# Inaugural-Dissertation

zur Erlangung der Doktorwürde
der Naturwissenschaftlich-Mathematischen Gesamtfakultät
der Ruprecht-Karls-Universität Heidelberg

vorgelegt von
Diplom-Mathematiker Markus Speth
aus Ravensburg

Tag der mündlichen Prüfung: 23. Juli 2014

# Exact Solutions for Discrete Graphical Models: Multicuts and Reduction Techniques

Betreuer:
Prof. Dr. Gerhard Reinelt
Prof. Dr. Christoph Schnörr

## Abstract

In the past years, discrete graphical models have become a major conceptual tool to model the structure of problems in image processing – example applications are image segmentation, image labeling, stereo vision, and tracking problems. It is therefore crucial to have techniques which are able to handle the occurring optimization problems and to deliver good solutions. Because of the hardness of these inference problems, so far mainly fast *heuristic* methods were used which yield *approximate* solutions.

In this thesis we present *exact* methods for obtaining *optimal* solutions for the energy minimization problem of discrete graphical models; image segmentation serves as the main application. Since these problems are $\mathcal{NP}$-hard in general, it is clear that in order to be able to handle problem sizes occurring in real-world applications one has to either (a) reduce the size of the problems or (b) restrict oneself to special problem classes. Concerning (a), we develop a combination of existing and new preprocessing steps which transform models into equivalent yet less complex ones. Concerning (b), we introduce the so-called multicut approach to image analysis: This is a generalization of the min $s$-$t$ cut method which allows for solving models of a certain structure significantly faster than previously possible or even solving them to global optimality for the first time at all. On the whole, we present methods which solve $\mathcal{NP}$-hard problems to proven optimality and which in some cases are as fast or even faster than approximative methods.

## Zusammenfassung

In den letzten Jahren entwickelten sich diskrete graphische Modelle zu einem grundlegenden Hilfsmittel, um die Struktur in der Bildverarbeitung auftretender Probleme zu modellieren – Beispielanwendungen sind etwa Bildsegmentierung, Bildlabeling, Stereosehen und Tracking-Probleme. Es ist daher essentiell, über Techniken zu verfügen, die mit den auftretenden Optimierungsproblemen umgehen und gute Lösungen liefern können. Aufgrund der Schwere dieser Probleme wurden bisher hauptsächlich *heuristische* Verfahren eingesetzt, die *Näherungslösungen* liefern.

In der vorliegenden Arbeit präsentieren wir *exakte* Methoden, um *optimale* Lösungen des Energieminimierungsproblems diskreter graphischer Modelle zu erhalten; unsere Hauptanwendung ist dabei die Bildsegmentierung. Da die Probleme im Allgemeinen $\mathcal{NP}$-schwer sind, ist es klar, dass man, um in der Praxis auftretende Problemgrößen behandeln zu können, entweder (a) die Größe der Probleme reduzieren oder (b) sich auf spezielle Problemklassen beschränken muss. Hinsichtlich (a) entwickeln wir eine Kombination von existierenden und neuen Vorverarbeitungsschritten, die ein Modell in ein äquivalentes, aber weniger komplexes umwandeln. Bezüglich (b) führen wir den sogenannten Multicut-Ansatz in die Bildverarbeitung ein: Dabei handelt es sich um eine Verallgemeinerung des Min-$s$-$t$-Cut-Verfahrens, die es ermöglicht, Probleme einer gewissen Struktur signifikant schneller als bisherige Methoden oder sogar erstmals überhaupt global optimal zu lösen. Insgesamt präsentieren wir Methoden, die $\mathcal{NP}$-schwere Probleme beweisbar optimal lösen und die in manchen Fällen genauso schnell oder sogar schneller sind als Näherungsverfahren.

## Acknowledgments

Above all, I want to thank my supervisor Prof. Gerhard Reinelt. He aroused my interest in combinatorial optimization and offered me the possibility to do my PhD in his group. I got to know him as a supportive and reliable person, taking care of the students under his guidance.

My second supervisor, Prof. Christoph Schnörr, introduced me to the field of image analysis in general and graphical models in particular. I am thankful to him for offering a complementing perspective from the viewpoint of application.

I owe special gratitude to Jörg Kappes for his help throughout the past years – he always had an open door for me and was willingly sharing his experience. Thorsten Bonato and Stefan Wiesberg helped me to get Thorsten's max cut code running with my instances. The evaluation of the superpixel algorithms was done in collaboration with Borislav Antic. Stefan Lörwald provided support with the new version of PORTA. Thanks to all of you!

Being in between two research groups, I had a lot of people helping me with administrative tasks. Our secretaries Catherine Proux-Wieland, Anke Sopka, Karin Tenschert, Barbara Werner, and Evelyn Wilhelm were taming the bureaucratic beasts to make life easier. Thanks to our system administrators who were responsible for keeping our machines running: Georgios Nikolis, Adrian Dempwolff, Lucas Appelmann, Klaus Hornung, and Jacqueline Hammer.

The members of our research group – Achim Hildenbrandt, Thomas Metz, Tuan Nam Nguyen, Marcus Oswald, Hanna Seitz, Khoa Vo, Pei Wang, and Stefan Wiesberg – provided a great and relaxed working atmosphere and, as well as the members of the groups "Database System Research", "Parallel and Distributed Systems", and "Image & Pattern Analysis", enriched working days with discussions about scientific and non-scientific topics.

I had the pleasure to be a member of the Research Training Group RTG 1653 "Spatio/Temporal Probabilistic Graphical Models and Applications in Image Analysis". I am grateful to the German Research Foundation (DFG) for this support and to all colleagues associated to the group for being so helpful. Especially, I want to thank my fellow PhD students Borislav Antic, Fabian Bachl, Eva-Maria Didden, Johannes Dueck, Dominic Edelmann, Jochen Fiedler, Bernhard Kausler, Gabriell Máté, Annette Möller, Fabian Rathke, and Bernhard Schmitzer – I definitely would have missed something without you.

For proofreading parts of this thesis I thank Stefan Wiesberg, Jörg Kappes, and Kira Augenstein – your comments were indeed helpful and improved this work. Of course, all remaining errors are my own.

My parents Monika and Josef Speth deserve my heartfelt thanks for their unconditional support in all situations.

Finally, I thank Kira Augenstein for her loving support.

# Contents

# List of Figures

# List of Tables

# List of Algorithms

# 1 Introduction

## 1.1 Overview and Motivation

In this thesis, we deal with the following general problem: Given a discrete graphical model $\mathcal{M} = (\mathcal{G}, X, \theta)$, find a labeling $x \in X$ that has minimal energy, i.e., solve the MAP problem

$$\min_{x \in X} \sum_{f \in \mathcal{F}} \theta(x_{\mathrm{nb}(f)}),$$

see Chap. 3 for exact definitions.

This problem is $\mathcal{NP}$-hard in general and many important problems from combinatorial optimization can be modeled as a MAP problem. We will see this in Sec. 3.2 for the satisfiability problem and the max cut problem, but it is also possible for the traveling salesman problem, quadratic Boolean optimization, and others. Due to its generality it found various applications, most prominently in the field of computer vision and image analysis. Image segmentation, image labeling, stereo vision, tracking problems – all can be modeled as an energy minimization problem in a graphical model.

Image segmentation is a fundamental problem of image analysis and will serve as our main application. In Fig. 1.1, an example of a segmentation is shown.

The image segmentation problem itself is not well-defined since several “correct” segmentations may exists. We here do not deal with the problem of obtaining data from the input images by means of feature extraction – we assume the data is given and we have to solve the problem at hand.

The methods from combinatorial optimization we apply here are linear and integer

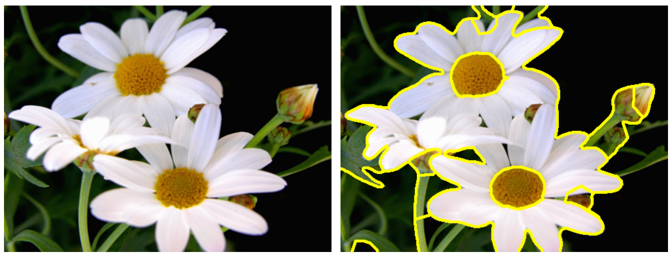


Figure 1.1: Example of an image segmentation: On the left the input image, on the right a segmentation of the image into meaningful segments.

linear programming techniques and cutting-plane procedures, and all is done in view of applicability for real-world problems.

## 1.2 Related Work

Dating back to the work of Schlesinger [Sch76] the marginal polytope and the local polytope relaxation are an important research area [WJ08]. Algorithms working in the dual domain have been suggested, either block-coordinate descent methods [Wer10; Kol06] or subgradient or bundle methods [KPT11; KSS12]. Even if these methods solve the dual, they have to reconstruct a relaxed primal solution and then round this to a solution of the original integer program.

Move-making methods build another type of algorithms, the most popular being $\alpha$-expansion [BVZ01], $\alpha$-$\beta$-swap [BVZ01], and FastPD [KT07].

In the last years combinatorial methods based on cutting-plane and branch-and-bound techniques have come more into focus in the computer vision community [ABK12a; OD11; FSW11; STL+12]. The main advantage of these methods is that they provide globally optimal integer solutions if no runtime restrictions are specified.

Concerning the multicut problem we will deal with, relevant work on this topic are recent publications dealing with closedness constraints for image segmentation [AKB+11; AKB+12b], contour completion [MLH12], ensemble segmentation [AG12; MLH12], and the convex hull of feasible multicuts from the optimization point of view [KNK+13; YIF12].

The multiway cut problem [CR91], which we will treat as a special case of the multicut problem, has been dealt with in [DJP+94], [DJP+92], and [Mar12].

A more detailed overview over the most important algorithms in given in Sec. 3.5.

## 1.3 Contribution

Our main contributions of this thesis are the following:

- We propose a set of preprocessing steps, which, when combined, lead to significant speedups. Random binary grid instances containing more than 4 million variables can be solved in 90 seconds. Problems from public datasets become solvable to optimality for the first time at all. (Sec. 4.1, Sec. 4.2)
- We provide a fair and neutral comparison of up-to-date superpixel algorithms that was not available before. The methods are tested on a public dataset and evaluated using standard measures. (Sec. 4.4)
- We introduce the multicut approach to computer vision and show that label permutation invariant functions are relevant for applications since they generalize

the popular Potts functions. These expose symmetries and we show that exploiting this structure pays off. Using the multicut algorithm, solving problems of this type is feasible while other approaches fail. (Chap. 5)

- All methods presented in the thesis are extensively evaluated on various datasets and compared to state-of-the-art algorithms. For the multicut cutting-plane algorithm also several relaxations are examined. (Chap. 6)

## 1.4 Organization

This thesis is organized as follows:

We start in Chap. 2 by giving basic definitions of concepts used in the whole thesis, ranging from graph theory to polyhedral theory to integer linear programming.

In Chap. 3 we give an overview on discrete graphical models, starting with their definition, special cases, and transformations. We show that the energy minimization problem (also known as MAP problem) in a graphical model is $\mathcal{NP}$-hard in general and explain its connection to cut problems. Furthermore, we discuss the formulation of the MAP problem of general graphical models as a linear program and as an integer linear program. After a brief section on the origins of graphical models, we end the chapter with an overview of the most important algorithms for solving the energy minimization problem.

Chap. 4 is devoted to techniques for reducing the complexity of graphical models which is advisable due to the $\mathcal{NP}$-hardness of the corresponding optimization problem. We show how preprocessing steps can be combined to make large problems tractable, together with experiments proving the effectiveness of the proposed methods. Additionally, for our main application image segmentation, we present an overview of state-of-the-art superpixel algorithms and provide an extensive comparison of them using various metrics on a benchmark dataset.

In Chap. 5, we introduce the multicut algorithm for solving the energy minimization problem in special graphical models. The models for which this approach is suited contain factors with label permutation invariant functions: Their energy does not depend on the labeling itself but only on the partition it induces. Therefore, permuting the labels does not change their function value. We show how second-order models can be modeled as a multicut problem in which variables $y_e$ for each edge $e$ of the multicut graph are used. In this graph, every edge corresponds to a pairwise (or unary) factor. In Sec. 5.3 we extend the framework to higher-order models and show how corresponding higher-order factors can be taken into account in a memory-efficient way by exploiting symmetries. We detail separation procedures for finding violated constraints in Sec. 5.4 and show how they can be implemented efficiently. Rounding mechanisms will be discussed in Sec. 5.4.3. We conclude the framework with our cutting-plane method presented in Sec. 5.4.4.

Extensive computational experiments are provided in 6, both for the reduction techniques of Chap. 4 and for the multicut approach of Chap. 5. We use various publicly available datasets to test our methods. Thanks to the reduction techniques, several problems become solvable to optimality that were not tractable before. For the multicut problems, we compare our algorithm to several standard solution methods. We included models of second-order as well as higher-order models, both for the supervised and unsupervised case.

Parts of this thesis have already been published in the following papers:

[KSA+11] Jörg H. Kappes, Markus Speth, Björn Andres, Gerhard Reinelt, and Christoph Schnörr. Globally optimal image partitioning by multicuts. In *Proceedings of the International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, 2011

[KSR+13a] Jörg H. Kappes, Markus Speth, Gerhard Reinelt, and Christoph Schnörr. Higher-order segmentation via multicuts. `http://arxiv.org/abs/1305.6387`. 2013

[KSR+13b] Jörg H. Kappes, Markus Speth, Gerhard Reinelt, and Christoph Schnörr. Towards efficient and exact MAP-inference for large scale discrete computer vision problems via combinatorial optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013

To be more precise:

- Sec. 4.1 and Sec. 4.2 contain material from [KSR+13b].
- Ch. 5 contains material from the multicut papers [KSA+11] and [KSR+13a].
- The results of the experiments and applications of Ch. 6 have already been published in [KSR+13b] and [KSR+13a].

# 2 Background

In this chapter we present basic concepts needed throughout this thesis. Most of them are standard and can be found in any textbook of the respective field. Our reasons for nonetheless including them here are twofold: First, to introduce our notational conventions, and second, to make the present thesis more self-contained.

## 2.1 General Notation

Sets will usually be denoted by capital letters, whereas small letters will be used for vectors and scalars. If $B$ is a *subset* of $A$ this is denoted by $B \subseteq A$, if the inclusion is *proper* we write $B \subset A$. The *power set*, i.e., the set of all subsets, of a set $A$ is denoted by $\mathcal{P}(A)$. The set of all subsets of a fixed size $k$ is denoted by $\binom{A}{k}$. A *partition* of a set $A$ is a collection of non-empty, pairwise disjoint sets $\{A_1, \ldots, A_n\}$ with $\bigcup_{i=1}^{n} A_i = A$. We denote the set of natural numbers (including zero) by $\mathbb{N}$, the set of real numbers is denoted by $\mathbb{R}$. set partition

A vector will usually be a column vector. If $a$ is a column vector, its transpose, i.e., the corresponding row vector, will be denoted by $a^T$ and vice versa. For two vectors $a = (a_1, \ldots, a_n)^T$ and $b = (b_1, \ldots, b_n)^T$, we denote their *inner product* by $a^T b := \sum_{i=1}^{n} a_i b_i$. The vector of all zeros and the vector of all ones of appropriate dimensions are denoted by $\mathbf{0}$ and $\mathbf{1}$, respectively. inner product

For a logical expression $\tau$ we define an *indicator function* $\mathbb{I}[\tau]$ that is 1 if $\tau$ is true and 0 otherwise. indicator function

When examining the growth of a function, we are often only interested in the order of its growth. For this we use the *Landau notation*: For two functions $f, g : \mathbb{R} \to \mathbb{R}$, we write $f \in \mathcal{O}(g)$ if there exists a $c > 0$ and an $x_0 \in \mathbb{R}$ such that $|f(x)| \leq c|g(x)|$ for all $x \geq x_0$. Landau notation

## 2.2 Graph Theory

We will define our problems on graphs, which are well-suited as mathematical representations of objects and their relationships. We here introduce those concepts from graph theory that we will need later on. The definitions are mostly taken from the introductory chapters of [Die97] and [Bol98], which are excellent references for further reading.

### 2.2.1 Graphs

graph A *graph* $G = (V, E)$ is a pair of two disjoint finite sets $V$ and $E$ such that $E \subseteq \binom{V}{2}$. The elements $v \in V$ are called *nodes* (or *vertices*) and the elements $e \in E$ are called *edges*. The set of all nodes and the set of all edges of a graph $G$ are denoted by $V(G)$ and $E(G)$, respectively. Most of the time, an edge $e = \{u, v\}$ will simply be denoted by $uv$, so $e = uv = vu$. For $e = uv \in E$ the nodes $u$ and $v$ are called the *endnodes* of $e$. Two nodes $u$ and $v$ are *adjacent* if $uv \in E$ and a node $v$ is *incident* to an edge $e$, and vice versa, if $v \in e$. The *neighborhood* of a node $v$ is the set $\mathrm{nb}(v) := \{u \in V \mid uv \in E\}$ and the cardinality of $\mathrm{nb}(v)$ is called the *degree* of $v$.

neighborhood

degree

isomorphic Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there exists a bijection $\varphi : V \to V'$ such that $uv \in E$ if and only if $\varphi(u)\varphi(v) \in E'$ for all $u, v \in V$. We do not distinguish between isomorphic graphs, i.e., we write $G = G'$.

subgraph A graph $G' = (V', E')$ is a *subgraph* of a graph $G = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. In this case, we also say that $G$ *contains* $G'$. If additionally $V' = V$ then $G'$ is said to be a *spanning* subgraph of $G$.

### 2.2.2 Paths, Cycles, and Connectivity

path A *path* between two nodes $v_0$ and $v_n$ in a graph $G$ is a subgraph $P = (V_P, E_P)$ of $G$ where $V_P = \{v_i \mid 0 \leq i \leq n\}$ and $E_P = \{v_i v_{i+1} \mid 0 \leq i \leq n-1\}$. The *length* of a path $P$ is $|E(P)|$. A path that has the smallest length among all paths between two nodes $u$ and $v$ is called a *shortest path* between them. If such a shortest path exists, we call its length the *distance* of $u$ and $v$, otherwise, we say their distance is $\infty$.

shortest path

distance

cycle A *cycle* in a graph $G$ is a subgraph $C = (V_C, E_C)$ of $G$ where $V_C = \{v_i \mid 1 \leq i \leq n\}$, $n \geq 3$, and $E_C = \{v_i v_{i+1} \mid 1 \leq i \leq n-1\} \cup \{v_1 v_n\}$. The *length* of a cycle $C$ is $|E(C)|$. If a graph $G$ contains a cycle $C$, then an edge $uv \in E(G)$ is a *chord* of $C$ in $G$ if $u, v \in V(C)$ and $uv \notin E(C)$. A cycle contained in a graph $G$ is *chordless* if it has no chord in $G$.

chord

connected Two nodes in a graph are *connected* if the graph contains a path between them. We call a graph *connected* if there exists a path between $u$ and $v$ for all nodes $u, v \in V$, otherwise it is called *disconnected*. Maximal connected subgraphs of a graph are called its *(connected) components*.

### 2.2.3 Cuts and Multicuts

cut For a graph $G = (V, E)$ and a subset of the nodes $S \subseteq V$, we define a *cut* of $G$ by $\delta(S) := \{uv \in E \mid u \in S, v \in V \setminus S\}$. If $s \in S$ and $t \in V \setminus S$, we call $\delta(S)$ an *$s$-$t$-cut* of $G$. We extend this definition to $\delta(S_1, \dots, S_n) := \{uv \in E \mid u \in S_i, v \in S_j, 1 \leq i < j \leq n\}$ for disjoint sets $S_i \subseteq V$, $1 \leq i \leq n$ with $n \geq 2$. If $\{S_1, \dots, S_n\}$ is a partition of $V$, the set $\delta(S_1, \dots, S_n)$ is called a *multicut* of $G$ and the sets $S_1, \dots, S_n$ are the *shores* of the multicut. Since the cut $\delta(S)$ can also be written as $\delta(S, V \setminus S)$ if $\emptyset \neq S \neq V$ it is therefore also a multicut. Additionally, the empty set $\emptyset$ is also a multicut.

$s$-$t$-cut

multicut

### 2.2.4 Special Classes of Graphs

A graph $G = (V, E)$ is called *complete* if $E = \binom{V}{2}$, i.e., if all pairs of nodes are adjacent. If a graph contains no cycles it is called *acyclic* or a *forest*; it is a *tree* if it is a connected forest. If there is a partition $\{V_1, V_2\}$ of $V$ such that $E \subseteq V_1 \times V_2$ then $G$ is *bipartite*. complete tree bipartite

We define planarity only informally: A graph is called *planar* if it is possible to draw it in the plane such that no edges are crossing. For a more formal definition, we refer to [Die97]. planar

A *grid graph* is a graph $G = (V, E)$ with $V = \{(i, j) \mid 1 \leq i \leq n_1, 1 \leq j \leq n_2\}$ and $E = \{(i, j)(i + 1, j) \mid 1 \leq i < n_1, 1 \leq j \leq n_2\} \cup \{(i, j)(i, j + 1) \mid 1 \leq i \leq n_1, 1 \leq j < n_2\}$; its size is $n_1 \times n_2$. grid graph

A *weighted graph* is a tuple $G = (V, E, w)$ where $(V, E)$ is a graph which has an associated *weight function* $w : E \to \mathbb{R}$ which assigns a *weight* $w(e)$ to each edge $e$ of the graph. In a weighted graph, we define the *length* of a path to be the sum of the edge weights of all edges of the path. Accordingly, the definitions of *shortest path* and *distance* change. We extend the domain of the weight function $w$ in a natural way to arbitrary sets of edges: For $E' \subseteq E$, we define $w(E') := \sum_{e \in E'} w(e)$ and call this value the weight of $E'$. weighted graph

A *hypergraph* $H = (V_H, E_H)$ is a pair of two disjoint finite sets $V_H$ and $E_H$ such that $E_H \subseteq \mathcal{P}(V_H)$. This is a generalization of ordinary graphs: Here, the elements of $E_H$, the *hyperedges*, are subsets of $V_H$ of arbitrary cardinality. Again, a node $v \in V_H$ is incident to a hyperedge $e \in E_H$, and vice versa, if $v \in e$. Every hypergraph has an associated bipartite ordinary graph that reflects the incidence relations among the nodes and hyperedges. This *incidence graph* $G = (V, E)$ of a hypergraph $H = (V_H, E_H)$ is defined by $V := V_H \cup E_H$ and $E := \{\{v, e\} \mid v \in V_H, e \in E_H, v \in e\}$. hypergraph incidence graph

## 2.3 Algorithms and Complexity

To compare the efficiency of algorithms, a measure of their quality is needed. In this section we briefly introduce some basic concepts from complexity theory, most importantly the notion of $\mathcal{P}$ and $\mathcal{NP}$. A very thorough work on this topic is the seminal book by Garey and Johnson [GJ79]. For this overview we used the corresponding chapters of [GJ79], [Sch03], and [KV00].

### 2.3.1 Algorithms

Although a precise definition of an algorithm is possible by using the concept of Turing machines (or an equivalent machine model), we refrain here from doing so due to the involved formalism. We therefore define an *algorithm* $A$ only in an informal way as a set of instructions which transforms some input data $X$ into output data $A(X)$. The set of algorithm instructions is required to be finite and each instruction has to consist of a sequence of simple and elementary steps.

To be able to handle different kinds of input data uniformly, we will encode it: Let $\Sigma$ be an *alphabet*, i.e., a finite set, with $|\Sigma| \geq 2$ and let $\Sigma^*$ denote the set of all *words* over $\Sigma$, i.e., the set of all finite sequences consisting of elements of $\Sigma$. For a word $x \in \Sigma^*$ we denote by $|x|$ the *length* of $x$. An *encoding scheme* enc is a mapping that transforms an arbitrary object $X$ into a representation as a word $x \in \Sigma^*$, so $\mathrm{enc} : X \mapsto x$. Since an exact specification of the encoding scheme is not necessary as long as it is reasonable (which means that the length of the encoding has to be bounded by a polynomial in the length of the shortest encoding possible), we assume in the following an arbitrary but fixed reasonable encoding scheme. The *input size* of an input $X$ is the length of its encoding $|\operatorname{enc}(X)|$.

alphabet

encoding scheme

input size

time complexity

The *time complexity* $\mathrm{time}_A : \mathbb{N} \to \mathbb{N}$ of an algorithm $A$ is the number of elementary steps that it carries out at most during its execution and is given in terms of the input size:

$$\mathrm{time}_A(n) := \max \left\{ m \;\middle|\; \begin{array}{l} \text{there is an input } X \text{ with input size } n \text{ such that } A \\ \text{needs } m \text{ elementary steps to compute } A(X) \end{array} \right\}.$$

If there exists a polynomial $p$ such that $\mathrm{time}_A(n) \leq p(n)$ for all $n \in \mathbb{N}$ then we call $A$ a *polynomial-time algorithm* or simply *polynomial*.

polynomial-time algorithm

### 2.3.2 Decision Problems and Optimization Problems

Informally, a decision problem is a problem that can be answered by “yes” or “no”. To formalize this, we say that a *decision problem* is a set $\Pi \subseteq \Sigma^*$. The corresponding “yes”-“no” question is: Given an input $x \in \Sigma^*$, is $x \in \Pi$?

decision problem

An *optimization problem* is given by an arbitrary set $X$ and a function $f : X \to \mathbb{R}$ and asks for an optimal solution $x^*$, i.e., for an element $x^* \in X$ with $f(x^*) = \min_{x\in X} f(x)$ (or $f(x^*) = \max_{x\in X} f(x)$) if such an element exists. Every optimization problem can be transformed into a decision problem by attaching an additional parameter $K \in \mathbb{R}$ to its input and asking: “Is there an $x \in X$ with $f(x) \leq K$ (or $f(x) \geq K$)?”

optimization problem

### 2.3.3 $\mathcal{P}$ and $\mathcal{NP}$

If there exists a polynomial algorithm for a decision problem $\Pi \subseteq \Sigma^*$, i.e., if there exists an algorithm and a polynomial $p$ such that for all $x \in \Sigma^*$ the algorithm decides in at most $p(|x|)$ steps whether $x \in \Pi$ or not, then $\Pi$ is called *polynomially solvable*. The set of all polynomially solvable decision problems is denoted by $\mathcal{P}$.

polynomially solvable

For example, let $\Pi_{\mathrm{HC}}$ be the set of all graphs that contain a Hamiltonian cycle[1]

[1] A *Hamiltonian cycle* in a graph is a cycle that contains all nodes of the graph. A graph containing a Hamiltonian cycle is called a *Hamiltonian graph*.

Hamiltonian graph

(using an appropriate encoding). Since so far no polynomial-time algorithm is known that decides whether a given graph is Hamiltonian or not, it is not known whether $\Pi_{\text{HC}} \in \mathcal{P}$ or not.

The class $\mathcal{NP}$ consists of all decision problems $\Pi \subseteq \Sigma^*$ for which there is a decision problem $\Pi' \subseteq \Sigma^*$ with $\Pi' \in \mathcal{P}$ and a polynomial $p$ such that $x \in \Pi$ if and only if there exists a $c \in \Sigma^*$ with $|c| \leq p(|x|)$ such that $xc \in \Pi'$ for all $x \in \Sigma^*$. The word $c$ is called a *certificate* for $x$. certificate

The problem $\Pi_{\text{HC}}$ is contained in $\mathcal{NP}$: If $x$ is the encoding of a given Hamiltonian graph $G$, we can set $c$ to the encoding of a Hamiltonian cycle of $G$. This is a certificate for $x$ since it can be checked in polynomial time that it indeed encodes a Hamiltonian cycle of $G$.

By setting $c$ to the empty word, one can see easily that $\mathcal{P}$ is a subset of $\mathcal{NP}$. However, it is still an open question whether the classes $\mathcal{P}$ and $\mathcal{NP}$ are equal or not.

### 2.3.4 $\mathcal{NP}$-Completeness

A decision problem $\Pi \subseteq \Sigma^*$ is said to be *(polynomially) reducible* to a decision problem $\Pi' \subseteq \Sigma^*$ if there exists a polynomial algorithm $A$ such that for all $x \in \Pi$ the relation $x \in \Pi \Leftrightarrow A(x) \in \Pi'$ holds. As a consequence, if $\Pi$ is reducible to $\Pi'$ and $\Pi' \in \mathcal{P}$, then also $\Pi \in \mathcal{P}$. polynomially reducible

A decision problem is called *$\mathcal{NP}$-hard* if every problem in $\mathcal{NP}$ is reducible to it. If $\Pi \in \mathcal{NP}$ and $\Pi$ is $\mathcal{NP}$-hard then $\Pi$ is called *$\mathcal{NP}$-complete*. Informally, this means that $\mathcal{NP}$-complete problems are the "hardest" problems in $\mathcal{NP}$ and that $\mathcal{NP}$-hard problems are "at least as hard" as any problem in $\mathcal{NP}$. $\mathcal{NP}$-hard $\mathcal{NP}$-complete

A depiction of the relations between the different complexity classes can be seen in Fig. 2.1, both for the case $\mathcal{P} \neq \mathcal{NP}$ and $\mathcal{P} = \mathcal{NP}$.

## 2.4 Polyhedral Theory

The optimization problems we deal with in this thesis will be formulated as integer linear programs (as explained in Sec. 2.5). However, the methods for solving such problems cannot be understood without knowledge of some basic concepts of polyhedral theory. These will be given in this section which is based on the first pages of the books of Ziegler [Zie95] and Brøndsted [Brø83].

### 2.4.1 Inequalities and Hyperplanes

A set $C \subseteq \mathbb{R}^d$ is called *convex* if $\{\lambda x + (1-\lambda)y \mid 0 \leq \lambda \leq 1\} \subseteq C$ for all $x, y \in C$. convex

The *convex hull* of a finite set of points $X = \{x_1, \ldots, x_k\} \subseteq \mathbb{R}^d$ is given by the set $\operatorname{conv}(X) := \left\{\sum_{i=1}^k \lambda_i x_i \mid \lambda_i \geq 0 \text{ for } 1 \leq i \leq k, \sum_{i=1}^k \lambda_i = 1\right\}$. One can show that convex hull

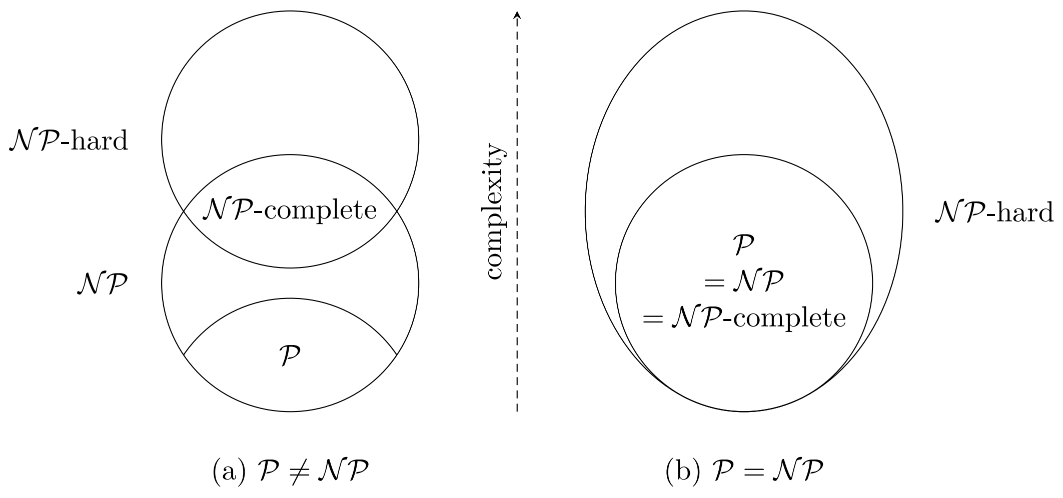


Figure 2.1: Schematic depiction of the various complexity classes for the two cases (a) $\mathcal{P} \neq \mathcal{NP}$ and (b) $\mathcal{P} = \mathcal{NP}$.

the convex hull of a set $X$ is the smallest convex set that contains $X$, which is the intersection of all convex sets $C \subseteq \mathbb{R}^d$ with $X \subseteq C$.

inequality — For $a \in \mathbb{R}^d$, $a \neq \mathbf{0}$, and $b \in \mathbb{R}$, an *inequality* is the logical expression $a^T x \leq b$ which depends on a variable $x \in \mathbb{R}^d$.

halfspace, hyperplane — Every inequality $a^T x \leq b$ naturally defines a *(closed) halfspace* $\{x \in \mathbb{R}^d \mid a^T x \leq b\}$, i.e., the set of points for which it is *valid*, and a *hyperplane* $\{x \in \mathbb{R}^d \mid a^T x = b\}$, i.e., the set of points for which it is *tight*.

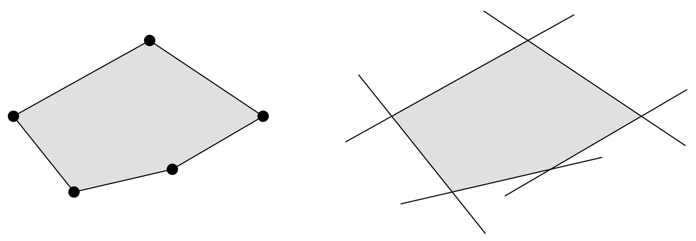
### 2.4.2 Polytopes

polytope, polyhedron, bounded — A *polytope* $P$ in $\mathbb{R}^d$ can be defined in two ways: A *$V$-polytope* is the convex hull of a finite set of points $X \subseteq \mathbb{R}^d$, i.e., $P = \operatorname{conv}(X)$. This is the so-called inner description of a polytope. An *$H$-polytope* is a bounded polyhedron: A *polyhedron* $P$ is the intersection of finitely many halfspaces, i.e., $P = \{x \in \mathbb{R}^d \mid a_i^T x \leq b_i, 1 \leq i \leq m\}$ where $a_i^T x \leq b_i$ are finitely many inequalities. A polyhedron $P$ is *bounded* if there is no $y \in \mathbb{R}^d$, $y \neq \mathbf{0}$, with $\{x + \lambda y \mid x \in \mathbb{R}^d, \lambda \geq 0\} \subseteq P$. This definition is called the outer description of a polytope. For $A \in \mathbb{R}^{m \times d}$ and $b \in \mathbb{R}^m$ we define

$$P(A, b) := \{x \in \mathbb{R}^d \mid Ax \leq b\}$$

which is an $H$-polytope where $a_i$ are the rows of $A$ and $b_i$ are the entries of $b$.

A central theorem of polyhedral theory, which is due to Minkowski [Min96] and Weyl [Wey35], states that $P \subseteq \mathbb{R}^d$ is a $V$-polytope if and only if it is an $H$-polytope.



(a) $V$-polytope (b) $H$-polytope

Figure 2.2: Both (a) and (b) show the same polytope: In (a) it is depicted as the convex hull of a finite set of points and in (b) as the intersection of finitely many halfspaces.

We will therefore simply denote them as polytopes in the following. A depiction of a $V$-polytope and an $H$-polytope can be seen in Fig. 2.2.

### 2.4.3 Faces and Facets

An inequality $a^T x \leq b$ is *valid* for a polytope $P$ if it is valid for all $x \in P$. Every valid inequality of a polytope $P$ defines a *face* $F := P \cap \{x \in \mathbb{R}^d \mid a^T x = b\}$ of $P$. Such a face $F$ is called *proper* if $F \neq P$ and $F \neq \emptyset$, it is called a *vertex* (or an *extreme point*) of $P$ if $|F| = 1$, and a *facet* if it is a proper face that is maximal with respect to inclusion. It can be shown that $P = \text{conv}(\{x \mid \{x\} \text{ is a vertex of } P\})$ for all polytopes $P \subseteq \mathbb{R}^d$.

face

vertex

facet

For a polytope $P$ and a valid inequality $a^T x \leq b$, the inequality is *facet-defining* if $P \cap \{x \in \mathbb{R}^d \mid a^T x = b\}$ is a facet. Facet-defining inequalities are the most important ones: Every (full-dimensional) polytope is the intersection of the halfspaces defined by its facet-defining inequalities and this representation is minimal with respect to the number of halfspaces.

facet-defining

## 2.5 Integer Linear Programming

In this section we will discuss integer linear programs (ILPs), which are a powerful tool to solve optimization problems of a certain kind. Since the emergence of this theory in the 1940s, integer linear programming has developed quickly and was used in many different areas such as production planning, scheduling, and network design.

Before discussing ILPs, we will cover linear programs (LPs), not only because both are strongly related but also because LPs are necessary for solving ILPs.

An in-depth treatment of this topic can be found in the books of Schrijver [Sch86] and Korte and Vygen [KV00].

### 2.5.1 Linear Programming

linear program (LP)

A *linear program (LP)* is an optimization problem of the following type: Given a matrix $A \in \mathbb{R}^{m\times n}$ and vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, find a vector $x \in \mathbb{R}^n$ with $Ax \leq b$ such that $c^Tx$ is minimal or decide that no such vector exists. We will usually write an LP in the form

$$\begin{aligned} &\min c^T x \\ \text{s.t.}\quad &Ax \leq b. \end{aligned}$$

The definition given above specifies the so-called standard form of an LP. Linear programs in non-standard form can for example ask for maximization instead of minimization, can contain equality restrictions instead of only inequality restrictions, or can require a sign for some variables. However, all these variants can be transformed into standard form.

constraint
feasible region
feasible
objective function
optimal solution
optimal value

For a linear program given by $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and $c \in \mathbb{R}^n$, the inequalities $a_i x \leq b_i$, where $a_i$ is the $i$-th row of $A$ and $b_i$ is the $i$-th entry of $b$, are called *(linear) constraints*. The set $P(A, b) = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ is called the *feasible region* and its elements the *feasible points* of the LP. Note that by definition the feasible region of an LP is a polytope, so linear programming deals with minimizing a linear function over a polytope. If the feasible region is non-empty, then the LP is called *feasible*, it is *infeasible* otherwise. The function $x \mapsto c^T x$ is the *objective function* of the LP. If for every $\alpha \in \mathbb{R}$ there exists a feasible point $x$ with $c^T x > \alpha$, then the LP is called *unbounded*. An *optimal solution* of an LP is a feasible point $x^* \in \mathbb{R}^n$ such that $c^T x^* \leq c^T x$ for all feasible points $x$. For an optimal solution $x^* \in \mathbb{R}^n$, the value $c^T x^*$ is called the *optimal value* of the LP. It can be shown that if the LP is neither infeasible nor unbounded, there always exists an optimal solution.

An important observation is that if an LP has an optimal solution, then there always exists an optimal solution which is a vertex of the feasible region. Additionally, for an optimal solution $x^*$, there always exists a facet $F$ of the feasible region such that $x^* \in F$.

### 2.5.2 Solution Methods for LPs

There are basically three methods for solving linear programs – the simplex method, the ellipsoid method, and the interior point method. We shortly describe them:

simplex method

**Simplex Method** The *simplex method* was developed by Dantzig in 1947 and published in 1951 [Dan51]. It was the first algorithm that was able to solve arbitrary linear programs to optimality and for a long time remained the only one. It first determines a vertex of the feasible region of the LP. Starting with this initial solution, it then generates a sequence of vertices by traversing the feasible region from the current vertex to a neighboring one with a better objective function value. If this is not possible anymore, an optimal solution has been found. Several variants of the simplex method are possible, but all of them analyzed so far have exponential runtime in the worst case. However, on average and in practice its runtime is polynomial.

**Ellipsoid Method** In 1979, Khachiyan was able to prove that linear programs belong to the class of polynomially solvable problems by adapting a method developed shortly before for nonlinear optimization. His *ellipsoid method* [Kha79] constructs a series of ellipsoids, each containing at least one optimal solution. The ellipsoids are decreasing in size, and after a polynomial number of steps they are small enough to derive an optimal solution. Although the method has polynomial runtime, it is too inefficient to be used in practice.

ellipsoid method

**Interior Point Method** Shortly after the ellipsoid method, another algorithm for linear programs was developed – the *interior point method* of Karmarkar [Kar84] is not only polynomial, but also applicable in practice. As its name suggests, it does not generate a tour along the vertices of the feasible region as the simplex method but a sequence of points in the interior of the polytope which converges to an optimal solution.

interior point method

For the purpose in this thesis, the simplex method – despite its theoretical complexity – is the fastest and therefore still the most widely used method today due to its additional properties that can be exploited. However, in general the interior point method is also competitive and and sometimes even faster for solving LPs.

### 2.5.3 Integer Linear Programming

An *integer linear program (ILP)* can be seen as a simple extension of a linear program. Again, we are given a matrix $A \in \mathbb{R}^{m\times n}$ and vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$, again we want to find a vector $x$ which satisfies $Ax \leq b$ and maximizes $c^T x$, however, in the case of an ILP we require $x$ to be integral. The standard form of an ILP is therefore

integer linear program (ILP)

$$\begin{aligned} &\min c^T x \\ \text{s.t.}\quad &Ax \leq b \\ &x \in \mathbb{Z}^n, \end{aligned}$$

where the requirements $x_1 \in \mathbb{Z}, \dots, x_n \in \mathbb{Z}$ are called *integrality constraints*.

integrality constraint

Despite their similarity to linear programs, ILPs are considerably harder: General ILPs are known to be $\mathcal{NP}$-complete. The decision version of an ILP was among the first set of problems for which $\mathcal{NP}$-hardness was shown [Kar72].

Integer linear programs are a quite general tool to model optimization problems. Virtually all combinatiorial problems can be formulated as an ILP, and many of them have a very natural formulation.

Most of the terms we defined for LPs can directly be applied for ILPs, with the exception of the feasible region: For ILPs the *feasible region* is naturally defined

feasible region

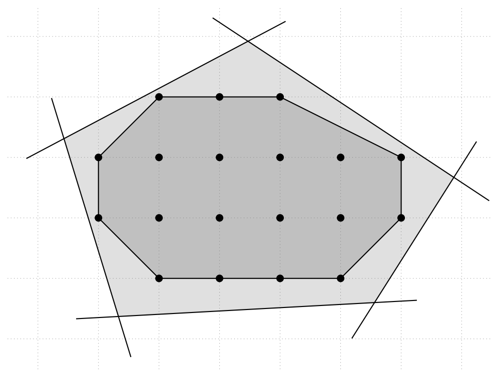


Figure 2.3: The integer hull $P_I(A,b)$, which is the convex hull of the feasible region of the ILP, is contained in $P(A,b)$.

as $\{x \in \mathbb{Z}^n \mid Ax \leq b\}$. If $P(A,b)$ is bounded, the feasible region of the ILP is finite. In this case, we can define its *integer hull* $P_I(A,b)$ as

integer hull

$$P_I(A,b) := \text{conv}(\{x \in \mathbb{Z}^n \mid Ax \leq b\}),$$

i.e., as the convex hull of the feasible region. This is depicted in Fig. 2.3. By definition, the integer hull is a polytope which means that the problem $\min \{c^T x \mid x \in P_I(A,b)\}$ is a linear program. Since we know that if an LP has an optimal solution, there also exists an optimal solution which is a vertex of its feasible region, we can conclude that

$$\min \{c^T x \mid Ax \leq b, x \in \mathbb{Z}^n\} = \min \{c^T x \mid x \in P_I(A,b)\}.$$

Thus, we have written our integer linear program (left-hand side) as a linear program (right-hand side). Although linear programs can be solved in polynomial time as we have seen in Sec. 2.5.2, this does not mean that we can solve the ILP in polynomial time. The description of $P_I(A,b)$ requires exponentially many halfspaces which means that the input size of the LP is already exponentially large.

### 2.5.4 Solution Methods for ILPs

Before describing two methods for solving integer linear programs (the cutting-plane procedure and the branch-and-bound algorithm), we now introduce the notion of a relaxation.

relaxation

In general, a *relaxation* of an optimization problem $\min_{x \in X} f(x)$ is also an optimization problem $\min_{x \in \bar{X}} f(x)$ with $\bar{X} \supseteq X$. An immediate consequence is that if $x^*$ is an optimal solution of the original problem and $\bar{x}^*$ is an optimal solution of the relaxed problem, then

$$f(\bar{x}^*) \leq f(x^*),$$

i.e., solving a relaxation yields a lower bound for the original problem. A relaxation can be used for its own sake, i.e., only for obtaining a lower bound, or as a subproblem in an algorithm for the original problem where the bounds are used in the further computations. The motivation for considering relaxations is of course the assumption that the relaxed problem is easier to solve than the original one. For an ILP

$$\min \{c^T x \mid Ax \leq b, x \in \mathbb{Z}^n\}$$

a natural relaxation is the *LP-relaxation*

LP-relaxation

$$\min \{c^T x \mid Ax \leq b, x \in \mathbb{R}^n\}$$

which is widely used. Other obvious relaxations include problems of the form

$$\min \{c^T x \mid A'x \leq b', x \in \mathbb{Z}^n\}$$

where $A'x \leq b'$ is a subsystem of $Ax \leq b$, i.e., where some of the inequality constraints of the original ILP were left out.

**Cutting-Plane Procedure** We now describe the *cutting-plane procedure*: When solving an ILP, we can start with solving a relaxation of the problem. If the obtained solution $\bar{x}^*$ satisfies all constraints of the ILP it is also the optimal solution of the original problem. However, this will usually not be the case. The basic idea of a cutting-plane procedure is to then derive a constraint which is not satisfied by $\bar{x}^*$ but which is valid for all feasible points of the ILP. This constraint is then added to the relaxation and the process is iterated. As soon as all constraints of the ILP are valid for $\bar{x}^*$ we are done.

cutting-plane procedure

The constraint (when it is an inequality) which is added to the relaxation is called a *cutting-plane* since it “cuts” $\bar{x}^*$ from $P_I(A, b)$; the process of finding this cutting-plane is called *separation*.

cutting-plane
separation

In the original cutting-plane procedure developed in the 1950s by Gomory [Gom58], the initial relaxation is the LP-relaxation. This original procedure is outlined in Alg. 1. The crucial point is clearly the separation step: It can be shown that in general the separation is polynomially solvable if and only if the original problem is polynomially solvable. Gomory was able to specify an easy way to generate such cutting-planes which guarantees a finite number of iterations until an integral solution is found. In practice, however, this number can be quite large and when used in its pure version numerical problems can occur. Nonetheless, Gomory cuts are used in many algorithms and off-the-shelf solvers.

**Branch-and-Bound** The *branch-and-bound* technique is a general approach for solving hard optimization problems. As its name suggests, it consists of two main steps: branching and bounding.

branch-and-bound

To simplify the notation in the following paragraphs, we will identify an optimization problem $\min_{x \in X} f(x)$ with its domain $X$ (i.e., in case of an LP or an ILP with its

**Algorithm 1** Cutting-Plane Procedure

```
1: function CUTTING-PLANE($A, b$)
2:     $\bar{A} \leftarrow A$, $\bar{b} \leftarrow b$
3:     compute $\bar{x}^* \in \text{argmin}\{c^T x \mid x \in P(\bar{A}, \bar{b})\}$
4:     if $\bar{x}^* \notin P_I(A, b)$ then
5:         find $a \in \mathbb{R}^n$ and $b_0 \in \mathbb{R}$ with $a^T \bar{x}^* > b_0$ and $a^T x \leq b_0$ for all $x \in P_I(A, b)$
6:         $\bar{A} \leftarrow (\bar{A}, a)^T$, $\bar{b} \leftarrow (\bar{b}, b_0)^T$
7:         goto 3.
8:     return $\bar{x}^*$
```

feasible region) since we assume that $f$ is the same for all problems we consider here. Furthermore, we here use the shorthand notation $\bar{X}$ to denote a relaxation of $X$.

separation

For an optimization problem $X$, a *separation* of $X$ is a set of optimization problems $X_i$, $i \in I$, with $\bigcup_{i \in I} X_i = X$ where $I$ is a finite index set. This splitting of an optimization problem into a separation is called branching. To solve $X$, we can also solve $X_i$, $i \in I$, since obviously it holds that $\min_{x \in X} f(x) = \min_{i \in I}(\min_{x \in X_i} f(x))$.

The main idea of the algorithm is to reduce the computational effort by not having to solve all $X_i$ but only some of them without losing optimality. This is done by the bounding step: Let $x'$ be a feasible solution of our problem $X$ which yields an upper bound $U$, i.e., $\min_{x \in X} f(x) \leq U = f(x')$. For a relaxation $\bar{X}_i$ of $X_i$ it obviously holds that $\min_{x \in \bar{X}_i} f(x) \leq \min_{x \in X_i} f(x)$. Now assume that $U \leq \min_{x \in \bar{X}_i} f(x)$: In this case, it follows that $U \leq \min_{x \in X_i} f(x)$ which means that we do not have to solve $X_i$ at all since we know that we cannot obtain a better solution than $x'$ – it was sufficient to solve the (easier) relaxation $\bar{X}_i$.

The exact procedure is described in Alg. 2. Of course one has to make sure that the algorithm terminates by choosing a reasonable procedure for both the branching and the bounding step. Additionally, also the choice of the active problem influences the quality of the algorithm.

For solving ILPs, this method was described by Land and Doig [LD60] and by Dakin [Dak65]. As the bounding step they use the LP-relaxation of the current problem. In the branching step, they choose an index $j$ such that in the solution $\bar{x}^*$ of the LP-relaxation of the problem to be separated the entry $\bar{x}^*_j$ is not integral. The separation they create consists of two new problems which each have one new constraint: In the first problem, the constraint $x_j \leq \lfloor \bar{x}^*_j \rfloor$ is added, and in the second one, $x_j \geq \lceil \bar{x}^*_j \rceil$ is used. Obviously, this defines a separation.

**Algorithm 2** General Branch-and-Bound Algorithm

```
function BRANCH-AND-BOUND($X_0$, $f$)
    create an active node for $X_0$
    $U \leftarrow \infty$
    while active nodes exist do
        let $X$ be an active node
        if a relaxation of $X$ has already been examined then
            let $X_i$, $i \in I$, be a separation of $X$
            create active nodes for all $X_i$, $i \in I$
            deactivate $X$
        else
            let $\bar{X}$ be a relaxation of $X$
            let $\bar{x}^*$ be an optimal solution of $\bar{X}$ (if existent)
            switch depending on $\bar{X}$ and $\bar{x}^*$ do
                case $\bar{X} = \emptyset$
                    deactivate $X$
                case $\bar{X}$ is unbounded
                    $L(X) \leftarrow -\infty$
                case $\bar{x}^* \notin X$
                    $L(X) \leftarrow f(\bar{x}^*)$
                    deactivate $X$ if $L(X) \geq U$
                case $\bar{x}^* \in X$
                    $L(X) \leftarrow f(\bar{x}^*)$
                    $U \leftarrow \min\{U, f(\bar{x}^*)\}$
                    deactivate $X$ and all nodes $X'$ with $L(X') \geq U$
    if $U < \infty$ then
        return $U$ and the corresponding feasible point
```

# 3 Energy Minimization in Discrete Graphical Models

Energy minimization in discrete graphical models, sometimes abbreviated as DGM, are a general type of problem where we search for the “best” out of a finite set of labelings. In fact, nearly any problem that can be formulated as labeling a finite set of discrete variables can be written as a DGM.

## 3.1 Discrete Graphical Models

A *discrete graphical model* $\mathcal{M} = (\mathcal{G}, X, \theta)$ is a triple consisting of

discrete graphical model

- a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$, which determines the structure of the model, more precisely: which variables directly interact with each other,
- the set of possible labelings $X$,
- and a vector $\theta$ of functions $\theta_f$ for all $f \in \mathcal{F}$ constituting the energy function $J : X \to \mathbb{R}$, which assigns an energy to every possible labeling of the model.

Since in this thesis we solely deal with *discrete* graphical models, we will sometimes drop the term “discrete” and simply call them graphical models.

### 3.1.1 Factor Graphs

We will define the structure of discrete graphical models by factor graphs, which have been introduced by Kschischang et al. in 2001 [KFL01]. They are a convenient way to explicitly represent the structural properties of the associated energy function, in contrast to earlier approaches where some of this information was only given implicitly.

A *factor graph* is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ where $(\mathcal{V} \cup \mathcal{F}, \mathcal{E})$ is a bipartite graph with a set of *variable nodes* $\mathcal{V}$, a set of *factors* $\mathcal{F}$ with $\mathcal{V} \cap \mathcal{F} = \emptyset$, and a set of edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{F}$ that defines the relation between those.[1] An example of a factor graph can be seen in Fig. 3.1.

factor graph

[1] Alternatively, it is possible to represent the structure by a hypergraph as follows: Given a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$, the hypergraph representing the same model as $\mathcal{G}$ is $H = (\tilde{V}, \tilde{E})$ with $\tilde{V} := \mathcal{V}$ and $\tilde{E} := \{\text{nb}(f) \mid f \in \mathcal{F}\}$. Note that $(\mathcal{V} \cup \mathcal{F}, \mathcal{E})$ is exactly the incidence graph of $H$.

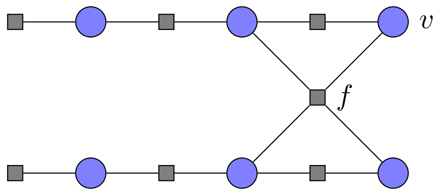


Figure 3.1: Graphical depiction of a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$: Circles correspond to variable nodes $v \in \mathcal{V}$, squares to factor nodes $f \in \mathcal{F}$. Since $\mathcal{G}$ contains factors of order 1, 2, and 4, the order of the graphical model described by it is 4.

For reasons of simplicity, we usually assume that the neighborhood of a factor is unique, i.e., that $\operatorname{nb}(f_1) \neq \operatorname{nb}(f_2)$ for all $f_1, f_2 \in \mathcal{F}$ with $f_1 \neq f_2$. This is no restriction as will be shown in Sec. 3.1.4. Using this convention also allows us to easily name a specific factor: For this, we denote the (unique) factor $f$ with $\operatorname{nb}(f) = \{v_1, v_2, \ldots, v_r\}$ by $f_{v_1 v_2 \ldots v_r}$.

order

We define the *order* of a factor by its degree, i.e., the order of $f \in \mathcal{F}$ is $|\operatorname{nb}(f)|$. Sometimes we will split up the set of factors according to their degree: We will denote all factors in $\mathcal{F}$ with degree $r$ by $\mathcal{F}_r$. Factors in $\mathcal{F}_1$ and $\mathcal{F}_2$ are commonly referred to as *unary* and *pairwise factors*, respectively, whereas factors of order greater than two are called *higher-order factors*. The *order* of a discrete graphical model is the maximal degree among the factors of its factor graph.

unary, pairwise and higher-order factors

underlying graph

For a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ that only contains factors of order at most two, we define its *underlying graph* by $G = (V, E)$ where $V := \mathcal{V}$ and $E := \{\operatorname{nb}(f) \mid f \in \mathcal{F}_2\}$, i.e., the pairwise factors are represented by edges in $G$ and the unary factors are not explicitly present. It is sometimes convenient to specify a factor graph by its underlying graph.

### 3.1.2 Variables and Labelings

label set

For every variable node $v \in \mathcal{V}$ there is a variable $x_v$ corresponding to $v$. Each variable $x_v$, $v \in \mathcal{V}$, can take values in a corresponding *label set* $X_v$, where $X_v$ is a non-empty, finite set. The elements of $X_v$ are called *labels*. Since only the cardinality of the label set and not the labels themselves is important, we will often simply use integers as labels, so a label set with $|X_v| = k$ will be identified with $\{1, 2, \ldots, k\}$. We will use the shorthands $X_A := \prod_{v \in A} X_v$ and $x_A := (x_v)_{v \in A}$ for $A \subseteq \mathcal{V}$ as well as $X := X_{\mathcal{V}}$ and $x := x_{\mathcal{V}}$. We call $x$ a *labeling* of our discrete graphical model. For simplicity of notation, we will sometimes assume that all $X_v$ have the same cardinality (i.e., are equal) and denote this common label set by $L$, so that $X = L^{|\mathcal{V}|}$. We can do so without loss of generality as will be shown in Sec. 3.1.4.

labeling

If $|X_v| = 2$ for all $v \in \mathcal{V}$, the discrete graphical model is said to be *binary*, if $|X_v| > 2$ for some $v \in \mathcal{V}$, then the model is called a *multi-label* model.

binary model
multi-label model

### 3.1.3 Energy Functions

Each factor $f \in \mathcal{F}$ has an associated energy function $\theta_f : X_{\mathrm{nb}(f)} \to \mathbb{R}$. Using these, we can define the *energy function* $J : X \to \mathbb{R}$ of the discrete graphical model, which assigns an energy to every labeling of the model. It is given by

energy function

$$J(x) := \sum_{f \in \mathcal{F}} \theta_f(x_{\mathrm{nb}(f)}).$$

Please note that when we use the shorthand notation $f_{v_1 \dots v_r}$ for the factor $f$ with $\mathrm{nb}(f) = \{v_1, \dots, v_r\}$, then the order of the arguments of $\theta_{f_{v_1 \dots v_r}}$ is always meant to correspond to the order of the elements $v_1, \dots, v_r$. This means for example that although $f_{uv} = f_{vu}$, the values of $\theta_{f_{uv}}(0,1)$ and $\theta_{f_{vu}}(0,1)$ are not necessarily the same since the first term gives the energy contribution of this factor in the case $x_u = 0$ and $x_v = 1$ whereas the second term covers the case $x_v = 0$ and $x_u = 1$.

Some functions of a particular type are of special interest since despite their simplicity they are expressive enough to be useful in applications.

An energy function $\theta_{f_{uv}}$ is called a *Potts function* if $f_{uv}$ is a pairwise factor with $\{u, v\} = \mathrm{nb}(f)$ and the energy function is given by

Potts function

$$\theta_{f_{uv}}(x_u, x_v) := \beta \cdot \mathbb{I}[x_u \neq x_v] = \begin{cases} \beta, & \text{if } x_u \neq x_v, \\ 0, & \text{otherwise,} \end{cases} \quad \text{for all } (x_u, x_v) \in X_u \times X_v,$$

where $\beta \in \mathbb{R}$. The value $\beta$ is called a *disagreement term*. A graphical model is called a *Potts model* if it is of order at most two and all its energy functions associated to pairwise factors are Potts functions.

Potts model

An *Ising function* is a special case of a Potts function where $X_u = X_v$ with $|X_u| = |X_v| = 2$. A graphical model is called an *Ising model* if all its energy functions are Ising functions. Ising models are closely related to the max cut problem, as will be shown in Sec. 3.2.2.

Ising function
Ising model

Potts functions can be generalized in different ways. One natural possibility is to extend their definition to higher-order factors: An energy function of a factor $f$ is a *higher-order Potts function* if $\theta_f$ is given by

higher-order Potts function

$$\theta_f(l_1, l_2, \dots, l_r) := \begin{cases} \beta_1, & \text{if } l_1 = l_2 = \dots = l_r, \\ \beta_2, & \text{otherwise,} \end{cases} \quad \text{for all } (l_1, l_2, \dots, l_r) \in X_{\mathrm{nb}(f)},$$

where $r > 2$ is the order of $f$ and $\beta_1, \beta_2 \in \mathbb{R}$.

submodular

Another important class are submodular functions: If $f$ is a pairwise factor with $X_v = \{0, 1\}$ for all $v \in \mathrm{nb}(f)$ then the energy function $\theta_f$ is *submodular* if

$$\theta_f(0,0) + \theta_f(1,1) \leq \theta_f(0,1) + \theta_f(1,0).$$

If $f$ is a higher-order factor then $\theta_f$ is submodular if all its projections on two variables are submodular.

## 3.1.4 Energy Minimization

Now we have defined all components of a discrete graphical model and can formulate our goal: finding a labeling with the lowest energy.

MAP problem

**Problem 3.1** (MAP Problem)**.** Given a discrete graphical model $\mathcal{M}$, the *energy minimization problem* (or *MAP problem*) is to determine a labeling $x^*$ that has the lowest energy among all possible labelings of $\mathcal{M}$, i.e., for which

$$x^* \in \operatorname*{argmin}_{x \in X} J(x)$$

holds.

Such a labeling $x^*$ is called an *optimal solution* or an *optimal labeling* of $\mathcal{M}$. The corresponding decision problem to this optimization problem is $\mathcal{NP}$-hard as we will see in Sec. 3.2.1.

inference

MAP solution

The process of finding $x^*$ is also called *inference*, an optimal labeling $x^*$ is also called a *maximum a posteriori (MAP) solution*. These terms originate from probability theory where graphical models serve as a modeling tool for certain distributions. This will be explained in Sec. 3.4.

After having formulated our goal – computing a labeling that is optimal with respect to the energy function – we can now give some simple transformations for modifying the graphical model without influencing the optimal labelings.

Assume that in a given graphical model there are two factors $f'$ and $f''$ that have the same neighborhood $N$. We can simply replace $f'$ and $f''$ by a new factor $f$ with the same neighborhood and an energy function that is defined as $\theta_f(x_N) := \theta_{f'}(x_N) + \theta_{f''}(x_N)$.

If the label sets $X_v$ do not have the same cardinality for all $v \in \mathcal{V}$, we can replace them by a common label set $L := \{1, \dots, k\}$ where $k := \max_{v \in \mathcal{V}} |X_v|$. Let $X_{\text{old}} := \prod_{v \in \mathcal{V}} X_v$ be the old label set of the model and $X_{\text{new}} := L^{|\mathcal{V}|}$ be the new label set. The energy functions have to be redefined as

$$\theta_f(x_{\mathrm{nb}(f)}) := \begin{cases} \theta_f(x_{\mathrm{nb}(f)}), & \text{if } x \in X_{\text{old}}, \\ M, & \text{if } x \in X_{\text{new}} \setminus X_{\text{old}} \end{cases}$$

where $M$ is a constant that is large enough.

It is also worth mentioning that any graphical model $\mathcal{M}$ with a factor graph $(\mathcal{V}, \mathcal{F}, \mathcal{E})$ and an energy function $J$ can be reformulated as a graphical model $\mathcal{M}'$ that contains only a single factor – a so-called *global factor*, which is connected to all variable nodes – by using the factor graph $(\mathcal{V}, \{f'\}, \{f'v \mid v \in \mathcal{V}\})$ and $\theta_{f'}(x) := J(x)$ for all $x \in X$. By doing so, all structural properties of the original factor graph get lost, which makes the computation of optimal solutions harder. It is therefore favorable to model a problem with a graphical model of an order as low as possible.

global factor

## 3.2 Connection to Other Problems

In this section we will show how well-known problems from Logic and Combinatorial Optimization can be formulated as graphical models:

First, this is shown for the satisfiability problem, which proves that energy minimization in graphical models is $\mathcal{NP}$-hard.

Second, we show that both the maximum cut and the minimum *s*-*t* cut problem can be written as a graphical model of low order and, even more important for the following chapters, how graphical models of a certain structure can be transformed into one of these classical optimization problems. By applying these transformations, we can solve certain graphical models using the methods available for computing cuts.

### 3.2.1 Satisfiability Problem

The *satisfiability problem (SAT)* deals with the satisfiability of a set of clauses over Boolean variables. When Cook introduced the concept of $\mathcal{NP}$-completeness, this was the first problem he showed to be $\mathcal{NP}$-complete [Coo71]. We will now give a transformation of SAT into a discrete graphical model. Since this transformation is polynomial, it follows that energy minimization in discrete graphical models is $\mathcal{NP}$-hard.

We first need some definitions: Let $U = \{u_1, \ldots, u_n\}$ be a finite set of variables, let $\bar{U} := \{\bar{u} \mid u \in U\}$, $U \cap \bar{U} = \emptyset$, and let $t$ be a *truth assignment* for $U$, i.e., a function $t : U \to \{T, F\}$. An element $l \in U \cup \bar{U}$ is called a *literal*. For a literal $l$ and a truth assignment $t$, if $l = u$ for a $u \in U$, then $l$ is said to be true if and only if $t(u) = T$; if $l = \bar{u}$ for a $u \in U$, then $l$ is said to be true if and only if $t(u) = F$. A subset $c \subseteq U \cup \bar{U}$ is a *clause* over $U$. Given a truth assignment $t$, a clause $c$ is true (or satisfied) if $l$ is true for at least one $l \in c$. A set $C$ of clauses is *satisfiable* if there is a truth assignment $t$ such that all $c \in C$ are satisfied by $t$.

truth assignment
literal
clause
satisfiable

**Problem 3.2** (Satisfiability Problem (SAT))**.** The *satisfiability problem* is a decision problem that asks for the following: Given a finite set of variables $U$ and a finite set $C$ of clauses over $U$, is there a truth assignment $t$ for $U$ for which all clauses in $C$ are satisfied, i.e., is $C$ satisfiable?

satisfiability problem (SAT)

The satisfiability problem can be modeled as a discrete graphical model $\mathcal{M} = (\mathcal{G}, X, \theta)$ [WJ08]: We define a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ by

$$\begin{aligned} \mathcal{V} &:= U, \\ \mathcal{F} &:= \{f_c \mid c \in C\}, \end{aligned}$$

and

$$\mathcal{E} := \{(f_c, u) \mid c \in C, u \in c\}.$$

The label sets and energy functions are given by

$$X_u := \{T, F\} \qquad \text{for all } u \in \mathcal{V}$$

and

$$\theta_{f_c}(x_{\text{nb}(f_c)}) := \begin{cases} 1, & \text{if } x_u = F \text{ for all } u \in \text{nb}(f_c) \text{ with } u \in c \\ & \text{and } x_u = T \text{ for all } u \in \text{nb}(f_c) \text{ with } \bar{u} \in c, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for all } c \in C.$$

An example of an instance of SAT and the corresponding factor graph can be seen in Fig. 3.2. The label set is $\{T, F\}$ for all variables, so that a labeling $x$ directly corresponds to a truth assignment. The energy functions are designed in a way that $\theta_{f_c}(x_{\text{nb}(f_c)})$ is 1 if the truth assignment corresponding to $x$ does not satisfy $c$ and 0 otherwise. So $J(x)$, the energy of the whole model, is the number of unsatisfied clauses of the labeling $x$. Therefore, the equivalence

$$\min_{x \in X} J(x) = 0 \;\Leftrightarrow\; C \text{ is satisfiable}$$

holds. To decide whether $C$ is satisfiable or not we can compute an optimal solution of $\mathcal{M}$ and check whether its energy is 0 or not. We have therefore reduced SAT to energy minimization in a graphical model and since this reduction was polynomial the latter is $\mathcal{NP}$-hard.

### 3.2.2 Maximum Cuts and Minimum $s$-$t$ Cuts

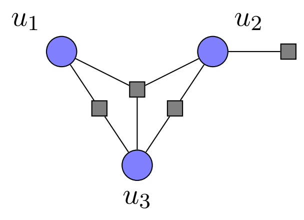
The max cut problem is one of the best known problems from Combinatorial Optimization and its associated decision problem was one of the first problems that was shown to be $\mathcal{NP}$-complete [Kar72].

max cut problem

**Problem 3.3** (Max Cut Problem)**.** Given a weighted graph $G = (V, E, w)$, the *max cut problem* consists of finding a cut of $G$ that has the maximal weight among all cuts, i.e., a maximizer of

$$\max_{S \subseteq V} w(\delta(S)).$$

$$\text{SAT instance } C = \Big\{\{u_1, \bar{u}_2, u_3\},\ \{u_1, u_3\},\ \{u_2, \bar{u}_3\},\ \{\bar{u}_2\}\Big\}$$



$$\begin{aligned} \varphi_{f_{u_2u_3}}(T,T) &:= 0 \\ \varphi_{f_{u_2u_3}}(T,F) &:= 0 \\ \varphi_{f_{u_2u_3}}(F,T) &:= 1 \\ \varphi_{f_{u_2u_3}}(F,F) &:= 0 \end{aligned}$$

Figure 3.2: Transformation of an instance of the satisfiability problem into a discrete graphical model. The variables of SAT correspond to the variable nodes in the factor graph, the clauses to the factor nodes. The energy functions are 1 if and only if the corresponding clause is unsatisfied as is exemplarily shown for $f_{u_2u_3}$.

The max cut problem can be formulated as a discrete graphical model $\mathcal{M} = (\mathcal{G}, X, \theta)$ as follows: Let $G = (V, E, w)$ be the weighted graph defining a max cut instance. We set $\mathcal{G} := (\mathcal{V}, \mathcal{F}, \mathcal{E})$ where

$$\begin{aligned} \mathcal{V} &:= V, \\ \mathcal{F} &:= \{f_{uv} \mid uv \in E\}, \end{aligned}$$

and

$$\mathcal{E} := \{uf_{uv}, vf_{uv} \mid uv \in E\},$$

i.e., we simply use $G$ as the underlying graph of $\mathcal{G}$. For a depiction of the structure of the factor graph see Fig. 3.3. The label-space of the associated variables $x_v$ is $X_v := \{0, 1\}$ for all $v \in \mathcal{V}$ and the energy functions corresponding to the factors are

$$\theta_{f_{uv}}(x_u, x_v) := \begin{cases} -w(uv), & \text{if } x_u \neq x_v, \\ 0, & \text{otherwise,} \end{cases}$$

for all $f_{uv} \in \mathcal{F}$.

Since $\mathcal{G}$ contains only factors of order two and the energy functions are given by disagreement terms, $\mathcal{M}$ is an Ising model.

Now let $x^*$ be an optimal labeling of $\mathcal{M}$, i.e., a minimizer of $\sum_{f \in \mathcal{F}} \theta_f(x_{\mathrm{nb}(f)})$. Then $\delta(S)$ with $S := \{v \in V \mid x^*_v = 0\}$ obviously defines a maximal cut of $G$.

After having transformed a max cut instance into a graphical model, we now turn to the converse: modeling a graphical model of a certain kind as a max cut problem. Assume we are given a binary graphical model of order at most two with factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ and label sets $X_v = \{0, 1\}$ for all $v \in \mathcal{V}$.

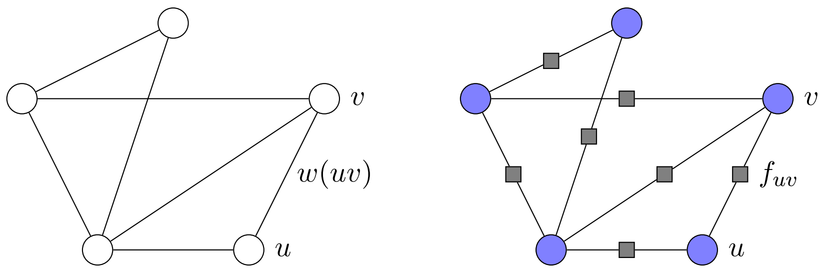


Figure 3.3: Transformation of a max cut problem given by $G = (V, E, w)$ into a graphical model $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$. The graph $G$ is used as the underlying graph of $\mathcal{G}$, so factors are added for each edge of $G$.

To ease our notation, we assume without loss of generality that there are unary and pairwise factors for all nodes and pairs of nodes, respectively, i.e., $f_v \in \mathcal{F}$ and $f_{uv} \in \mathcal{F}$ for all $u, v \in \mathcal{V}$. This is no restriction since we can simply add missing factors and define the corresponding energy functions to be 0 for all arguments.

The graph defining the max cut instance will be a complete graph with nodes $\mathcal{V}$ and an additional node $v_0$. The partition implied by a cut will naturally define a labeling: All nodes in one shore will be labeled with 0, all nodes in the other shore with 1. The additional node $v_0$ is necessary for removing the ambiguity of which shore will get which label. This transformation was similarly already shown in [Ham65]. Our presentation is based on [SK08], but we give an explicit proof.

Let $G := (V, E, w)$ be a complete graph with $V := \{v_0\} \cup \mathcal{V}$ where $v_0 \notin \mathcal{V}$ and $E = E' \cup E''$ where $E' := \{uv \mid u, v \in \mathcal{V}\}$ and $E'' := \{v_0 v \mid v \in \mathcal{V}\}$. We define the weight function $w$ by

$$w(uv) := \frac{1}{2}\Big(\theta_{f_{uv}}(0,1) + \theta_{f_{uv}}(1,0) - \theta_{f_{uv}}(0,0) - \theta_{f_{uv}}(1,1)\Big) \qquad \text{for all } uv \in E'$$

and

$$w(v_0 v) := \theta_{f_v}(1) - \theta_{f_v}(0) + \sum_{\substack{u \in \mathcal{V} \\ u \neq v}} \Big(\theta_{f_{uv}}(0,1) - \theta_{f_{uv}}(0,0) - w(uv)\Big) \quad \text{for all } v_0 v \in E''.$$

Now let $C \subseteq E$ be a cut in $G$. Then there exists a unique set $S_0 \subseteq \mathcal{V}$ with $\delta(\{v_0\} \cup S_0) = C$. Let $S_1 := \mathcal{V} \setminus S_0$, so $V = \{v_0\} \cup S_0 \cup S_1$.

Then we can write the weight $w(C)$ of the cut as

$$
\begin{aligned}
&w(C)\\
&= \sum_{\substack{u \in \{v_0\} \cup S_0 \\ v \in S_1}} w(uv)\\
&= \sum_{v \in S_1} w(v_0 v) + \sum_{\substack{u \in S_0 \\ v \in S_1}} w(uv)\\
&= \sum_{v \in S_1} \Bigg( \theta_{f_v}(1) - \theta_{f_v}(0) + \sum_{\substack{u \in S_0 \cup S_1 \\ u \neq v}} \left( \theta_{f_{uv}}(0,1) - \theta_{f_{uv}}(0,0) - w(uv) \right) \Bigg) + \sum_{\substack{u \in S_0 \\ v \in S_1}} w(uv)\\
&= \sum_{v \in S_1} \Big( \theta_{f_v}(1) - \theta_{f_v}(0) \Big) + \sum_{\substack{v \in S_1 \\ u \in S_0}} \Big( \theta_{f_{uv}}(0,1) - \theta_{f_{uv}}(0,0) \Big)\\
&\quad + \sum_{\substack{v \in S_1 \\ u \in S_1 \\ u \neq v}} \Big( \theta_{f_{uv}}(0,1) - \theta_{f_{uv}}(0,0) \Big) - \sum_{\substack{v \in S_1 \\ u \in S_0}} w(uv) - \sum_{\substack{v \in S_1 \\ u \in S_1 \\ u \neq v}} w(uv) + \sum_{\substack{u \in S_0 \\ v \in S_1}} w(uv)\\
&= \sum_{v \in S_1} \Big( \theta_{f_v}(1) - \theta_{f_v}(0) \Big) + \sum_{\substack{v \in S_1 \\ u \in S_0}} \Big( \theta_{f_{uv}}(0,1) - \theta_{f_{uv}}(0,0) \Big)\\
&\quad + \sum_{\substack{uv \in E \\ u,v \in S_1}} \Big( \theta_{f_{uv}}(0,1) - \theta_{f_{uv}}(0,0) + \theta_{f_{uv}}(1,0) - \theta_{f_{uv}}(0,0) \Big) - \sum_{\substack{uv \in E \\ u,v \in S_1}} 2w(uv)\\
&= \sum_{v \in S_1} \Big( \theta_{f_v}(1) - \theta_{f_v}(0) \Big) + \sum_{\substack{v \in S_1 \\ u \in S_0}} \Big( \theta_{f_{uv}}(0,1) - \theta_{f_{uv}}(0,0) \Big)\\
&\quad + \sum_{\substack{uv \in E \\ u,v \in S_1}} \Big( \theta_{f_{uv}}(1,1) - \theta_{f_{uv}}(0,0) \Big)\\
&= J(x) - J(\mathbf{0}),
\end{aligned}
$$

where the labeling $x$ is defined by

$$
x_v := \begin{cases} 0, & \text{if } v \in S_0, \\ 1, & \text{if } v \in S_1, \end{cases}
$$

for all $v \in \mathcal{V}$. So the weight of the cut $C$ only differs by $-J(\mathbf{0})$, which is a constant, from the energy of $x$. However, to compute a labeling with *minimum* energy with an algorithm for *maximum* cuts, we have to negate the weights $w$. Then a maximum cut directly corresponds to an optimal labeling $x^*$ and the weight of the cut only differs by a constant from $J(x^*)$.

Also the min $s$-$t$ cut problem can be modeled by a graphical model and (partly) vice versa.

min $s$-$t$ cut problem

**Problem 3.4** (Min $s$-$t$ Cut Problem)**.** Given a weighted graph $G = (V, E, w)$ with $w(uv) \geq 0$ for all $uv \in E$ and two nodes $s, t \in V$, the *min $s$-$t$ cut problem* consists of finding an $s$-$t$ cut of $G$ that has the minimal weight among all other $s$-$t$ cuts, i.e., a minimizer of

$$\min_{\substack{S \subseteq V \\ s \in S,\, t \in V \setminus S}} w(\delta(S)).$$

In contrast to the max cut problem, the min $s$-$t$ cut problem is solvable in polynomial time [JF56].

As before, it is possible to convert a min $s$-$t$ cut instance given by $G$ into a graphical model. We again use $G$ as the underlying graph for the factor graph and add pairwise factors for all edges in $G$. Additionally, we have to add two unary factors to the nodes $s$ and $t$ to ensure that they are labeled differently.

For the opposite direction, we can use a binary graphical model of order at most two. For the min $s$-$t$ cut graph, this time we have to add two additional nodes connected to all other nodes. The construction of the weights is similar as before. However, to indeed obtain an instance of the min $s$-$t$ cut problem, we have to ensure that the weights $w(uv)$ are non-negative. It turns out that this is only possible when the energy functions of the graphical model are submodular.

## 3.3 LP and ILP Formulations

The energy minimization problem in a discrete graphical model can be written as an LP and as an ILP. A commonly used LP optimizes over the so-called *marginal polytope*. The ILP is based on a relaxation of the marginal polytope, the *local polytope*: Adding integrality constraints to this relaxation makes it exact.

### 3.3.1 The Marginal Polytope

To formulate the MAP problem in a graphical model $\mathcal{M}$ as a linear program, we define the following indicator variables: For every factor $f \in \mathcal{F}$, every possible labeling $x' \in X_{\mathrm{nb}(f)}$, and every labeling $x \in X$ we set

$$\varphi_f(x', x) := \mathbb{I}[x_{\mathrm{nb}(f)} = x'].$$

We combine all these variables to a common vector

$$\varphi(x) := \Big( (\varphi_f(x', x))_{x' \in X_{\mathrm{nb}(f)}} \Big)_{f \in \mathcal{F}}$$

and do the same for the energy function values

$$\theta := \left( (\theta_f(x'))_{x' \in X_{\mathrm{nb}(f)}} \right)_{f \in \mathcal{F}}.$$

The dimension of $\varphi(x)$ and $\theta$ is $d = \sum_{f \in \mathcal{F}} |X_{\mathrm{nb}(f)}|$.

Using these, we can write the energy function $J$ as

$$\begin{aligned} J(x) &= \sum_{f \in \mathcal{F}} \theta_f(x_{\mathrm{nb}(f)}) \\ &= \sum_{f \in \mathcal{F}} \sum_{x' \in X_{\mathrm{nb}(f)}} \theta_f(x')\varphi_f(x', x) \\ &= \theta^T \varphi(x), \end{aligned}$$

where the linear programming formulation becomes apparent: Using the vectors $\varphi(x)$ for all labelings $x \in X$ as vertices, we define the *marginal polytope* as

marginal polytope

$$\mathbb{M}(\mathcal{M}) := \mathrm{conv}(\{\varphi(x) \mid x \in X\}).$$

An example of a vertex of the marginal polytope can be seen in Fig. 3.4.

Now, the equivalence

$$\min_{x \in X} J(x) = \min_{\mu \in \mathbb{M}} \theta^T \mu$$

is clear, making energy minimization in $\mathcal{M}$ equivalent to optimizing a linear function over the marginal polytope, which can be written as an LP. However, the definition above is only a representation as a $V$-polytope and not as an $H$-polytope.

Note that the indicator variables $\varphi(x)$ we used here are the so-called standard overcomplete representation [WJ08] of the labeling $x$, which means that it is not the most compact form of encoding a labeling since there are several linear dependencies among the entries, e.g., $\sum_{x' \in X_{\mathrm{nb}(f)}} \varphi_f(x', x) = 1$ for all $f \in \mathcal{F}$ and all $x \in X$. However, it is a very convenient one.

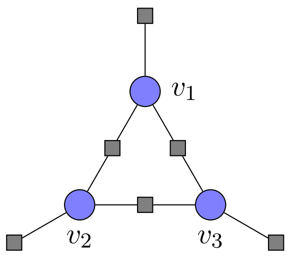
For special models, even more dependencies among the entries of $\varphi(x)$ hold. Recall that for an Ising model $\mathcal{M}$, there are only pairwise factors and the energy functions are given as disagreement terms. Since $\theta_f(0,0) = \theta_f(1,1) = 0$ for all $f \in \mathcal{F}$, the corresponding entries of $\varphi(x)$ are not needed to compute the energy of a labeling. Furthermore, since $\theta_f(0,1) = \theta_f(1,0)$, the full information is contained in the vector $\psi(x) := (\psi_f(x))_{f \in \mathcal{F}}$, where $\psi_f(x)$ is defined as $\psi_f(x) := \varphi_f((0,1), x) + \varphi_f((1,0), x)$. So instead of optimizing over the marginal polytope, we can restrict ourselves to

$$\mathbb{C}(\mathcal{M}) := \mathrm{conv}(\{\psi(x) \mid x \in X\}).$$

This is exactly the *cut polytope*, as for example given in [DL97].

cut polytope

The name “marginal polytope” again originates in probability theory and will be motivated in Sec. 3.4.



$$\mu = \varphi(x) = \varphi(0,1,0) = \left(\begin{array}{c} \mu_{f_1}(0) \\ \mu_{f_1}(1) \\ \hline \mu_{f_2}(0) \\ \mu_{f_2}(1) \\ \hline \mu_{f_3}(0) \\ \mu_{f_3}(1) \\ \hline \mu_{f_{12}}(0,0) \\ \mu_{f_{12}}(0,1) \\ \mu_{f_{12}}(1,0) \\ \mu_{f_{12}}(1,1) \\ \hline \mu_{f_{13}}(0,0) \\ \mu_{f_{13}}(0,1) \\ \mu_{f_{13}}(1,0) \\ \mu_{f_{13}}(1,1) \\ \hline \mu_{f_{23}}(0,0) \\ \mu_{f_{23}}(0,1) \\ \mu_{f_{23}}(1,0) \\ \mu_{f_{23}}(1,1) \end{array}\right) = \left(\begin{array}{c} 1 \\ 0 \\ \hline 0 \\ 1 \\ \hline 1 \\ 0 \\ \hline 0 \\ 1 \\ 0 \\ 0 \\ \hline 1 \\ 0 \\ 0 \\ 0 \\ \hline 0 \\ 0 \\ 1 \\ 0 \end{array}\right)$$

Figure 3.4: Vertex of the marginal polytope: For a graphical model $\mathcal{M}$ with the factor graph depicted on the left and the common label set $L := \{0,1\}$, so $X = \{0,1\}^3$, the vector $\mu$ on the right is the vertex of $\mathbb{M}(\mathcal{M})$ corresponding to the labeling $x = (0,1,0)$.

### 3.3.2 The Local Polytope Relaxation and the ILP Model

Directly optimizing over the marginal polytope is not possible since we only know its inner description as the convex hull of its vertices and not its outer description in terms of halfspaces. Even if this were known, due to the huge number of inequalities this approach were prohibited.

We therefore first introduce a relaxation of the MAP problem. We here once more assume without loss of generality that for all $v \in \mathcal{V}$ a unary factor $f_v \in \mathcal{F}$ exists (if not present, a factor with function value 0 can be added). The relaxation is given as follows:

$$
\begin{aligned}
& \min \theta^T \mu \\
\text{s.t.} \quad & \sum_{x' \in X_{\mathrm{nb}(f)}} \mu_f(x') = 1 && \text{for all } f \in \mathcal{F} && (3.1a) \\
& \sum_{\substack{x' \in X_{\mathrm{nb}(f)} \\ x'_v = x_v}} \mu_f(x') = \mu_{f_v}(x_v) && \text{for all } f \in \mathcal{F},\ v \in \mathrm{nb}(f),\ x_v \in X_v && (3.1b) \\
& \mu_f(x') \geq 0 && \text{for all } f \in \mathcal{F},\ x' \in X_{\mathrm{nb}(f)} && (3.1c)
\end{aligned}
$$

The feasible region of this LP is called the *local polytope* $\mathbb{L}(\mathcal{M})$ and the LP itself the *local polytope relaxation*. It is the first in a hierarchy of relaxations – details on this can be found in [WJ08]. For pairwise models, this LP was first introduced by Schlesinger in 1976 [Sch76]. In this case, the constraints (3.1b) simplify to

local polytope

$$\sum_{x_u \in X_u} \mu_{f_{uv}}(x_u, x_v) = \mu_{f_v}(x_v) \qquad \text{for all } u, v \in \mathcal{V} \text{ with } f_{uv} \in \mathcal{F} \text{ and all } x_v \in X_v.$$

That it is indeed a relaxation, i.e., that $\mathbb{L}(\mathcal{M}) \supseteq \mathbb{M}(\mathcal{M})$, is clear since $\varphi(x)$ satisfies all constraints for all $x \in X$ and both sets are polytopes. Therefore

$$\min_{\mu \in \mathbb{L}(\mathcal{M})} \theta^T \mu \leq \min_{\mu \in \mathbb{M}(\mathcal{M})} \theta^T \mu$$

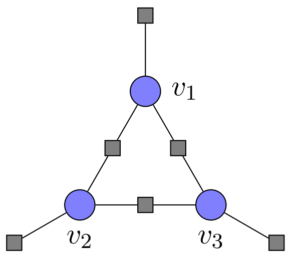
holds. In some simple cases even equality holds: When $\mathcal{M}$ is a pairwise model and the underlying graph is a tree, then $\mathbb{L}(\mathcal{M}) = \mathbb{M}(\mathcal{M})$ [WJ08]. It is also known that if $\mu^*$ is an optimal solution of the local polytope relaxation and all entries of $\mu^*$ are integral, then it is also an optimal solution of the MAP problem, and that all vertices of the marginal polytope are also vertices of the local polytope. However, in general there are also additional (fractional) vertices of $\mathbb{L}(\mathcal{M})$, which can also be optimal solutions for certain graphical models.

The simplest model, for which an optimal solution is fractional, is shown in Fig. 3.5 – its factor graph is the same that we used in the last section, three nodes and unary and pairwise factors for all nodes and pairs of nodes, respectively [WJ08]. The common label set is again $\{0, 1\}$. The fractional vector $\mu^*$ is in $\mathbb{L}(\mathcal{M})$ as can be checked easily: The constraints (3.1a) and (3.1c) are satisfied and the constraints (3.1b) in this case are

$$
\begin{aligned}
\mu_{f_{uv}}(0,0) + \mu_{f_{uv}}(1,0) = \mu_{f_v}(0) \\
\mu_{f_{uv}}(0,1) + \mu_{f_{uv}}(1,1) = \mu_{f_v}(1)
\end{aligned}
$$

for all $u, v \in \mathcal{V}$, which are also true for $\mu^*$. If the energy functions are given by

$$\theta_{f_v}(x_v) := 0 \qquad\qquad \text{for all } v \in \mathcal{V},\ x_v \in \{0, 1\}$$



$$\theta_{f_v}(x_v) = 0$$

$$\theta_{f_{uv}}(x_u, x_v) = \begin{cases} 0, & \text{if } x_u \neq x_v, \\ 1, & \text{otherwise.} \end{cases}$$

$$\mu^* = \begin{pmatrix} \mu_{f_1}(0) \\ \mu_{f_1}(1) \\ \hline \mu_{f_2}(0) \\ \mu_{f_2}(1) \\ \hline \mu_{f_3}(0) \\ \mu_{f_3}(1) \\ \hline \mu_{f_{12}}(0,0) \\ \mu_{f_{12}}(0,1) \\ \mu_{f_{12}}(1,0) \\ \mu_{f_{12}}(1,1) \\ \hline \mu_{f_{13}}(0,0) \\ \mu_{f_{13}}(0,1) \\ \mu_{f_{13}}(1,0) \\ \mu_{f_{13}}(1,1) \\ \hline \mu_{f_{23}}(0,0) \\ \mu_{f_{23}}(0,1) \\ \mu_{f_{23}}(1,0) \\ \mu_{f_{23}}(1,1) \end{pmatrix} = \begin{pmatrix} 0.5 \\ 0.5 \\ \hline 0.5 \\ 0.5 \\ \hline 0.5 \\ 0.5 \\ \hline 0 \\ 0.5 \\ 0.5 \\ 0 \\ \hline 0 \\ 0.5 \\ 0.5 \\ 0 \\ \hline 0 \\ 0.5 \\ 0.5 \\ 0 \end{pmatrix}$$

Figure 3.5: Fractional vertex of $\mathbb{L}(\mathcal{M})$: The fractional vertex $\mu^*$ satisfies all constraints of the local polytope relaxation and is a vertex of $\mathbb{L}(\mathcal{M})$. For the energy function $\theta$, it is an optimal solution with a lower value than an optimal solution of the MAP problem.

and

$$\theta_{f_{uv}}(x_u, x_v) := \begin{cases} 0, & \text{if } x_u \neq x_v, \\ 1, & \text{otherwise,} \end{cases} \qquad \text{for all } u, v \in \mathcal{V},\ x_u, x_v \in \{0, 1\},$$

the value of an optimal solution of the corresponding MAP problem is 1. However, $\theta^T \mu^* = 0$ and so $\min_{\mu \in \mathbb{L}(\mathcal{M})} \theta^T \mu < \min_{\mu \in \mathbb{M}(\mathcal{M})} \theta^T \mu$.

For this model, the marginal polytope has 8 vertices (since the number of labelings is $|L|^3 = 8$), whereas the local polytope has 12 vertices, so 4 of these are fractional. We used PORTA [Chr97] to compute the number of vertices of $\mathbb{L}(\mathcal{M})$ for larger values of $|L|$, the results can be seen in Tab. 3.1. While for $|L| = 5$, the marginal polytope has 125 vertices, the local polytope has 853725 vertices.

As already mentioned, all integral vertices of $\mathbb{L}(\mathcal{M})$ are also vertices of $\mathbb{M}(\mathcal{M})$. This directly leads to an integer linear programming formulation of the MAP problem by

Table 3.1: Number of vertices of $\mathbb{M}(\mathcal{M})$ and $\mathbb{L}(\mathcal{M})$ for a graphical model with the factor graph depicted in Fig. 3.5 for different cardinalities of the label set $L$. Since the only integral vertices $\mathbb{L}(\mathcal{M})$ contains are the vertices of $\mathbb{M}(\mathcal{M})$, the difference between the two numbers is the number of non-integral vertices of $\mathbb{L}(\mathcal{M})$. The numbers were computed with PORTA [Chr97].

| $\lvert L\rvert$ | number of vertices of $\mathbb{M}(\mathcal{M})$ | number of vertices of $\mathbb{L}(\mathcal{M})$ |
|---|---|---|
| 2 | 8 | 12 |
| 3 | 27 | 207 |
| 4 | 64 | 8992 |
| 5 | 125 | 853725 |

simply additionally requiring integrality. So the ILP

$$\begin{aligned} &\min\ \theta^T\mu \\ \text{s.t.}\quad &\mu \in \mathbb{L}(\mathcal{M}) \\ &\mu \in \{0,1\}^d \end{aligned}$$

is equivalent to the energy minimization problem in $\mathcal{M}$.

## 3.4 Probabilistic View of Graphical Models

In this section, we give a brief motivation of the terms used in graphical models by explaining their origins. More details can be found in the book of Koller [KF09].

The content presented here is independent of the rest of the thesis. We will use the same notation for different concepts which turn out to be the same.

Let $x = (x_1, \dots, x_n)$ be a random vector consisting of random variables $x_i$ taking values in $X_i \subseteq \mathbb{R}$, $i = 1, \dots, n$, let $X := \prod_{i=1}^n X_i$, and let $\varphi : X \to \mathbb{R}^d$ be a $d$-dimensional vector of *potential functions* or *sufficient statistics*. Then we can define a set of probability distributions called *exponential family* parametrized by $\tilde\theta \in \mathbb{R}^d$ associated with the potential functions $\varphi$, which is given by

potential functions

exponential family

$$p_{\tilde\theta}(x) := \exp(\tilde\theta^T\varphi(x) - A(\tilde\theta)), \quad x \in X,\ \tilde\theta \in \Omega,$$

where $A(\tilde\theta)$ denotes the *log partition function* which is defined by

log partition function

$$A(\tilde\theta) := \log \int_X \exp(\tilde\theta^T\varphi(x))\nu(dx)$$

for some measure $\nu$ and $\Omega$ is the set of valid *canonical parameters* $\tilde\theta$ defined as $\Omega := \{\tilde\theta \in \mathbb{R}^d \mid A(\tilde\theta) < \infty\}$.

canonical parameters

The log partition function $A(\tilde{\theta})$ is simply a normalizing factor to ensure that $p_{\tilde{\theta}}$ is indeed a probability distribution.

discrete exponential family

A *discrete exponential family* is an exponential family where all $X_i$ are discrete finite sets. In the discrete case, $\nu$ is the counting measure, so the log partition function can be written as

$$A(\tilde{\theta}) = \log \sum_{x \in X} \exp(\tilde{\theta}^T \varphi(x)),$$

and $\Omega = \mathbb{R}^d$ holds.

Let $\varphi$ be the vector of potential functions of a discrete exponential family, and let $p$ be an arbitrary probability distribution on $X$. Then we define the vector $\mu = (\mu_1, \dots, \mu_d)$ of *mean parameters* associated to $\varphi$ by

mean parameters

$$\mu_j := \mathrm{E}_p[\varphi_j(x)] = \sum_{x \in X} \varphi_j(x) p(x)$$

for $j = 1, \dots, d$, where $\mathrm{E}_p$ denotes the expectation function defined by $p$. Note that we did not require $p$ to be a member of an exponential family, in particular, not of the exponential family associated to $\varphi$.

We now can consider the set of all possible mean parameters, i.e., the set of all $\mu$ for which a probability distribution $p$ exists:

$$\mathbb{M} := \Big\{ \mu \in \mathbb{R}^d \mid \text{there is probability distribution } p \text{ on } X \text{ such that } \mu = \mathrm{E}_p[\varphi(x)] \}.$$

The definition of $\mathbb{M}$ can be reformulated to

$$\mathbb{M} = \Big\{ \mu \in \mathbb{R}^d \mid \text{there is a } p : X \to \mathbb{R}_0^+ \text{ with } \sum_{x \in X} p(x) = 1 \text{ such that } \mu = \sum_{x \in X} p(x) \varphi(x) \Big\}$$

which is equivalent to

$$\mathbb{M} = \operatorname{conv}(\{\varphi(x) \mid x \in X\})$$

marginal polytope

so that it becomes apparent that $\mathbb{M}$ is a polytope: It is called the *marginal polytope*.

For a member of a discrete exponential family given by a canonical parameter $\tilde{\theta}$, a state $x^* \in X$ is called a *MAP solution* if it has the highest probability of all possible states, i.e., if

$$x^* \in \operatorname*{argmax}_{x \in X} \; p_{\tilde{\theta}}(x).$$

This is equivalent to requiring

$$x^* \in \operatorname*{argmax}_{x \in X} \; \tilde{\theta}^T \varphi(x).$$

Everything introduced in this section of course directly corresponds to the components of a discrete graphical model with the same identifiers: The random variables correspond to the nodes of a factor graph, the set of possible states $X$ corresponds to the label set, and so on. We set the energy function $\theta$ of the graphical model by $\theta := -\tilde{\theta}$ so that

$$\operatorname*{argmin}_{x \in X} \ \theta^T \varphi(x) = \operatorname*{argmax}_{x \in X} \ \tilde{\theta}^T \varphi(x),$$

and a MAP solution $x^*$ therefore is equivalent to an optimal labeling in the graphical model and finding such a solution is the energy minimization problem.

## 3.5 Overview of Existing Inference Methods

There is a vast number of algorithms for energy minimization in graphical models – some are quite general, others are specialized to restricted models, some compute optimal solutions and others only approximations. Since a complete overview is too much to include here, we present a short description of those that are relevant for the rest of this thesis. For a more complete coverage, see [KAH+13].

### 3.5.1 Polynomially Solvable Cases

**Min $s$-$t$ Cut Approach** This approach was first used in computer vision in 1989 by Greig et al. [GPS89]. It works for binary models of order two with submodular energies, which are transformed to min $s$-$t$ cut problems as described in Sec. 3.2.2. Because of the equivalence of minimum $s$-$t$ cuts and maximum flows, mostly algorithms for the max flow problem are used where many different versions have been developed for special structures of the factor graph and the energy functions [BK04].

**Perfect Matching Approach** Schraudolph et al. describe a way to compute a maximum cut in a planar graph by exploiting a correspondence between maximum cuts and minimum perfect matchings [SK09]. The algorithm works for either binary pairwise models without unary factors whose underlying graph is planar or for binary pairwise models whose underlying graph is outerplanar.

**Junction-Tree Algorithm** The junction-tree algorithm first builds a tree whose nodes nodes correspond to subsets of the nodes of the factor graph of the graphical model. The structure depends solely on the structure of the factor graph. Then dynamic programming is used on this graph to compute a labeling. The runtime of the junction-tree algorithm is polynomial in the tree-width of the graph.

### 3.5.2 Approximative Methods

**Alpha-Expansion ($\alpha$-Exp)** This method was introduced by Boykov [BVZ01] and is applicable to multi-label graphical models of order at most two where the energy functions of the second-order factors are given by a metric. It is a move-making algorithm that starts with an arbitrary labeling and then computes a series of minimum cuts to improve this. In each iteration, a label $\alpha \in L$ is chosen and every variable can then either keep its current label or get label $\alpha$. This is repeated until there are no further changes. Alpha-expansion provides a guarantee for the labeling it produces in relation to an optimal labeling, which depends on the energy function values.

**Sequential Tree-Reweighted Message Passing (TRWS)** Developed by Kolmogorov in 2006 [Kol06], the sequential tree-reweighted message passing algorithm improves an earlier message passing algorithm [WJW05] and has a connection to the local polytope relaxation. It produces a sequence of labelings with decreasing energy value and additionally provides a lower bound in each iteration. TRWS is suited for models of order at most two. It is a fast and universal it is a block-coordinate descent method that is widely used, however, it can get stuck in local minima. An implementation of the method is available from the original author.

**Fast Primal-Dual (FastPD)** FastPD is an algorithm by Komodakis and Tziritas [KT07] and uses, as its name suggests, primal and dual formulations of the problem. It is a move-making algorithm and can be seen as a generalization of $\alpha$-Exp. It can converge to a non-optimal point. However, since it also works in the dual domain, it at least also provides a lower bound. It uses of the dual solutions to reparametrize the energy function, which leads to a significant speedup. The code is publicly available from the authors.

**Quadratic Pseudo-Boolean Optimization (QPBO)** Originally used to name the type of problem, QPBO now stands for a method to solve such problems. It dates back to a work of Hammer [HHS84] and was developed further by Rother et al. [RKL+07]. QPBO can be used for binary pairwise models and is equivalent to the local polytope relaxation. For permuted submodular energy functions it yields optimal solutions. In general, it provides *partial optimality*: For a subset of the variables, it computes labels that occur in an optimal solution, i.e., it is guaranteed that there is an optimal solution that coincides with the QPBO solution on this subset. It is described in more detail in Sec. 4.1.1.

**Reweighted Perfect Matching (RPM)** This method can be used for graphical models of max cut type. In [Sch10], Schraudolph extends the algorithm from [SK09] to non-planar graphs. In order for this to work, a so-called consistent collection of graphs that builds a cycle basis for the input graph is needed. Finding such a collection is difficult in general, however, for grid graphs the author gives one that performs well. As part of the *isinf* library, the code is free for non-commercial research and education purposes. Although the author claims that the algorithm is exact, his proof remains unclear.

### 3.5.3 Exact Methods for $\mathcal{NP}$-hard Models

**Integer Linear Programming (ILP)** A general representation of a discrete graphical model is an integer linear program as explained in Sec. 3.3.2: Additional to the constraints of the local polytope relaxation it contains integrality constraints. This method is implemented in [ABK12a] and imposes no further restrictions on $\mathcal{G}$ or $\theta_f(\cdot)$. During optimization a sequence of linear programs is solved and integer constraints are enforced iteratively by applying cutting-plane or branch-and-bound techniques. The code of [ABK12a] is publicly available under the MIT license and uses the commercial optimization library CPLEX, which is free for academic use.

**MPLP using Cycle Constraints (MPLP-C)** MPLP (Max Product Linear Programming) is a dual decomposition method introduced by Globerson and Jaakkola [GJ07]. Sontag et al. [SCL12] give an extension of it which searches for violated constraints corresponding to cycles of length 3 or 4. This leads to a tighter relaxation than the local polytope relaxation.

**Breadth-Rotating AND/OR Branch-and-Bound (BRAOBB)** Otten et al. suggested a depth-first search branch-and-bound algorithm over AND/OR search spaces using mini-bucket heuristics for bounding [OD11]. Contrary to naive depth-first search, which processes one branch of the tree after another, BRAOBB processes all branches "simultaneously" in a round-robin style. This leads to a better anytime behavior. BRAOBB was the winner of the Probabilistic Inference Challenge 2011 [PIC11]. The source code is freely available under the GPL.

**Max Cut by Branch-and-Cut (MCBC)** In [Bon11], Bonato developed a method for solving max cut problems to optimality using a branch-and-cut framework. For graphical models that can be transformed into max cut problems (see Sec. 3.2.2), this can used for the MAP problem. In addition to using the standard cycle relaxation for the cut polytope he employs special separation and lifting techniques for deriving further inequalities that tighten the relaxation. The algorithm is in particular very well suited for sparse graphs. We applied it to computer vision problems for the first time in [KSR+13b]. The code is not publicly available, but the author kindly provided us with the possibility to run our experiments.

# 4 Reduction Techniques

One of the main challenges when dealing with real-world applications of graphical models are the mere problem sizes that occur: Problem formulations in image analysis or computer vision usually use one variable per image pixel or video voxel. So even for medium-sized images, this yields graphical models with hundreds of thousands of variables. Solving these to optimality is often intractable despite the advance in both algorithms and computer hardware.

In this chapter, we will describe two approaches to reformulate a graphical model with the aim of obtaining problems of smaller size that are easier to solve.

In Sec. 4.1 we will introduce a set of preprocessing steps which transform a graphical model into a smaller but equivalent one – here, equivalent means that an optimal solution of the reduced problem is also an optimal solution of the original problem. While each of these steps is well-known or quite simple, it is their combination which yields good results as will be demonstrated in Sec. 4.2.

Sec. 4.3 and Sec. 4.4 are devoted to superpixels which are a valid way to simplify models for our main application image segmentation. Several existing methods for creating superpixels are presented and evaluated. When using superpixels, several pixels are first grouped together and then treated as one variable in the graphical model. Although this does not yield an equivalent model, this preprocessing step can save a lot of computation time while still delivering qualitatively good results.

The findings of Sec. 4.1 and Sec. 4.2 have already been published in [KSR+13b].

## 4.1 Exact Model-Reduction

Assume we are given a graphical model with a factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$. How can we modify the model in order to make it solvable more efficiently while at the same time maintaining the optimal solutions? We propose some simple reduction techniques which can also be combined easily.

### 4.1.1 Partial Optimality

An important observation when trying to solve graphical models that are based on real-world data is that their computational complexity often only depends on a relatively small part of the problem instance. Especially problems from computer vision tend to

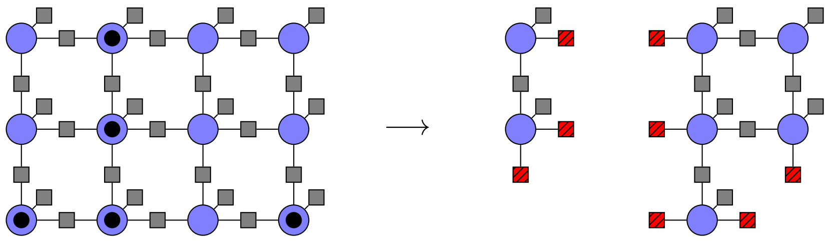


Figure 4.1: Partial optimality reduction: After applying a partial optimality algorithm, the nodes with known value (black dots) can be removed after modifying the factors connected to them appropriately (red).

contain large parts that can be solved easily. This means that algorithms that provide *partial optimality* work well in these cases.

partial optimality

Existing partial optimality algorithms are based on roof duality [BH02], e.g., [RKL+07; Kov03; KSR+08], or on iterative pruning, e.g., [SSK+13].

The output of a partial optimality algorithm is a partition $\mathcal{V} = \mathcal{V}^\circ \cup \mathcal{V}^\bullet$ together with a vector $\bar{x} \in X_{\mathcal{V}^\bullet}$ such that for every optimal solution $(x^*_{\mathcal{V}^\circ}, x^*_{\mathcal{V}^\bullet}) \in X$ of $\mathcal{G}$ also $(x^*_{\mathcal{V}^\circ}, \bar{x}) \in X$ is an optimal solution of $\mathcal{G}$. Here, $\mathcal{V}^\bullet$ represents the “easy” part of the problem that could be solved by the partial optimality algorithm and $\mathcal{V}^\circ$ is the “difficult” part which remains to be solved in an additional step. This is done in a straightforward way by defining a modified graphical model in the following way: The factor graph $\mathcal{G}' = (\mathcal{V}', \mathcal{F}', \mathcal{E}')$ is given by

$$\begin{aligned} \mathcal{V}' &:= \mathcal{V}^\circ, \\ \mathcal{F}' &:= \{f \in \mathcal{F} \mid \operatorname{nb}(f) \cap \mathcal{V}^\circ \neq \emptyset\}, \\ \text{and} \quad \mathcal{E}' &:= \mathcal{E} \cap (\mathcal{V}' \times \mathcal{E}'). \end{aligned}$$

This is sketched in Fig. 4.1. The set of labels $X'$ is the product of the label sets of the unknown nodes, so $X' := \prod_{v \in \mathcal{V}^\circ} X_v$. For all factors $f \in \mathcal{F}'$, their neighborhood in $\mathcal{G}$ can be written as $\operatorname{nb}_\mathcal{G}(f) = \{v_1, \ldots, v_i, v_{i+1}, \ldots, v_r\}$ with $1 \leq i \leq r$ such that $\{v_1, \ldots, v_i\} \subseteq \mathcal{V}^\circ$ and $\{v_{i+1}, \ldots, v_r\} \subseteq \mathcal{V}^\bullet$. The energy functions $\theta'_f$ are then defined as

$$\begin{aligned} \theta'_f &: X_{\operatorname{nb}_\mathcal{G}(f) \setminus \mathcal{V}^\bullet} \to \mathbb{R}, \\ \theta'_f(x_1, \ldots, x_i) &:= \theta_{f_{v_1,\ldots,v_r}}(x_1, \ldots, x_i, \bar{x}_{i+1}, \ldots, \bar{x}_r), \end{aligned}$$

where $\theta$ are the original energy functions.

With these changes, a solution $x^*$ of the modified model together with the output $\bar{x}$ of the partial optimality algorithm gives an optimal solution $(x^*, \bar{x})$ of the original model. The total energies of the solutions differ by the function values of the factors $f \in \mathcal{F}$ with $\operatorname{nb}(f) \subseteq \mathcal{V}^\bullet$, so the total energy $J(x^*, \bar{x})$ of the composed solution can be

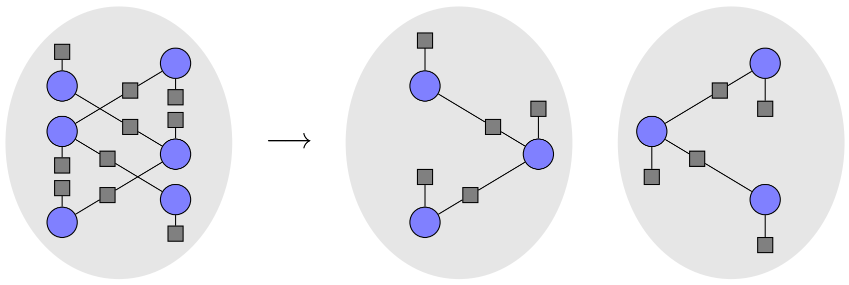


Figure 4.2: Connected component reduction: Obviously, each connected component of a factor graph can be treated separately.

computed as

$$J(x^*, \bar{x}) = J'(x^*) + \sum_{\{f \in \mathcal{F} | \mathrm{nb}(f) \subseteq \mathcal{V}^\bullet\}} \theta_f(\bar{x}_{\mathrm{nb}(f)}).$$

In principle, any algorithm can be used to compute $x^*$, and indeed, this approach has already been used in [AKT10] with *approximative* methods. In contrast, we here systematically exploit partial optimality to make *exact* combinatorial methods feasible for large problem sizes. That this is feasible can be seen in the experiments of Sec. 4.2 and Chap. 6.

### 4.1.2 Connected Components

If the factor graph $\mathcal{G}$ is disconnected, the individual connected components can clearly be treated separately as sketched in Fig. 4.2. We suggest to use a preprocessing step that detects connected components in polynomial time, e.g., by using depth-first search [CLR90], such that all of them can be solved independently.

Although this might seem obvious, it is not automatically taken into account by many solvers. As a proof of concept, we used test instances of the following type: Each instance is a binary graphical model of order two. The underlying graphs each have three connected components which are complete graphs of equal size. We used models with 15, 30, . . . , 90 variables and created ten instances for each size. The values of the unary and pairwise factors were chosen randomly.

In Fig. 4.3, we show the average runtime per size for these models: They have been solved by an ILP solver, once without any modification (ILP) and once by first splitting them up into their connected components (ILP-c). It can be seen clearly that first detecting the connected components leads to a huge speedup. The instances with 90 variables took on average only about ten seconds with ILP-c compared to nearly one hour with ILP.

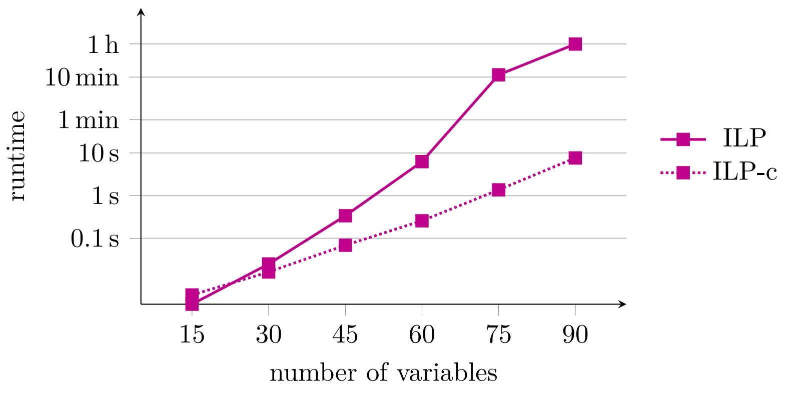


Figure 4.3: Average runtimes of ten binary instances per size. Each instance consists of three complete graphs of equal size. Standard ILP solvers are not able to capture this. Processing each connected component independently (ILP-c) leads to a significant speedup. Note that the time axis is logarithmic.

### 4.1.3 Tentacle Elimination

bridge We call an edge $e \in \mathcal{E}$ a *bridge* if the number of connected components of $\mathcal{G}$ increases when $e$ is removed. After removing a bridge, the resulting components can be treated separately.

When doing so while fixing the variable incident to $e$ for all its possible labels, one side of the bridge can be shrunken to a unary factor representing the optimal values of this subgraph, see Fig. 4.4.

More formally: Assume that $\mathcal{G}$ is connected and that $e' = (v', f') \in \mathcal{E}$ is a bridge where $v' \in \mathcal{V}$ and $f' \in \mathcal{F}$. After removing $e'$, the factor graph $\mathcal{G}$ consists of two components $\mathcal{G}_1 = (\mathcal{V}_1, \mathcal{F}_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{F}_2, \mathcal{E}_2)$ where w.l.o.g. $v' \in \mathcal{V}_1$ and $f' \in \mathcal{F}_2$. Let $X_{v'} = \{l_1, \ldots, l_k\}$ and $\mathrm{nb}(f') = \{v', v_1, \ldots, v_r\}$. We are now defining $k$ graphical models: Their structure is given by $\mathcal{G}_2$, their label set by $\prod_{v \in \mathcal{V}_2} X_v$, and their energy functions by the corresponding functions of the original model with the exception of $\theta_{f'}$: Instead of $\theta_{f'}$, for the $i$-th model, $1 \le i \le k$, we use the function $\theta^i_f$ defined as

$$\theta^i_{f_{v_1,\ldots,v_r}}(x_1, \ldots, x_r) := \theta_{f'_{v',v_1,\ldots,v_r}}(l_i, x_1, \ldots, x_r).$$

We implicitly assume that these models are easily solvable so that it is still efficient to solve all of them. Let $x^i \in X_{\mathcal{V}_2}$ be an optimal solution of the $i$-th model with energy $J^i$ for $1 \le i \le k$.

We can now replace the component $\mathcal{G}_2$ in the original model: We define a graphical model with the factor graph $(\mathcal{V}_1, \mathcal{F}_1 \cup \{f''\}, \mathcal{E}_1 \cup \{(v', f'')\})$. The label sets are the same

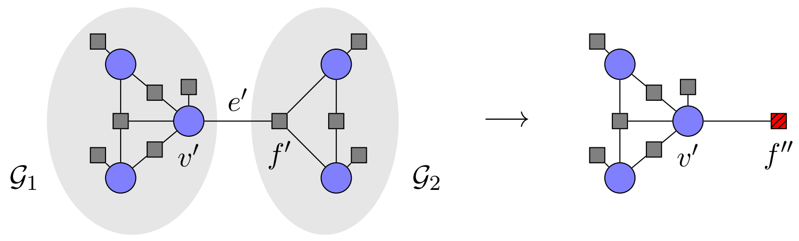


Figure 4.4: Bridge reduction: The right side of the original graph can be shrunken to a unary factor $f''$ by $|X_{v'}|$ small optimization problems. After solving the problem corresponding to the reduced graph, the full solution can be recovered.

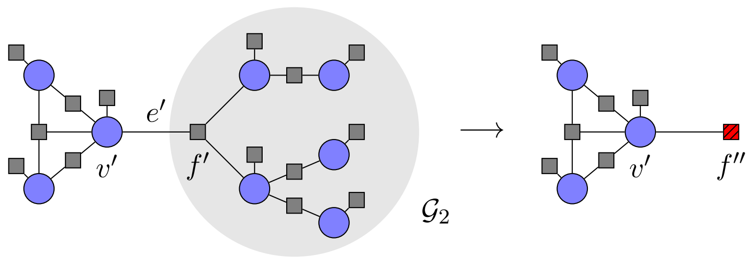


Figure 4.5: Tentacle reduction: In cases where $\mathcal{G}_2$ is acyclic, i.e., is a tentacle, dynamic programming can be used to replace this subgraph by a single unary factor $f''$.

as in the original model and the additional factor $f''$ has the associated energy function

$$\theta_{f''}(l_i) := J^i.$$

If $x^* \in X_{\mathcal{V}_1}$ is an optimal solution of this model and $x^*_{v'} = l_j$, then $(x^j, x^*)$ is an optimal solution of the original model.

So the described procedure yields a simplified and smaller model by replacing $\mathcal{G}_2$ by $f''$. However, this is only beneficial if the computation of the optimal solutions of the $k$ subproblems is fast, i.e., if the structure or size of $\mathcal{G}_2$ is simple enough. A special case where this holds is when $\mathcal{G}_2$ is acyclic – we then call $\mathcal{G}_2$ a *tentacle*, see also Fig. 4.5. Since acyclic graphical models can be solved in polynomial time by dynamic programming, tentacles can be eliminated very efficiently. tentacle

To demonstrate this efficiency, we again created test instances that exhibit this. The instances are again pairwise binary models where the values of the unary and pairwise factors were chosen randomly. The models have 100, 200, 400, . . . , 25600 variables, we created ten instances per size. Their structure is as follows: The underlying graph of each instance consists of a complete graph with 20 nodes, all remaining nodes were

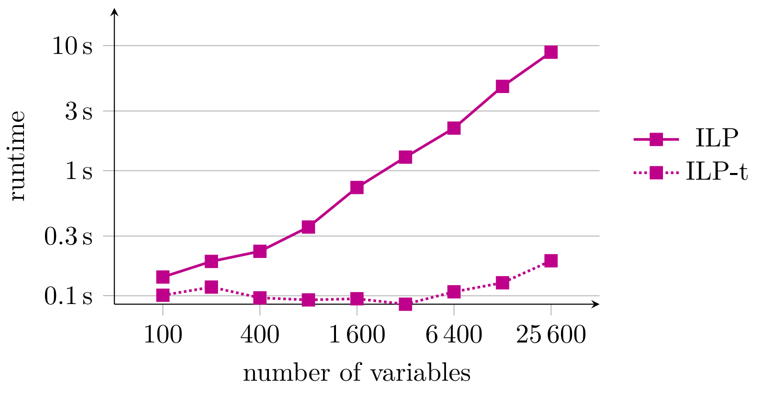


Figure 4.6: Average runtimes of ten binary instances per size. Each instance consists of a complete graph with 20 nodes, all other nodes are part of tentacles. This is not exploited by standard ILP solvers, so eliminating the tentacles first (ILP-t) clearly pays off. Note that both axes are logarithmic.

added iteratively and connected to an already present node by a single edge.

The average runtime per size can be seen in Fig. 4.6: ILP gives the times needed by a standard ILP solver. For ILP-t, the same solver was used but all tentacles were eliminated first. We can again see a clear improvement: The instances of size 25 600 took 8.8 seconds on average when solved normally but only 0.2 seconds when applying tentacle elimination first.

The basic idea behind bridge and tentacle elimination is also known as variable conditioning and variable elimination, see [KF09] and the references therein for an overview. The main difference to our work is that we suggest to use such techniques only for fast solvable substructures and not for the complete model.

## 4.2 Evaluation of Combined Reduction Methods

We created pairwise models where the underlying graph is a grid graph of size $32 \times 32$, $64 \times 64$, …, $2\,048 \times 2\,048$. The problems are binary, i.e., $|X_v| = 2$ for all $v \in \mathcal{V}$. Unary and pairwise factors were added for all nodes and edges of the grid; the values of $\theta_f(\cdot)$ and $\theta_f(\cdot,\cdot)$ were drawn uniformly at random from the interval $[0, 1]$ for all $f \in \mathcal{F}$. We created ten instances per size. Problems of this type can be transformed into pure max cut problems as described in [SK09]. For MCBC and RPM, we transform the problems into max cut instances and solve those.

As expected, the runtimes decrease by orders of magnitudes when we apply QPBO to get partial optimal solutions first, see. Fig. 4.7. Please note that this is not possible for RPM since it needs certain embeddings which are easy to compute for the original

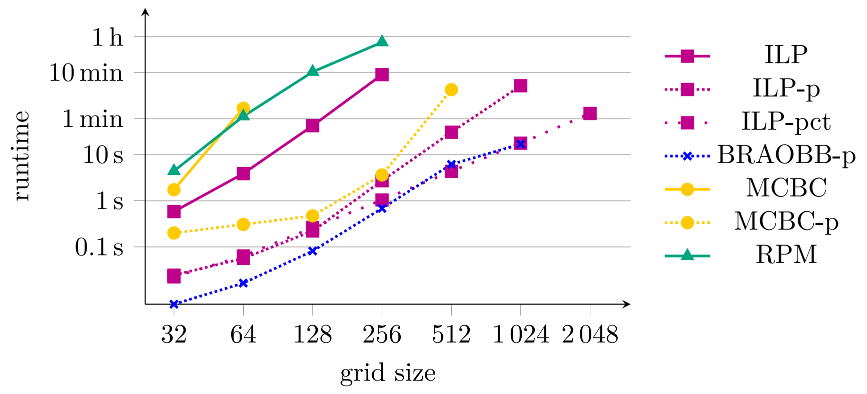


Figure 4.7: Average runtimes of those instances that could be solved in less than one hour (compare Fig. 4.8). For all methods reduction makes them applicable to larger instances. Overall, ILP-pct and BRAOBB-p perform best.

grid problems but not for the reduced ones. The optimality ratio, i.e., the percentage of nodes that were already correctly labeled by QPBO, was 97.6% on average.

However, we achieve another tremendous reduction of the runtime by taking into account that the reduced problems are mostly disconnected and treating the connected components as independent problems. For the $1\,024 \times 1\,024$ instances, the number of components is between 2 497 and 2 669 with sizes in the range of 4 to 83.

We do not apply the connected component reduction for BRAOBB explicitly because this is already taken into account by the method itself. Therefore, the performance of BRAOBB-p is similar to ILP-pct. For MCBC, we only apply the partial optimality reduction.

As can also be seen in Fig. 4.8, the reduction methods show their full potential when applied to large scale problems. We are able to solve instances of size $2\,048 \times 2\,048$ in less than 90 seconds. Overall, BRAOBB-p and ILP-pct perform best.

## 4.3 Reduction via Superpixels

The reduction methods we have presented so far are applicable to general graphical models and not limited to a certain type of problem. Since the main application in this thesis is image segmentation, we will now introduce a technique which was developed solely for segmenting images: Superpixels are a grouping of several pixels of an image into one object which is then treated as a single variable. This preprocessing of grouping the pixels cannot be reversed in the optimization step of the reduced model, implying that this is not an equivalent reformulation: An optimal solution of the reduced model is not necessarily an optimal solution of the original model.

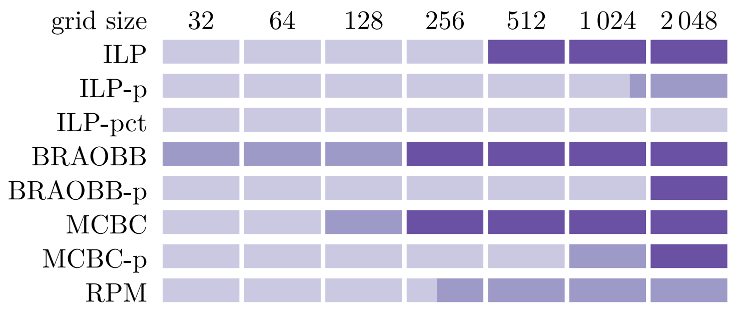


Figure 4.8: Fraction of the ten instances per grid size that could be solved within one hour (□), that could not be solved due to the time limit of one hour (▨), and that could not be solved due to an out-of-memory error (■). Using the three proposed reduction techniques makes methods applicable to larger instances. Overall, ILP-pct scales best.

Several algorithms for obtaining superpixel have been published. However, what was lacking so far was a neutral comparison of them, i.e., one which was not conducted by one of the developers of a superpixel algorithm.

We will therefore present five methods for creating superpixels, four of which having been developed recently (2009–2011) and the fifth one being an established method from 2003. All methods will be compared based on fixed criteria which measure their quality. This way, we will obtain an unbiased view on the different techniques allowing us to choose the appropriate method for our applications.

No precise definition of a superpixel beyond being a semantic grouping of pixels exists. It is clear that superpixels should an oversegmentation and that each each superpixel should belong to the same real-world object. Further desired properties are that superpixels

- are compact,
- contain pixels of similar color and texture,
- are uniform in size and shape,
- and are computationally efficient to compute.

Superpixels are usually used as a preprocessing step to speed up computations. Additionally, they can help to reduce redundancies. They are used in applications like image segmentation and parsing, depth estimation, or object localization. Overall, superpixels contribute to the computational efficiency.

We will now give a brief summary of the superpixel algorithms used in our comparison.

**Superpixels Based on Normalized Cuts** Superpixel segmentation based on normalized cuts was first proposed by Ren and Malik [RM03]. Mori used the same technique and published a freely available version of his code [Mor]. The method is used by many application as a preprocessing step. NCuts has an excellent segmentation quality due to the global optimization criterion it is based on and we can directly control the number of superpixels. The algorithm is known to quite slow compared to other approaches, which we will also see in our experiments. There is also no control of the compactness of the superpixels.

**TurboPixels** TurboPixels is a geometric flow based algorithm proposed by Levinshtein et al. [LSK+09]. The method dilates regularly spaced seeds and adapts them to a local image structure. It uses curve evolution and geometric flow techniques and it delivers simply connected and compact superpixels which are also uniform in size. There exists an extension of TurboPixels to the spatio-temporal domain to compute supervoxels. It has a theoretically sound formulation, however, it is a bit slower than the fastest methods available.

**SLIC Superpixels** SLIC stands for *simple linear iterative clustering* and was introduced by Achanta et al. [ASS+10]. It is a special case of the $k$-means algorithm adapted to the task of superpixel segmentation. SLIC clusters pixels in a joint color and location space, where they are represented as $(labxy)$, where $(lab)$ are the components in CIELAB color space and $(xy)$ are coordinates in the image plane. SLIC is a very fast method for superpixel generation and delivers a good segmentation performance. The superpixels are uniformly sized and compact, we have direct control of the number of superpixels and their compactness.

**Veksler's Method for Superpixels by Energy Minimization** This approach to the superpixel problem was published by Veksler et al. [VBM10]. It is formulated as a pairwise discrete graphical model where the underlying graph is the 8-connected image graph. The optimization is done with $\alpha$-expansion, where only two iterations are needed. It gives very good results while being efficient and easily parallelizable since it contains independent subproblems. Since it uses an explicit energy, it is easy to modify – Veksler et al. also give several variants of their approach. However, it is not as fast as SLIC.

**Superpixels via Pseudo-Boolean Optimization** Introduced by Zhang [ZHM+11] in 2011, this superpixel algorithm uses pseudo-Boolean optimization. It needs only two runs: One run divides the image in horizontal stripes, one in vertical stripes. The superpixels are then generated by intersecting these stripes. Since these two computations are independent, there is no connectivity or size constraint for the superpixels. This is a very fast method whose runtime is independent from the number

Figure 4.9: Example images from the Berkeley Segmentation Dataset.

of superpixels. However, it is of rather low quality, as we will see, since it produces a lot of small fragments.

## 4.4 Evaluation and Comparison

### 4.4.1 Experimental Setup

We tested publicly available implementations of NCuts, SLIC, TurboPixels, Veksler's method, and PBO-SP. Additionally, we included a method squares for having a baseline in the comparisons: This simply divides the input image into squares of the same size, independent of the image data. We wrote a wrapper code in MATLAB to be able to call all methods in a consistent way and also did the evaluation in MATLAB.

We took 50 images from the Berkeley segmentation dataset (BSDS) [MFT+01] as our input data. This dataset consists of color images of size $481 \times 321$ of different real-world scenes, see Fig. 4.9 for example images. Additionally, each image in the dataset has at least five ground truth segmentations done by humans. As already mentioned, since the segmentation problem is not well-defined also these ground truth segmentations can differ a lot, see Fig. 4.10 for examples.

For each image and each method, a segmentation into 150, 250, . . . , 950 superpixels was generated. Please note that it is not possible for all methods to specify the number of superpixels exactly. In these cases we chose the input parameters in a way that the number of superpixels was approximately correct. The measures shown in the plots on the next pages are averaged over all images and (if it depends on the ground truth segmentation) over all ground truths.

Our tests were run on a PC with an Intel E5400 DualCore 2.7 GHZ processor and 8 GB RAM.

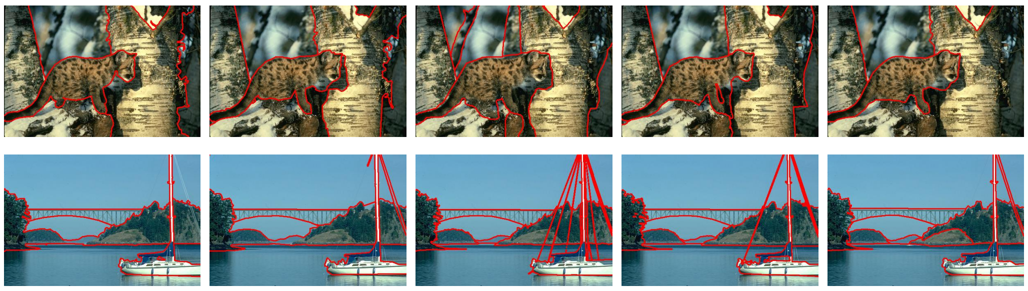


Figure 4.10: For two images of the Berkeley segmentation dataset five segmentations done by humans are shown. Clearly there is a lot of variation in the segmentations.

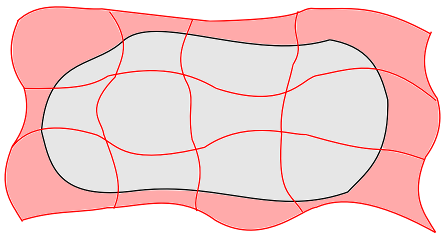


Figure 4.11: Illustration of the "bleeding" of superpixels, which is measured by the undersegmentation error. The object with the black border is a ground truth segment, the red lines are superpixel boundaries. The areas shaded red count towards the undersegmentation error.

### 4.4.2 Undersegmentation Error

undersegmentation error

The *undersegmentation error* measures the amount of "bleeding" of superpixels when they are placed over ground truth segments. This is illustrated in Fig. 4.11. Given ground truth segments $g_1, \ldots, g_M$ and superpixels $s_1, \ldots, s_L$, the undersegmentation error $U$ is defined as

$$U := \frac{1}{N}\left(\sum_{1\leq i\leq M} \sum_{\substack{1\leq j\leq L\\ s_j\cap g_i\neq\emptyset}} |s_j| - N\right),$$

where $N$ is the number of pixels of the image.

The computed undersegmentation errors can be seen in Fig. 4.12. As can be seen, NCuts achieves the lowest undersegmentation error. Veksler's method and SLIC show similar segmentation performance, whereas TurboPixels shows a somewhat weaker performance when the number of superpixels is small. PBO-SP has the highest undersegmentation error at whole range of superpixel counts.

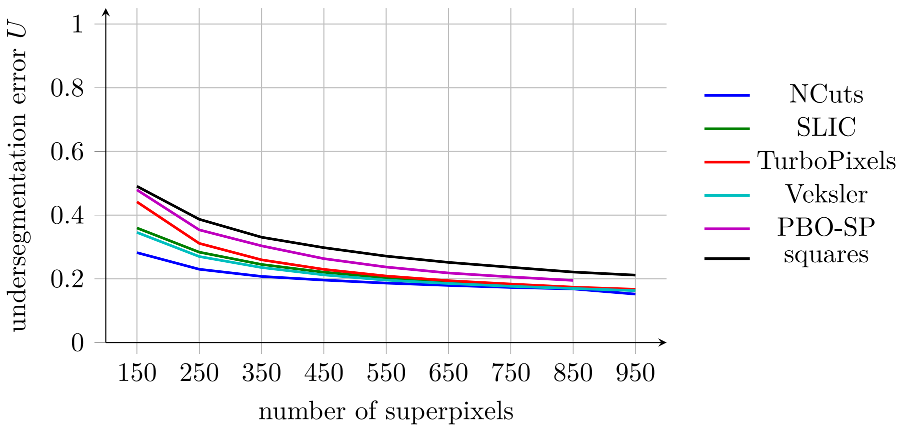


Figure 4.12: Undersegmentation error.

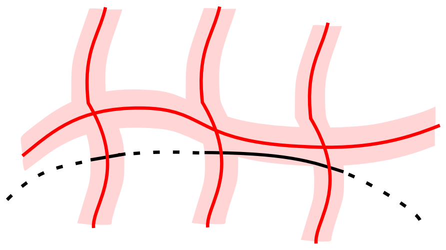


Figure 4.13: Illustration of the boundary recall: The boundary of a ground truth segment (black) counts towards the boundary recall if it is within a distance of $t$ to the boundary of a superpixel segment (red).

### 4.4.3 Boundary Recall

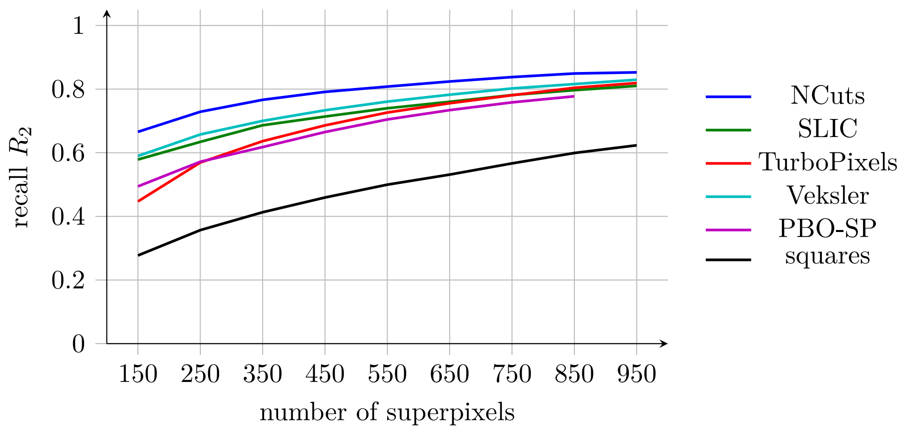
boundary recall

The *boundary recall* computes the fraction of pixels on the boundary of ground truth segments that falls within a small distance of at least one superpixel boundary, see Fig. 4.13. Given ground truth segments $g_1, \ldots, g_M$, superpixels $s_1, \ldots, s_L$, and a distance $t$, the boundary recall $R_t$ is defined as

$$R_t := \frac{\sum_p \mathbb{I}[d(p, \partial s_j) \leq t \text{ for some } j] \cdot \mathbb{I}[p \in \partial g_i \text{ for some } i]}{\sum_p \mathbb{I}[p \in \partial g_i \text{ for some } i]},$$

where $\partial s_j$ and $\partial g_i$ denote the border of the segments and $d$ is the smallest distance of a pixel $p$ to the border of a segment.

Fig. 4.14 shows the computed recall values for $t = 1$ and $t = 2$. NCuts achieves the best boundary recall for both values of $t$. Veksler's method and SLIC again produce similar results, whereas TurboPixels has a lower boundary recall for small numbers



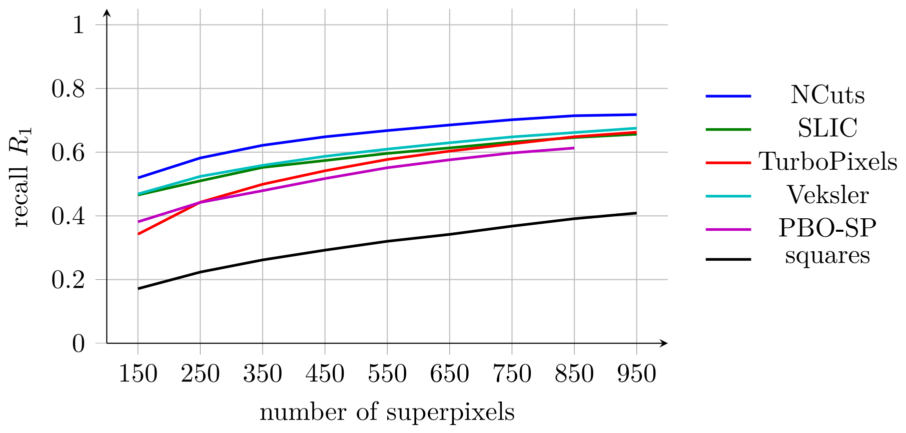


Figure 4.14: Recall for $t = 1$ and $t = 2$.

of superpixels, but its performance improves as the number of superpixels increases. PBO-SP once again has the worst segmentation performance. If the distance constraint is relaxed from $t = 1$ to $t = 2$, all methods show an improved recall rate by roughly 15% with no changes in their relative order.

### 4.4.4 Superpixel Size Uniformity

It is preferable that superpixels are uniform in size and shape. We therefore plot the average histogram of superpixel size, normalized with respect to the expected superpixel size $\frac{N}{K}$, where $N$ is the number of pixels of the image and $K$ is the number of superpixels, see Fig. 4.15.

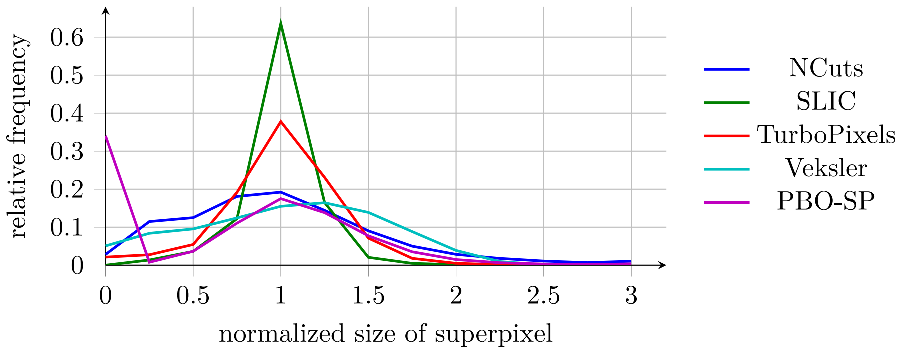


Figure 4.15: Normalized size of the superpixels.

SLIC has the least variability in the size of superpixels, and also TurboPixels produces very uniform superpixels. NCuts and Veksler's method produce a greater variability in the size of produced superpixels since these methods lack the control of the compactness of the superpixel. PBO-SP produces a bimodal distribution peaked at 1 and 0. As this method doesn't have any connectivity constraint, it produces many tiny superpixels.

### 4.4.5 Runtimes

We plotted the runtimes of the methods in Fig. 4.16. NCuts is extremely slow compared to the other methods, which makes it unacceptable in many real-world applications. Its runtime increases with the number of superpixels, whereas for the other methods the running time is roughly constant. TurboPixels and Veksler's method have medium runtimes of about 14 seconds and 5 seconds, respectively. SLIC and PBO-SP are the fastest methods with running times of about 0.3 seconds each.

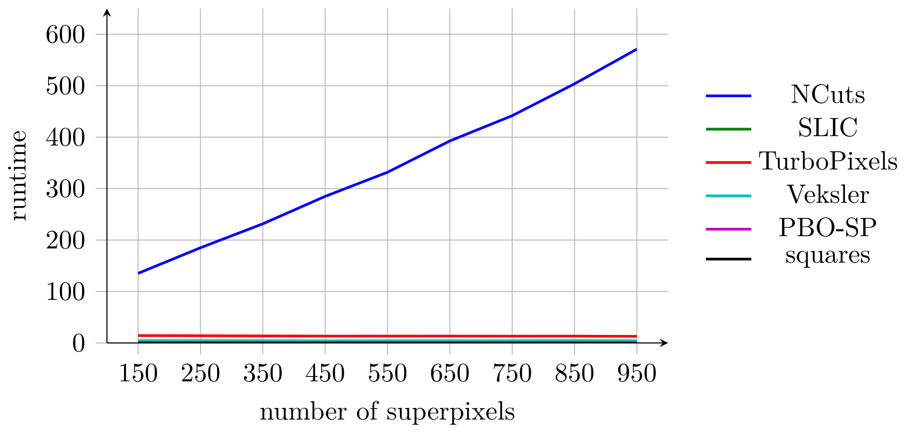
### 4.4.6 Summary

When time is not an essential resource, according to this comparison you should choose NCuts as your superpixel algorithm. It delivered the best quality, however, it is by far the slowest method.

SLIC constitutes a good alternative: Its quality is nearly as good as NCuts and it has the best runtime in our evaluation. Also TurboPixels and Veksler have their benefits: Both can be used for computing supervoxels and Veksler is easily modifiable, and both have acceptable runtime.

Only the results of PBO-SP were disappointing: It clearly was the worst competitor, mostly due to its property of producing many small superpixel segments.

In Fig. 4.17, the superpixel segmentation of all algorithms considered are shown side-by-side for two different numbers of superpixels.



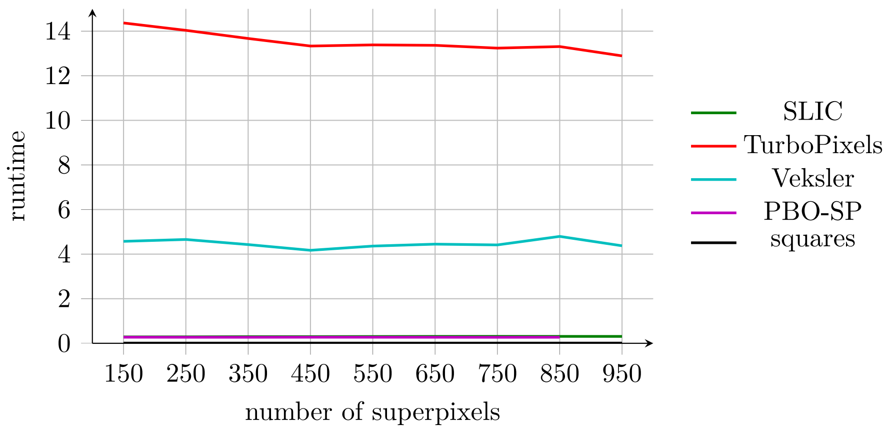


Figure 4.16: Runtimes of the superpixel algorithms: In the top plot it can be seen that NCuts is clearly the slowest method. The bottom plot is simply a zoomed in version of the top one to show the differences of the other methods.

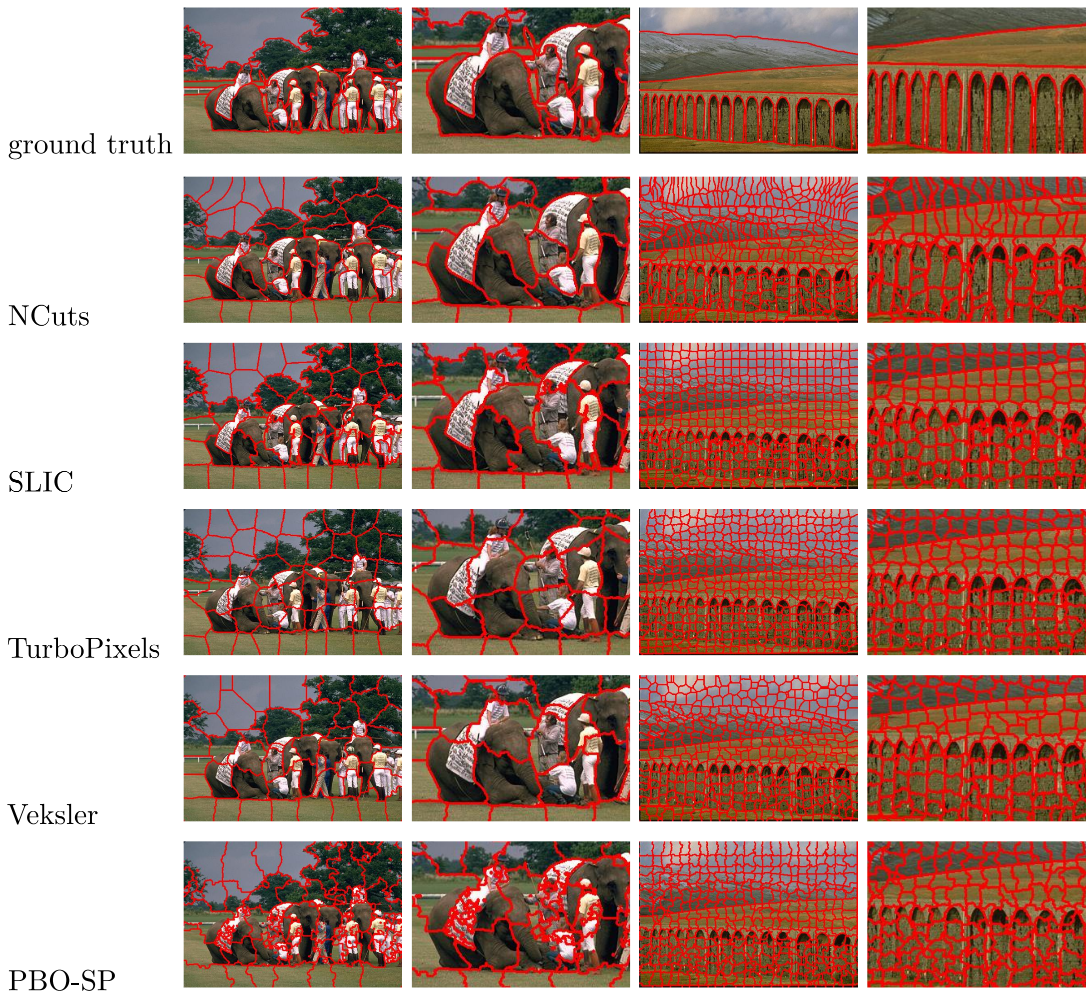


Figure 4.17: Qualitative comparison of superpixel algorithms: The two left columns show a superpixel segmentation into $K = 50$ superpixels, the two right columns into $K = 450$ superpixels. In both cases, the second column is a zoomed in version of the first one.

# 5 Multicuts for Discrete Graphical Models

After having seen reduction techniques for rather general graphical models, we will now turn our view to a specific class of models, which we will solve with specialized methods.

A partition of a graph can be represented by a labeling of its nodes. However, such a representation is far from unique: Any permutation of the labels yields the same partition. What is uniquely defined, however, is the set of edges with endnodes in different partition sets – such a set of edges is called a multicut.

We will exploit this fact to achieve an efficient encoding of suitable functions, namely functions invariant to label permutation. Because of the involved symmetries, standard methods for treating labeling problems like TRWS do not perform well on problems with label permutation invariant functions.

After defining the problem types involving label permutation invariant functions in Sec. 5.1, we show how we can state this class of problems as a multicut problem in the second-order case. Also higher-order factors can be incorporated in this framework quite efficiently, see Sec. 5.3.

In Sec. 5.4, we will explain in detail how we solve multicut problems in an (integer) linear programming formulation. Various types of inequalities together with separation procedures for them are presented, which build the basis for a cutting-plane algorithm. Additionally, we also show dedicated rounding methods for solutions of relaxations.

Of course the reduction techniques explained in the previous chapter can also be used with the models treated here as will be demonstrated in Chap. 6.

The main content of this chapter has already been published in [KSR+13a].

## 5.1 Problem Formulation

The most important concept for this chapter are energy functions that are invariant to label permutations. For a function of this class the function values only depend on the partition induced by the labels of the variables rather than on the labeling itself. They generalize Potts functions in a natural way and are especially suited to be handled by the multicut approach. Many problems of interest are covered by models involving functions of this class. They are defined as follows:

For a factor $f \in \mathcal{F}$ of order $r$ with $X_{\mathrm{nb}(f)} = L^r$ for some common label set $L$ the associated energy function $\theta_f : L^r \to \mathbb{R}$ is called *invariant to label permutations* if for all

label permutation invariant function

$x', x'' \in L^r$ with $x'_u = x'_v \Leftrightarrow x''_u = x''_v$ for all $u, v \in X_{\mathrm{nb}(f)}$ the equality $\theta_f(x') = \theta_f(x'')$ holds.

A permutation invariant function of a factor of order two can be written as

$$\theta_f(x_u, x_v) = \begin{cases} \alpha_1, & \text{if } x_u \neq x_v, \\ \alpha_2, & \text{otherwise} \end{cases}$$

By setting $\beta_f = \alpha_1 - \alpha_2$ this is equivalent to

$$\theta_f(x_u, x_v) = \beta_f \cdot \mathbb{I}[x_u \neq x_v] + \alpha_2,$$

which is the sum of a Potts function and the constant term $\alpha_2$. Since a constant does not influence the optimization problem, we assume in the following without loss of generality that all permutation invariant functions of factors of order two are given as Potts functions with disagreement terms $\beta_f$.

We now introduce the two types of graphical models we are investigating in this chapter. Both involve functions that are invariant to label permutations, but differ in the structure of their factor graphs and label sets.

### Supervised case

**Problem 5.1.** In the supervised case, we deal with the energy minimization problem

$$\min_{x \in X} \sum_{f \in \mathcal{F}_1} \theta_f(x_{\mathrm{nb}(f)}) + \sum_{r \geq 2} \sum_{f \in \mathcal{F}_r} \theta_f(x_{\mathrm{nb}(f)})$$

in a graphical model $\mathcal{M}$ where we require the energy functions $\theta_f$ to be invariant to label permutations for all factors $f \in \mathcal{F}_r$ with $r \geq 2$. The unary factors $f \in \mathcal{F}_1$ can have arbitrary energy functions.

If $\mathcal{M}$ is a graphical model as required in Prob. 5.1 and it is of order at most two, i.e., $\mathcal{F}_r = \emptyset$ for all $r > 2$, then it is a Potts model (see Sec. 3.1.3) since we can then write the energy functions with disagreement terms as explained before.

We focus on related higher-order models separately in Sec. 5.3.

### Unsupervised case

**Problem 5.2.** As in the supervised case in Prob. 5.1, in the unsupervised case we require all energy functions of factors of order greater than one to be invariant to label permutations. However, here we do not allow unary factors at all, so $\mathcal{F}_1 = \emptyset$. Additionally, we have a common label set $L$ with cardinality $|\mathcal{V}|$ for all variables, i.e., $L = \{1, \dots, |\mathcal{V}|\}$. We can therefore write the problem as

$$\min_{x \in \{1,\dots,|\mathcal{V}|\}^{|\mathcal{V}|}} \sum_{r \geq 2} \sum_{f \in \mathcal{F}_r} \theta_f(x_{\mathrm{nb}(f)}).$$

In the second-order case, Prob. 5.2 is known as the *pairwise correlation clustering problem* [BBC04], where a set of variables has to be partitioned into clusters such that the sum of the weights of pairs of nodes in different clusters is minimized.

The main difference between the two cases is that in the supervised case 5.1, the labels usually have a meaning: They correspond to classes about which some information is known. In applications, usually the unary factors indicate the likelihood of a variable belonging to certain class. That is why this type is called a *supervised* problem. In the unsupervised case 5.2, there is no such inherent meaning linked to the labels. We therefore do not have unary factors. Consequently, the common label set is $\{1, \dots, |\mathcal{V}|\}$ which means that in the extreme cases all variables can have the same label or every variable can have its own label. The number of labels actually used solely depends on the factors of order $r \geq 2$.

As shown in [KSA+11] for the second-order case, solving Prob. 5.2 with solvers commonly used for Prob. 5.1, e.g., TRWS [Kol06], does not work, since the large state-space and label permutation invariant functions cause large sets of optimal solutions.

We study efficient methods for solving both Prob. 5.1 and Prob. 5.2 in the general case – multicuts play a key role in modeling their structure.

## 5.2 Multicuts

Multicuts and functions invariant to label permutations have a close connection: Multicuts are a way of representing a partition of the node set of a graph and such a partition generated by a labeling is what a label permutation invariant function depends on.

### 5.2.1 Basic Definitions

We first repeat the definition of a multicut: For a graph $G = (V, E)$, let $\{S_1, \dots, S_k\}$ be a partition of $V$. We call the edge set

$$\delta(S_1, \dots, S_k) := \{uv \in E \mid u \in S_i, v \in S_j, 1 \leq i < j \leq k\}$$

a *multicut* and the sets $S_i$ the *shores* of the multicut. An edge $e \in \delta(S_1, \dots, S_k)$ is called a *cut edge*.

multicut
cut edge

To obtain a polyhedral representation of multicuts, we define *incidence vectors* $\chi(E') := (\chi_e(E'))_{e \in E} \in \{0,1\}^{|E|}$ for each subset $E' \subseteq E$ by

incidence vectors

$$\chi_e(E') := \begin{cases} 1, & \text{if } e \in E', \\ 0, & \text{if } e \in E \setminus E'. \end{cases}$$

The *multicut polytope* $\mathbb{MC}(G)$ is then given by the convex hull of these vectors:

multicut polytope

$$\mathbb{MC}(G) := \operatorname{conv}\left(\{\chi(\delta) \mid \delta \text{ is a multicut of } G\}\right).$$

For details on the geometry of this and related polytopes, we refer to [DGL91].

multicut problem

**Problem 5.3** (Multicut Problem)**.** The *multicut problem* is to find a multicut in a weighted graph $G = (V, E, w)$ for which the sum of the weights of cut edges is minimal. Since all vertices of the multicut polytope correspond to multicuts, this amounts to solving the linear program

$$\min_{y \in \mathbb{MC}(G)} \sum_{e \in E} w(e) y_e.$$

Please note that the terms *multicut* and *multicut problem* are not used consistently in the literature. They are sometimes also used to refer to a generalization of the maximum flow problem, for an overview see [CLR05]. The multicut problem, as we stated it here, is related to the $k$ cut problem [GH94], however, we do not fix the number of partition sets and therefore also allow arbitrary edge weights.

In order to apply linear programming techniques, we have to represent $\mathbb{MC}(G)$ as the intersection of halfspaces given by a system of inequalities. Since the multicut problem is $\mathcal{NP}$-hard [BBC04; GJ79], we cannot expect to find a system of polynomial size. But, as we will see later, partial systems may already support effectively solving the multicut problem.

Before discussing how Prob. 5.3 can be solved efficiently, we will show how the problems 5.1 and 5.2 can be transformed into Prob. 5.3.

### 5.2.2 Multicuts for Second-order Models

We first restrict ourselves to second-order models before treating higher-order models in Sec. 5.3.

#### Supervised case

Let $\mathcal{M} = (\mathcal{G}, X, \theta)$ be an instance of Prob. 5.1 of order two. As explained, we assume that the energy functions of the second-order factors are given by disagreement terms $\beta_f$ for $f \in \mathcal{F}_2$. Additionally, we assume without loss of generality that all variables have the same label set $L = \{1, \ldots, k\}$.

Every labeling $x \in X$ naturally defines a partition $\{S_1, \ldots, S_k\}$ of the nodes $\mathcal{V}$ where $S_i := \{v \in \mathcal{V} \mid x_v = i\}$ for $1 \leq i \leq k$. The energy of a labeling is the sum of the energy of the unary factors for assigning a label to the nodes plus the disagreement terms $\beta_f$ for each second-order factor connecting nodes with different labels.

For the graph defining the instance of the multicut problem we define additional nodes $T := \{t_l \mid l \in L\} = \{t_1, \ldots, t_k\}$ and the weighted graph $G = (V, E, w)$ with

$$V := \mathcal{V} \cup T$$

and

$$E := \{\mathrm{nb}(f) \mid f \in \mathcal{F}_2\} \cup \{tv \mid t \in T, v \in V\} \cup \{t_i t_j \mid t_i, t_j \in T, t_i \neq t_j\}.$$

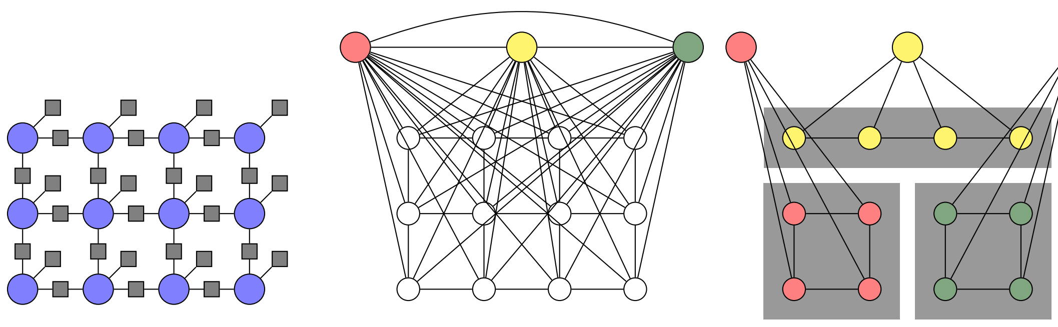


Figure 5.1: Transformation of a supervised graphical model into a multicut problem: The factor graph $\mathcal{G}$ of order two on the left belongs to an instance of Prob. 5.1. The graph in the middle is the input graph of the corresponding multicut problem. For each of the $k = 3$ labels a terminal node is added to the underlying graph of $\mathcal{G}$ and connected to all nodes. On the right, one possible multicut is indicated by showing all edges that are *not* part of it. The label of each internal node is determined by the (unique) terminal node that is in the same connected component.

The nodes in $\mathcal{V}$ are called *internal nodes*, the ones in $T$ are *terminal nodes*. Edges between internal nodes are *internal edges*, edges between a terminal node and an internal node are *terminal edges*, and edges between terminal nodes are *inter-terminal edges*.

internal and terminal nodes

internal, terminal, and inter-terminal edges

Certain multicuts of $G$ will correspond to a labeling of $\mathcal{M}$ as follows: The terminal nodes represent the $k$ labels of $L$ and label $l$ is assigned to variable $x_v$ if the terminal edge $t_l v$ is not part of the multicut, i.e., if $t_l$ and $v$ are in the same shore. For this to be sensible, not all multicuts will be allowed: Since only a single label should be assigned to each variable, $k-1$ terminal edges incident to each internal node $v$ have to be part of the multicut. Also, inter-terminal edges have to belong to different shores. We will see in Sec. 5.4.2 how this is enforced. An depiction of the graph construction can be seen in Fig. 5.1.

It remains to define the weight function of $G$ such that that the weight of a multicut as described above equals the energy of the labeling it induces. For an internal edge $uv$, we set $w(uv) := \beta_{f_{uv}}$. Inter-terminal edges have weight 0. For the terminal edges, let $\mathbf{1}\mathbf{1}^T$ be the matrix of all ones and $I$ be the identity matrix, both of size $k \times k$. Then the weights $w(t_l v)$, $l \in L$, $v \in \mathcal{V}$, are given by

$$\begin{pmatrix} w(t_1 v) \\ \vdots \\ w(t_k v) \end{pmatrix} := \frac{1}{k-1}(\mathbf{1}\mathbf{1}^T - I) \begin{pmatrix} \theta_{f_v}(l) \\ \vdots \\ \theta_{f_v}(k) \end{pmatrix}.$$

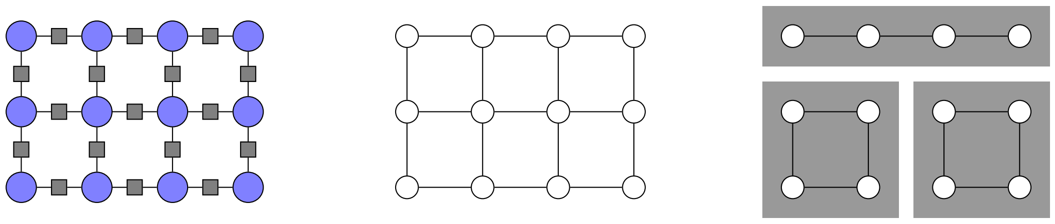


Figure 5.2: Transformation of an unsupervised graphical model into a multicut problem: For the factor graph on the left containing only pairwise factors, its underlying graph is used for the multicut problem (middle). On the right, one possible multicut is indicated by showing all edges that are *not* part of it. Here all multicuts correspond to a labeling. The exact labels of the nodes are not important since all energy functions only depend on the induced partition.

### Unsupervised case

Reformulating Prob. 5.2 in the second-order case into a multicut problem is straightforward. Let again $\mathcal{M} = (\mathcal{G}, X, \theta)$ be an instance of Prob. 5.2. Since there are no unary factors we can simply use the underlying graph of $\mathcal{G}$ and the disagreement terms $\beta_f$ to define the graph for the multicut problem and make use of the one-to-one correspondence between a partition and a multicut. So let $G := (V, E, w)$ with

$$\begin{aligned} V &:= \mathcal{V}, \\ E &:= \{\mathrm{nb}(f) \mid f \in \mathcal{F}\}, \end{aligned}$$

and

$$w(uv) := \beta_{f_{uv}} \qquad \text{for all } uv \in E.$$

Accordingly, the weight of a multicut of $G$ is the sum of all $\beta_f$ over factors $f$ connecting variables in different shores, which equals the energy of the labeling induced by the multicut – see [CR91] for a formal proof. An illustration of this can be seen in Fig. 5.2.

## 5.3 Multicuts for Higher-order Models

In the last section, we have shown how to transform second-order graphical models into an input graph $G$ of the multicut problem so that we can solve the MAP problem with the linear program

$$\min_{y \in \mathbb{MC}(G)} \sum_{e \in E} w(e) y_e.$$

The constraints for ensuring that $y \in \mathbb{MC}(G)$ will be detailed in Sec. 5.4.

Before doing so, we will now show how higher-order factors with energy functions invariant to label permutations can be included in this framework. For this, we can use the same variables $y_e$ that are already present in the formulation and have to add only a few additional auxiliary variables and inequalities. This is possible by encoding the function as a sum of indicator functions of all possible partitions of the neighborhood of such a higher-order factor.

For special functions, e.g., higher-order Potts functions, the representation is even more efficient than in the general case.

### 5.3.1 General Label Permutation Invariant Functions

#### Definition

Let $\mathcal{M}$ be a graphical model, let $f$ be a factor of $\mathcal{M}$ of order $r > 2$ whose energy function is invariant to label permutations, and let $\mathrm{nb}(f) = \{v_1, \dots, v_r\}$. We assume that $X_v = L$ for all $v \in \mathrm{nb}(f)$. The energy function of $f$ therefore only depends on the partition of $\{v_1, \dots, v_r\}$ induced by the labeling $x' \in L^r$.

Each possible partition of the $r$ nodes is uniquely represented by an indicator vector $(\chi_{v_s v_t})_{1 \le s < t \le r} \in \{0,1\}^{r(r-1)/2}$ over all pairs of nodes by

$$(\chi)_{v_s v_t} := \begin{cases} 1, & \text{if } v_s \text{ and } v_t \text{ are in the same shore,} \\ 0, & \text{otherwise,} \end{cases} \qquad \text{for all } 1 \le s < t \le r.$$

Clearly, not every vector $\chi \in \{0,1\}^{r(r-1)/2}$ corresponds to a partition – indeed, those vectors are exactly the vertices of $\mathbb{MC}(K_r)$, where $K_r$ denotes the complete graph with $r$ nodes. Their number is given by the Bell numbers $B(r)$ [Aig79][1]. This observation raises the issue of an efficient representation of these functions, independent of the number of labels.

Let us denote for $i = 1, \dots, B(r)$ by $\chi_i^r \in \{0,1\}^{r(r-1)/2}$ the indicator vector of the $i$-th partition of the $r$ nodes, i.e., $\chi_i^r$ is the $i$-th vertex of $\mathbb{MC}(K_r)$, for some arbitrary order. Furthermore, we define a mapping $\tau^r : L^r \to \{0,1\}^{r(r-1)/2}$ from a labeling $x' \in L^r$ to the partition indicator. With this we can represent the energy function $\theta_f$ by a parameter $\beta \in \mathbb{R}^{B(r)}$ by

$$\theta_f(x') = \beta_i \qquad \text{if } \tau^r(x') = \chi_i^r. \tag{5.1}$$

We call such functions *generalized higher-order Potts functions* since they generalize (second-order) Potts functions.

generalized higher-order Potts function

[1] The first Bell numbers are $B(2) = 2$, $B(3) = 5$, $B(4) = 15$, $B(5) = 52$, $B(6) = 203$, and $B(7) = 877$.

### Reduction Theorem

In order to incorporate generalized higher-order Potts functions into our multicut framework, we introduce the following reduction theorem. The basic idea of this theorem dates back to the work of Glover and Woolsey [GW74].

**Theorem 5.1** (Reduction Theorem)**.** Any pseudo-Boolean function $g : \{0,1\}^M \to \mathbb{R}$ given by $g(z) = \prod_{i\in B^+} z_i \cdot \prod_{i\in B^-}(1-z_i)$, with $|B^+ \cup B^-| = M$ and $B^+ \cap B^- = \emptyset$, can be transformed into an optimization problem with

(a) a single Boolean auxiliary variable $s \in \{0,1\}$ and two linear inequalities

$$\begin{aligned} \min_{z\in\{0,1\}^M, s\in\{0,1\}} \quad & s \\ \text{s.t.} \quad Ms &\le \sum_{i\in B^+} z_i + \sum_{i\in B^-}(1-z_i) && (5.2a) \\ s &\ge 1 - M + \sum_{i\in B^+} z_i + \sum_{i\in B^-}(1-z_i) && (5.2b) \end{aligned}$$

or

(b) a single auxiliary variable $s \in [0,1]$ and $M+1$ inequalities

$$\begin{aligned} \min_{z\in\{0,1\}^M, s\in[0,1]} \quad & s \\ \text{s.t.} \quad & s \le z_i && \text{for all } i \in B^+ \quad (5.3a) \\ & s \le (1-z_i) && \text{for all } i \in B^- \quad (5.3b) \\ & s \ge 1 - M + \sum_{i\in B^+} z_i + \sum_{i\in B^-}(1-z_i). && (5.3c) \end{aligned}$$

*Proof.* The function $g(z)$ takes the value 1 if and only if $z_i = 1$ for all $i \in B^+$ and $z_i = 0$ for all $i \in B^-$, otherwise $g(z) = 0$. It remains to show that the systems of inequalities together with $s \in \{0,1\}$ or $s \in [0,1]$ restrict the feasible set such that $s = g(z)$.

We set $\kappa := \big|\{i \in B^+ \mid z_i = 0\} \cup \{i \in B^- \mid z_i = 1\}\big|$.

(a) Inequalities (5.2a) and (5.2b) imply $s \le 1 - \frac{k}{M}$ and $s \ge 1 - k$. Since $s \in \{0,1\}$, it follows that $s = 1$ if $\kappa = 0$ and $s = 0$ if $\kappa > 0$.

(b) If $\kappa > 0$, the inequalities (5.3a) and (5.3b) imply $s \le 0$. Since $s \in [0,1]$ it follows that $s = 0$. In the case $\kappa = 0$, we have $s \ge 1$ because of (5.3c) and therefore $s = 1$. □

A crucial observation is that case (b) of the reduction theorem implies integrality of $s$ if $z_i \in \{0,1\}$ for all $z_i$, whereas in case (a) this has to be enforced separately by $s \in \{0,1\}$. Consequently, case (b) leads to tighter relaxations by only enforcing $s \in [0,1]$.

While reduction (b) thus seems to be preferable, due to a lower number of constraints, method (a) can be nonetheless appealing for some (I)LP techniques, e.g., for the dual simplex method. In our experiments, we therefore use all $M+2$ constraints (5.2a),(5.2b), (5.3a), and (5.3b) (note that (5.2b) is the same as (5.3c)), and let the solver choose the active constraint set.

### Reduction

In order to apply Theorem 5.1 to a label permutation invariant function (5.1) of order $r$ parametrized by $\beta \in \mathbb{R}^B(r)$ we rewrite it as a sum of pseudo-Boolean functions $g_i$ as required by the theorem by

$$\begin{aligned}\theta_f(x') &= \sum_{i=1}^{B(r)} \beta_i \cdot \mathbb{I}[\tau^r(x') = \chi_i^r] \\ &= \sum_{i=1}^{B(r)} \beta_i \cdot \underbrace{\prod_{\substack{s,t\\1\leq s<t\leq r}} \mathbb{I}[(\tau^r(x'))_{v_s v_t} = (\chi_i^r)_{v_s v_t}]}_{=g_i(\tau^r(x'))} .\end{aligned}$$

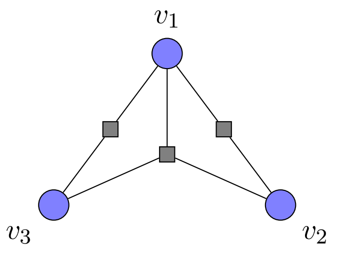
We apply the reduction theorem to each of the $B(r)$ binary functions $g_i(z)$ where $z = \tau^r(x')$. Consequently, a function $\theta_f(x')$ of order $r$ requires $B(r)$ auxiliary variables and for each of them at most $M+1 = r(r-1)/2+1$ inequalities. These auxiliary variables are connected to the labels via the Boolean expressions $\mathbb{I}[(\tau^r(x'))_{v_s v_t} = (\chi_i^r)_{v_s v_t}]$ and correspond to the edge variables $y$ used in the multicut problem 5.3.

If for an expression $\mathbb{I}[(\tau^r(x'))_{v_s v_t} = (\chi_i^r)_{v_s v_t}]$ there is no corresponding edge $v_s v_t$ in $G$, we add this edge to $G$ with weight 0 and therefore also the variable $y_{v_s v_t}$.

Summing up, to include a label permutation invariant factor of order $r$ into our multicut framework, we require at most $r(r-1)/2$ edge variables $y_e$ (if they are not included yet), $B(r)$ auxiliary variables, and $B(r) \cdot (r(r-1)/2+2)$ linear inequalities. In many cases more compact representations are obtained, see Sec. 5.3.2. Note that the size of the representation does not depend on the number of states per variable.

We observed in our experiments that additionally enforcing that all auxiliary variables corresponding to a higher-order factor sum up to 1 significantly speeds up optimization. This leads to a single equality constraint for each higher-order term.

Fig. 5.3 illustrates an example of a factor graph containing a factor of order three.



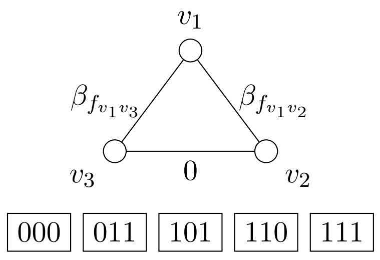


Figure 5.3: The factor graph $\mathcal{G}$ on the left contains a factor of order three. To include it in our multicut framework, $B(3) = 5$ auxiliary variables corresponding to the possible partitions of its neighbors are added (they are indicated by $\chi_i^3$, $i = 1, \ldots, 5$, in the figure). Since no factor $f$ with $\mathrm{nb}(f) = \{v_2, v_3\}$ is contained in $\mathcal{G}$, the edge $v_2 v_3$ with weight 0 is added to the multicut graph.

### 5.3.2 Higher-order Potts Functions

#### Definition

A special case of label permutation invariant functions are functions $\theta_f$ taking only two different values, depending on whether all nodes in the neighborhood of the factor $f$ have the same label or not. They can therefore be parametrized by a vector $\beta \in \mathbb{R}^2$ and are defined as

$$\theta_f(x'_1, \ldots, x'_r) = \begin{cases} \beta_1, & \text{if } x'_1 = \cdots = x'_r, \\ \beta_2, & \text{otherwise,} \end{cases}$$

for all $(x'_1, \ldots, x'_r) \in L^r$.

higher-order Potts function

We call such functions *higher-order Potts functions* since they constitute the simplest generalization of (second-order) Potts functions to the higher-order case. Such functions are general enough to model the costs of a hypergraph partitioning [KNK+11], in which the cost for a hyperedge is included in the overall cost function if the hyperedge connects at least two shores.

#### Reduction

We can reformulate such functions in a pseudo-Boolean form as

$$\theta_f(x') = \beta_2 + (\beta_1 - \beta_2) \prod_{e \in E'} (1 - y_e)$$

where $E'$ is a subset of the edges of the multicut graph $G$ that spans $\mathrm{nb}(f)$. If $G' = (\mathrm{nb}(f), E \cap (\mathrm{nb}(f) \times \mathrm{nb}(f)))$ is disconnected we have to add some edges with weight 0. We point out our empirical observation that using a spanning graph that includes all edges of $G'$ instead of an arbitrary spanning tree leads to shorter runtimes.

As before, we apply the reduction theorem to add a higher-order Potts function to our model. This only requires a single auxiliary variable.

## 5.4 Cutting-Plane Approach and Separation Procedures

### 5.4.1 Approach

Determining a multicut with minimal weight is $\mathcal{NP}$-hard in general. However, if given data induces some structure then it is plausible to expect such problems to be easier solvable in practice than problems without any structure.

Let $G = (V, E, w)$ be a weighted graph obtained from converting a problem of type 5.1 or 5.2 into a multicut problem as described in Sec. 5.2 and Sec. 5.3. To solve this, we use a cutting-plane approach to iteratively tighten a relaxation of the form

$$\min_{y \in Y} \sum_{e \in E} w(e) y_e.$$

We will use both linear programs and integer linear programs of this type. When dealing with an LP, we have $\mathbb{MC}(G) \subseteq Y$, so that $Y$ is superset of the multicut polytope $\mathbb{MC}(G)$ (see Sec. 5.2.1). In case of an ILP we have $\mathbb{MC}(G) \cap \{0,1\}^{|E|} \subseteq Y \subseteq \{0,1\}^{|E|}$.

In each step we solve such a relaxation, detect violated constraints from a pre-specified finite list (see Sec. 5.4.2) and augment the constraint system accordingly. This procedure is repeated until no more violated constraints are found.

After each iteration we obtain a lower bound as the solution of the (I)LP and an upper bound by mapping the obtained solution to the feasible region (rounding, see Sec. 5.4.3).

### 5.4.2 Relaxation, Constraints

#### Initial Constraints

We start with a polytope that enforces all variables $y_e$ to be lower and upper bounded by 0 and 1, respectively:

$$y_e \in [0,1] \qquad \text{for all } e \in E. \tag{5.4}$$

In case of Prob. 5.1 when terminal nodes are present in $G$, we additionally enforce for each internal node $v \in V \setminus T$ that (in case of an ILP) exactly one incident edge is in the multicut, i.e.,

$$\sum_{t \in T} y_{tv} = |T| - 1 \qquad \text{for all } v \in V \setminus T \text{ (if } T \neq \emptyset). \tag{5.5}$$

Furthermore, we add the compulsory constraints

$$y_{tt'} = 1 \qquad \text{for all } t, t' \in T \text{ with } t \neq t',$$

forcing different terminal nodes to belong to different shores.

### Integer Constraints

A more restrictive alternative to (5.4) are the integer constraints

$$y_e \in \{0,1\} \qquad\qquad \text{for all } e \in E. \tag{5.6}$$

In general, using constraints (5.6) renders inference problems more difficult. On the other hand, finding violated constraints can be much simpler for Boolean variables than for less tight LP relaxations. This may well compensate the additional costs[2] for solving an ILP instead of an LP.

### Cycle Constraints

The problem of inconsistent edge-labelings has been considered in the literature, either motivated by closing contours [MLH12; AKB+11] or as tightening the multicut polytope relaxation via cycle constraints [CR93; NJ09; KNK+11; KSA+11]. In both cases inconsistent cycles are detected. If integer constraints are enforced, an inconsistent cycle is a cycle that contains exactly a single active edge, which obviously violates transitivity. This can be generalized to the relaxed non-Boolean case $y_e \in [0,1]$ [CR93].

cycle inequalities

A system of *cycle inequalities* that necessarily has to be satisfied by consistent labelings is given by

$$\sum_{e \in E(P)} y_e \geq y_{uv} \qquad \text{for all } uv \in E \text{ and all paths } P \text{ between } u \text{ and } v. \tag{5.7}$$

It is well known [CR93] that the constraint is facet-defining for the underlying polytope if and only if the cycle consisting of $P$ and $uv$ is chordless.

While for fully connected graphs, the constraints (5.7) can be represented by a polynomial number of triangle constraints [CR93; GW89; BDG+08], the separation procedure reduces to a sequence of shortest path problems in the general case [CR93]. Given $y$, the naive approach searches for each edge $uv \in E$ the shortest path from $u$ to $v$ in the weighted graph $G_y := (V, E, y)$. If this path is shorter than $y_{uv}$, it represents the most violated constraint of the form (5.7) for $uv$. Using a basic implementation of the Dijkstra algorithm (as we do) the cost for one search is $\mathcal{O}(|V|^2)$. This can be reduced to $\mathcal{O}(|E| + |V| \log |V|)$ by using Fibonacci heaps.

To reduce the number of shortest path searches we exploit the following three ideas:

**Efficient Bounds on the Shortest Path (B)** Instead of searching for each edge $uv \in E$ a shortest path from $u$ to $v$ in the graph $G_y = (V, E, y)$ we can compute a lower bound on the path length for all $uv \in E$ in $\mathcal{O}(|E| + |V|)$. To this end, we determine the connected components of the graph $G_{<\gamma} := (V, \{e \in E \mid y_e < \gamma\})$. If two nodes $u, v \in V$ are not in the same connected component of $G_{<\gamma}$, the length of a shortest path from $u$

[2]Note that sometimes solving the ILP is even faster than the LP.

to $v$ is greater than or equal to $\gamma$. Choosing $\gamma = 1$ yields a preprocessing procedure that enables to omit many shortest path searches. Furthermore, if the edge between two nodes has weight 0, this is obviously the shortest path since all edge weights $y_e$ are non-negative.

**Shortest Path in Binary Weighted Graph (I)** If the edge weights are either 0 or 1, simple breadth-first search can be applied instead of the Dijkstra algorithm. The computational effort can be reduced further by restricting the search to the graph $G_{=0} := (V, \{e \in E \mid y_e = 0\})$ since any path including an edge with weight 1 cannot be shorter than the edge between the two nodes which is 0 or 1.

**Finding Chordless Shortest Paths/Facet-Defining Constraints (F)** A path between the two nodes forming an edge is called *chordless* if the cycle consisting of the path and the edge has no chord. Shortest path search can be easily extended so as to determine the shortest chordless paths: Every node except for the endnode is not updated by the Dijkstra algorithm if the path from this node to the starting node is chordal. This increases the costs by a factor bounded by $|V|$. In view of cycle constraints, the corresponding constraints are facet-defining.

Our experiments, discussed in Chap. 6, show that the joint application of all three improvement methods B, I, and F leads to better runtimes in nearly all cases.

## Terminal Cycle Constraints

We can further reduce the costs for shortest path searches based on the following observation: In the presence of terminal nodes there exists no cycle $C$ of length greater than three that is chordless and contains a terminal node. This follows from the fact that there are edges from all nodes to all terminal nodes.

As a result, we can ignore all cycle constraints where the induced cycle has a length greater than three and includes a terminal node. All facet-defining cycle constraints that include a terminal node are then given by

$$y_{tu} + y_{tv} \geq y_{uv} \qquad \text{for all } uv \in E,\ t \in T, \tag{5.8a}$$

$$y_{tu} + y_{uv} \geq y_{tv} \qquad \text{for all } uv \in E,\ t \in T, \tag{5.8b}$$

$$y_{tv} + y_{uv} \geq y_{tu} \qquad \text{for all } uv \in E,\ t \in T. \tag{5.8c}$$

As a consequence we can exclude the terminal nodes when searching for general cycle constraints, which results in a graph that has $|T| \cdot |V|$ fewer edges.

Table 5.1: Comparison of terminal cycle constraints and multi terminal constraints: For a toy example consisting of two nodes and four labels, we show the implications of (5.8a)–(5.8c) and (5.9) on $y_{v_1 v_2}$ for a few values of $y_e$ for the terminal edges $e$. In the example in the third row, the constraints (5.9) improve the bound.

| $(y_{tv_1})_{t\in T}$ | $(y_{tv_2})_{t\in T}$ | (5.8a)–(5.8c) imply | (5.9) implies |
|---|---|---|---|
| $(1,1,1,0)$ | $(1,1,0,1)$ | $1 \leq y_{v_1v_2} \leq 1$ | $1 \leq y_{v_1v_2}$ |
| $(1,1,\frac{1}{2},\frac{1}{2})$ | $(1,1,\frac{1}{2},\frac{1}{2})$ | $0 \leq y_{v_1v_2} \leq 1$ | $0 \leq y_{v_1v_2}$ |
| $(\frac{1}{2},\frac{1}{2},1,1)$ | $(1,1,\frac{1}{2},\frac{1}{2})$ | $\frac{1}{2} \leq y_{v_1v_2} \leq \frac{3}{2}$ | $1 \leq y_{v_1v_2}$ |
| $(1,\frac{2}{10},\frac{3}{10},\frac{5}{10})$ | $(1,\frac{1}{10},\frac{2}{10},\frac{7}{10})$ | $\frac{2}{10} \leq y_{v_1v_2} \leq \frac{3}{10}$ | $\frac{2}{10} \leq y_{v_1v_2}$ |

### Multi Terminal Constraints

Călinescu et al. [CKR00] suggested another class of non-facet-defining inequalities that further tightens the relaxation:

$$y_{uv} \geq \sum_{t\in S} (y_{tu} - y_{tv}) \qquad \text{for all } uv \in E,\ S \subseteq T. \tag{5.9}$$

Intuitively, these constraints enforce each internal edge to be at least as active as all its pairs of terminal edges indicate. Since $\sum_{t\in T}(y_{tu} - y_{tv}) = 0$, we only consider differences in the direction from $u$ to $v$. An alternative representation of (5.9) exploiting symmetry is

$$y_{uv} \geq \sum_{t\in T} \frac{1}{2} |y_{tu} - y_{tv}| \qquad \text{for all } uv \in E. \tag{5.10}$$

In order to see why multi terminal constraints are useful, let us consider a tiny toy example of a graphical model with two nodes and four labels. Overall, the multicut graph has eight terminal edges $(tv_1)_{t\in T}$ and $(tv_2)_{t\in T}$ and one internal edge $v_1v_2$. We inspect a few values of $y$ and check if (5.9) is implied by (5.8a)–(5.8c) or not in Tab. 5.1. In the third row of the table the multi terminal constraints tighten the relaxation. It can be shown that these constraints may tighten the relaxation only if at least four terminal nodes are present.

### Odd-Wheel Constraints

While cycle constraints are sufficient to obtain optimal solutions if integer constraints are enforced, we may tighten the relaxation in the case $y_e \in [0,1]$ by adding more complex constraints.

One such a class of constraints for which the separation procedure can be carried out efficiently are odd-wheel constraints. A *wheel* $W = (V_W, E_W)$ is a graph with a selected

wheel

center node $c \in V_W$. There are edges between all other nodes and the center, and the remaining edges build a cycle containing all nodes in $V_W \setminus \{c\}$. An *odd-wheel* is a wheel with an odd number of non-center nodes. The odd-wheel constraints are given by

odd-wheel

$$\sum_{uv \in E_W, u,v \neq c} y_{uv} - \sum_{v \in V_W \setminus \{c\}} y_{cv} \leq \left\lfloor \frac{||V_W| - 1|}{2} \right\rfloor$$

for all odd-wheels $W = (V_W, E_W)$ that are subgraphs of $G$.

Deza et al. [DGL92] could prove that odd-wheel constraints are facet-defining for $||V_W| - 1| \geq 3$. As described in detail by Deza and Laurent [DL97] and Nowozin [Now09], the search for violated odd-wheel constraints can be reduced to a polynomial number of shortest path searches if the current solution does not violate any cycle constraints.

In our experiments, we found that with increasing sparsity odd-wheel constraints tighten the relaxation less. This is intuitively plausible since in densely connected graphs significantly more odd-wheels exist that could be violated.

The overall gain of including odd-wheel constraints was not better than with the previously proposed methods.

### 5.4.3 Rounding Fractional Solutions

Relaxations of the multicut problem yield solutions that may be fractional and therefore infeasible. The objective value then will be a lower bound of the optimal value. The procedure to map an infeasible solution to the feasible set is called *rounding*. For the resulting multicut, a corresponding labeling has to be determined.

rounding

Let in the following $y^*$ be a solution of a relaxation of the multicut problem.

#### Supervised Case

In the presence of terminal nodes, i.e., for Prob. 5.1, when $y^*$ is the solution of an ILP we assign to each node $v \in \mathcal{V}$ the label corresponding to the unique terminal node $t$ with $y^*_{tv} = 0$. This idea extends to the LP case by assigning to node $v$ the label $l$ with the lowest value of $y^*_{t_l v}$, i.e., the nearest corner in the corresponding simplex, see Fig. 5.4 (a):

$$x_v := \operatorname*{argmin}_{l \in L} \; y^*_{t_l v} \qquad\qquad \text{for all } v \in V \setminus T.$$

This heuristic *nearest label rounding* method has two drawbacks, however. Firstly, it does not provide any performance guarantee. Secondly, nearby nodes that favor two or more labels nearly equally might be randomly assigned to different labels due to numerical inaccuracy. This is particularly problematic when homogeneously labeled regions are preferred.

nearest label rounding

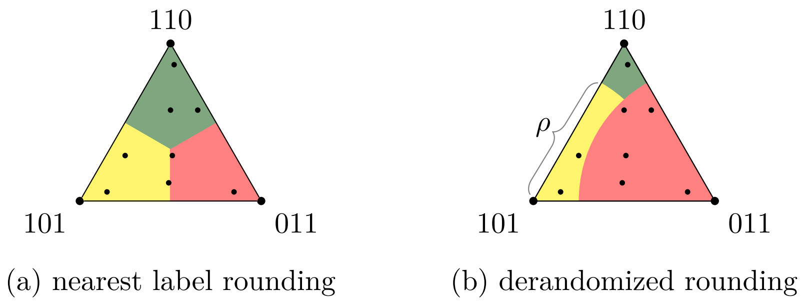


Figure 5.4: Illustration of the two rounding schemes for the multicut formulation of Prob. 5.1 for the vector $(y^*_{tv})_{t\in T}$. Nearest label rounding shown in (a) assigns each point in the simplex to the nearest vertex. In (b) one iteration of derandomized rounding is shown, exemplarily for $\rho = 0.75$.

Contrary to this local procedure, Călinescu et al. [CKR00] suggested a randomized rounding procedure that provides optimality bounds for Potts models with positive disagreement terms. Given a threshold $\rho \in [0, 1]$, they iterate over all labels in a fixed order and assign label $l$ to node $v$ if $y^*_{t_l v} \leq \rho$ and no label was assigned to $v$ before. In case no label was assigned to node $v$ in the end, the last label with respect to the ordering of the labels is assigned to $v$. This rounding procedure is sketched by Fig. 5.4 (b).

randomized rounding

A *randomized rounding* procedure would apply this for all $\rho \in [0, 1]$ and select the labeling with the lowest energy. Since $[0, 1]$ is uncountable, Călinescu et al. suggested a derandomized version. This is based on the observation that we only have to consider $|T| \cdot |V \setminus T|$ different threshold parameters, namely the values of the variables $y^*_{tv}$ for all $t \in T$ and $v \in V \setminus T$. Since this set can still be quite large, we also consider a heuristic approximation that we call *pseudo-derandomized rounding*, using a small number of equidistant thresholds, in practice $0, 0.01, 0.02, \ldots, 0.99, 1$.

pseudo-derandomized rounding

Concerning the tightness of the relaxation, Călinescu et al. [CKR00] pointed out that the integrality ratio of the relaxed LP for the second-order multiway cut problem with positive disagreement terms, exploiting cycle, terminal and multi terminal constraints, is $\frac{3}{2} - \frac{1}{k}$. This is superior to the $\alpha$-expansion algorithm [BVZ01] and the work of Dahlhaus et al. [DJP+92], which guarantees only a ratio of $2 - \frac{2}{k}$.

Empirically, we observed for these types of models that derandomized rounding and pseudo-derandomized rounding usually lead to results that are slightly better than when using nearest label rounding. While pseudo-derandomization does empirically not give results worse than original derandomization, it is much faster, but does not come along with theoretical guarantees. Fig. 5.5 shows results for two instances taken from [KAH+13]. While for the synthetic instances rounding matters, for real world examples the differences are negligible.

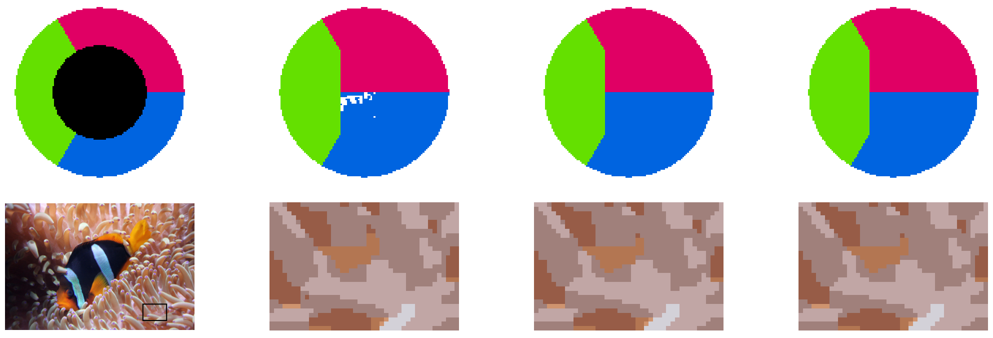

(a) input data (b) nearest label rounding (c) pseudo-derandomized rounding (d) derandomized rounding

Figure 5.5: Illustration of different rounding results (nearest label rounding, pseudo-derandomized rounding, and derandomized rounding) after solving the LP relaxation with terminal, multi terminal, and cycle inequalities for the instances “inpainting” (top row) and “clownfish” (bottom row) from [KAH+13]. Derandomized rounding and pseudo-derandomized rounding give similar results. Simply rounding to the nearest label can give inferior results (top row). However, for real applications the differences of the labelings are marginal (bottom row).

### Unsupervised Case

In absence of terminal nodes, i.e., for Prob. 5.2, if $y^*$ is integral we compute the connected components of $G_{=0} = (V, \{e \in E \mid y^*_e = 0\})$. We enumerate them and define $\#CC_{G_{=0}}(v)$, $v \in V$, to be the number of the connected component $v$ belongs to. This number is then used as the label of $v$, so

$$x_v := \#CC_{G_{=0}}(v) \qquad \text{for all } v \in V.$$

If $y^*$ is not integral we first have to map it to a vertex of the multicut polytope. To this end, we determine the connected components of $G_{\leq\kappa} = (V, \{e \in E \mid y^*_e \leq \kappa\})$ and define the feasible projection $\hat{y}$ by

$$\hat{y}_{uv} = \begin{cases} 0, & \text{if } \#CC_{G_{\leq\kappa}}(u) = \#CC_{G_{\leq\kappa}}(v), \\ 1, & \text{otherwise.} \end{cases}$$

The labeling then is given by

$$x_v := \#CC_{G_{\leq\kappa}}(v) \qquad \text{for all } v \in V.$$

Since this procedure tends to remove dangling edges, it seems to be reasonable to select $\kappa$ smaller than 0.5. This was empirically confirmed by our experiments.

### 5.4.4 Multicut Cutting-Plane Algorithm

Alg. 3 provides a compact description of our complete multicut approach, summarizing the present section. In addition to the specification of the objective function in terms of a graphical model $\mathcal{M}$, we expect a list of separation procedure sets $\mathcal{S}$ as input parameters. For example, $\mathcal{S}_1$ could represent simple cycle constraints separation, $\mathcal{S}_2$ integrality constraints, and $\mathcal{S}_3$ cycle constraints separation specialized to integer solutions.

As specified by Alg. 3, we construct the weighted graph $G$, introduce auxiliary variables for higher-order factors (as detailed in previous sections), and initialize the constraint set $\mathcal{C}$ by a simple relaxation of the feasible set.

We apply all separation procedures in $\bigcup_{j=1}^{i} \mathcal{S}_i$ to find violated constraints and add these to $\mathcal{C}$ until no more are found. Then we proceed with the next set $\mathcal{S}_{i+1}$.

The (integer) linear program in line 7 is solved by CPLEX 12.2, a standard off-the-shelf LP-solver. Finally, we compute an optimal labeling $x \in X$ from the multicut solution $y^*$.

**Algorithm 3** Multicut Algorithm MC

```
1:  Given: graphical model $\mathcal{M}$, list of separation procedure sets $\mathcal{S}$
2:  construct $G = (V, E, w)$ from $\mathcal{M}$ (see Sec. 5.2.2)
3:  add auxiliary variables for higher-order factors (see Sec. 5.3)
4:  initialize the constraint set $\mathcal{C}$ (see Sec. 5.4.2)
5:  for $i = 1, \ldots, |\mathcal{S}|$ do
6:      repeat
7:          compute $y^* \in \operatorname{argmin}_{y \in \mathcal{C}} \sum_{e \in E} w(e) y_e$
8:          $\bar{\mathcal{C}} \leftarrow$ violated constraints found by separation procedures $\bigcup_{j=1}^{i} \mathcal{S}_j$ for $y^*$
9:          $\mathcal{C} \leftarrow \mathcal{C} \cup \bar{\mathcal{C}}$
10:     until $\bar{\mathcal{C}} = \emptyset$
11: compute a labeling $x \in X$ based on $y^*$
```

The implementation of Alg. 3 turned out to be involved, due to several pitfalls necessitating some care. When solving the (I)LP one should not expect that the solution is feasible. Sometimes we observed negative values of $y_e$ and therefore project solutions always to $[0, 1]^{|E|}$. Also Boolean constraints were sometimes slightly violated. Most importantly, due to numerical reasons, constraints should only be added if they are significantly violated, i.e., the constraint $a \leq b$ is only added if $a \leq b - \epsilon$ does not hold. Ignoring this may not only lead to infinite loops for some instances, but may also significantly increase runtime. The parameter $\epsilon$ should be chosen depending on the precisions of the (I)LP solver. We use $\epsilon = 10^{-8}$.

# 6 Experiments

We here give application of the method presented in the thesis.

These experiments have already been published in [KSR+13b] and [KSR+13a].

## 6.1 Application of Reduction Techniques

In the following, we denote the algorithms introduced in Sec. 3.5 by ILP, BRAOBB, MCBC, and RPM, and the multicut algorithm 3 by MC followed by the reduction type (p, c, t) from Sec. 4.1 if preprocessing was applied. Additionally, we compare to TRWS [Kol06], FastPD [KT07], and $\alpha$-expansion [BVZ01] – all provided by the original authors of the papers. TRWS is stopped after 1 000 iterations. For evaluation we count how often a method provides the best energy value among all competing methods (*best*) and how often the gap between the energy value and the lower bound was less than $10^{-7}$ (*ver. opt*). For the synthetic models we use an Intel Pentium E5400, 2.7 GHz, 8 GB RAM, for the other a Xeon W3550, 3.07 GHz, 12 GB RAM. For MCBC we use a Xeon E5420, 2.5 GHz, 16 GB RAM for all experiments.

### 6.1.1 Decision Tree Fields

Decision tree fields (DTF) [NRB+11] are discriminatively learned conditional random fields (CRF). We consider the Chinese character models provided along with [NRB+11][1]. The dataset consists of 100 binary problems with first and second order terms. Variables are connected to more than 30 other variables by pairwise factors. Common approximations using local polytope relaxations do not perform well on these problems, and standard ILP solvers are not able to guarantee optimality in one hour, even if partial optimality is used to reduce the problem size. For the Chinese character dataset, partial optimality has reduced the problem size from 4 992–17 856 to 502–1 093 variables.

We set a time limit of one hour per instance. Using MCBC [Bon11] with the partial optimality reduction, we were able to verify optimality for 56 instances. Overall, we obtain superior results to all other methods, followed by ILP-pct. This is a significant progress – see Tab. 6.1. BRAOBB-p, which performed very good on the synthetic grid data, does not perform well on this dataset. We believe that this is caused by the larger size of the subproblems as well as the high connectivity which results in a higher treewidth.

[1] http://www.nowozin.net/sebastian/papers/DTF_CIP_instances.zip

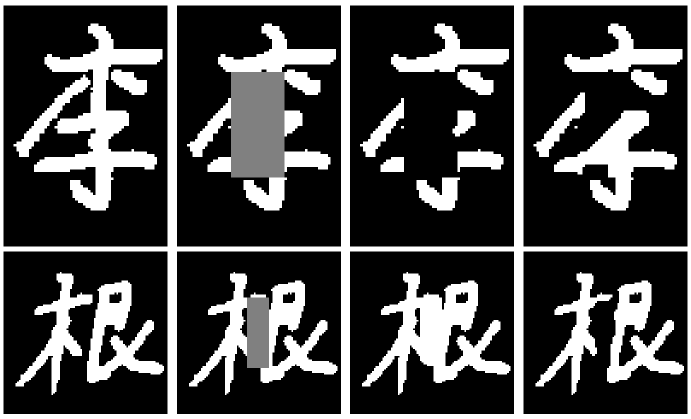


Figure 6.1: Examples from the Chinese character dataset. From left to right: Original image, occluded image, TRWS solution, MCBC solution.

Table 6.1: Results on the Chinese character dataset [NRB+11]

| algorithm | avg. runtime | avg. energy | avg. bound | best | ver. opt |
|---|---|---|---|---|---|
| ILP-pct | 3 581.42 | −49 542.87 | −50 071.87 | 30% | 0% |
| BRAOBB-p | 3 600.00 | −49 415.55 | $-\infty$ | 0% | 0% |
| MCBC-p | 2 053.89 | **− 49 550.10** | **−49 612.38** | **92%** | **56%** |
| TRWS | 100.13 | −49 496.84 | −50 119.41 | 2% | 0% |
| QPBO | **0.16** | −49 501.95 | −50 119.38 | 0% | 0% |
| SA [NRB+11] | n/a | −49 533.02 | $-\infty$ | 13% | 0% |

### 6.1.2 Multi-Label Potts Models

We also evaluate our approach on multi-label instances. To obtain partial optimality, we use the method of Kovtun [Kov03] and therefore restrict ourselves to second order Potts functions. While more general methods exist [KSR+08; SH11; KS11] these do not scale as well and will be subject to further work.

First, we consider the three **color segmentation** instances used in [AKT10]. While the standard multiway cut method [KSA+11] takes on average more than two minutes, we obtain runtimes comparable to state-of-the-art methods by using partial optimality for problem reduction and contrary to those verified globally optimal solutions, see Tab. 6.2. The reduction obtained by partial optimality was between 95% and 99.9%.

The differences between the energy values of the methods are quite small. Fig. 6.2 shows the calculated optimal labeling by MC (middle) and the differently labeled pixels by TRWS. Differences exist in boundary regions that might be not that important for applications. However, on this dataset MC-pct is comparable to approximative

Table 6.2: Results on the three color segmentation instances [AKT10]

| algorithm | avg. runtime | avg. energy | avg. bound | best | ver. opt |
|---|---|---|---|---|---|
| MC | 149.43 | **308 472 274.3** | **308 472 274.3** | **3/3** | **3/3** |
| MC-pct | **1.86** | **308 472 274.3** | **308 472 274.3** | **3/3** | **3/3** |
| TRWS | 150.47 | 308 472 310.6 | 308 472 270.4 | 2/3 | 1/3 |
| TRWS-pct | 3.90 | 308 472 274.3 | 308 472 274.3 | 2/3 | 2/3 |
| FastPD | **0.45** | 308 472 275.0 | $-\infty$ | 2/3 | 0/3 |
| FastPD-pct | **1.62** | 308 472 274.7 | $-\infty$ | 2/3 | 0/3 |
| $\alpha$-Exp | 6.42 | 308 472 275.6 | $-\infty$ | 2/3 | 0/3 |
| $\alpha$-Exp-pct | **1.72** | **308 472 274.3** | $-\infty$ | **3/3** | 0/3 |

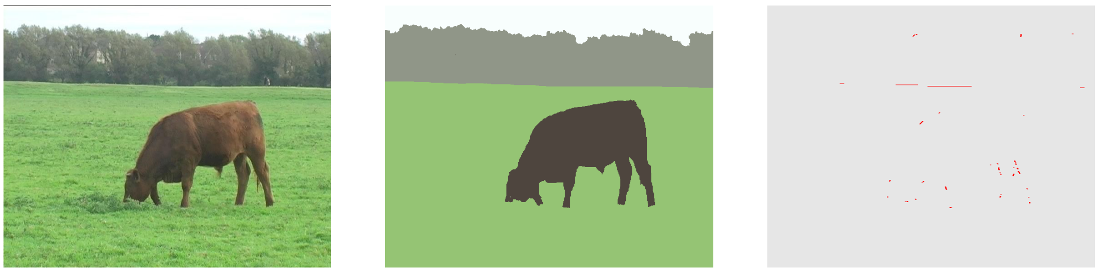


Figure 6.2: Color segmentation example from [AKT10]. **Left:** Original image. **Middle:** Optimal segmentation. **Right:** Pixels labeled differently by TRWS (256 out of 414 720 pixels).

state-of-the-art methods in terms of runtime and provides optimality. Furthermore, we would like to point out that most of the runtime of MC-pct is spent for the calculation of partial optimality.

We also apply model reduction for approximative methods as in [AKT10]. This improves the runtimes and energies of TRWS and $\alpha$-Exp as can also be seen in Tab. 6.2. For FastPD we oberve an increase in runtime.

Second, we investigate large scale **3D MRI brain segmentation**. We use simulated 3D MRI brain data [BW] and calculate the five color modes in the histogram. Our model contains five labels corresponding to the five intensity modes. The local data-term penalizes the $L_1$-distance between the voxel intensities and the mode intensities. We use a pairwise Potts term to penalize the boundary length using the 6-neighborhood in the 3D grid [BK04]. We choose a T1 pulse sequence with 3% noise relative to the brightest tissue and 20% intensity non-uniformity, see [BW] for details. The simulated slice thickness was set to 3, 5, 7, and 9 mm, which results in $181 \times 217 \times 60$, 36, 26, and 20 voxel volumes, respectively.

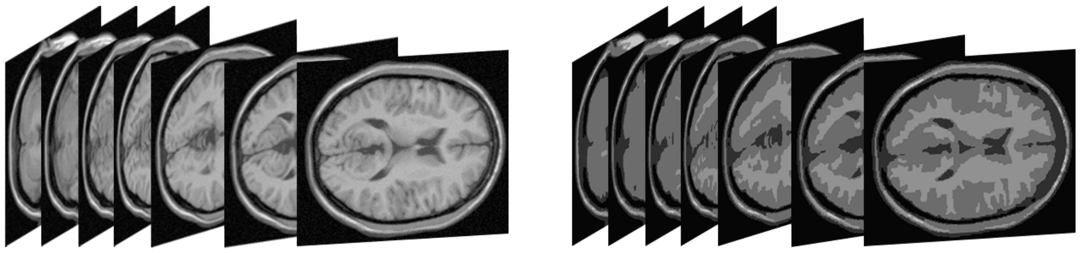


Figure 6.3: **Left:** Input data of the 3D MRI brain instance. It contains more than one million variables with five states. **Right:** Segmentation using the $L_1$-distance to the five intensity modes as data term together with a boundary length regularization.

Table 6.3: Results on the brain dataset with 7 mm slices [BW]

| algorithm | runtime | energy | bound | best | ver. opt |
|---|---|---|---|---|---|
| MC | 1 370.80 | **12 661 506** | **12 661 506** | **yes** | **yes** |
| MC-pct | 21.64 | **12 661 506** | **12 661 506** | **yes** | **yes** |
| TRWS | 589.67 | 12 661 590 | **12 661 506** | no | no |
| TRWS-pct | 335.27 | 12 661 572 | **12 661 506** | no | no |
| FastPD | **1.80** | 12 663 105 | $-\infty$ | no | no |
| FastPD-pct | 8.60 | 12 662 871 | $-\infty$ | no | no |
| $\alpha$-Exp | 53.25 | 12 662 909 | $-\infty$ | no | no |
| $\alpha$-Exp-pct | 9.91 | 12 662 793 | $-\infty$ | no | no |

For the instance with a thickness of 7 mm, MC requires 20 minutes for optimization. We can reduce the runtime to less than half a minute by using partial optimality, connected components, and tentacle elimination, see Tab. 6.3. FastPD (both with and without reduction) and $\alpha$-Exp-pct provide results that are a bit faster than that of MC-pct but worse in terms of energy. Notably, TRWS found the optimal bound but was not able to find the optimal integer solution during 1 000 iterations. For MC-pct, 951 982 variables are eliminated by partial optimality and 7 391 by tentacle elimination. Of the remaining 2 467 subproblems, the largest one contains 45 023 variables.

The runtimes for the different problem sizes are shown in Fig. 6.4. As already seen for the 7 mm instance, MC-pct is quite fast and at the same time provides verified optima for all problems. MC fails on the largest instance due to the lack of enough memory. For the approximate methods we show the gap to the optimal energy in Fig. 6.5. TRWS performs good but has worse runtimes than MC-pct.

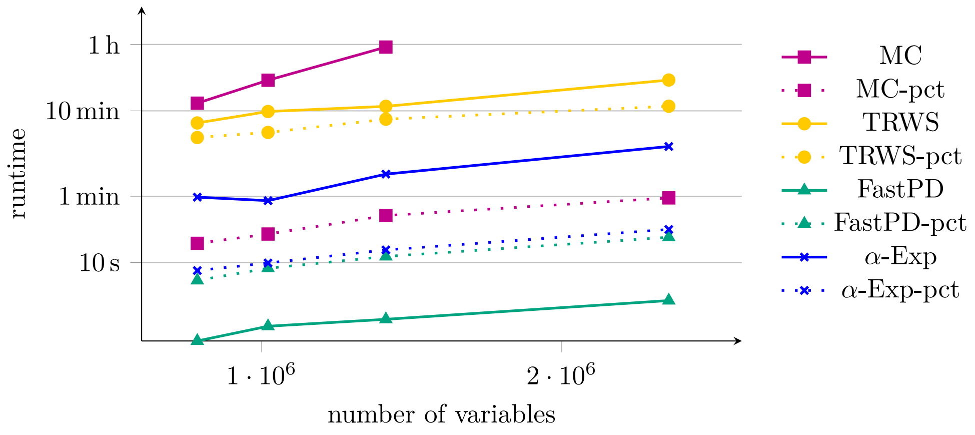


Figure 6.4: Runtimes for the brain dataset for different slice thicknesses which result in different numbers of variables. While for MC, the speedup caused by model-reduction is large, it is moderate or not present for approximative methods.

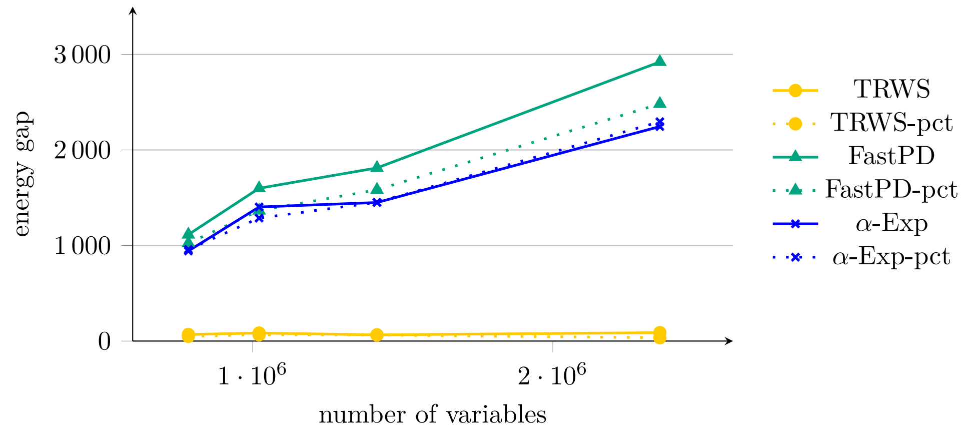


Figure 6.5: Energy gaps to the optimal energy for the brain dataset for different slice thicknesses which result in different numbers of variables. TRWS gives remarkably good but non-optimal solutions.

Table 6.4: Abbreviations for the separation procedures.

| | |
|---|---|
| **I** | integer constraints |
| **C** | cycle inequalities separation |
| **CF** | facet-defining cycle inequalities separation |
| **CI** | cycle inequalities separation for ILP |
| **CIF** | facet-defining cycle inequalities separation for ILP |
| **OW** | odd-wheel inequalities separation |
| **T** | terminal inequalities separation |
| **MT** | multi terminal inequalities separation |
| **TI** | terminal inequalities separation for ILP |
| **. . . B** | bounding for the shortest path search was used |

## 6.2 Application of Multicut Algorithm

### 6.2.1 Set-Up, Implementation Details

We implemented the separation procedures and reduction methods described above using C++ and the OpenGM2-library [ABK12b] for the factor graph representation, and CPLEX for solving ILPs and LPs in the inner loop of the iteration.

Our multicut approach encompasses a variety of algorithms which differ in the used inequalities, in the separation procedures, and in the order these procedures are applied. The abbreviations for single separation procedures are listed as Tab. 6.4.

For example, MC-CFB-I-CIF indicates:

- application of the multicut algorithm (MC) based on
- searching for violated facet-defining cycle inequalities (CF) using bounding (B),
- enforcing integer constraints (I), and finally
- searching for facet-defining cycle inequalities violated by the current *Boolean* solution (CIF), based on Breadth-First-Search instead of the Dijkstra algorithm.

We report for each dataset results averaged over all its instances:

1. the mean runtime: *runtime*,
2. the mean value of the integer solution after rounding: *value*,
3. the mean lower bound: *bound*,
4. how often the method found an integer solution with an objective value not larger than $10^{-6}$ compared to the overall best method for this instance: *best*, and

5. how often the method provided a gap between the objective value of the integer solution and the lower bound, that was smaller than $10^{-6}$: *ver. opt*, which we interpret as globally optimal for our instances.

In the *unsupervised case*, we compared the proposed methods with our implementation of the Kernighan-Lin (KL) algorithm [KL70] for the second-order case, as well as with iterative conditional mode (ICM) [Bes86] and Lazy Flipper (LF) [AKB+12a]. For *planar graphs*, an optimal segmentation with only four labels exists, and methods for the supervised case can be applied.

In the *supervised case*, we compared with TRWS [Kol06], $\alpha$-expansion [BVZ01] and FastPD [KT07] – using in each case code provided by the respective authors of these papers. Furthermore, we compared to commercial LP- and ILP-solvers in the nodal domain, LBP, TRBP, and $\alpha$-fusion, as provided by OpenGM2.

### 6.2.2 Probabilistic Image Segmentation

The probabilistic image segmentation framework is due to Andres et al. [AKB+11] and belongs to the class of unsupervised image segmentation problems. These problem instances involve $156 \cdots 3764$ superpixels. For all pairs of adjacent superpixels, the likelihood that their common part of the superpixel boundary is part of the segmentation, is learned offline by a random forest. This results in a Potts model with positive and negative coupling constraints. While the connection to Potts models is not mentioned in [AKB+11], they use a similar optimization scheme as in the present work. They introduced a higher-order model as well as a second-order one. Only the latter has been made publicly available in [KAH+13].

**Second-order Case.** As shown in Tab. 6.5, for this dataset, we profit from using ILP subproblems. This reduces the mean runtime to less than 3 seconds and is therefore empirically faster than LP-based cutting-plane methods and the heuristic KL-algorithm. ICM and LF perform worse than KL. With increasing search space LF outperforms KL. For a search-depth greater than 1 we make use of the fact that the instances are planar and an optimal solution with four labels exists. The same trick is used to make TRWS applicable. Additionally, we fix the first variable and initialize messages randomly. Even this does not help to prevent TRWS from running into poor local fix-points. In both cases the label reduction is marked by the postfix *L4*.

Concerning the multicut approach, odd-wheel constraints only marginally improve the results. LP-based cutting-plane methods find the optimal solution for 35 of 100 instances and are slower than ILP-based methods, too.

**Higher-order Case.** The third-order models from [AKB+11] are hard to solve with relaxations, hence rounding becomes more important. The additional third-order factors favor smooth boundary continuation. Since this sometimes conflicts with local boundary probabilities, the problem becomes more involved.

Table 6.5: Second-order probabilistic image segmentation [AKB+11; KAH+13]

| algorithm | runtime | value | bound | best | ver. opt |
|---|---|---|---|---|---|
| KL | 4.96 s | 4608.57 | $-\infty$ | 0.0% | 0.0% |
| ICM | 6.03 s | 4705.07 | $-\infty$ | 0.0% | 0.0% |
| LF1 | 2.35 s | 4705.01 | $-\infty$ | 0.0% | 0.0% |
| LF2-L4 | 0.13 s | 4627.38 | $-\infty$ | 0.0% | 0.0% |
| LF3-L4 | 3.16 s | 4581.83 | $-\infty$ | 0.0% | 0.0% |
| LF4-L4 | 176.47 s | 4555.73 | $-\infty$ | 0.0% | 0.0% |
| TRWS-L4 | 0.84 s | 4889.23 | 4096.53 | 0.0% | 0.0% |
| MC-C | 14.02 s | 4447.47 | 4442.34 | 35.0% | 35.0% |
| MC-CB | **4.71 s** | 4447.47 | 4442.34 | 35.0% | 35.0% |
| MC-CF | 11.35 s | 4447.47 | 4442.34 | 35.0% | 35.0% |
| MC-CFB | 5.16 s | 4447.47 | 4442.34 | 35.0% | 35.0% |
| MC-C-OW | 14.08 s | 4447.41 | 4442.34 | 35.0% | 35.0% |
| MC-CB-OW | **4.81 s** | 4447.41 | 4442.34 | 35.0% | 35.0% |
| MC-CF-OW | 11.45 s | 4447.41 | 4442.34 | 35.0% | 35.0% |
| MC-CFB-OW | 5.19 s | 4447.41 | 4442.34 | 35.0% | 35.0% |
| MC-I-CI | 2.78 s | 4442.64 | 4442.64 | 100.0% | 100.0% |
| MC-I-CIF | **2.20 s** | 4442.64 | 4442.64 | 100.0% | 100.0% |
| MC-C-I-CI | 15.00 s | 4442.64 | 4442.64 | 100.0% | 100.0% |
| MC-CFB-I-CIF | 5.69 s | 4442.64 | 4442.64 | 100.0% | 100.0% |

Table 6.6: Third-order probabilistic image segmentation [AKB+11]

| algorithm | runtime | value | bound | best | ver. opt |
|---|---|---|---|---|---|
| ICM | 10.79 *(8.11)* s | 6030.49 | $-\infty$ | 0.0% | 0.0% |
| LF | 4.17 *(3.11)* s | 6030.29 | $-\infty$ | 0.0% | 0.0% |
| MC-C | 43.82 *(9.33)* s | 6657.32 | 5465.15 | 0.0% | 0.0% |
| MC-CB | 42.86 *(9.26)* s | 6657.32 | 5465.15 | 0.0% | 0.0% |
| MC-CF | 26.68 *(8.06)* s | 6658.28 | 5465.15 | 0.0% | 0.0% |
| MC-CFB | 25.00 *(6.64)* s | 6658.28 | 5465.15 | 0.0% | 0.0% |
| MC-C-OW | 43.71 *(11.16)* s | 6657.12 | 5465.29 | 0.0% | 0.0% |
| MC-CB-OW | 43.38 *(9.66)* s | 6657.12 | 5465.29 | 0.0% | 0.0% |
| MC-CF-OW | 27.62 *(8.15)* s | 6658.08 | 5465.29 | 0.0% | 0.0% |
| MC-CFB-OW | 25.55 *(7.40)* s | 6658.08 | 5465.29 | 0.0% | 0.0% |
| MC-I-C | 689.79 *(41.43)* s | 5627.52 | 5627.52 | 100.0% | 100.0% |
| MC-I-CFB | 469.87 *(33.02)* s | 5627.52 | 5627.52 | 100.0% | 100.0% |
| MC-I-CI | 119.64 *(31.73)* s | 5627.52 | 5627.52 | 100.0% | 100.0% |
| MC-I-CIF | 72.81 *(27.39)* s | 5627.52 | 5627.52 | 100.0% | 100.0% |
| MC-C-I-CI | 125.33 *(33.63)* s | 5627.52 | 5627.52 | 100.0% | 100.0% |
| MC-CFB-I-CIF | 82.00 *(25.60)* s | 5627.52 | 5627.52 | 100.0% | 100.0% |

As shown in Tab. 6.6, local search methods give better results than relaxed solutions after rounding. Our exact multicut scheme was able to solve all instances to optimality. Notably, one instance was significantly harder than all others and took more than half of the overall runtime for MC-I-C and MC-I-CFB.

Overall, a few instances are significantly harder than others. This is apparent by the large difference of the mean runtime to the median runtime (the latter is shown in parentheses in Tab. 6.6).

### 6.2.3 Higher-order Hierarchical Image Segmentation

The hierarchical image segmentation framework was suggested by Kim et al. [KNK+11] and also belongs to the class of unsupervised image segmentation problems. Contrary to the work of Andres et al. [AKB+11], they learn their model-parameters by a structured support vector machine (S-SVM). Furthermore, higher-order Potts terms force selected regions to belong to the same cluster. The 715 instances of this dataset, published as part of [KAH+13], contain factors of order up to a few hundred and 122–651 variables.

The results are summarized as Table 6.7. Surprisingly, our LP-based methods perform better than the original algorithm used in [KNK+11], even though the algorithms are identical. Maybe this was caused by the different LP solver they used, or by

Table 6.7: Higher-order hierarchical image segmentation [KNK+11].

| algorithm | runtime | value | bound | best | ver. opt |
|---|---|---|---|---|---|
| ICM | 1.90 s | −585.60 | $-\infty$ | 0.0% | 0.0% |
| LF | 1.00 s | −585.60 | $-\infty$ | 0.0% | 0.0% |
| MC-C | 0.23 s | −625.97 | −628.89 | 19.9% | 13.7% |
| MC-CB | 0.12 s | −625.97 | −628.89 | 19.9% | 13.7% |
| MC-CF | 0.20 s | −625.97 | −628.89 | 19.9% | 13.7% |
| MC-CFB | **0.11 s** | −625.97 | −628.89 | 19.9% | 13.7% |
| MC-C-OW | 0.24 s | −625.98 | −628.89 | 20.1% | 14.0% |
| MC-CB-OW | 0.14 s | −625.98 | −628.89 | 20.1% | 14.0% |
| MC-CF-OW | 0.21 s | −625.98 | −628.89 | 20.1% | 14.0% |
| MC-CFB-OW | **0.13 s** | −625.98 | −628.89 | 20.1% | 14.0% |
| MCR [KNK+11] | 0.38 s | −624.35 | −629.03 | 16.4% | 10.2% |
| MC-CI | 1.14 s | −628.16 | −628.16 | 100.0% | 100.0% |
| MC-CIF | 1.04 s | −628.16 | −628.16 | 100.0% | 100.0% |
| MC-C-CI | 0.85 s | −628.16 | −628.16 | 100.0% | 100.0% |
| MC-CFB-CIF | **0.62 s** | −628.16 | −628.16 | 100.0% | 100.0% |

some floating-point problems inside their separation procedure. The use of odd-wheel constraints marginally improves the results. Best results are obtained by using integer cutting-planes after having solved the LP. The use of the bounding as part of the post-processing reduces runtime by a factor of 2. The differences to only using facet-defining constraints are negligible.

### 6.2.4 Modularity Clustering

We also considered a clustering problem from outside the field of computer vision, which contrary to the previous models considered so far, involves a fully connected graph. Modularity clustering [BDG+08] means the problem of clustering an undirected unweighted graph into "meaningful" subsets, which amounts to optimization problems related to fully connected Potts model. For our experiments, we used the datasets[2] *dolphins*, *football*, *karate*, and *lesmis*, with 62, 115, 34, and 77 data-points, respectively.

As shown in Tab. 6.8, for modularity clustering, the use of facet-defining inequalities as well as odd-wheel constraints significantly improves the results. We attribute this to the high connectivity of the graph. In such dense graphs more likely violated odd-wheel inequalities exist. Likewise, more *non*-facet-defining cycle inequalities exist as well, and

[2] http://www-personal.umich.edu/~mejn/netdata/

Table 6.8: Modularity clustering [BDG+08]

| algorithm | runtime | value | bound | best | ver. opt |
|---|---|---|---|---|---|
| KL | 0.01 s | −0.5251 | $-\infty$ | 2/4 | 0/4 |
| ICM | 0.12 s | 0.0000 | $-\infty$ | 0/4 | 0/4 |
| LF | 0.05 s | 0.0000 | $-\infty$ | 0/4 | 0/4 |
| MC-C | 47.99 s | −0.5204 | −0.5294 | 1/4 | 1/4 |
| MC-CB | 48.33 s | −0.5204 | −0.5294 | 1/4 | 1/4 |
| MC-CF | 1.02 s | −0.5204 | −0.5294 | 1/4 | 1/4 |
| MC-CFB | 0.91 s | −0.5204 | −0.5294 | 1/4 | 1/4 |
| MC-C-OW | 72.05 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-CB-OW | 72.42 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-CF-OW | 12.26 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-CFB-OW | 11.60 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-I-C | 152.20 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-I-CI | 14.57 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-I-CIF | 6.31 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-I-CCFDB | 6.56 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-C-I-CI | 58.24 s | −0.5282 | −0.5282 | 4/4 | 4/4 |
| MC-CFB-I-CIF | 1.31 s | −0.5282 | −0.5282 | 4/4 | 4/4 |

adding those only blows up the system of inequalities.

As observed by Nowozin and Jegelka [NJ09], odd-wheel inequalities usually tighten sufficiently the polytope. Furthermore, we observed for this dataset, as in [NJ09], numerical problems if the allowed feasibility and optimality tolerances were set too large. However, the experiments showed that our proposed integer cycle inequalities perform better than odd-wheel separation, especially if we start from the LP-relaxation with cycle inequalities, see Tab. 6.8.

### 6.2.5 Supervised Image Segmentation

An elementary approach to supervised image segmentation, or image labeling, is to apply locally a statistical classifier, trained offline beforehand, to raw image data or to locally extracted image features. This is complemented by a non-local prior term, the most common form of which favours short boundaries of the segments partitioning the image domain. Such terms can be approximated by pairwise Potts terms [BK03] and

Table 6.9: Supervised image segmentation [AKT10]

| algorithm | runtime | value | bound | best | ver. opt |
|---|---|---|---|---|---|
| MC-T-MT-I-T | 149.43 s | 308 472 274.3 | 308 472 274.3 | 3/3 | 3/3 |
| MC*-T-MT-I-T | 1.86 s | 308 472 274.3 | 308 472 274.3 | 3/3 | 3/3 |
| ILP | † | † | † | † | † |
| ILP* | 1.91 s | 308 472 274.3 | 308 472 274.3 | 3/3 | 3/3 |
| MC-T-MT | 115.14 s | 308 472 274.3 | 308 472 274.3 | 3/3 | 3/3 |
| MC*-T-MT | 1.76 s | 308 472 274.3 | 308 472 274.3 | 3/3 | 3/3 |
| LP | † | † | † | † | † |
| LP* | 2.17 s | 308 472 274.3 | 308 472 274.3 | 3/3 | 3/3 |
| TRWS [Kol06] | 150.47 s | 308 472 310.6 | 308 472 270.4 | 2/3 | 1/3 |
| TRWS* | 3.90 s | 308 472 274.3 | 308 472 274.3 | 2/3 | 2/3 |
| FastPD [KT07] | 0.45 s | 308 472 275.0 | $-\infty$ | 2/3 | 0/3 |
| FastPD* | 1.62 s | 308 472 274.7 | $-\infty$ | 2/3 | 0/3 |
| $\alpha$-Exp [BVZ01] | 6.42 s | 308 472 275.6 | $-\infty$ | 2/3 | 0/3 |
| $\alpha$-Exp* | 1.72 s | 308 472 274.3 | $-\infty$ | 3/3 | 0/3 |

lead to an energy function of the form

$$\sum_{f\in\mathcal{F}_1} -\log(p_{ne(f)}(x_{ne(f)}|I)) + \sum_{f\in\mathcal{F}_2} \beta\,\mathbb{I}(x_{ne(f)_1} \neq x_{ne(f)_2}).$$

As recently shown by Kappes et al. [KSR+13b], such models can be evaluated globally optimal and very fast by first determining partial optimality, leading to a reduced inference problem in terms of remaining unlabelled connected image components, followed by solving each of these smaller problems independently.

We use the labels "*" to mark when these preprocessing steps were applied and "†" to mark whenever the memory requirement exceeded 12 GB.

As dataset we used the color segmentation instances of Alahari et al. [AKT10]. The results are summarized as Table 6.9.

While standard (I)LP solvers often suffer from their large memory requirements, the multicut approach outperformed all other approaches. Since for all instances the local polytope relaxation returned optimal integer solutions, MC-T-MT could solve them in polynomial time. When we resorted to the model reduction *, the subproblems became small for these problem instances, and (I)LP solvers could be conveniently applied. Our multicut approach then was only marginally faster. Despite global optimality, however, the runtime was comparable to algorithms for approximate inference that do not guarantee global optimality.

### 6.2.6 Higher-Order Supervised Image Segmentation

We studied image segmentation with junction regularisation as problem instances that benefit from the application of higher-order generalized Potts functions.

Rather than merely penalizing the boundary length of segments, this approach aims at improving segmentation results by additionally penalizing points where the boundaries of three or more segments meet:

$$\theta^I(x_1, x_2, x_3, x_4) = \begin{cases} \lambda, & \text{if } |\{x_1, x_2, x_3, x_4\}| > 2, \\ 0, & \text{else.} \end{cases}$$

The overall energy for labeling then is given by

$$\sum_{f \in \mathcal{F}_1} \theta^1_f(x_{ne(f)}) + \sum_{f \in \mathcal{F}_2} \theta^2_f(x_{ne(f)}) + \sum_{f \in \mathcal{F}_4} \theta^I(x_{ne(f)}),$$

where $\theta^1$ denotes the $L_1$-norm of the difference between intensity of a pixel and a pixel-label, $\theta^2$ the same second-order terms as in the pairwise case, and $\mathcal{F}_4$ the set of all factors over four pixels that build a cycle in the image grid.

Setting $\lambda$ to 0 yields standard second-order model with boundary length regularization, whereas setting $\lambda \to \infty$ yields a model that enforces segments to be surrounded by one single segment.

The results of an empirical evaluation for 10 synthetic $32 \times 32$ images are summarized as Table 6.10.

Approximate inference methods performed quite good, but among those only LBP-LF2 (Lazzy Flipper initialed with the solution of LBP) was able to provide nearly optimal results. While the multicut approach is on par when relaxations were considered, it became quite slow compared to a ILP applied to labeling in the nodal domain, when a globally optimal solution was enforced.

We believe there are two major reasons: First, the relaxation "prefers" less integral solutions due to the higher-order terms and therefore becomes harder to solve for LP-based methods. Second, we observe that CPLEX solves the ILP mainly by branching and probing in order to avoid solving LPs. This is also the reason why ILP is faster than LP.

Table 6.10: Supervised image segmentation with inclusion priors

| algorithm | runtime | value | bound | best | ver. opt |
|---|---|---|---|---|---|
| ICM | 0.03 s | 1556.20 | $-\infty$ | 0/10 | 0/10 |
| LBP-LF2 | 12.20 s | 1400.62 | $-\infty$ | 8/10 | 0/10 |
| $\alpha$-FUSION | 0.07 s | 1587.13 | $-\infty$ | 0/10 | 0/10 |
| LBP | 12.28 s | 1800.67 | $-\infty$ | 3/10 | 0/10 |
| TRBP | 13.93 s | 2000.67 | $-\infty$ | 2/10 | 0/10 |
| LP | 25.04 s | 3900.59 | 1400.33 | 1/10 | 1/10 |
| MC-T-MT | 18.55 s | 1739.29 | 1399.49 | 1/10 | 0/10 |
| ILP | 7.33 s | 1400.57 | 1400.57 | 10/10 | 10/10 |
| MC-T-MT-I-T | 66.58 s | 1400.57 | 1400.57 | 10/10 | 10/10 |

# Bibliography

[ABK12a] Björn Andres, Thorsten Beier, and Jörg H. Kappes. OpenGM: A C++ library for discrete graphical models. `http://arxiv.org/abs/1206.0111`. 2012 (cited on pages 2, 37).

[ABK12b] Björn Andres, Thorsten Beier, and Jörg H. Kappes. OpenGM2. `http://hci.iwr.uni-heidelberg.de/opengm2/`. 2012 (cited on page 78).

[AG12] Amir Alush and Jacob Goldberger. Ensemble segmentation using efficient integer linear programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(10):1966–1977, 2012 (cited on page 2).

[Aig79] Martin Aigner. *Combinatorial Theory*. Springer, 1979 (cited on page 61).

[AKB+11] Björn Andres, Jörg H. Kappes, Thorsten Beier, Ullrich Köthe, and Fred A. Hamprecht. Probabilistic image segmentation with closedness constraints. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2011 (cited on pages 2, 66, 79–81).

[AKB+12a] Björn Andres, Jörg H. Kappes, Thorsten Beier, Ullrich Köthe, and Fred A. Hamprecht. The lazy flipper: Efficient depth-limited exhaustive search in discrete graphical models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012 (cited on page 79).

[AKB+12b] Björn Andres, Thorben Kröger, Kevin L. Briggman, Winfried Denk, Natalya Korogod, Graham Knott, Ullrich Köthe, and Fred A. Hamprecht. Globally optimal closed-surface segmentation for connectomics. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012 (cited on page 2).

[AKT10] Karteek Alahari, Pushmeet Kohli, and Philip H. S. Torr. Dynamic hybrid algorithms for MAP inference in discrete MRFs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(10):1846–1857, 2010 (cited on pages 41, 74, 75, 84).

[ASS+10] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. SLIC Superpixels. Technical report. École Polytechnique Fédérale de Lausanne, 2010 (cited on page 47).

[BBC04] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation clustering. *Machine Learning*, 56(1–3):89–113, 2004 (cited on pages 57, 58).

[BDG+08] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Görke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE Transactions on Knowledge and Data Engineering*, 20(2):172–188, 2008 (cited on pages 66, 82, 83).

[Bes86] Julian Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society, Series B*, 48(3):259–302, 1986 (cited on page 79).

[BH02] Endre Boros and Peter L. Hammer. Pseudo-Boolean optimization. *Discrete Applied Mathematics*, 123(1–3):155–225, 2002 (cited on page 40).

[BK03] Yuri Boykov and Vladimir Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2003 (cited on page 83).

[BK04] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004 (cited on pages 35, 75).

[Bol98] Béla Bollobás. *Modern Graph Theory*. Springer, 1998 (cited on page 5).

[Bon11] Thorsten Bonato. *Contraction-based Separation and Lifting for Solving the Max-Cut Problem*. Optimus Verlag, 2011 (cited on pages 37, 73).

[Brø83] Arne Brøndsted. *An Introduction to Convex Polytopes*. Springer, 1983 (cited on page 9).

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001 (cited on pages 2, 35, 70, 73, 79, 84).

[BW] BrainWeb: Simulated brain database. `http://brainweb.bic.mni.mcgill.ca/brainweb/` (cited on pages 75, 76).

[Chr97] Thomas Christof. PORTA – POlyhedron Representation Transformation Algorithm. `http://www.iwr.uni-heidelberg.de/groups/comopt/software/PORTA/`. 1997 (cited on pages 32, 33).

[CKR00] Gruia Călinescu, Howard Karloff, and Yuval Rabani. An improved approximation algorithm for multiway cut. *Journal of Computer and System Sciences*, 60(3):564–574, 2000 (cited on pages 68, 70).

[CLR05] Marie-Christine Costa, Lucas Létocart, and Frédéric Roupin. Minimal multicut and maximal integer multiflow: A survey. *European Journal of Operational Research*, 162(1):55–69, 2005 (cited on page 58).

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Press, 1990 (cited on page 41).

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1971 (cited on page 23).

[CR91] Sunil Chopra and Mendu R. Rao. On the multiway cut polyhedron. *Networks*, 21(1):51–89, 1991 (cited on pages 2, 60).

[CR93] Sunil Chopra and Mendu R. Rao. The partition problem. *Mathematical Programming*, 59(1–3):87–115, 1993 (cited on page 66).

[Dak65] R. J. Dakin. A tree-search algorithm for mixed integer programming problems. *The Computer Journal*, 8(3):250–255, 1965 (cited on page 16).

[Dan51] George B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 339–347. John Wiley & Sons, 1951 (cited on page 12).

[DGL91] Michel M. Deza, Martin Grötschel, and Monique Laurent. Complete descriptions of small multicut polytopes. In Peter Gritzmann and Bernd Sturmfels, editors, *Applied Geometry and Discrete Mathematics: The Victor Klee Festschrift*, pages 221–252. American Mathematical Society, 1991 (cited on page 58).

[DGL92] Michel M. Deza, Martin Grötschel, and Monique Laurent. Clique-web facets for multicut polytopes. *Mathematics of Operations Research*, 17(4):981–1000, 1992 (cited on page 69).

[Die97] Reinhard Diestel. *Graph Theory*. Springer, 1997 (cited on pages 5, 7).

[DJP+92] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiway cuts (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992 (cited on pages 2, 70).

[DJP+94] Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM Journal on Computing*, 23(4):864–894, 1994 (cited on page 2).

[DL97] Michel M. Deza and Monique Laurent. *Geometry of Cuts and Metrics*. Springer, 1997 (cited on pages 29, 69).

[FSW11] Vojtěch Franc, Sören Sonnenburg, and Tomáš Werner. Cutting plane methods in machine learning. In Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright, editors, *Optimization for Machine Learning*, pages 185–218. MIT Press, 2011 (cited on page 2).

[GH94] Olivier Goldschmidt and Dorit S. Hochbaum. A polynomial algorithm for the k-cut problem for fixed k. *Mathematics of Operations Research*, 19(1):24–37, 1994 (cited on page 58).

[GJ07] Amir Globerson and Tommi Jaakkola. Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2007 (cited on page 37).

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979 (cited on pages 7, 58).

[Gom58] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958 (cited on page 15).

[GPS89] D. M. Greig, B. T. Porteous, and Allan H. Seheult. Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society, Series B*, 51(2):271–279, 1989 (cited on page 35).

[GW74] Fred Glover and Eugene Woolsey. Converting the 0-1 polynomial programming problem to a 0-1 linear program. *Operations Research*, 22(1):180–182, 1974 (cited on page 62).

[GW89] Martin Grötschel and Yoshiko Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1):59–96, 1989 (cited on page 66).

[Ham65] Peter Hammer. Some network flow problems solved with pseudo-Boolean programming. *Operations Research*, 13(3):388–399, 1965 (cited on page 26).

[HHS84] Peter L. Hammer, Pierre Hansen, and Bruno Simeone. Roof duality, complementation and persistency in quadratic 0-1 optimization. *Mathematical Programming*, 28(2):121–155, 1984 (cited on page 36).

[JF56] Lester R. Ford Jr. and Delbert R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956 (cited on page 28).

[KAH+13] Jörg H. Kappes, Björn Andres, Fred A. Hamprecht, Christoph Schnörr, Sebastian Nowozin, Dhruv Batra, Sungwoong Kim, Bernhard X. Kausler, Jan Lellmann, Nikos Komodakis, and Carsten Rother. A comparative study of modern inference techniques for discrete energy minimization problems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013 (cited on pages 35, 70, 71, 79–81).

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972 (cited on pages 13, 24).

[Kar84] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984 (cited on page 13).

[KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009 (cited on pages 33, 44).

[KFL01] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001 (cited on page 19).

[Kha79] Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademii Nauk SSSR*, 244:1093–1096, 1979 (cited on page 13).

[KL70] Brian W. Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell Systems Technical Journal*, 49(2):291–307, 1970 (cited on page 79).

[KNK+11] Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang D. Yoo. Higher-order correlation clustering for image segmentation. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2011 (cited on pages 64, 66, 81, 82).

[KNK+13] Sungwoong Kim, Sebastian Nowozin, Pushmeet Kohli, and Chang D. Yoo. Task-specific image partitioning. *IEEE Transactions on Image Processing*, 22(2):488–500, 2013 (cited on page 2).

[Kol06] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568–1583, 2006 (cited on pages 2, 36, 57, 73, 79, 84).

[Kov03] Ivan Kovtun. Partial optimal labeling search for a NP-hard subclass of (max, +) problems. In *Proceedings of the DAGM Symposium*, 2003 (cited on pages 40, 74).

[KPT11] Nikos Komodakis, Nikos Paragios, and Georgios Tziritas. MRF energy minimization and beyond via dual decomposition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):531–552, 2011 (cited on page 2).

[KS11] Fredrik Kahl and Petter Strandmark. Generalized roof duality for pseudo-Boolean optimization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2011 (cited on page 74).

[KSA+11] Jörg H. Kappes, Markus Speth, Björn Andres, Gerhard Reinelt, and Christoph Schnörr. Globally optimal image partitioning by multicuts. In *Proceedings of the International Conference on Energy Minimization Methods in Computer Vision and Pattern Recognition (EMMCVPR)*, 2011 (cited on pages 4, 57, 66, 74).

[KSR+08] Pushmeet Kohli, Alexander Shekhovtsov, Carsten Rother, Vladimir Kolmogorov, and Philip H. S. Torr. On partial optimality in multi-label MRFs. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2008 (cited on pages 40, 74).

[KSR+13a] Jörg H. Kappes, Markus Speth, Gerhard Reinelt, and Christoph Schnörr. Higher-order segmentation via multicuts. `http://arxiv.org/abs/1305.6387`. 2013 (cited on pages 4, 55, 73).

[KSR+13b] Jörg H. Kappes, Markus Speth, Gerhard Reinelt, and Christoph Schnörr. Towards efficient and exact MAP-inference for large scale discrete computer vision problems via combinatorial optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013 (cited on pages 4, 37, 39, 73, 84).

[KSS12] Jörg H. Kappes, Bogdan Savchynskyy, and Christoph Schnörr. A bundle approach to efficient MAP-inference by Lagrangian relaxation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012 (cited on page 2).

[KT07] Nikos Komodakis and Georgios Tziritas. Approximate labeling via graph cuts based on linear programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1436–1453, 2007 (cited on pages 2, 36, 73, 79, 84).

[KV00] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer, 2000 (cited on pages 7, 11).

[LD60] Alisa H. Land and Alison G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960 (cited on page 16).

[LSK+09] Alex Levinshtein, Adrian Stere, Kiriakos N. Kutulakos, David J. Fleet, Sven J. Dickinson, and Kaleem Siddiqi. TurboPixels: Fast superpixels using geometric flows. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2290–2297, 2009 (cited on page 47).

[Mar12] Dániel Marx. A tight lower bound for planar multiway cut with fixed number of terminals. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2012 (cited on page 2).

[MFT+01] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2001 (cited on page 48).

[Min96] Hermann Minkowski. *Geometrie der Zahlen*. Teubner, 1896 (cited on page 10).

[MLH12] Yansheng Ming, Hongdong Li, and Xuming He. Connected contours: A new contour completion model that respects the closure effect. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012 (cited on pages 2, 66).

[Mor] Greg Mori. http://www.cs.sfu.ca/~mori/research/superpixels/ (cited on page 47).

[NJ09] Sebastian Nowozin and Stefanie Jegelka. Solution stability in linear programming relaxations: Graph partitioning and unsupervised learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2009 (cited on pages 66, 83).

[Now09] Sebastian Nowozin. Learning with Structured Data: Applications to Computer Vision. PhD thesis. Technische Universität Berlin, 2009 (cited on page 69).

[NRB+11] Sebastian Nowozin, Carsten Rother, Shai Bagon, Toby Sharp, Bangpeng Yao, and Pushmeet Kohli. Decision tree fields. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2011 (cited on pages 73, 74).

[OD11] Lars Otten and Rina Dechter. Anytime AND/OR depth-first search for combinatorial optimization. In *Proceedings of the Annual Symposium on Combinatorial Search (SOCS)*, 2011 (cited on pages 2, 37).

[PIC11] The probabilistic inference challenge (PIC 2011). http://www.cs.huji.ac.il/project/PASCAL/. 2011 (cited on page 37).

[RKL+07] Carsten Rother, Vladimir Kolmogorov, Victor Lempitsky, and Martin Szummer. Optimizing binary MRFs via extended roof duality. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2007 (cited on pages 36, 40).

[RM03] Xiaofeng Ren and Jitendra Malik. Learning a classification model for segmentation. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2003 (cited on page 47).

[Sch03] Alexander Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency*. Springer, 2003 (cited on page 7).

[Sch10] Nicol N. Schraudolph. Polynomial-time exact inference in NP-hard binary MRFs via reweighted perfect matching. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010 (cited on page 36).

[Sch76] Michail I. Schlesinger. Syntactic analysis of two-dimensional visual signals in noisy conditions. *Kibernetika*, 4:113–130, 1976 (cited on pages 2, 31).

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1986 (cited on page 11).

[SCL12] David Sontag, Do Kook Choe, and Yitao Li. Efficiently searching for frustrated cycles in MAP inference. In *Proceedings of the Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2012 (cited on page 37).

[SH11] Alexander Shekhovtsov and Václav Hlaváč. On partial optimality by auxiliary submodular problems. In *Control Systems and Computers*, 2011 (cited on page 74).

[SK08] Nicol N. Schraudolph and Dmitry Kamenetsky. Efficient exact inference in planar Ising models. `http://arxiv.org/abs/0810.4401`. 2008 (cited on page 26).

[SK09] Nicol N. Schraudolph and Dmitry Kamenetsky. Efficient exact inference in planar Ising models. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*, 2009 (cited on pages 35, 36, 44).

[SSK+13] Paul Swoboda, Bogdan Savchynskyy, Jörg H. Kappes, and Christoph Schnörr. Partial optimality via iterative pruning for the Potts model. In *Proceedings of the International Conference on Scale Space and Variational Methods in Computer Vision (SSVM)*, 2013 (cited on page 40).

[STL+12] Min Sun, Murali Telaprolu, Honglak Lee, and Silvio Savarese. Efficient and exact MAP-MRF inference using branch and bound. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012 (cited on page 2).

[VBM10] Olga Veksler, Yuri Boykov, and Paria Mehrani. Superpixels and supervoxels in an energy optimization framework. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2010 (cited on page 47).

[Wer10] Tomáš Werner. Revisiting the linear programming relaxation approach to Gibbs energy minimization and weighted constraint satisfaction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1474–1488, 2010 (cited on page 2).

[Wey35] Hermann Weyl. Elementare Theorie der konvexen Polyeder. *Commentarii Mathematici Helvetici*, 7:290–306, 1935 (cited on page 10).

[WJ08] Martin J. Wainwright and Michael I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305, 2008 (cited on pages 2, 24, 29, 31).

[WJW05] Martin J. Wainwright, Tommi Jaakkola, and Alan S. Willsky. MAP estimation via agreement on trees: Message-passing and linear programming. *IEEE Transactions on Information Theory*, 51(11):3697–3717, 2005 (cited on page 36).

[YIF12] Julian Yarkony, Alexander Ihler, and Charless C. Fowlkes. Fast planar correlation clustering for image segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2012 (cited on page 2).

[ZHM+11] Yuhang Zhang, Richard Hartley, John Mashford, and Stewart Burn. Superpixels via pseudo-Boolean optimization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2011 (cited on page 47).

[Zie95] Günter M. Ziegler. *Lectures on Polytopes.* Springer, 1995 (cited on page 9).

# Index

# Nomenclature

| | |
|---|---|
| $\mathbf{0}$ | vector of all zeros, 5 |
| $\mathbf{1}$ | vector of all ones, 5 |
| $\binom{A}{k}$ | set of all subsets of $A$ with size $k$, 5 |
| $\subset$ | proper subset, 5 |
| $\subseteq$ | subset, 5 |
| $\mathcal{E}$ | edges of a factor graph, 19 |
| $\mathcal{F}$ | factors of a factor graph, 19 |
| $\mathcal{F}_r$ | factors of order $r$, 20 |
| $\mathcal{G}$ | factor graph, 19 |
| $\mathcal{M}$ | discrete graphical model, 19 |
| $\mathcal{NP}$ | set of all polynomially verifiable decision problems, 9 |
| $\mathcal{O}$ | Landau notation, 5 |
| $\mathcal{P}$ | set of all polynomially solvable decision problems, 8 |
| $\mathcal{P}(A)$ | power set of $A$, 5 |
| $\mathcal{V}$ | nodes of a factor graph, 19 |
| $\mathbb{C}(\mathcal{M})$ | cut polytope, 29 |
| $\mathbb{I}$ | indicator function, 5 |
| $\mathbb{L}(\mathcal{M})$ | local polytope, 31 |
| $\mathbb{M}(\mathcal{M})$ | marginal polytope, 29 |
| $\mathbb{MC}(G)$ | multicut polytope of $G$, 57 |
| $\mathbb{N}$ | set of natural numbers (including zero), 5 |

| | |
|---|---|
| $\mathbb{R}$ | set of real numbers, 5 |
| $\chi$ | incidence vector, 57 |
| $\delta(S_1, \ldots, S_n)$ | set of edges with endnodes in different sets $S_i$, 6 |
| $\theta_f$ | energy function for factor $f$, 21 |
| $\Pi$ | decision problem, 8 |
| $\Sigma$ | alphabet, 8 |
| $\Sigma^*$ | set of all words over $\Sigma$, 8 |
| $\mathrm{conv}(\cdot)$ | convex hull, 9 |
| $e$ | edge of a graph, 6 |
| enc | encoding scheme, 8 |
| $f_{v_1 v_2 \ldots v_r}$ | factor with neighborhood $\{v_1, v_2, \ldots, v_r\}$, 20 |
| $k$ | cardinality of label set, 20 |
| $\mathrm{nb}(v)$ | neighborhood of a node $v$, 6 |
| $r$ | order of a factor, 20 |
| time | time complexity of an algorithm, 8 |
| $uv$ | edge of a graph, 6 |
| $v$ | node of a graph, 6 |
| $w$ | weight function, 7 |
| $x$ | labeling of a discrete graphical model, 20 |
| $x_v$ | variable holding the label of node $v$, 20 |
| $y$ | variable of the multicut (I)LP, 58 |
| $A$ | algorithm, 7 |
| $E$ | edge set of a graph, 6 |
| $E(G)$ | edge set of a graph $G$, 6 |
| $G$ | graph, 6 |

| | |
|---|---|
| $J$ | energy function, 21 |
| $L$ | common label set, 20 |
| $P$ | polytope, 10 |
| $T$ | terminal nodes, 59 |
| $V$ | node set of a graph, 6 |
| $V(G)$ | node set of a graph $G$, 6 |
| $X$ | label set of a discrete graphical model, 20 |
| $X_v$ | label set of node $v$, 20 |