Applied Combinatorial Optimization

or Learnable Combinatorial Optimization

Bogdan Savchynskyy
Heidelberg University
bogdan.savchynskyy@iwr.uni-heidelberg.de

July 6, 2026

Contents

Preface

The course is devoted to combinatorial optimization, which includes but not limited to algorithms on graphs, integer linear programming, pseudo-boolean optimization, matroids and submodularity. A distinctive feature of this course is its motivation by machine learning applications, which shifts the optimization accent from attaining an optimal solution to a problem, to obtaining an accurate enough solution very fast. The reason for this shift is complexity of models used in modern artificial intelligence-related branches and the lesson they teach us: Better results can be easier attained by more accurate models rather than by more accurate optimization. More accurate models can be obtained by learning, and learning is an iterative process. Should this process include optimization, the latter must be fast enough to be run on each learning iteration. For even more accurate modeling, parameters of the considered combinatorial problems must me learned. Naturally, we cover this aspect in the lecture as well. The material selection for the course emerges from lecturer's experience of combinatorial optimization in computer vision and machine learning. The goals of the course are:

- competent modeling of real-life combinatorial problems and usage of existing program packages to solve them;
- learn typical combinatorial optimization techniques and have a sufficient background for an independent literature search;
- learn how scalable, fast and accurate combinatorial solvers are built; understand the basics of convex analysis, convex optimization, convex duality theory, (integer) linear programs and their geometry.
- learn the cutting-edge results in the area of learning combinatorial solvers, i.e. estimating cost vectors and problem structure based on the training set.

Aknowledgements

The text of this lecture draft has been notably improved by my students Ali Momeni, Georg Mix, Laura Saiz, Elizaveta Mironovich and Amelie-Christine Strupp. Yet, the most extensive and valuable correction work was done by Emily Tempus, which is outstanding, even if compared to others.

Notation

To simplify reading of the monograph, some frequently used notations are collected here. Some of them, which we assume to be standard, are used without additional notice in the text. Others, typically more specialized, are introduced in the monograph. For those we point out the section and the page they are defined in.

Standard notation

$\mathbb{N}$	the set of natural numbers
$\mathbb{Z}$	the set of integer numbers
$\mathbb{R}$	the set of real numbers
$\mathbb{R}^n$	an n-dimensional vector space over the field of real numbers
$\mathbb{R}^n_+$, $\mathbb{R}^n_{\geq 0}$	the set of vectors with non-negative coordinates in $\mathbb{R}^n$, i.e. $\{x \in \mathbb{R}^n \colon x_i \geq 0,\ i = 1, 2, \ldots, n\}$; for $n = 1$ the notation simplifies to $\mathbb{R}_+$
$\mathbb{R}^n_{>0}$	the set of n-dimentional vectors with strictly positive coordinates, *i.e.*, $\{x \in \mathbb{R}^n \colon x_i > 0,\ i = 1, 2, \ldots, n\}$
$\mathcal{A}^*$	the set of vectors of arbitrary length with elements from the set $\mathcal{A}$
$x \in \mathcal{A}^{\mathcal{B}}$	For any set $\mathcal{A}$ and any finite set $\mathcal{B}$, this notation stands for a vector x with $\lvert\mathcal{B}\rvert$ coordinates indexed by elements of $\mathcal{B}$, for each $b \in \mathcal{B}$ it holds that $x_b \in \mathcal{A}$. The only exception from this rule is the notation $\Delta^{\mathcal{B}}$, see below.
$x \geq y$	comparison operations are applied coordinate-wise to vectors and point-wise to functions
$[\![\cdot]\!]$	denotes the Iverson brackets, that is, for any predicate A it holds that $[\![A]\!] = 1$ if A is true, otherwise $[\![A]\!] = 0$
$\langle c, x\rangle$	the inner product, i.e. $\langle c, x\rangle = \sum_{i=1}^n c_i x_i$
∇f	gradient of the function f
$O(\cdot)$	for two functions $f \colon \mathbb{N} \to \mathbb{N}$ and $g \colon \mathbb{N} \to \mathbb{N}$ one writes $f = O(g)$, if there is a constant $c > 0$ and a number $n_0 \in \mathbb{N}$ such that $f(n) \leq c \cdot g(n)$ for all $n \geq n_0$
$\binom{A}{k}$	the set containing all possible k-tuples of different elements of the set A. The number of such k-tuples is equal to the binomial coefficient $\binom{\lvert A\rvert}{k}$.
$[n]$	the set $\{1, \ldots, n\}$.

Standard abbreviations

w.r.t.	with respect to
w.l.o.g.	without loss of generality
s.t.	subject to

Notation defined in the lecture draft

$\mathcal{N}(u)$	set of graph vertexes incident to vertex u, see Section 1.5.1, page 37
$\delta(\mathbf{y})$	binary representation of the labeling $\mathbf{y}$, i.e. a binary vector with non-zero coordinates corresponding to the labels y_u, $u \in \mathcal{Y}_u$, and label pairs (y_u, y_v), $uv \in \mathcal{E}$, see Section 3.3 on page 85;
$\mathcal{I}$	the set of indexes of the cost vector of a graphical model; $\lvert\mathcal{I}\rvert$ is equal to the number of coordinates of the cost vector, see §1.5.1 on page 40
$\operatorname{vrtx}(P)$	set of vertexes of the polyhedron P, see Definition 2.20 on page 53
Δ^n	n-dimensional simplex, see Definition 2.22 on page 54
$\Delta^{\mathcal{X}}$	$\lvert\mathcal{X}\rvert$-dimensional simplex, with coordinates indexed by elements of $\mathcal{X}$, see Definition 2.22 on page 54
$\operatorname{conv}(X)$	convex hull of X, see Definition 2.30 on page 57
$\operatorname{mi}[\theta_w]$	binary vector with non-zero coordinates corresponding to locally minimal values of the cost vector θ_w, see page 133
$\operatorname{nz}[\boldsymbol{\mu}]$	binary vector with non-zero coordinates corresponding to the non-zero coordinates of $\boldsymbol{\mu}$, see page 133
$\mathcal{J}$	the set of indexes of the Lagrange dual vector for the MAP-inference problem; $\lvert\mathcal{J}\rvert$ is equal to the number of coordinates in the dual vector, see §5.5.1 on page 125
$\left\langle \frac{1}{2} \right\rangle$, $\langle 0.7 \rangle$	angular brackets are used in figures for coordinates of primal relaxed solutions of the labeling problem, see e.g. Figure 9.7.

1 Introduction

1.1 What is a combinatorial problem?

Definition 1.1. Let $S \subseteq \mathbb{Z}^n$ and $f\colon S \to \mathbb{R}$. A problem that can be represented in the form

$$\min_{\mathbf{x}\in S} f(\mathbf{x})\,. \tag{1.1}$$

we will call *combinatorial*.

Example 1.2 (Shortest path in a graph)**.** Consider a graph $(\mathcal{V}, \mathcal{E})$ with $\mathcal{V}$ and $\mathcal{E}$ being the sets of nodes and edges respectively. Let also $\mathbf{c} = (c_e\colon e \in \mathcal{E}) \in \mathbb{R}^{\mathcal{V}}_{>0}$ be the *cost vector* defining the length of its edges. Let us show that the problem of finding a path with the shortest total length of edges between two graph nodes $u, v \in \mathcal{V}$ is combinatorial.

To this end associate a binary integer (*indicator*) variable $x_e \in \{0, 1\}$ with each edge $e \in \mathcal{E}$ and set it to 1 if the respective edge belongs to the path, otherwise set it to zero. Each path can now be encoded by a binary integer vector $\mathbf{x} = (x_e\colon e \in \mathcal{E})$ of the length $|\mathcal{E}|$. The subset of all binary vectors that encode paths between u and v corresponds to the set $S \subseteq \{0, 1\}^{\mathcal{V}} \subseteq \mathbb{Z}^{\mathcal{Y}}$ and the length of the path is equal to $f(\mathbf{x}) := \langle \mathbf{c}, \mathbf{x}\rangle$. This turns the problem into format (1.1), which finalizes the proof.

Example 1.3 (Hamiltonian cycle problem)**.** The *Hamiltonian cycle problem* consist in answering the question whether there is a simple (without self-intersections) cycle containing all nodes of a given undirected graph $(\mathcal{V}, \mathcal{E})$.

As in Example 1.2, let $\mathbf{x} = (x_e\colon e \in \mathcal{E})$ be the indicator vector associated with the edges of the graph. The set S be the subset consisting of all such vectors corresponding to the simple cycles containing

all nodes. Assign $f(\mathbf{x}) := [\![\mathbf{x} = \mathbf{0}]\!]$. Since for any $\mathbf{x} \in S$ it holds $\mathbf{x} \neq \mathbf{0}$, minimization of f essentially consists of finding out whether S is empty or not. Overall, this implies that the respective problem is combinatorial.

Example 1.4 (Traveling salesman problem)**.** Let $(\mathcal{V}, \mathcal{E})$ be a fully connected graph and $\mathbf{c} = (c_e \colon e \in \mathcal{E}) \in \mathbb{R}^{\mathcal{V}}_{>0}$ be the *cost vector* defining the length of its edges. The traveling salesman problem is the problem of finding a simple cycle with the lowest total length in this graph.

As in Example 1.3, let $\mathbf{x} = (x_e \colon e \in \mathcal{E})$ be the indicator vector associated with the edges of the graph. The set S be the subset consisting of all such vectors corresponding to the simple cycles containing all nodes. By assigning $f(\mathbf{x}) := \langle \mathbf{c}, \mathbf{x} \rangle$ we turn the problem into format (1.1) thus proving that it is combinatorial.

Example 1.5 (Pseudo-boolean optimization)**.** Let $\bar{x} := (1 - x)$ for any $x \in \{0, 1\}$. The problem of the form

$$\min_{x \in \{0,1\}^n} \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n} b_i \bar{x}_i + \sum_{i=1}^{n} \sum_{j=i}^{n} c_{ij} x_i x_j + \sum_{i=1}^{n} \sum_{j=i}^{n} d_{ij} x_i \bar{x}_j \,, \quad (1.2)$$

where $a_i, b_i, c_{ij}, d_{ij} \in \mathbb{R}$, satisfies conditions of Definition 1.1 directly and is, therefore, *combinatorial*.

Example 1.6 (Integer quadratic problem)**.** The *quadratic integer optimization* problem of the form

$$\min_{x \in \{0,1\}^n} \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n} \sum_{j=i}^{n} c_{ij} x_i x_j \,, \quad (1.3)$$

where $a_i, b_i, c_{ij} \in \mathbb{R}$, satisfies conditions of Definition 1.1 directly and is, therefore, *combinatorial*.

At the same time, a related *quadratic ~~integer~~ optimization* problem

$$\min_{x \in [0,1]^n} \sum_{i=1}^{n} a_i x_i + \sum_{i=1}^{n} \sum_{j=i}^{n} c_{ij} x_i x_j \,, \quad (1.4)$$

as the set $[0,1]^n$ is continuous and can't be represented as (at most) countable $S \subseteq \mathbb{Z}^n$.

Similarly, it can be shown that the following problems are combinatorial as well:

Exercise 1.7 (Knapsack problem)**.** Let $[n]$ be the list of items one can put into a knapsack. Each item $i \in [n]$ has a volume w_i and a cost $c_i > 0$. The knapsack has a *maximum total volume* or *capacity* W. The goal is to maximize the cost of items put into the knapsack, whereas their total volume does not exceed W. A modification of the above problem, referred to as *binary knapsack* assumes that each item from the list can be selected at most once. In contrast, in the non-binary knapsack the number of items of the same type is not limited. Show that both binary and non-binary knapsack problems are combinatorial.

Exercise 1.8 (Prime factorization)**.** Given a natural number, represent it as a product of prime numbers. Show that this problem is combinatorial.

Exercise 1.9 (Stable or independent set problem)**.** A subset of graph nodes is called *stable* or *independent* if none two of them are adjacent. Given a graph $\mathcal{G}$ and a natural number k. Answer the question whether there is an independent set of the size k.

Exercise 1.10 (Maximum independent set cardinality problem)**.** Given a graph $\mathcal{G}$, find the cardinality of its largest independent set.

1.2 Big-O notation

Consider two functions $f, g\colon \mathbb{N} \to \mathbb{N}$ that map *problem size* to the *number of operations*. We write $f \in O(g)$ if

$$\exists c \in \mathbb{R}_+,\ n_0 \in \mathbb{N} :\quad f(n) \le cg(n) \quad \forall n \ge n_0\,. \tag{1.5}$$

Example 1.11. $\underbrace{3n^2 + 7n}_{f} = O(\underbrace{n^2}_{g})$. Indeed, $f(n) \le 4g(n)$ for $n \ge 7$.

1.3 NP-completeness and NP-hardness

We now make precise what it means for a problem to be "computationally hard". The whole theory is built on the simplest possible kind of problem: those whose answer is a single bit, *Yes* or *No*. We first develop the complexity classes for such *decision problems*, and only afterwards extend the notions to general *computational problems* and, in particular, to optimization problems such as the traveling salesman problem from Example 1.4. Throughout, $\mathcal{A}^*$ denotes the set of all finite strings over a set $\mathcal{A}$, and the *size* of an input is the length of the binary string encoding it. We always assume a fixed, efficient binary encoding of the input: for instance, a graph is given by its adjacency list, and an integer by its binary representation.

1.3.1 Decision problems

Definition 1.12. A *decision problem* is a mapping

$$f\colon \mathrm{dom} f \to \{0,1\}\,, \qquad \mathrm{dom} f \subseteq \{0,1\}^*\,,$$

such that the membership question "is $\mathbf{x} \in \mathrm{dom} f$?" can be decided in polynomial time. The elements of $\mathrm{dom} f$ are called *instances*; an instance $\mathbf{x}$ with $f(\mathbf{x}) = 1$ is a *yes-instance*, and one with $f(\mathbf{x}) = 0$ is a *no-instance*. The instance $\mathbf{x}$ is also called a *question*, and the output bit the *answer*.

The requirement that $\mathrm{dom} f$ be polynomially decidable deserves a comment. An algorithm is handed an *arbitrary* binary string, whereas f is defined only on $\mathrm{dom} f$. Before it can even attempt to answer the question, the algorithm must first recognise whether the string *is* a legitimate question at all – *e.g.* whether it encodes an undirected graph. Demanding that "is $\mathbf{x} \in \mathrm{dom} f$?" be answerable in polynomial time guarantees that none of the difficulty of the problem is smuggled into the recognition of its instances; all of it lives in the yes/no question itself.

Example 1.13 (Decision problems)**.** The following are decision problems.

- *Hamiltonian cycle*: given an undirected graph, does it contain a simple cycle through all of its nodes? See Example 1.3.
- *Decision-TSP*: given a complete graph $(\mathcal{V}, \mathcal{E})$ with integer edge lengths $\mathbf{c} = (c_e \colon e \in \mathcal{E})$ and a number $k \in \mathbb{N}$, does there exist a round trip (a simple cycle through all nodes) of total length at most k? This is the decision variant of Example 1.4.
- *Path*: given a graph and two of its nodes u, v, does a path between u and v exist? See Example 1.2.
- *Prime*: given a natural number n in binary, is n prime?
- *Subset-sum*: given items with integer volumes and a capacity W, is there a subset of items whose total volume equals exactly W? See Exercise 1.7.
- The *stable set* problem from Exercise 1.9: given a graph and a number $k \in \mathbb{N}$, does it contain a stable (independent) set of size k, *i.e.* a set of k nodes no two of which are connected by an edge?

For every one of these the instance set is easy to recognise: checking that a string encodes a graph, a graph together with a number, or a natural number is clearly possible in polynomial time.

Let n denote the size of the instance, as in Definition 1.12.

Definition 1.14. A decision problem f belongs to the class P, or is *polynomially solvable*, if there exists $k \in \mathbb{N}$ such that $f(\mathbf{x})$ can be computed for every instance $\mathbf{x}$ in time $O(n^k)$.

For example, deciding whether a path between two nodes exists can be done in polynomial time, so this problem is in P; likewise *Prime* is in P. In contrast, no polynomial-time algorithm is known for *Hamiltonian cycle* or *Decision-TSP*.

1.3.2 The class NP

For *Hamiltonian cycle* we do not know a polynomial-time *solution* algorithm, but if somebody claims that a particular graph *is* Hamiltonian, they can convince us cheaply: they exhibit the cycle, and we verify in polynomial time that it is indeed a simple cycle through all nodes. Such a convincing object is called a certificate. This is the idea behind the class NP.

Definition 1.15. A decision problem f belongs to the class *NP* if there is a polynomial p with the following property: for every yes-instance $\mathbf{x}$ (*i.e.* $f(\mathbf{x}) = 1$) there exists a *certificate* (or *proof*) $\mathbf{c} \in \{0,1\}^*$ of size at most $p(n)$ that can be verified in polynomial time; for no-instances no such certificate exists.

For instance, a Hamiltonian cycle is a certificate for *Hamiltonian cycle*, a round trip of length $\leq k$ is a certificate for *Decision-TSP*, an explicit u–v path is a certificate for *Path*, and a stable set of size k is a certificate for the stable set problem. Clearly $P \subseteq \mathrm{NP}$, since a polynomial-time solution algorithm produces the answer itself, and the (empty) certificate is then trivially verifiable. It is the most famous open problem of the field whether, conversely, $\mathrm{NP} \subseteq P$, *i.e.* whether $P = \mathrm{NP}$; it is widely believed that $P \neq \mathrm{NP}$.

Why "non-deterministic polynomial". The letters NP stand for *non-deterministic polynomial*, which refers to an equivalent way of describing the class. Imagine an algorithm that, in addition to its input, is allowed to read a string of *guessed* bits; its computation may depend on these guesses. Such an algorithm is called *non-deterministic* if

- on every no-instance it answers *No* for *every* possible string of guessed bits, and
- on every yes-instance there exists *at least one* string of guessed bits that makes it answer *Yes*.

Note the asymmetry, and the absence of any probability bound: a single lucky guess is enough. A decision problem is in NP if and only

if it admits such a non-deterministic algorithm running in polynomial time. The bridge between the two descriptions is that the *certificate* of Definition 1.15 and the *lucky string of guessed bits* are the same object: "guess the bits that lead to acceptance and check them" is exactly "be handed a certificate and verify it". On a yes-instance some guess equals a valid certificate, so the algorithm can accept; on a no-instance no certificate exists, so it always rejects. This is why "non-deterministic polynomial" is a faithful name for NP.

1.3.3 The class co-NP

Definition 1.15 is deliberately asymmetric: it demands an efficiently verifiable proof only for yes-instances. Swapping the roles of yes and no yields a genuinely different class, which serves as our first example of problems that are (presumably) *not* in NP.

Definition 1.16. The *complement* of a decision problem $f\colon \mathrm{dom} f \to \{0,1\}$ is the decision problem $\bar{f}$ on the same instance set with $\bar{f}(\mathbf{x}) = 1 - f(\mathbf{x})$, *i.e.* yes- and no-instances are interchanged. A decision problem belongs to the class *co-NP* if its complement belongs to NP.

Equivalently, a problem is in co-NP if every *no*-instance has a polynomially verifiable certificate. Consider the complement of *Hamiltonian cycle*:

> *Non-Hamiltonian*: given an undirected graph, is it true that it has *no* Hamiltonian cycle?

By definition *Non-Hamiltonian* is in co-NP, because a Hamiltonian cycle certifies its no-instances (*i.e.* the Hamiltonian graphs). However, it is not at all clear how one would certify a *yes*-instance: what short proof convinces us that a graph has *no* Hamiltonian cycle whatsoever? No such certificate is known, and it is a wide-open question whether *Non-Hamiltonian* belongs to NP. It is commonly conjectured that NP $\neq$ co-NP.

Clearly the complement of a problem in P is again in P, so $P \subseteq \mathrm{NP} \cap \text{co-NP}$. A problem lying in $\mathrm{NP} \cap \text{co-NP}$ has a *good characterization*: both its yes- and its no-instances admit efficiently checkable certificates. For example, *Prime* is in $\mathrm{NP} \cap \text{co-NP}$ (and in fact in P).

1.3.4 Polynomial transformations, NP-hardness and NP-completeness

To compare the difficulty of two decision problems we map one to the other.

Definition 1.17. Let f and g be decision problems. A mapping $T\colon \mathrm{dom}f \to \mathrm{dom}g$ is a *transformation* of f to g if for all $\mathbf{x} \in \mathrm{dom}f$

$$g(T(\mathbf{x})) = 1 \quad \text{if and only if} \quad f(\mathbf{x}) = 1\,,$$

i.e. yes-instances are mapped to yes-instances and no-instances to no-instances. If, in addition, T is computable in polynomial time, then f is said to be *polynomially transformable* to g.

A polynomial transformation of f to g shows that f is "no harder than" g: a polynomial-time algorithm for g yields one for f by first applying T. Polynomial transformations are transitive.

Definition 1.18. A decision problem is called *NP-hard* if every problem in NP is polynomially transformable to it. If, in addition, the problem itself belongs to NP, it is called *NP-complete*.

By Definition 1.18 all NP-complete problems are polynomially transformable to one another. Hence any single NP-complete problem is representative of the whole class: if *one* NP-complete problem were solvable in polynomial time, then so would be every problem in NP, *i.e.* $P = \mathrm{NP}$. Cook's theorem [1] provides the first NP-complete problem (*satisfiability*); from it, by exhibiting suitable transformations, a great many problems have been shown NP-complete [2], among them *Hamiltonian cycle*, *Decision-TSP*, *Subset-sum* and the stable set problem.

1.3.5 Computational problems and reductions

Decision problems are the special case in which the answer is a single bit. In general the output may be an arbitrary binary string.

Definition 1.19. A mapping of the form $f\colon \mathrm{dom} f \to \{0,1\}^*$ with $\mathrm{dom} f \subseteq \{0,1\}^*$, for which "is $\mathbf{x} \in \mathrm{dom} f$?" is decidable in polynomial time, is called a *computational problem*. The input is a *problem description* (a binary string whose length is the problem size), and the output is a binary *solution* string. Decision problems (Definition 1.12) are exactly the computational problems whose output is a single bit.

Many computational problems are naturally attached to the same combinatorial object. The traveling salesman and Hamiltonian cycle problems illustrate several distinct flavours.

Example 1.20 (Optimization, evaluation, counting)**.** For the traveling salesman problem on a complete graph $(\mathcal{V}, \mathcal{E})$ with integer edge lengths $\mathbf{c}$ (Example 1.4) one may ask:

- *Optimization*: find an *optimal* round trip, *i.e.* a simple cycle through all nodes of minimum total length. The output is the tour itself (an edge set).
- *Evaluation*: find the *value* of an optimal round trip, *i.e.* the minimum total length. The output is a number.
- *Decision* (or *recognition*): this is *Decision-TSP* – given k, is the optimal value at most k? The output is a single bit.

A fourth flavour is *counting*, *e.g.* given a graph, *how many* distinct Hamiltonian cycles does it contain? The output is again a number, but the problem is of a different nature: even when the corresponding decision problem is easy, counting the solutions may be much harder.

Transformation (Definition 1.17) applies only to decision problems and maps an instance to a *single* instance of the target problem. The following notion generalises it to arbitrary computational problems and allows the target problem to be invoked *repeatedly*.

Definition 1.21. A computational problem f *(polynomially) reduces* to a computational problem g if there exists a (polynomial-time) algorithm for f that may, at any of its steps, invoke an algorithm for g, where *each such invocation is counted as a single elementary operation*, regardless of the actual cost of computing g. Such an algorithm is called a *(polynomial) reduction*, and the invoked algorithm for g is called an *oracle* for g.

The convention that an oracle call counts as one operation is what makes the reduction's running time independent of how hard g itself is. The point is the following: if f polynomially reduces to g *and* g admits a polynomial-time algorithm, then substituting that algorithm for the oracle yields a polynomial-time algorithm for f. Indeed, a polynomial reduction makes at most polynomially many oracle calls, each on an instance of at most polynomial size, so the total work is bounded by a composition of polynomials, which is again a polynomial. Every polynomial transformation is, in particular, a polynomial reduction (one that makes a single oracle call and returns its answer); reductions are transitive.

Example 1.22. The maximum stable set cardinality problem (Exercise 1.10) – given a graph, find the size of its largest stable set, an evaluation problem – reduces to the stable set decision problem of Exercise 1.9. The reduction queries the stable set oracle for $k = |\mathcal{V}|, |\mathcal{V}|-1, \ldots,$ decreasing k by one until the answer first becomes *Yes*; that value of k is the maximum cardinality. Each oracle call counts as one operation, and there are at most $|\mathcal{V}|$ of them, so the reduction is polynomial.

1.3.6 NP-hard and NP-easy problems

Definition 1.23. A computational problem f is called *NP-hard* if every problem in NP polynomially reduces to it. It is called *NP-easy* if it polynomially reduces to some problem in NP. A problem that is both NP-hard and NP-easy is called *NP-equivalent*.

Definition 1.23 generalises NP-hardness from decision problems (Definition 1.18) to all computational problems: for a decision problem, polynomial reducibility of all of NP to it is the natural relaxation of polynomial transformability. (The two notions of NP-hardness need not coincide even for decision problems, since a reduction is more powerful than a transformation; but every NP-complete problem is in particular NP-hard in the sense of Definition 1.23.) NP-hard problems are at least as hard as everything in NP; an NP-easy problem is no harder. Unless $P = \mathrm{NP}$, no NP-hard problem has a polynomial-time algorithm.

In practice one proves a computational problem NP-hard by exhibiting a polynomial reduction *from* a known NP-complete decision problem *to* it.

Definition 1.24. A computational problem g is *NP-hard* if there exists an NP-complete decision problem that polynomially reduces to g.

Definition 1.24 is equivalent to Definition 1.23: since every problem in NP transforms (hence reduces) to the NP-complete one, which in turn reduces to g, transitivity gives a reduction from all of NP to g.

Example 1.25 (The TSP is NP-hard)**.** Let $(\mathcal{V}, \mathcal{E})$ be a graph in which we seek a Hamiltonian cycle. Construct the complete graph on the same vertex set $\mathcal{V}$, assigning length 0 to the edges in $\mathcal{E}$ and length 1 to all other edges. The original graph has a Hamiltonian cycle if and only if the complete graph has a round trip of total length 0. Thus *Hamiltonian cycle* reduces to the traveling salesman problem (in either its decision, evaluation or optimization form), and since *Hamiltonian cycle* is NP-complete, the traveling salesman problem is NP-hard.

1.3.7 NP-optimization problems

We now formalise the kind of optimization problem we are interested in and relate its three forms. Following the convention of this monograph, we assume the objective takes *non-negative integer* values.

Definition 1.26. An *NP-optimization problem* consists of

- a polynomial-time decidable instance set $\mathcal{X} \subseteq \{0,1\}^*$;
- for each instance $\mathbf{x} \in \mathcal{X}$ a non-empty set $S_{\mathbf{x}} \subseteq \{0,1\}^*$ of *feasible solutions*, where solutions have size polynomially bounded in the size of $\mathbf{x}$, and membership "is $\mathbf{y} \in S_{\mathbf{x}}$?" is decidable in polynomial time;
- a *cost* (or value) function $c(\mathbf{x}, \mathbf{y}) \in \mathbb{N}$ computable in polynomial time;
- a goal, min or max.

We write $\mathrm{OPT}(\mathbf{x}) := \mathrm{goal}\{c(\mathbf{x}, \mathbf{y}) \colon \mathbf{y} \in S_{\mathbf{x}}\}$, and call $\mathbf{y} \in S_{\mathbf{x}}$ with $c(\mathbf{x}, \mathbf{y}) = \mathrm{OPT}(\mathbf{x})$ an *optimal solution*.

The traveling salesman problem fits this scheme: $\mathbf{x}$ encodes the weighted graph, $S_{\mathbf{x}}$ is the set of round trips, $c(\mathbf{x}, \mathbf{y})$ is the length of the trip $\mathbf{y}$, and the goal is min. To each such problem we associate three related computational problems.

- The *optimization problem*: given $\mathbf{x}$, output an optimal solution $\mathbf{y}$ (find an optimal TSP tour).
- The *evaluation problem*: given $\mathbf{x}$, output the value $\mathrm{OPT}(\mathbf{x})$ (find the length of an optimal TSP tour).
- The *decision/recognition problem*: given $\mathbf{x}$ and $k \in \mathbb{N}$, decide whether $\mathrm{OPT}(\mathbf{x}) \leq k$ (for min; for max, whether $\mathrm{OPT}(\mathbf{x}) \geq k$). This is *Decision-TSP*.

Relations between the three forms

Optimization $\Rightarrow$ evaluation and decision (trivial). Given an optimal solution $\mathbf{y}$, its value $c(\mathbf{x}, \mathbf{y}) = \mathrm{OPT}(\mathbf{x})$ is computed in polynomial time (the evaluation answer), and comparing it with k answers the decision problem. Both reductions make a single call to the optimization oracle.

Decision ⇒ evaluation (binary search). Conversely, the value $\mathrm{OPT}(\mathbf{x})$ can be pinned down by binary search on the threshold k, using the decision oracle. Because c is computable in polynomial time and outputs a non-negative integer, there is a polynomial $l(\cdot)$ with $0 \leq \mathrm{OPT}(\mathbf{x}) \leq 2^{l(\mathbf{x})}$; thus $l(\mathbf{x})$ bounds the number of bits of the optimal value $\mathrm{OPT}(\mathbf{x})$. Starting from the interval $[\alpha, \beta] = [0, 2^{l(\mathbf{x})}]$, each query "is $\mathrm{OPT}(\mathbf{x}) \leq \lfloor \frac{\alpha+\beta}{2} \rfloor$?" halves the interval; after $l(\mathbf{x}) + 1$ queries the interval contains a single integer, which is $\mathrm{OPT}(\mathbf{x})$. This is a polynomial reduction of the evaluation problem to the decision problem.

Evaluation ⇒ optimization for TSP (edge fixing). Recovering an optimal solution from an oracle for the value is the most delicate of the three reductions, so before treating the general case we warm up on the traveling salesman problem, where the construction is particularly concrete. Here the passage from the optimal value to an actual optimal solution is especially transparent, and the plain evaluation oracle already suffices. The reason is structural – a TSP tour is nothing but a subset of the given edges, so recovering it amounts to deciding, edge by edge, whether the edge belongs to an optimal tour.

Compute first $\mathrm{OPT}(\mathbf{x})$, the length of an optimal tour, and maintain a *current graph* $\mathcal{G}'$, initially the input graph itself; we write $\mathrm{OPT}(\mathcal{G}')$ for the length of an optimal tour in $\mathcal{G}'$. Now process the edges $e_1, \ldots, e_m$ one at a time. To test edge e_i, *delete* it from $\mathcal{G}'$ and ask the evaluation oracle for the value $\mathrm{OPT}(\mathcal{G}' - e_i)$ of the smaller graph $\mathcal{G}' - e_i$.

- If $\mathrm{OPT}(\mathcal{G}' - e_i) = \mathrm{OPT}(\mathbf{x})$, then some optimal tour of $\mathcal{G}'$ avoids e_i: discard e_i permanently, replacing $\mathcal{G}'$ by $\mathcal{G}' - e_i$.
- If $\mathrm{OPT}(\mathcal{G}' - e_i) > \mathrm{OPT}(\mathbf{x})$ (including the case where $\mathcal{G}' - e_i$ has no tour at all), then every optimal tour of $\mathcal{G}'$ uses e_i: keep e_i in $\mathcal{G}'$ and move on.

Deleting an edge can never lower the optimum, so the current graph always satisfies $\mathrm{OPT}(\mathcal{G}') = \mathrm{OPT}(\mathbf{x})$ and, in particular, still contains an optimal tour. This makes "the reduced graph" unambiguous: it is

simply the input graph with the discarded edges removed. After all m edges have been processed, the surviving edges of $\mathcal{G}'$ are exactly the edges of an optimal tour, which we output. Indeed, $\mathcal{G}'$ still contains some optimal tour T; and had a surviving edge e lain outside T, then at the moment e was tested the tour T was still present, so $\mathrm{OPT}(\mathcal{G}'-e) = \mathrm{OPT}(\mathbf{x})$ and e would have been discarded – a contradiction. Hence every surviving edge lies on T, and $\mathcal{G}' = T$. The procedure makes $m+1$ calls to the evaluation oracle and is therefore polynomial. A plain decision oracle is equally enough: run the binary search first to obtain $\mathrm{OPT}(\mathbf{x})$; then, since removing edges can only raise the optimum, replace each equality test "$\mathrm{OPT}(\mathcal{G}' - e_i) = \mathrm{OPT}(\mathbf{x})$?" by the single threshold query "$\mathrm{OPT}(\mathcal{G}' - e_i) \leq \mathrm{OPT}(\mathbf{x})$?".

Enriched decision ⇒ optimization (general case). What made the edge-fixing argument work is that a TSP tour is a subset of a *known*, polynomially large ground set – the edges of the given graph – so we could enumerate its coordinates in advance and settle each with a single oracle call. A general NP-optimization problem offers no such fixed ground set: its solutions are arbitrary strings whose very length may vary from instance to instance, and there is nothing to fix "one edge at a time". To handle this general case we fall back on a construction that grows the output string itself, bit by bit; binary search recovers the optimal *value*, but a plain threshold oracle never hands us an actual optimal *solution*, so to recover a witness we enrich the decision problem with an auxiliary argument.

For two binary strings we use the *lexicographic* order: $\mathbf{y} \succeq_{\mathrm{lex}} \mathbf{s}$ means $\mathbf{y}$ is equal to or lexicographically not smaller than $\mathbf{s}$, where, to compare strings of different length, the shorter one is padded with the symbol -1 (a value below both 0 and 1). With this padding, appending a bit to a prefix moves *up* in the order, and the empty string ε is the global minimum: $\mathbf{y} \succeq_{\mathrm{lex}} \varepsilon$ for every $\mathbf{y}$.

The enriched decision problem Q takes *three* inputs:

> Given $\mathbf{x} \in \mathcal{X}$, a threshold $k \in \mathbb{N}$, and a string $\mathbf{s} \in \{0,1\}^*$: is there a feasible $\mathbf{y} \in S_{\mathbf{x}}$ with $c(\mathbf{x}, \mathbf{y}) \leq k$ (for min; $\geq k$ for max) *and* $\mathbf{y} \succeq_{\text{lex}} \mathbf{s}$?

The feasible solution $\mathbf{y}$ serves as a polynomial certificate, so Q is in NP. The string $\mathbf{s}$ acts as a *lexicographic lower bound on the solution itself*; it is the knob that turns the value-oracle into a solution-search. Note that with $\mathbf{s} = \varepsilon$ the constraint $\mathbf{y} \succeq_{\text{lex}} \varepsilon$ is vacuous, so $Q(\mathbf{x}, k, \varepsilon)$ reduces to the plain threshold question used in the binary search above.

The reduction of the optimization problem to Q has two phases.

1. *Find the optimum value.* Run the binary search above, always with $\mathbf{s} = \varepsilon$, to obtain $k^\star = \text{OPT}(\mathbf{x})$.

2. *Grow a solution bit by bit.* Fix $k = k^\star$ and build the solution string $\mathbf{s}$ one bit at a time. Initialise $\mathbf{s}_0 := \varepsilon$. For $i = 1, 2, \ldots, p$ (where p bounds the solution length) query Q on the two candidate prefixes $\mathbf{s}_{i-1}0$ and $\mathbf{s}_{i-1}1$ (the current prefix with a 0, respectively a 1, appended):

   - if both answers are *Yes*, set $\mathbf{s}_i := \mathbf{s}_{i-1}1$;
   - if only the first is *Yes*, set $\mathbf{s}_i := \mathbf{s}_{i-1}0$;
   - if both are *No*, stop and return $\mathbf{y} := \mathbf{s}_{i-1}$.

   We may stop at this point because both *No* answers mean that no optimal solution properly extends $\mathbf{s}_{i-1}$, so $\mathbf{s}_{i-1}$ is itself the sought optimal solution $\mathbf{y} \in S_{\mathbf{x}}$ with $c(\mathbf{x}, \mathbf{y}) = \text{OPT}(\mathbf{x})$.

The order "$\succeq_{\text{lex}}$" (equal-or-greater) rather than the strict "$\succ_{\text{lex}}$" is essential in both phases: in phase 1 it makes the query with $\mathbf{s} = \varepsilon$ trivial, so the value search works; in phase 2 it lets the constructed string be accepted once it *equals* an optimal solution, so the procedure can land on, and certify, a real witness. Altogether the reduction makes $O(l(\mathbf{x}) + p)$ oracle calls and is therefore polynomial.

Remark 1.27 (From integer to rational objective values)**.** A more general definition admits *rational* objective values, $c(\mathbf{x}, \mathbf{y}) \in \mathbb{Q}$, rather

than non-negative integers; arbitrary real values cannot be admitted, as there are uncountably many reals but only countably many binary strings. The restriction to non-negative integers, which we adopt for convenience, is no real loss of generality, and everything carries over to rational costs with only cosmetic changes. Indeed, because c is computable in polynomial time, every value $c(\mathbf{x}, \mathbf{y})$ is a rational of bounded bit-length, say with denominator dividing some $D = 2^{d(\mathbf{x})}$ for a polynomial d. Multiplying all costs by the common denominator D is an order-preserving, polynomial-time rescaling that turns them into integers without changing the set of optimal solutions; a further shift by a constant removes negative values. The only effect on the arguments above is on the granularity of the binary search: instead of stepping through consecutive integers, the search resolves $\mathrm{OPT}(\mathbf{x})$ down to the spacing $2^{-d(\mathbf{x})}$ between attainable values, which still costs only polynomially many oracle calls.

Every NP-optimization problem is NP-easy

The three reductions above combine into the following theorem.

Theorem 1.28. Every NP-optimization problem is NP-easy.

Proof sketch. Let P be an NP-optimization problem with optimization form "output an optimal solution". Consider the enriched decision problem Q defined above. We have already observed that Q is in NP, since a feasible solution $\mathbf{y}$ is a polynomially verifiable certificate. The two-phase procedure – binary search on the threshold to find $\mathrm{OPT}(\mathbf{x})$, followed by the bit-by-bit growth of the solution string – is a polynomial-time algorithm for P that uses Q only as an oracle, with $O(l(\mathbf{x}) + p(\mathbf{x}))$ calls in total. This is precisely a polynomial reduction of P to the NP problem Q, so P is NP-easy. □

In particular, the traveling salesman problem – in its optimization, evaluation and decision forms – is NP-easy, and by Example 1.25 also NP-hard; hence it is NP-equivalent. The same holds for essentially all optimization problems studied in this monograph: they are NP-easy because they reduce to an NP decision problem in the manner

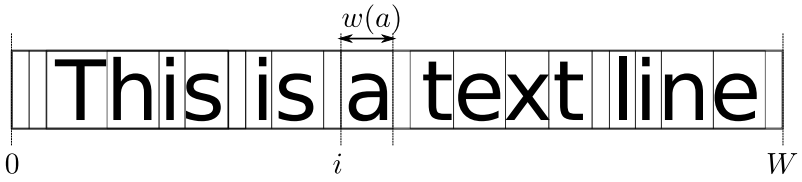


Figure 1.1: Line partitioning problem

above, and one proves them NP-hard by a reduction from a known NP-complete problem.

1.4 Dynamic programming

Dynamic programming is one of the classical techniques of combinatorial optimization. In its idea, it is an exact optimization technique. However, it has a broad usage also to obtain solution approximations and as a subroutine in more complex algorithms.

The goal of this section is not only to give an overview of dynamic programming, but also use it to illustrate the notions introduced above, such as O-notation and relation between polynomial algorithms and the complexity class NP.

1.4.1 Case study: Line partitioning problem

Consider an image of a text line as in Figure 1.1. Assume that the image of a given width W is a concatenation of an arbitrary sequence of (noisy) images of characters taken from a finite *alphabet* $\mathcal{A}$. Each character $a \in \mathcal{A}$ is associated with the *width* of its image in pixels and a function $f_i(a)$, $i = 0, \dots W - w(a)$, defining a cost of assigning the character a to the image interval with x-coordinates from i till $i + w(a)$. One can therefore interpret $\exp(-f_i(a))$, as a conditional probability of the character a being depicted in the above interval. We also assume that there is a *'space'* character in $\mathcal{A}$ with the width equal to *one*.

The goal is to partition the image into intervals corresponding to character from the alphabet $\mathcal{A}$ along the x-axis. Partitioning means, that there is no empty spaces between characters. In our case such empty spaces are modeled by the *space* character, potentially several in a row. Denote $\mathcal{A}^*$ the set of character sequences from $\mathcal{A}$ of arbitrary length. Let $\mathbf{a}_1^k$ denote the subsequence with indices $1 \dots, k$ of $\mathbf{a} \in \mathcal{A}^*$. Denote the total width of the subsequence as $w(\mathbf{a}_1^k) := \sum_{i=1}^k w(a_k)$ and let $w(\mathbf{a}_1^0) := 0$. Let also $|\mathbf{a}|$ stand for the number of characters in $\mathbf{a}$. The partitioning problem can be formalized as: Find a text line $\mathcal{A}^* \ni \mathbf{a}^*$ such that

$$\mathbf{a}^* \in \arg\min_{\mathbf{a}\in\mathcal{A}^*} \sum_{k=1}^{|\mathbf{a}|} f_{w(\mathbf{a}_1^{k-1})}(a_k)\,. \tag{1.6}$$

Note that the number of possible character sequences grows exponentially with the line width W. Yet it is possible to efficiently compute an optimal one defined by (1.6). Algorithm 1 implements the *dynamic programming* principle for this problem.

Algorithm 1 Dynamic programming for line partitioning

1: Initialize: $q_0 := 0$
2: **for** $i = 1$ **to** W **do**
3: $\quad \{z_i, \}q_i := \{\arg\} \min\limits_{a\in\mathcal{A}:\ i-w(a)\geq 0} \big(q_{i-w(a)} + f_{i-w(a)}(a)\big)$
4: **end for**
5: **return** Optimal costs: q_W

The notation

$$\{z_i, \}q_i := \{\arg\} \min_{a\in\mathcal{A}:\ i-w(a)\geq 0} \big(q_{i-w(a)} + f_{i-w(a)}(a)\big) \tag{1.7}$$

is used here and further to denote

$$q_i := \min_{a\in\mathcal{A}:\ i-w(a)\geq 0} \big(q_{i-w(a)} + f_{i-w(a)}(a)\big) \tag{1.8}$$

$$z_i \in \arg\min_{a\in\mathcal{A}:\ i-w(a)\geq 0} \big(q_{i-w(a)} + f_{i-w(a)}(a)\big)\,. \tag{1.9}$$

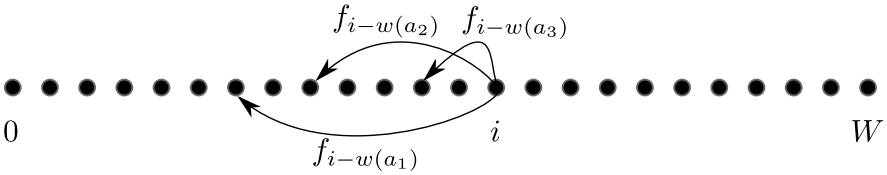


Figure 1.2: Line partitioning problem as shortest path on a graph between nodes W and 0. The length of several edges is shown.

An optimal character sequence can be obtained by backtracking the values of z_i starting with z_W, as specified by Algorithm 2.

Algorithm 2 Backtracking for line partitioning

1: Initialize: $t := N$, $i = 1$
2: **while** $t > 0$ **do**
3: $\quad a'_i := z_t$
4: $\quad t := t - w(z_t)$
5: $\quad i := i + 1$
6: **end while**
7: Flip the character's order: $a_i := (|\mathbf{a}'| - i + 1)$ for $i \in [|\mathbf{a}'|]$.
8: **return a**

The function q is called *Bellman function* in context of dynamic programming. Since Algorithm 1 performs W iterations and within each a minimum from at most $|\mathcal{A}|$ numbers is selected, its total complexity is $O(W|\mathcal{A}|)$.

Trasformation to the shortest path on a directed acyclic graph The line partitioning problem and Algorithm 1 can be seen as the problem of finding a shortest path on a directed acyclic graph, see Figure 1.2. The set of nodes of this graph $\mathcal{V} = \{0, \ldots, W\}$ and the set of edges $\mathcal{E} = \{(i, i - w(a))\colon a \in \mathcal{A},\ i - w(a) \geq 0\}$. Each edge $(i, i - w(a))$ is assigned the cost $f_{i-w(a)}(a)$.

1.4.2 Dynamic programming: General approach

In general, the dynamic programming approach consists of the following four steps:

1. Find out the state space I and the recursive update formula:

$$q_i = F(\{q_j \colon j \in J(i) \subseteq I\}), \ i \in I. \tag{1.10}$$

   In the line partitioning example from Section 1.4.1 the state space is the set of the discrete x-coordinates from 0 to W and the update formula is given by Line 3 of Algorithm 1. The set $J(i)$ is the set of outgoing edges for the node i in the graph in Figure 1.2, e.g., $J(i) = \{i - w(a) \colon a \in \mathcal{A}, \ i - w(a) \geq 0\}$.

2. Build the *dependency graph* $(\mathcal{V}, \mathcal{E})$, where

$$\mathcal{V} = I, \ \mathcal{E} = \{(i, j) \colon i \in I, \ j \in J(i)\} \tag{1.11}$$

   The graph in Figure 1.2 is the dependency graph for the above example.

3. Is the graph acyclic? Ok. Otherwise - failure.
   The graph in Figure 1.2 is acyclic.

4. Compute q_i in an inverse topological order.
   The computation order $0, 1, \ldots, W$ is the inverse topological order of the graph in Figure 1.2.

The computational complexity of the general dynamic programming algorithm is $O(|\mathcal{E}|)$, as the complexity for one node of the dependency graph is proportional to the number of edges starting in it, and each node of the graph is processed once.

1.4.3 Case study: Knapsack problem

The knapsack problem from Exercise 1.7 can be addressed with the dynamic programming as well. It is particularly easy to do if assuming

that the item volumes $w_i \leq W$ are positive integers. First we will treat the non-binary variant of the problem.

Let the state-space be all possible total volumes of the items in knapsack, *e.g.*, the numbers from 0 to W. The volume u, $0 \leq u \leq W$, can be obtained by packing an item $i \in [n]$ in addition to other items of the total volume $u - w_i$. This leads to the recursive formula

$$q_u = \max_{i \in [n]\colon u - w_i \geq 0} (q_{u-w_i} + c_i) \tag{1.12}$$

with the initial state $q_0 = 0$. This leads to the dependency graph $(\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{0, \ldots, W\}$ and $\mathcal{E} = \{(u, u - w_i)\colon u \in \mathcal{V},\ i \in [n],\ u - w_i \geq 0\}$. The order $0, 1, \ldots, W$ is inverse topological for this graph, which leads to Algorithm 3.

Algorithm 3 Dynamic programming for knapsack

1: Initialize: $q_0 := 0$, $q_u = -\infty$ for $u = 1, \ldots, W$
2: **for** $i = 1$ **to** W **do**
3: $\quad \{z_u, \} q_u := \{\arg\} \max_{i \in [n]\colon u - w_i \geq 0} (q_{u-w_i} + c_i)$
4: **end for**
5: **return** Optimal costs: $\max_{u \in \{0,\ldots,W\}} q_u$

The optimization in the last line is required, as the optimal knapsack need not be full, *i.e.*, the total volume of the items inside can be lower. This may happen even with strictly positive costs of the items, *e.g.*, if there is no such item combination that exactly fills the volume W. To exclude such a non-existing solution from the output, we initialize q_u with $-\infty$ for all u but 0.

Computational complexity, is P=NP ? The computational complexity of Algorithm 3 can be obtained similarly to those for the line partitioning: The dependency graph has $W+1$ nodes, and the update formula for each node selects the best one from at most n items. This leads to $O(nW)$.

As the knapsack problem is known to be NP-hard (also for integer item volumes), the seemingly polynomial complexity of Algorithm 3

raises the question whether $P = NP$. This is not the case, as for a polynomial algorithm its complexity must be a polynom from the problem description size. The latter is a binary vector or, what is the same, a bit sequence. In particular, to represent the total knapsack volume W one requires $k = O(\log_2 W)$ bits. With respect to this number the complexity $O(nW)$ reads as $O(n2^k)$, which is not polynomial anymore.

Dynamic Programming for the Binary Knapsack

The case of the binary knapsack is slightly more complex than its non-binary counterpart. Here, the state space is characterized by the current filled volume and the subset of items already considered. Since items can be packed in any order, we fix an arbitrary one by enumerating all items. Thus, the state is defined by the current volume u, with $0 \leq u \leq W$, and the item index i. This means that only items with indices in $\{1, \ldots, i\}$ are eligible for inclusion in the knapsack.

The Bellman function $q(i, u)$ depends on these two parameters and can be recursively computed as

$$q(i,u) = \max \left\{ \underbrace{q(i-1,u)}_{x_i=0}, \underbrace{q(i-1,u-w_i)+c_i}_{x_i=1} \right\}. \tag{1.13}$$

In words, this means:

- If the i-th item is *not* packed ($x_i = 0$), the total cost remains $q(i-1,u)$, with the same filled volume.
- If the i-th item *is* packed ($x_i = 1$), the total cost becomes $q(i-1,u-w_i)+c_i$, since the cost increases by c_i and the volume increases from $u - w_i$ to u.

The set of nodes $\mathcal{V}$ of the dependency graph $(\mathcal{V}, \mathcal{E})$ is defined as

$$\mathcal{V} = \{(i,u) \mid i \in \{0,1,\ldots,n\},\ u \in \{0,1,\ldots,W\}\}. \tag{1.14}$$

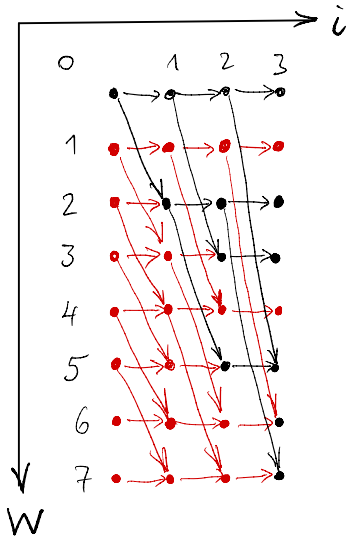


Figure 1.3: Dependency graph definded by the knapsack inequality $2x_1 + 3x_2 + 5x_3 \leq W$. Red nodes have the weight $-\infty$, red edges start in red nodes. Essentially, only black nodes and edges define feasible solutions.

Each node (i, u), with $i \in [n]$ and $u \in \{0, \ldots, W\}$, serves as the endpoint of edges that originate from $(i-1, u)$ and, if $u - w_i \geq 0$, from $(i-1, u-w_i)$.

By introducing appropriate initialization conditions for the Bellman function and processing the nodes in inverse topological order, one obtains Algorithm 4.

An example of the dependency graph is shown in Figure 1.3.

Algorithm complexity is defined by the number of edges in the dependency graph and the latter is the product of the number of nodes

Algorithm 4 Dynamic programming for binary knapsack

1: Initialize: For $i \in \{0, 1, \ldots, n\}$ assign $q(i, 0) := 0$
2: For $i \in \{0, 1, \ldots, n\}$, $u \in [W]$ assign $q(i, u) := -\infty$
3: **for** $u = 1$ **to** W **do**
4: **for** $i = 1$ **to** n **do**
5: **if** $u - w_i \geq 0$ **then**
6: $$q(i, u) = \max \left\{ \underbrace{q(i-1, u)}_{x_i=0}, \underbrace{q(i-1, u-w_i) + c_i}_{x_i=1} \right\}$$
7: **else**
8: $q(i, u) = q(i-1, u)$
9: **end if**
10: **end for**
11: **end for**
12: **return** Optimal costs: $\max_{u \in \{0,\ldots,W\}} q(n, u)$

$O(nW)$ by the number 2 of outgoing edges for each node. In total, this results in $O(nW)$.

1.5 Labeling problem

The labeling problem is one of the main running examples in this monograph. Therefore, we will introduce this problem in detail. Even in more details this problem is treated in [3].

There are many problems in computer science, which can be formulated as a *labeling problem*. Examples can be found in bio-informatics, communication theory, statistical physics, computer vision, signal processing, information retrieval and machine learning.

Labeling problems as a modeling tool naturally appear when

- the target object (the object we model) consists of *many small parts*,
- each part must be labeled by a label from a *finite set*, and

- parts (and, therefore, their labels) are *mutually dependent*.

Example 1.29 (Image segmentation)**.** Image segmentation is a classical image analysis task: Each pixel of an input image must be assigned a label of an object visible in the image. For instance, if we consider images of street scenes, these labels could belong to the set $\{\texttt{pedestrian}, \texttt{car}, \texttt{tree}, \texttt{building}\}$.

The target object is an image, i.e. a two-dimensional array of pixels. Each pixel constitutes an elementary part of the image and must be labeled with a label from a finite set. The simplest assumption about image segments, i.e. groups of pixels having the same label, is the so called "compactness assumption". It states that it is more probable that neighboring pixels are labeled with the same label than with different ones.

Example 1.30 (Depth reconstruction)**.** Depth reconstruction is another important image analysis problem. In the classical setting there are (at least) two images taken from different viewpoints. The task is to match pixels from these two images to each other. Assuming the positions of cameras and their focal lengths are known, this allows us to estimate depth of the scene, which was photographed with the cameras.

As in the previous example, the target object is a two-dimensional pixel array, where each pixel constitutes an elementary part of the object and must be labeled with a label from a finite set. Here, the meaning of the labels is different: Each label represents depth information of the associated pixel in an image, i.e. how far the depicted observation is placed from the camera. Usually the set of labels is chosen as natural numbers in a given interval, for instance, $\{0, 1, \ldots, 255\}$.

Assuming that the observed surface is smooth, one would expect the difference $|s - s'|$ between labels s and s' in neighboring pixels to be small. The opposite would mean a jump in depth, or, in other words, non-smoothness of the surface.

Example 1.31 (Cell tracking problem in bio-imaging)**.** Given is a sequence of images that show the development of a living organism

from an early embryo consisting of only a few cells to a fully grown animal. During this sequence, the images show moving and splitting cells.

Under the assumption that the image is already *pre-segmented*, i.e. the cells were already found in each image, the task at hand is to track each individual cell and its descendants from the first to the last frame.

The cells are the elementary parts of the considered object. Each cell in a given image frame is labeled with pointers to one or two cells in the next frame. One pointer means that the cell only moved, and two pointers correspond to a cell division. The simplest tracking model forbids two different cells to have the same descendants. This rule defines dependencies between object parts.

1.5.1 Basic definitions

Graph Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be an undirected graph consisting of a finite set of *nodes* $\mathcal{V}$ and a set of edges $\mathcal{E} \subseteq \binom{\mathcal{V}}{2}$. The set $\mathcal{E}$ will also be called *a neighborhood structure* of $\mathcal{V}$. For convenience, we will typically use lower case letters u and v for nodes of the graph, and write uv to denote an edge $\{u, v\} \in \mathcal{E}$ connecting u and v. Since the graph is undirected, uv and vu denote the same edge. The notation $\mathcal{N}(u)$ will be used for the set of nodes $\{v \mid uv \in \mathcal{E}\}$ connected to the node u.

The graph $\mathcal{G}$ is considered as a model of the considered target object, where the nodes represent the elementary object parts and edges stand for mutually dependencies between them.

In Examples 1.29 and 1.30 the graph $\mathcal{G}$ may have the grid structure of the underlying two-dimensional pixel array. In Example 1.31 cells of one image frame are neighbors, since their labels depend on each other.

Labels and unary costs A finite *set of labels* $\mathcal{Y}_u$ is associated with each node $u \in \mathcal{V}$. Our preference for each label is expressed by the *unary cost function* $\boldsymbol{\theta}_u \colon \mathcal{Y}_u \to \mathbb{R}$, which is defined for each node $u \in \mathcal{V}$. The value $\theta_u(s)$ determines the *cost*, which we pay for assigning label

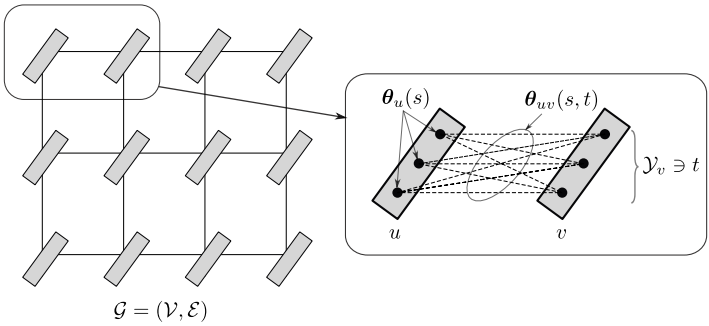


Figure 1.4: Example of a labeling problem with grid structure. On the left, graph nodes are denoted with inclined rectangles, lines connecting nodes correspond to graph edges. On the right, two neighboring nodes are shown. Black circles inside rectangles correspond to the labels s in the node u (left rectangle) and t in the node v (right rectangle). Dashed lines correspond to each label pair s, t with an assigned pairwise cost $\theta_{uv}(s,t)$.

$s \in \mathcal{Y}_u$ to the node u. Sometimes we will use very high costs to implicitly forbid certain labels or label pairs. The notation ∞ will be used to denote such high costs.

Unary costs are usually defined by what is known from observation. In Example 1.29, typically, the color distribution in the vicinity of a given pixel defines the cost of each possible label. The difference between color distributions from two or more images of the same scene taken from different viewpoints determines the unary costs in Example 1.30. Unary costs are often called the "data term" to emphasize that they depend on the input data or observation.

Dependence and pairwise costs Dependencies between labels assigned to different graph nodes are modeled with *pairwise cost functions* $\boldsymbol{\theta}_{uv} \colon \mathcal{Y}_u \times \mathcal{Y}_v \to \mathbb{R}$, which are defined for each edge $uv \in \mathcal{E}$ of the graph.

A simple (although not always the best) way to model the compactness assumption in Example 1.29 is to assign

$$\theta_{uv}(s,t) = \begin{cases} 0, & s = t \\ \alpha, & s \neq t \end{cases} \tag{1.15}$$

for any pair of labels $(s,t) \in \mathcal{Y}_u \times \mathcal{Y}_v$ with some $\alpha > 0$. A simple way to model a smooth surface in depth reconstruction in Example 1.30 is to assign

$$\theta_{uv}(s,t) = |s-t|\,, \tag{1.16}$$

to penalize large differences between depth in the neighboring nodes.

In the cell tracking example the pairwise costs should forbid the same labels to be assigned to neighboring nodes when no cell division happens:

$$\theta_{uv}(s,t) = \begin{cases} 0, & s \neq t \\ \infty, & s = t\,. \end{cases} \tag{1.17}$$

This disallows that cells u and v "glue" to the same "parent" cell $s = t$. In case of cell division, this pairwise cost function can be extended in a natural way to disallow intersection of cell descendants.

These examples show that pairwise costs often incorporate the prior information about a considered object, therefore, they are often collectively referred to as the *prior*. However, this is not always the case. For instance, much better segmentation results can be obtained if the parameter α in (1.15) depends on the color distribution of the input image, i.e. on uv.

Costs and cost functions are also called *potentials* and *potential functions*. We prefer the term *cost* since it is more widely used in general optimization literature.

Since unary and pairwise costs are functions of discrete variables, they can be seen as vectors. Therefore we can treat the unary cost function $\boldsymbol{\theta}_u$ as a *unary cost* vector $(\theta_u(s),\ s \in \mathcal{Y}_u)$. Similar reasoning holds also for each pairwise cost function, which can be considered as a *pairwise cost* vector $\boldsymbol{\theta}_{uv} = (\theta_{uv}(s,t),\ (s,t) \in \mathcal{Y}_u \times \mathcal{Y}_v)$. Unless we use the word *vector* or *function*, the context will determine whether we

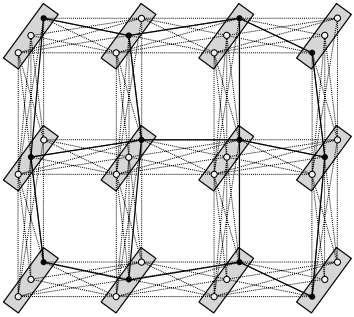


Figure 1.5: Labeling example. Selected labels are marked as black circles and connected with solid lines. Each black circle corresponds to a unary cost and each solid line to a pairwise cost in the sum in the labeling problem (1.18).

refer to a vector or a function $\boldsymbol{\theta}_u$ (or $\boldsymbol{\theta}_{uv}$). All unary vectors stacked together form the vector of all unary costs $\boldsymbol{\theta}_{\mathcal{V}} = (\boldsymbol{\theta}_u,\ u \in \mathcal{V})$. The vector $\boldsymbol{\theta}_{\mathcal{E}}$ of all pairwise costs is defined similarly as $(\boldsymbol{\theta}_{uv},\ uv \in \mathcal{E})$. Stacking together the latter two results in a long *cost vector* $\boldsymbol{\theta} = (\boldsymbol{\theta}_{\mathcal{V}}, \boldsymbol{\theta}_{\mathcal{E}})$ with dimension $\mathcal{I} := \sum_{u\in\mathcal{V}} |\mathcal{Y}_u| + \sum_{uv\in\mathcal{E}} |\mathcal{Y}_{uv}|$.

Labeling In the following, we will often use the notation $\mathcal{Y}_{\mathcal{A}}$ for all possible label assignments to a subset of nodes $\mathcal{A} \subseteq \mathcal{V}$. Formally, $\mathcal{Y}_{\mathcal{A}}$ stands for the Cartesian product $\prod_{u\in\mathcal{A}} \mathcal{Y}_u$. In particular, $\mathcal{Y}_{uv}$ denotes $\mathcal{Y}_u \times \mathcal{Y}_v$ and is the set of all possible pairs of labels in nodes u and v. A vector $y \in \mathcal{Y}_{\mathcal{V}}$ of labels assigned to *all* nodes of the graph is called *labeling*. We will refer to coordinates of this vector with the node index, i.e. y_u stands for the label assigned to the node u. One may also speak about *partial labelings*, if only a subset $\mathcal{A}$ of the nodes is labeled.

Definition 1.32 (Graphical model)**.** The triple $(\mathcal{G}, \mathcal{Y}_{\mathcal{V}}, \boldsymbol{\theta})$ consisting of a graph $\mathcal{G}$, discrete space of all labelings $\mathcal{Y}_{\mathcal{V}}$ and a corresponding cost vector $\boldsymbol{\theta}$, is called *a (discrete) graphical model.*

Definition 1.33 (Labeling problem)**.** The problem

$$\mathbf{y}^* = \arg\min_{\mathbf{y}\in\mathcal{Y}_\mathcal{V}} \left[E(\mathbf{y};\boldsymbol{\theta}) := \sum_{u\in\mathcal{V}} \theta_u(y_u) + \sum_{uv\in\mathcal{E}} \theta_{uv}(y_u, y_v) \right] \tag{1.18}$$

of finding a labeling $\mathbf{y}^*$ with minimal total cost will be called *labeling* or *maximum a posteriori (MAP) inference* problem for the graphical model $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \boldsymbol{\theta})$.

The labeling problem is also widely known as *energy minimization* for discrete graphical models, hence is objective function is often referred to as *energy*.

For the sake of notation we will sometimes use the short form of (1.18)

$$\mathbf{y}^* = \arg\min_{\mathbf{y}\in\mathcal{Y}_\mathcal{V}} \left[E(\mathbf{y};\boldsymbol{\theta}) := \sum_{w\in\mathcal{V}\cup\mathcal{E}} \theta_w(y_w) \right] \tag{1.19}$$

with y_w being equal to y_u, if w corresponds to a node, i.e. $w = u \in \mathcal{V}$, and y_{uv}, if w corresponds to an edge, i.e. $w = uv \in \mathcal{E}$.

Problems equivalent or very closely related to (1.18) have also other names depending on the corresponding community they are studied in: *maximum likelihood estimation (MLE) inference* (machine learning, natural language processing community), *weighted/valued/partial constraint satisfaction problem* (constraint satisfaction community).

1.5.2 Case study: Dynamic programming

Let the graph $\mathcal{G}$ be chain-structured. In this case the set of nodes can be totally ordered, i.e. $\mathcal{V} = \{1, \dots, n\}$. The set of edges contains pairs of neighboring nodes in the chain: $\mathcal{E} = \{(i, i+1) \colon i = 1, \dots, n-1\}$.

Consider Figure 1.6, where a chain-structured labeling problem is depicted. An optimal labeling starts in the first (left-most) node and ends in the last (right-most) one. To select its last label optimally assume we have computed the Bellman function $F_n \colon \mathcal{Y}_n \to \mathbb{R}$ such that $F_n(s) + \theta_n(s)$ denotes the cost of the best (minimal-cost) labeling with the very last label being s. In other words, the value $F_n(s)$ is

the cost of the labeling without the unary cost $\theta_n(s)$. Then the minimization $y_n = \arg\min_{s \in \mathcal{Y}_n}(F_n(s) + \theta_n(s))$, which can be performed by enumeration, allows us to determine the last label of an optimal labeling $\mathbf{y}$.

To find other labels the same considerations can be applied given that the functions $F_i \colon \mathcal{Y}_i \to \mathbb{R}$ are computed such that $F_i(s)$ equals the cost of the best partial labeling in the nodes $1, \ldots, i$ with the label s assigned to the i-th node .

Functions F_i can be recursively computed as

$$F_i(s) := \min_{t \in \mathcal{Y}_{i-1}} \left(F_{i-1}(t) + \theta_{i-1}(t) + \theta_{i-1,i}(t,s)\right) , \tag{1.20}$$

with $F_1(s) = 0$ for all $s \in \mathcal{Y}_1$.

These considerations can be derived from the following recursive representation of the energy of a chain-structured labeling problem, where functions $F_i \colon \mathcal{Y}_i \to \mathbb{R}$ for $i \in \mathcal{V}$ are introduced "on the fly":

$$\begin{aligned}
\min_{\mathbf{y} \in \mathcal{Y}_\mathcal{V}} E(\mathbf{y}; \boldsymbol{\theta}) &= \min_{y_1, \ldots, y_n} \left(\sum_{i=1}^{n-1} \Big(\theta_i(y_i) + \theta_{i,i+1}(y_i, y_{i+1})\Big) + \theta_n(y_n) \right) \\
&= \min_{y_2, \ldots, y_n} \Bigg(\underbrace{\min_{y_1 \in \mathcal{Y}_1} \Big(\theta_1(y_1) + \theta_{1,2}(y_1, y_2)\Big)}_{F_2(y_2)} \\
&\qquad + \sum_{i=2}^{n-1} \Big(\theta_i(y_i) + \theta_{i,i+1}(y_i, y_{i+1})\Big) + \theta_n(y_n) \Bigg) \\
&= \min_{y_3, \ldots, y_n} \Bigg(\underbrace{\min_{y_2 \in \mathcal{Y}_2} \Big(F_2(y_2) + \theta_2(y_2) + \theta_{2,3}(y_2, y_3)\Big)}_{F_3(y_3)} \\
&\qquad + \sum_{i=3}^{n-1} \Big(\theta_i(y_i) + \theta_{i,i+1}(y_i, y_{i+1})\Big) + \theta_n(y_n) \Bigg) \\
&\qquad \ldots \\
&= \min_{y_n \in \mathcal{Y}_n} \Big(F_n(y_n) + \theta_n(y_n)\Big) .
\end{aligned} \tag{1.21}$$

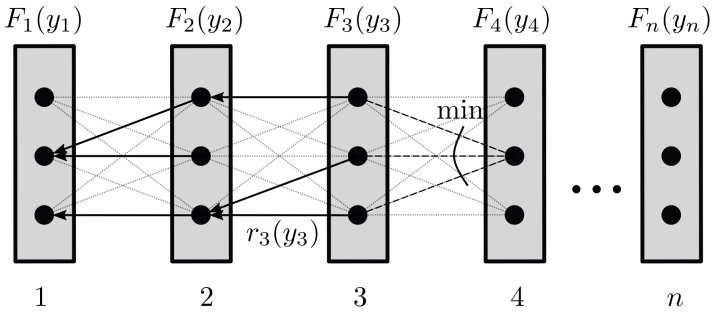


Figure 1.6: Illustration of dynamic programming on a chain: recursive expression (1.20) and Algorithm 5. Dash lines between nodes 4 and 3 connect the label y_4, which is currently computed, with all possible labels $y_3 \in \mathcal{Y}_3$. Minimization is performed over y_3 according to $F_4(y_4) = \min_{y_3 \in \mathcal{Y}_3} \Big(F_3(y_3) + \theta_3(y_3) + \theta_{3,4}(y_3, y_4) \Big)$. The arrows point to the result of this optimization during the previous steps of the algorithm and correspond to the values of $r_i(y_i)$. To reconstruct an optimal labeling one has to follow the arrows back from the last node to the first one. The starting label in the last node is chosen as $y_n = \arg\min_{s \in \mathcal{Y}_n} (F_n(s) + \theta_n(s))$.

The transformations (1.21) confirm that the energy of the optimal labeling can be computed as $\min_{\mathbf{y} \in \mathcal{Y}_\mathcal{V}} E(\mathbf{y}) = \min_{s \in \mathcal{Y}_n} (F_n(s) + \theta_n(s))$ and values $F_i(s)$ can be computed recursively as given by (1.20). We summarize these observations in Algorithm 5.

Values $F_i(s)$ are often called *forward min-marginals* or *forward messages*, since they represent an optimal labeling from the first node to the label s in the node i, and all unary/pairwise costs with indexes smaller than i are *marginalized out*.

Values $r_i(s)$ are pointers needed to recover the optimal labeling, when its energy is computed, as given by Algorithm 6. Note that $F_i(s)$ and $r_i(s)$ are obtained by the same minimization, therefore it must be performed only once.

Algorithm 5 Dynamic programming algorithm for chain-structured labeling problems

1: Given: $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \boldsymbol{\theta})$ - a chain-structured labeling problem
2: $F_1(s) := 0$, for all $s \in \mathcal{Y}_1$
3: **for** $i = 2$ **to** n **do**
4: $\quad F_i(s) := \min_{t \in \mathcal{Y}_{i-1}} (F_{i-1}(t) + \theta_{i-1}(t) + \theta_{i-1,i}(t, s))$, $\forall s \in \mathcal{Y}_i$
5: $\quad r_i(s) := \arg\min_{t \in \mathcal{Y}_{i-1}} (F_{i-1}(t) + \theta_{i-1}(t) + \theta_{i-1,i}(t, s))$, $\forall s \in \mathcal{Y}_i$
6: **end for**
7: $E^* = \min_{s \in \mathcal{Y}_n}(F_n(s) + \theta_n(s))$
8: **return** E^*, $\{r_i(s) \colon i = 2, \ldots, n,\ s \in \mathcal{Y}_i\}$

Algorithm 6 Reconstructing an optimal labeling

1: Given: $r_i(s) \colon i = 2, \ldots, n,\ s \in \mathcal{Y}_i$
2: $y_n = \arg\min_{s \in \mathcal{Y}_n}(F_n(s) + \theta_n(s))$
3: **for** $i = n$ **to** 2 **do**
4: $\quad y_{i-1} = r_i(y_i)$
5: **end for**
6: **return** $\mathbf{y}$

Computation complexity of each minimization in line 4 of Algorithm 5 is $O(|\mathcal{Y}_{i-1}|)$. One has to perform $|\mathcal{Y}_i|$ such minimizations on each iteration. Therefore, complexity of the algorithm reads

$$O\left(\sum_{i=2}^{n} |\mathcal{Y}_{i-1}||\mathcal{Y}_i|\right),$$

which results in $O(L^2|\mathcal{V}|) = O(L^2|\mathcal{E}|)$ if we assume that all nodes have an equal number L of labels. This is exactly the memory, which is required to define a chain-structured labeling problem: $|\mathcal{V}|$ unary cost vectors of the size L each plus $|\mathcal{E}|$ pairwise costs of the size L^2 each. In other words, dynamic programming for chain-structured labeling problems has linear complexity with respect to the model size.

1.6 Bibliography and further reading

The monograph [4] is a classical source for the description of classes P and NP, NP-completeness and related questions. The book [5] is a more modern reference for the subject. The work [2] is the first one providing reductions of different NP-complete problems to each other. In out treatment of NP-hardness we mostly follow the excellent text-book [6].

A historical reference for dynamic programming is the book [7]. Andrew Viterbi proposed a method for finding the most probable sequence of a hidden Markovian chain (which can be seen as a chain graphical model) in his seminal work [8]. Other terms like *believe propagation* [9], [10] and *message passing* are used for the dynamic programmig and its generalization for graphical models.

Acyclic graphical models and generalizations of dynamic programming are presented in great detail in the excellent text-book [11] which we recommend for further reading on the topic.

The dynamic programming algorithm for labeling problems is applicable not only to chain-structured graphical models, but to any acyclic, *i.e.*, tree-structured ones. The respective generalization can be found, *e.g.*, in [3], [11]. The monograph [3] covers additionally almost all other aspects, except those from Chapter 3, related to labeling problems presented in this lecture draft.

The line partitioning problem is a classical problem in the field of *optical character recognition*. Its formalization and generalization is presented in, *e.g.*, [11].

Most of the facets of the knapsack problem, including different variants of the dynamic programming algorithm, can be found in [12].

2 (Integer) linear programs

We start this chapter with a general notion of a (constrained) optimization problem and its relaxations. Further, we will focus on a specific, polyhedral constraints, which will lead us naturally to the linear programs. Finally, we will consider linear programs with integer constraints, known as integer linear programs.

2.1 Optimization problems

In the optimization literature the notation

$$\min_{\mathbf{x}\in X} f(\mathbf{x}) \tag{2.1}$$

is adopted for a problem of *minimizing* a function $f\colon X \to \mathbb{R}$ on a set $X \subseteq \mathbb{R}^n$. More precisely, it means that *the optimal value* of f defined as $f^* = \inf_{\mathbf{x}\in X} f(\mathbf{x})$ must be found. If $f^* = -\infty$, that is, there exists a sequence $\mathbf{x}^t \in X$ such that $f(\mathbf{x}^t) \overset{t\to\infty}{\longrightarrow} -\infty$, the problem is called *unbounded*. Another special case appears when X is an empty set. Then the problem is called *infeasible*, the notation $f^* = \infty$ is adopted to this case. The set X is called the *feasible set* of the problem (2.1), and $\mathbf{x} \in X$ is a *feasible point*. The fact that the problem (2.1) is neither infeasible nor unbounded, in other words, *feasible* and *bounded*, is denoted as $-\infty < f^* < \infty$. A vector $\mathbf{x}^* \in X$ such that $f(\mathbf{x}^*) = f^*$ is called an *(optimal) solution* or *optimal point* of the problem. In general, an optimal solution is not unique, or may not exist, even if $-\infty < f^* < \infty$.

Example 2.1. Although the optimal values of the problems

$$\min_{x\geq 1} e^{\frac{1}{x}}; \quad \min_{x>1} x \tag{2.2}$$

are finite and equal to 1, their optimal points do not exist.

Indeed, assume that a feasible point $x \geq 1$ is an optimal solution for the first problem. However, since $e^{\frac{1}{x+1}} < e^{\frac{1}{x}}$ and $x+1 \geq 1$ is feasible, the point x is not an optimal solution.

Similarly for the second problem, its objective value in $(1 + \frac{x-1}{2})$ is strictly smaller than in x for any feasible x, and $1 + \frac{x-1}{2}$ is feasible as soon as x is feasible.

In the following, we will mostly deal with problems where the existence of an optimal value implies also the existence of an optimal point.

A feasible point x such that $f(x)$ is approximately equal to f^* (denoted as $f(x) \approx f^*$) is often referred to as an *approximate solution* of the minimization problem (2.1).

The problem of the form (2.1) is referred to as a *constraint minimization* problem. The *constrained maximization* problem $\max_{\mathbf{x} \in X} f(x)$ is defined by substituting min with max, inf with sup, and $-\infty$ with ∞ and the other way around. Minimization and maximization problems together are also called *optimization* problems.

Example 2.2. The following optimization problems are infeasible, as their respective feasible sets are empty:

$$\min_{\substack{x \geq 1 \\ x \leq 0}} f(x); \qquad \max_{\substack{x \in \{0,1\} \\ x \geq 2}} f(x); \qquad \min_{\substack{x \in \{0,1\} \\ x \in [0.2, 0.9]}} f(x)\,. \tag{2.3}$$

Example 2.3. The following optimization problems are unbounded:

$$\min_{x \leq 0} x + 1; \qquad \min_{x \geq 0} 2 - x; \qquad \max_{x \leq 0} x^2\,. \tag{2.4}$$

Example 2.4. The following optimization problems have multiple optimal solutions

$$\min_{x \in [-1,1]} 1 - x^2; \qquad \min_{x \in \{0,1\}} |x - 0.5|; \qquad \max_{x \in [0,1]} 1\,. \tag{2.5}$$

In the rest of the monograph we will deal mostly with *linear objectives*, i.e. $f(\mathbf{x}) = \langle \mathbf{c}, \mathbf{x} \rangle$ for some vector $\mathbf{c} \in \mathbb{R}^n$. The feasible set X will usually be either finite or *polyhedral*. The definition of the latter will be given in Section 2.3.

2.1.1 Relaxations of optimization problems

Important optimization problems are often computationally intractable. Therefore, it is natural to consider tractable approximations of these problems to obtain an approximate solution of the problem or at least a bound on its optimal value. In the following, we define one of the most important types of such approximations:

Definition 2.5. The optimization problem

$$\min_{\mathbf{x}\in X'} g(\mathbf{x}) \tag{2.6}$$

is called a *relaxation* of the problem (2.1) if $X' \supseteq X$ and $g(\mathbf{x}) \leq f(\mathbf{x})$ for any $\mathbf{x} \in X$. The solution of the relaxed problem is often referred to as a *relaxed solution*.

Example 2.6. The problems $\min_{\mathbf{x}\in\mathbb{R}_{\geq 0}} f(\mathbf{x})$ and $\min_{\mathbf{x}\in[0,1]^n} f(\mathbf{x})$ are relaxations of the problem $\min_{\mathbf{x}\in\{0,1\}^n} f(\mathbf{x})$.

Example 2.7. The problem $\min_{x\in[0,1]} x^2$ is a relaxation of the problem $\min_{x\in[0,1]} |x|$. The advantage of this relaxation is the differentiability of x^2, which may often lead to faster optimization algorithms.

Example 2.8. The problems $\min\limits_{\mathbf{x}\in X\subset\mathbb{R}^n} \langle \mathbf{c}, \mathbf{x}\rangle$ and $\min\limits_{\mathbf{x}\colon A\mathbf{x}=\mathbf{b}} \langle \mathbf{c}, \mathbf{x}\rangle$ are relaxations of $\min\limits_{\substack{\mathbf{x}\in X\subset\mathbb{R}^n \\ A\mathbf{x}=\mathbf{b}}} \langle \mathbf{c}, \mathbf{x}\rangle$.

The most important property of a relaxation is provided by the following proposition:

Proposition 2.9. Any relaxation constitutes a lower bound, i.e.

$$\min_{\mathbf{x}\in X'} g(\mathbf{x}) \leq \min_{\mathbf{x}\in X} f(\mathbf{x}),$$

if $X \subseteq X'$ and $g(\mathbf{x}) \leq f(\mathbf{x})$ for $\mathbf{x} \in X$. Moreover, if $\mathbf{x}' \in X$ and $f(\mathbf{x}') = g(\mathbf{x}')$ holds for a relaxed solution $\mathbf{x}' \in \arg\min_{\mathbf{x}\in X'} g(\mathbf{x})$, then $\mathbf{x}'$ is an optimal solution of the non-relaxed problem $\min_{\mathbf{x}\in X} f(\mathbf{x})$ as well. In this case, one says that the lower bound provided by the relaxation is *tight*.

Proof. The first statement of the proposition is implied by the following inequalities $\min_{\mathbf{x}\in X'} g(\mathbf{x}) \leq \min_{\mathbf{x}\in X} g(\mathbf{x}) \leq \min_{\mathbf{x}\in X} f(\mathbf{x})$, which follow from Definition 2.5.

The second statement follows from $f(\mathbf{x}') = g(\mathbf{x}') = \min_{\mathbf{x}\in X'} g(\mathbf{x}) \leq \min_{\mathbf{x}\in X} f(\mathbf{x}) \leq f(\mathbf{x}')$. □

Relaxations, which provide tight lower bounds for *all* instances of optimization problems from a certain class, are called *tight relaxations*.

Example 2.10. Problem $\min\limits_{x\in[0,1]} cx$ is a tight relaxation for $\min\limits_{x\in\{0,1\}} cx$ for any constant c.

Definition 2.11. The relaxation $\min_{\mathbf{x}\in X'} g(\mathbf{x})$ is called *tighter* than the relaxation $\min_{\mathbf{x}\in \hat{X}'} \hat{g}(\mathbf{x})$, if it provides better lower bound, i.e. $\min_{\mathbf{x}\in X'} g(\mathbf{x}) \geq \min_{\mathbf{x}\in \hat{X}'} \hat{g}(\mathbf{x})$.

The first part of Proposition 2.9, in particular, implies that the relaxation $\min_{\mathbf{x}\in X'} g(\mathbf{x})$ is tighter than $\min_{\mathbf{x}\in \hat{X}'} \hat{g}(\mathbf{x})$ if the latter is a relaxation of the former one.

Definition 2.12. Two relaxations are called *equivalent*, if their bounds coincide.

The following simple proposition shows that uniqueness of the solution of a tight relaxation implies uniqueness of the solution of the non-relaxed problem:

Proposition 2.13. Let $\min_{\mathbf{x}\in X} f(\mathbf{x})$ be an optimization problem and $\min_{\mathbf{x}\in X'} g(\mathbf{x})$ be its relaxation. If $\mathbf{x}' \in X$ is the unique solution of the relaxed problem and $g(\mathbf{x}') = f(\mathbf{x}')$, then it is also the unique solution of the non-relaxed one.

Proof. Since $\mathbf{x}'$ is the unique relaxed solution, it holds that $f(\mathbf{x}') = g(\mathbf{x}') < g(\mathbf{x})$ for $\mathbf{x} \in X'\backslash\{\mathbf{x}'\}$. Therefore, $f(\mathbf{x}') < g(\mathbf{x}) \leq f(\mathbf{x})$ for all $\mathbf{x} \in (X\backslash\{\mathbf{x}'\}) \subseteq (X'\backslash\{\mathbf{x}'\})$, which implies that x' is the unique solution of the non-relaxed problem. □

2.2 Linear and affine spaces

Consider the vector space $\mathbb{R}^n$. We call vectors $\mathbf{x}^i \in \mathbb{R}^n$, $i \in [k]$ for some k, *linearly independent*, if

$$\sum_{i=1}^{k} a_i \mathbf{x}^i = 0 \tag{2.7}$$

implies $a_i = 0$ for all $i \in [k]$.

A *linear subspace* S in $\mathbb{R}^n$ is a subset of $\mathbb{R}^n$ closed under vector addition and multiplication.

A simple example of a linear subspace is a *linear hyperplane*, defined as the set of vectors $\mathbf{x}$ that satisfy $\langle \mathbf{a}, \mathbf{x} \rangle = 0$ for some $\mathbf{a} \in \mathbb{R}^n$, $\mathbf{a} \neq \mathbf{0}$.

It is known, that in general, a linear subspace S can be represented as an intersection of an arbitrary number m of linear hyperplanes:

$$S := \{\mathbf{x} \in \mathbb{R}^n \colon A\mathbf{x} = \mathbf{0}\}. \tag{2.8}$$

Here A is a $m \times n$ matrix.

Every linear subspace has a *dimension*, defined as the maximum number of linearly independent vectors in it. Equivalently, $\dim(S) = n - \operatorname{rank}(A)$.

An *affine subspace* S' of $\mathbb{R}^n$ is a linear subspace S translated by a vector $\mathbf{u}$: $S := \{\mathbf{x} + \mathbf{u} \colon \mathbf{x} \in S\}$. The *dimension* of S' is that of S. Equivalently, an *affine subspace* is the set of points satisfying a set of (inhomogeneous) equations

$$S' := \{\mathbf{x} \in \mathbb{R}^n \colon A\mathbf{x} = \mathbf{b}\} \tag{2.9}$$

for some matrix A and vector $\mathbf{b}$.

The dimension of *any subset* of $\mathbb{R}^n$ is the smallest dimension of any affine subspace which contains it. For example, any line segment has dimension 1; any set of k points, $k \leq n + 1$ has dimension at most $k - 1$.

Example 2.14. The dimension of the non-empty set $P := \{\mathbf{x} \in \mathbb{R}^n \colon A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq \mathbf{0}\} \neq \emptyset$ is $\dim(P) = n - \operatorname{rank}(A)$, *i.e.*, the same as of the affine subspace $\{\mathbf{x} \in \mathbb{R}^n \colon A\mathbf{x} = \mathbf{b}\}$.

The special case of an affine subspace is an *affine hyperplane*, defined as the set $\{\mathbf{x} \in \mathbb{R}^n \colon \langle \mathbf{a}, \mathbf{x} \rangle = b\}$ for some $\mathbf{a} \in \mathbb{R}^n \backslash \{\mathbf{0}\}$, $b \in \mathbb{R}$. The dimension of any affine hyperplane is $n-1$. The set $\{\mathbf{x} \in \mathbb{R}^n \mid \langle \mathbf{a}, \mathbf{x} \rangle \leq b\}$ is called an *affine half-space*.

In the following, we will omit the word *affine* when referring to affine hyperplanes or subspaces.

2.3 Linear constraints and (convex) polyhedra

In this section we consider a special type of constraint sets known as (convex) polyhedra. Polyhedra are the main components of linear and integer linear programs.

Definition 2.15. A *convex polyhedron* P in $\mathbb{R}^n$ is a set which can be represented by a finite set of linear inequalities, i.e. $P = \{\mathbf{x} \in \mathbb{R}^n \colon A\mathbf{x} \leq \mathbf{b}\}$ for a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $\mathbf{b} \in \mathbb{R}^m$. A bounded convex polyhedron is called a *convex polytope*.

Since in the following all considered polyhedra will be convex, we will omit this word when speaking about them.

As any subset of $\mathbb{R}^n$, polyhedra have their dimension. Polyhedra of dimension n are called *full-dimensional*.

Example 2.16. The set $[0,1]^n$, called an *n-dimensional cube*, is a polytope, since it can be represented as

$$\{\mathbf{x} \in \mathbb{R}^n \mid -x_i \leq 0,\ x_i \leq 1,\ i = 1 \ldots, n\}$$

and it is bounded. Note that the n-dimensional cube is full-dimensional.

Constraints $\mathbf{x} \in [0,1]^n$ constitute a special case of *box constraints*. The latter in general may specify a different feasible interval for each coordinate of $\mathbf{x}$.

Polyhedra can be represented also by a mixture of equalities and inequalities:

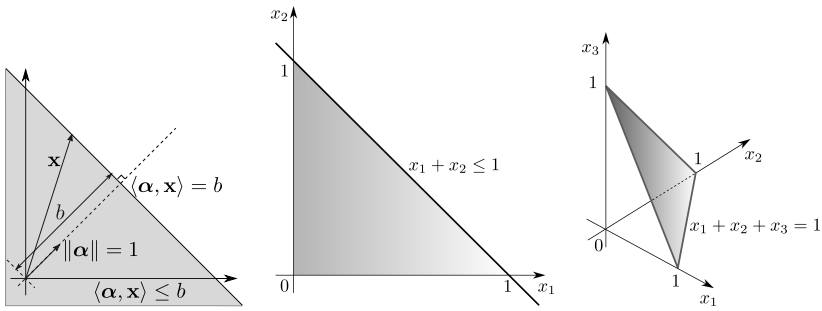


Figure 2.1: Examples of different polyhedra:
(left) Hyperspace $\langle \boldsymbol{\alpha}, \mathbf{x} \rangle \leq b$;
(middle) Polytope $\{x_1 \geq 0,\ x_2 \geq 0,\ x_1 + x_2 \leq 1\}$;
(right) Polytope (three-dimensional simplex)
$\Delta^3 = \{\mathbf{x} \in \mathbb{R}^3 \colon x_1 + x_2 + x_3 = 1,\ x_1, x_2, x_3 \geq 0\}$.

Proposition 2.17. The set of the type $\{\mathbf{x} \in \mathbb{R}^n \colon A\mathbf{x} = \mathbf{b},\ B\mathbf{x} \leq \mathbf{d}\}$, where A and B are arbitrary matrices and $\mathbf{b}, \mathbf{d}$ vectors of suitable dimensions, is a polyhedron in $\mathbb{R}^n$.

Proof. The proof follows from the fact that the equality constraints $A\mathbf{x} = \mathbf{b}$ can be equivalently represented as inequalities:

$$\{A\mathbf{x} \leq \mathbf{b},\ -A\mathbf{x} \leq -\mathbf{b}\}. \qquad \square$$

Example 2.18. A hyperplane $\langle \mathbf{a}, \mathbf{x} \rangle = b$ is a polyhedron, and so is an intersection of multiple hyperplanes $A\mathbf{x} = \mathbf{b}$. Indeed, as an intersection of hyperplanes $A\mathbf{x} = \mathbf{b}$ defines a affine subspace, which is unbounded unless it is empty or consists of a single point.

The set of polyhedra, as well as the set of polytopes are closed with respect to intersection:

Proposition 2.19. The intersection of two polyhedra is a polyhedron itself. Moreover, if one of the polyhedra is a polytope, the intersection is a polytope as well.

Proof. The proof of the first statement follows from the fact that an intersection of two polyhedra given by $A\mathbf{x} \leq \mathbf{b}$ and $B\mathbf{x} \leq \mathbf{d}$, respectively, is the set $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{b},\ B\mathbf{x} \leq \mathbf{d}\}$ constrained by both these sets of inequalities. The second statement is trivial as the intersection is a subset of the polytope at hand. □

2.3.1 Convex hulls and vertices of polyhedra

There is an important one-to-one relation between a polytope and the set of its vertices, which plays an important role in the analysis of (integer) linear programs. Below, we define this relation.

Definition 2.20. A vector $\mathbf{x}' \in \mathbb{R}^n$ is called a *vertex* of a polyhedron P if there exists a vector $\mathbf{c} \in \mathbb{R}^n$ such that the linear function $\langle \mathbf{c}, \mathbf{x} \rangle$ attains a unique maximum at $\mathbf{x}'$ on P.

In other words, $\mathbf{x}'$ is a vertex, if it is the unique solution of

$$\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle .$$

Definition 2.20 is illustrated in Figure 2.2. We will use the notation $\mathrm{vrtx}(P)$ for the set of vertices of a polyhedron P.

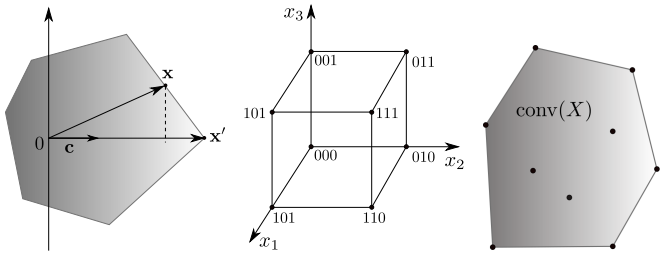
Note that since $\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle = -\min_{\mathbf{x} \in P}(-\langle \mathbf{c}, \mathbf{x} \rangle)$, the maximization problem $\max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle$ can be equivalently exchanged with

$$\min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle$$

in Definition 2.20.

Example 2.21. Let us show that $\{0,1\}^n \subseteq \mathrm{vrtx}([0,1]^n)$. Indeed, consider $\mathbf{z} \in \{0,1\}^n$ and a linear function $\langle \mathbf{c}, \mathbf{x} \rangle$ with $c_i = 2z_i - 1$. Then

$$\begin{aligned} \langle \mathbf{c}, \mathbf{x} \rangle = 2\sum_{i=1}^{n} z_i x_i - \sum_{i=1}^{n} x_i &= 2 \sum_{\substack{i=1,\dots,n: \\ z_i = 1}} x_i - \sum_{i=1}^{n} x_i \\ &= \sum_{\substack{i=1,\dots,n: \\ z_i = 1}} x_i - \sum_{\substack{i=1,\dots,n: \\ z_i = 0}}^{n} x_i \,. \end{aligned} \tag{2.10}$$



(a) (b) (c)

Figure 2.2: **(a)** Illustration of Definition 2.20 of a polyhedron vertex. The value of the scalar product $\langle \mathbf{c}, \mathbf{x} \rangle$ is the length of the orthogonal projection of a point $\mathbf{x}$ of the polyhedron onto the direction of the vector $\mathbf{c}$. The length of the projection of the vertex $\mathbf{x}'$ is the longest one. **(b)** Illustration of the statement $\{0,1\}^n = \mathrm{vrtx}([0,1]^n)$. **(c)** Illustration of the statement $\mathrm{vrtx}(\mathrm{conv}(X)) \subseteq X$. Each point in the finite set X is denoted by a dot.

Therefore, $\max_{\mathbf{x} \in [0,1]^n} \langle \mathbf{c}, \mathbf{x} \rangle = \max_{\mathbf{x} \in [0,1]^n} \sum_{i=1,\dots,n:\ z_i=1} x_i - \sum_{i=1,\dots,n:\ z_i=0} x_i = \sum_{i=1}^{n} z_i$, and the maximum is attained in the point $\mathbf{z}$ only. Therefore, according to Definition 2.20, $\mathbf{z}$ is a vertex of $[0,1]^n$.

Let $\mathbb{R}_+ := \{x \in \mathbb{R} \mid x \geq 0\}$ denote the set of non-negative real numbers and $\mathbb{R}_+^n = \prod_{i=1}^n \mathbb{R}_+$ stand for the non-negative cone of the n-dimensional vector space.

Definition 2.22. The polytope $\Delta^n := \{\mathbf{x} \in \mathbb{R}_+^n : \sum_{i=1}^n x_i = 1\}$ is called *n-dimensional (probability) simplex.*[1] For finite $\mathcal{X}$ we also use the notation $\Delta^{\mathcal{X}}$ for vectors from $\Delta^{|\mathcal{X}|}$ whose coordinates are indexed by elements of the set $\mathcal{X}$. Note that the dimension of the n-dimensional simplex is actually $n-1$, as follows from Example 2.14.

[1] Although dimensionality of the n-dimensional simplex as a manifold is $n-1$, we stick to the above name. For the purpose of this book the dimensionality of the space plays more important role than the dimensionality of the simplex itself.

A three-dimensional simplex is shown in Figure 2.1.

Example 2.23. Let us show that vectors $\mathbf{x}^i \in \mathbb{R}^n$, $i = 1, \dots, n$ such that $x^i_j = [\![i = j]\!]$ are vertices of the simplex Δ^n. Indeed, for $\mathbf{c} = \mathbf{x}^i$ the maximum $\max_{\mathbf{x} \in \Delta^n} \langle \mathbf{c}, \mathbf{x} \rangle$ is attained in a single point $\mathbf{x}^i$. Therefore, $\mathbf{x}^i \in \operatorname{vrtx}(\Delta^n)$.

Note also that $\mathbf{x}^i \in \mathbb{R}^n$, $i = 1, \dots, n$, are the only binary (with coordinates 0 and 1) vectors in Δ^n. In other words, $\{\mathbf{x}^i \in \mathbb{R}^n \mid i = 1, \dots, n\} = \Delta^n \cap \{0, 1\}^n$. This implies that $\Delta^n \cap \{0, 1\}^n \subseteq \operatorname{vrtx}(\Delta^n)$. Moreover, later we will show that these sets are equal, i.e. $\Delta^n \cap \{0, 1\}^n = \operatorname{vrtx}(\Delta^n)$.

Faces of polyhedra

Definition 2.24. A *face* P' of a polyhedron $P \subseteq \mathbb{R}^n$ is the set that can be expressed as $P' = \arg\min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle$ for some $\mathbf{c} \in \mathbb{R}^n$.

The most important types of faces are (see Figure 2.3):

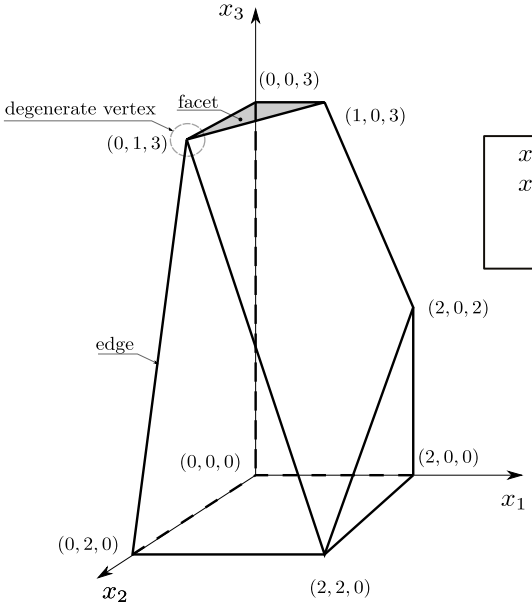
- *vertex* - dimension 0 (point)
- *edge* - dimension 1 (line) and
- *facet* - dimension $\dim(P) - 1$.

As we show later in Corollary 2.52 a face of a polytope is a polytope itself.

Definition 2.25. For any finite number N of points $\mathbf{x}^i \in \mathbb{R}^n$, $i = 1, \dots, N$ and any $\mathbf{p} \in \Delta^N$ the point $\sum_{i=1}^N p_i \mathbf{x}^i$ is called *a convex combination* of $\mathbf{x}^i$, $i = 1, \dots, N$.

Definition 2.26. A set $X \subseteq \mathbb{R}^n$ is called *convex*, if for any $\alpha \in [0, 1]$ and any two points $\mathbf{x}, \mathbf{z} \in X$ their convex combination $\alpha\mathbf{x} + (1 - \alpha)\mathbf{z}$ also belongs to X.

It is sometimes easier to use an equivalent definition:



$$\begin{array}{rcrcrcl} x_1 & + & x_2 & + & x_3 & \leq & 4 \\ x_1 & & & & & \leq & 2 \\ & & & & x_3 & \leq & 3 \\ & & 3x_2 & + & x_3 & \leq & 6 \end{array}$$

Figure 2.3: Faces of polyhedron.

Definition 2.27. A set $X \subseteq \mathbb{R}^n$ is called *convex*, if for any any finite number N of its points $\mathbf{x}^i \in X$, $i = 1, \ldots, N$, and any $\mathbf{p} \in \Delta^N$ it holds that $(\sum_{i=1}^N p_i \mathbf{x}^i) \in X$.

Convexity is closed w.r.t. the intersection:

Lemma 2.28. Let X be representable as the intersection of an arbitrary number of convex sets. Then X is convex as well.

Proof. Let $\mathbf{x}$ and $\mathbf{z}$ belong to the intersection. Then $\mathbf{x}$ and $\mathbf{z}$ belong to each of the sets. Consider now one of these sets. Then, for any $\alpha \in [0, 1]$ the point $\hat{\mathbf{x}} = \alpha\mathbf{x} + (1 - \alpha)\mathbf{z}$ belongs to this set as well due to its convexity. Since this consideration holds for any set involved in the intersection, $\hat{\mathbf{x}}$ belongs to the intersection as well. □

Proposition 2.29. Polyhedra are convex sets.

Proof. It is necessary to prove that the set $X = \{\mathbf{x} \colon A\mathbf{x} \leq \mathbf{b}\}$ is convex, where A is a matrix and $\mathbf{b}$ a vector of suitable dimensions. Let $p \in [0,1]$ and $\mathbf{x}, \mathbf{z} \in X$. The proof follows from

$$A(p\mathbf{x} + (1-p)\mathbf{z}) = pA\mathbf{x} + (1-p)A\mathbf{z} = p\mathbf{b} + (1-p)\mathbf{b} = \mathbf{b}\,.$$

□

Definition 2.30. For $X \subset \mathbb{R}^n$ the set $\{\mathbf{s} \in \mathbb{R}^n \mid \exists N \in \mathbb{N},\ \mathbf{p} \in \Delta^N \colon\ s = \sum_{i=1}^{N} p_i \mathbf{x}^i,\ \mathbf{x}^i \in X\}$ of points representable as a convex combination of a finite number of points of X is called the *convex hull* of X and will be denoted as $\mathrm{conv}(X)$.

The famous Caratheodory's theorem limits the maximum number of required points in Definition 2.30:

Theorem 2.31 (Caratheodory)**.** Let $X \subset \mathbb{R}^n$. Then any $\mathbf{x} \in \mathrm{conv}(X)$ can be represented as $\mathbf{x} = \sum_{i=1}^{n+1} p_i \mathbf{x}^i$, where $\mathbf{x}^i \in X$ (not necessary distinct) and $\mathbf{p} \in \Delta^{n+1}$.

Exercise 2.32. Prove that X is a convex set if and only if $\mathrm{conv}(X) = X$.

Exercise 2.33. Prove that $\mathrm{conv}(X)$ is a minimum convex set containing X, that is, for any convex set $C \supseteq X$ it holds that $\mathrm{conv}(X) \subseteq C$.

Proposition 2.34. Let $Z \subseteq \mathbb{R}^n$ and $X \subseteq Z$. Then $\mathrm{conv}(X) \subseteq \mathrm{conv}(Z)$.

Proof. Any $\mathbf{x} \in \mathrm{conv}(X)$ is representable as $\sum_{i=1}^{n+1} p_i \mathbf{x}^i$, $\mathbf{p} \in \Delta^{n+1}$ and $\mathbf{x}^i \in X$. Since $\mathbf{x}^i \in Z$, the point $\mathbf{x} = \sum_{i=1}^{n+1} p_i \mathbf{x}^i$ also belongs to $\mathrm{conv}(Z)$. □

Example 2.35. Let us show that simplex Δ^n is a convex hull of vectors $\{\mathbf{x}^i \in \mathbb{R}^n \mid i = 1, \ldots, n,\ x^i_j = [\![i = j]\!]\} = \Delta^n \cap \{0,1\}^n$.

Indeed, any vector $\mathbf{p} \in \Delta^n$ is representable as $\sum_{i=1}^{n} p_i \mathbf{x}^i$. The other way around, all vectors of the form $\sum_{i=1}^{n} p_i \mathbf{x}^i$ belong to Δ^n.

A simple but important technical lemma simplifies the computation of convex hulls in multidimensional spaces:

Lemma 2.36 (Convex hull of a Cartesian product)**.** Let $X = \prod_{i=1}^{n} X^i$ be the Cartesian product of $X^i \subseteq \mathbb{R}^{d_i}$, where d_i is the dimension of the subspace. Then $\operatorname{conv}(X) = \prod_{i=1}^{n} \operatorname{conv}(X^i)$.

Proof. It is sufficient to give the proof for $n = 2$, since $\prod_{i=1}^{n} X^i = X' \times X^n$, where $X' = \prod_{i=1}^{n-1} X^i \subseteq \mathbb{R}^{\sum_{i=1}^{n-1} d_i}$, and the statement of the lemma can be applied recursively. Therefore, let $X = Y \times Z$, $Y \in \mathbb{R}^{d_1}$, $Z \in \mathbb{R}^{d_2}$.

Proof of $\operatorname{conv}(X) \subseteq (\operatorname{conv}(Y) \times \operatorname{conv}(Z))$: Elements of $\operatorname{conv}(X)$ are representable as $\mathbf{x} = (\sum_{i=1}^{N} p_i \mathbf{y}^i, \sum_{i=1}^{N} p_i \mathbf{z}^i)$ with $N = \sum_{i=1}^{n} d_i$ and $\mathbf{p} \in \Delta^N$. By definition, $\sum_{i=1}^{N} p_i \mathbf{y}^i \in \operatorname{conv}(Y)$ and $\sum_{i=1}^{N} p_i \mathbf{z}^i \in \operatorname{conv}(Z)$. Therefore, $\mathbf{x} \in \operatorname{conv}(Y) \times \operatorname{conv}(Z)$.

Proof of $(\operatorname{conv}(Y) \times \operatorname{conv}(Z)) \subseteq \operatorname{conv}(X)$: Let $\mathbf{p} \in \Delta^k$ and $\mathbf{q} \in \Delta^m$. Elements of $\operatorname{conv}(Y) \times \operatorname{conv}(Z)$ are representable as

$$\mathbf{x} = (\sum_{i=1}^{k} p_i \mathbf{y}^i, \sum_{j=1}^{m} q_j \mathbf{z}^j) = \sum_{i=1}^{k} \sum_{j=1}^{m} p_i \mathbf{q}_j (\mathbf{y}^i, \mathbf{z}^j)\,. \tag{2.11}$$

Note that $p_i q_j \geq 0$ and

$$\sum_{i=1}^{k} \sum_{j=1}^{m} p_i q_j = \sum_{i=1}^{k} p_i \underbrace{\sum_{j=1}^{m} q_j}_{=1} = \sum_{i=1}^{k} p_i = 1\,. \tag{2.12}$$

Since $(\mathbf{y}^i, \mathbf{z}^j) \in X$, this implies $\mathbf{x} \in \operatorname{conv}(X)$. □

Example 2.37. Due to Lemma 2.36 the equality $\operatorname{conv}(\{0,1\}^n) = [0,1]^n$ follows directly from the fact that $\operatorname{conv}(\{0,1\}) = [0,1]$.

The following lemmas show that convexity is invariant with respect to linear mappings:[2]

[2] In this book we do not distinguish between linear and affine mappings and call both of them linear.

Lemma 2.38. Let X be a convex set. Then the set $Z := \{\mathbf{z} \mid \mathbf{z} = A\mathbf{x} + \mathbf{b},\ \mathbf{x} \in X\}$ is convex as well.

Proof. Let $\mathbf{z}, \mathbf{z}' \in Z$. Then they are representable as $\mathbf{z} = A\mathbf{x} + \mathbf{b}$ and $\mathbf{z}' = A\mathbf{x}' + \mathbf{b}$ for some $\mathbf{x}, \mathbf{x}' \in X$ respectively. Consider $p \in (0, 1)$ and $p\mathbf{z}+(1-p)\mathbf{z}' = p(A\mathbf{x}+\mathbf{b})+(1-p)(A\mathbf{x}'+\mathbf{b}) = A(p\mathbf{x}+(1-p)\mathbf{x}')+\mathbf{b}$. Since X is convex, $p\mathbf{x}+(1-p)\mathbf{x}' \in X$ and, therefore, $p\mathbf{z}+(1-p)\mathbf{z}' \in Z$. □

Lemma 2.39 (Convex hull of a linear mapping image)**.** $\operatorname{conv}\{A\mathbf{x}+\mathbf{b} \mid \mathbf{x} \in X\} = \{A\mathbf{x} + \mathbf{b} \mid x \in \operatorname{conv}(X)\}$, or in other words, convex hull commutes with linear mapping.

Proof. Let

$$Z := \operatorname{conv}\{A\mathbf{x} + \mathbf{b} \mid \mathbf{x} \in X\} \quad \text{and} \quad \hat{Z} := \{A\mathbf{x} + \mathbf{b} \mid \mathbf{x} \in \operatorname{conv}(X)\}.$$

$\hat{Z} \subseteq Z$: Let $\hat{\mathbf{z}} = A\mathbf{x}' + \mathbf{b}$, $\mathbf{x}' \in \operatorname{conv}(X)$. Therefore, $\mathbf{x}' = \sum_{i=1}^{N} p_i \mathbf{x}^i$ with $\mathbf{x}^i \in X$ and $\mathbf{p} \in \Delta^N$. Hence,

$$\begin{aligned}\hat{Z} \ni \hat{\mathbf{z}} = A\mathbf{x}' + \mathbf{b} &= A(\sum_{i=1}^{N} p_i \mathbf{x}^i) + (\sum_{i=1}^{N} p_i)\mathbf{b} \\ &= \sum_{i=1}^{N} p_i (A\mathbf{x}^i) + \sum_{i=1}^{N} p_i \mathbf{b} = \sum_{i=1}^{N} p_i (A\mathbf{x}^i + \mathbf{b}) \in Z\,. \quad (2.13)\end{aligned}$$

$Z \subseteq \hat{Z}$: Consider $\mathbf{z} = \sum_{i=1}^{N} p_i (A\mathbf{x}^i + \mathbf{b}) \in Z$, $\mathbf{x}^i \in X$, and apply the transformation above in the opposite direction.

□

From Definitions 2.26 and 2.30 it follows directly that the convex hull is a convex set. The inverse holds trivially, since each convex set can be seen as its own convex hull. A non-trivial result, stated in the theorem below, describes an important relation between *finite* sets and their convex hulls:

Theorem 2.40 (Minkowski (1896))**.** The set $P \subset \mathbb{R}^n$ is a polytope if and only if it is representable as the convex hull of a finite set of points.

Corollary 2.41. A polytope is the convex hull of its vertices.

Proofs of both statements are beyond the scope of this monograph. In Section 2.7 we reference the literature, where these proofs are given.

Whereas Corollary 2.41 claims that any polytope is uniquely defined by a finite set (of its vertices) by its convex hull, the following theorem describes vertices of a convex hull of a finite set of points:

Theorem 2.42. Let $X \subset \mathbb{R}^n$ be a finite set and $\mathbf{z}$ be a vertex of $\operatorname{conv}(X)$. Then $\mathbf{z} \in X$.

Theorem 2.42 is illustrated in Figure 2.2. To prove Theorem 2.42 we will need the following two simple but very useful Lemmata:

Lemma 2.43. Let $a \in \mathbb{R}^n$ and $p^* = \arg\min_{p \in \Delta^n} \sum_{i=1}^n p_i a_i$. Then $p_j^* = 0$ for all j such that $a_j > \min_{i=1,\dots,n}\{a_i\}$.

Proof. Let $i^* := \arg\min_{i=1,\dots,n}\{a_i\}$ and $a_* := a_{i^*}$. Let also $p_j^* > 0$ for some $a_j > a_*$ and

$$p_i' = \begin{cases} p_i^*, & i \notin \{i^*, j\}\,, \\ 0, & i = j\,. \\ p_{i^*}^* + p_j^*, & i = i^*\,. \end{cases} \tag{2.14}$$

Note that $p' \in \Delta^n$ if $p^* \in \Delta^n$. Then

$$\begin{aligned} \sum_{i=1}^n p_i' a_i &= p_{i^*}^* a_* + p_j^* a_* + \sum_{i \notin \{i^*, j\}} p_i^* a_i \\ &\overset{a_* < a_j}{<} p_{i^*}^* a_* + p_j^* a_j + \sum_{i \notin \{i^*, j\}} p_i^* a_i = \sum_{i=1}^n p_i^* a_i\,, \end{aligned} \tag{2.15}$$

which contradicts the definition of p^*. □

Let $P \subset \mathbb{R}^n$ be a polytope and $\mathbf{x}^* \in \mathrm{vrtx}(P)$ be its vertex. Corollary 2.41 implies that $\mathbf{x}^*$ can be represented as a convex combination of the vertices of P. That is, there exists $\mathbf{p} \in \Delta^{\mathrm{vrtx}(P)}$ such that

$$\mathbf{x}^* = \sum_{\mathbf{x} \in \mathrm{vrtx}(P)} p_{\mathbf{x}} \mathbf{x} \,. \tag{2.16}$$

Lemma 2.44. For any $\mathbf{p}$ that satisfies (2.16) it holds $p_{\mathbf{x}} = 0$ for all $\mathbf{x} \neq \mathbf{x}^*$, or, equivalently, $p_{\mathbf{x}^*} = 1$.

Proof. $\mathbf{x}^* \in \mathrm{vrtx}(P)$ implies that there is $\mathbf{c} \in \mathbb{R}^n$ such that

$$\langle \mathbf{c}, \mathbf{x}^* \rangle = \left\langle \mathbf{c}, \sum_{\mathbf{x} \in \mathrm{vrtx}(P)} p_{\mathbf{x}} \mathbf{x} \right\rangle = \sum_{\mathbf{x} \in \mathrm{vrtx}(P)} p_{\mathbf{x}} \langle \mathbf{c}, \mathbf{x} \rangle$$
$$\overset{\text{Def. 2.20}}{=} \max_{\mathbf{x} \in \mathrm{vrtx}(P)} \langle \mathbf{c}, \mathbf{x} \rangle \tag{2.17}$$

Moreover, the maximum in the the last item of (2.17) is attained uniquely. The latter implies $p_{\mathbf{x}} = 0$ for all $\mathbf{x} \neq \mathbf{x}^*$ according to Lemma 2.43. □

Let X be a finite set and $\mathbf{x}^* \in \mathrm{vrtx}(\mathrm{conv}(X))$. Then $\mathbf{x}^* \in \mathrm{conv}(X)$ and therefore

$$\mathbf{x}^* = \sum_{\mathbf{x} \in X} p_{\mathbf{x}} \mathbf{x} \,. \tag{2.18}$$

The following lemma generalizes the result of Lemma 2.44:

Lemma 2.45. For any $\mathbf{p}$ that satisfies (2.18) it holds $p_{\mathbf{x}} = 0$ for all $\mathbf{x} \neq \mathbf{x}^*$, or, equivalently, $p_{\mathbf{x}^*} = 1$.

Proof. According to Corollary 2.41 each $\mathbf{x}' \in X \subseteq \mathrm{conv}(X)$ is, in turn, representable as $\mathbf{x}' = \sum_{\mathbf{x} \in \mathrm{vrtx}(\mathrm{conv}(X))} q[\mathbf{x}']_{\mathbf{x}} \mathbf{x}$ for some $q[\mathbf{x}'] \in$

$\Delta^{\mathrm{vrtx}(\mathrm{conv}(X))}$. By plugging it into (2.18), we obtain

$$\mathbf{x}^* = \sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} \mathbf{x}' = \sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} \sum_{\mathbf{x} \in \mathrm{vrtx}(\mathrm{conv}(X))} q[\mathbf{x}']_{\mathbf{x}} \mathbf{x}$$
$$= \sum_{\mathbf{x} \in \mathrm{vrtx}(\mathrm{conv}(X))} \underbrace{\sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} q[\mathbf{x}']_{\mathbf{x}}}_{:=\alpha_{\mathbf{x}}} \mathbf{x} \,. \quad (2.19)$$

Note that $\alpha_{\mathbf{x}} \geq 0$ and

$$\sum_{\mathbf{x} \in \mathrm{vrtx}(\mathrm{conv}(X))} \alpha_{\mathbf{x}} = \sum_{\mathbf{x} \in \mathrm{vrtx}(\mathrm{conv}(X))} \sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} q[\mathbf{x}']_{\mathbf{x}}$$
$$= \sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} \underbrace{\sum_{\mathbf{x} \in \mathrm{vrtx}(\mathrm{conv}(X))} q[\mathbf{x}']_{\mathbf{x}}}_{=1, \text{ as } q[\mathbf{x}'] \in \Delta^{\mathrm{vrtx}(\mathrm{conv}(X))}} = \sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} = 1 \,. \quad (2.20)$$

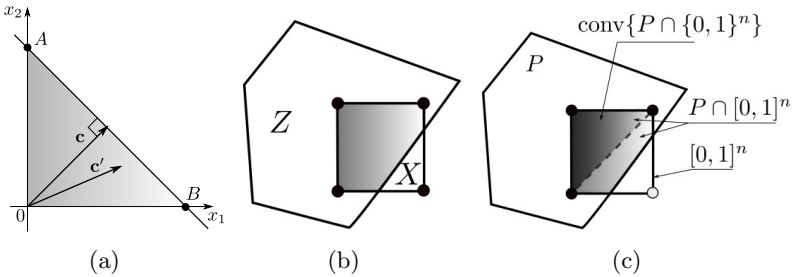
Hence, $\boldsymbol{\alpha} \in \Delta^{\mathrm{vrtx}(\mathrm{conv}(X))}$. Now, Lemma 2.44 applied to the polytope $\mathrm{conv}(X)$ implies that $\alpha_{\mathbf{x}} = [\![\mathbf{x} = \mathbf{x}^*]\!]$.

Assume now that $p_{\mathbf{x}^*} < 1$. Then there exists $\mathbf{x} \neq \mathbf{x}^*$ such that $p_{\mathbf{x}} > 0$ and since $q[\mathbf{x}']_{\mathbf{x}^*} < 1$ for all $\mathbf{x}' \neq \mathbf{x}^*$, it implies

$$1 = \alpha_{\mathbf{x}^*} = \sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} q[\mathbf{x}']_{\mathbf{x}^*} = \sum_{\mathbf{x}' \in X \setminus \{\mathbf{x}^*\}} p_{\mathbf{x}'} q[\mathbf{x}']_{\mathbf{x}^*} + p_{\mathbf{x}^*} q[\mathbf{x}^*]_{\mathbf{x}^*}$$
$$\overset{\substack{q[\mathbf{x}']_{\mathbf{x}^*} < 1 \\ p_{\mathbf{x}} > 0}}{<} \sum_{\mathbf{x}' \in X \setminus \{\mathbf{x}^*\}} p_{\mathbf{x}'} + p_{\mathbf{x}^*} = \sum_{\mathbf{x}' \in X} p_{\mathbf{x}'} = 1 \,. \quad (2.21)$$

The latter is a contradiction that proves that $p_{\mathbf{x}^*} = 1$. □

Proof of Theorem 2.42. According to Lemma 2.45 convex representations of vertices of $\mathrm{conv}(X)$ are trivial, hence they must belong to X. □



(a) (b) (c)

Figure 2.4: **(a)** Illustration of optimal solutions of a linear program. Given the polytope P with vertices $0, A$ and B, the point B is the unique solution of $\max_{\mathbf{x}\in P} \langle \mathbf{c}', \mathbf{x} \rangle$, whereas for the problem $\max_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x} \rangle$ (with $\mathbf{c}$ orthogonal to the line AB) any point on the line between points A and B is a solution. **(b)** Illustration of Proposition 2.48. Vertices of a polytope X remain vertices of the intersection $X \cap Z$, if they belong to this intersection. **(c)** Illustration of Corollary 2.65: the set of vertices of the feasible set of the LP relaxation ($P \cap [0,1]^n$) contains the vertices of the integer hull ($\mathrm{conv}(P \cup \{0,1\}^n)$).

Example 2.46. According to Example 2.16, $[0,1]^n$ is a polytope. Let us show that the set of its vertices is precisely the set $\{0,1\}^n$.

Since $\mathrm{conv}(\{0,1\}^n) = [0,1]^n$ (see Example 2.37), Theorem 2.42 implies $\mathrm{vrtx}([0,1]^n) \subseteq \{0,1\}^n$. The inverse inclusion $\{0,1\}^n \subseteq \mathrm{vrtx}([0,1]^n)$ is proven in Example 2.21, which finalizes the proof.

Example 2.47. $\Delta^n \cap \{0,1\}^n = \mathrm{vrtx}(\Delta^n)$. In other words, the binary vectors of a simplex are all its vertices.

Since $\Delta^n = \mathrm{conv}(\Delta^n \cap \{0,1\}^n)$ (see Example 2.35), Theorem 2.42 implies $\mathrm{vrtx}(\Delta^n) \subseteq \Delta^n \cap \{0,1\}^n$. The inverse inclusion $\Delta^n \cap \{0,1\}^n \subseteq \mathrm{vrtx}(\Delta^n)$ is proven in Example 2.23, which finalizes the proof.

The following simple lemma is useful to find vertices of polytopes obtained as an intersection of polyhedra:

Proposition 2.48. Let X be a polytope and Z be a polyhedron in $\mathbb{R}^n$, and $\mathbf{x} \in X \cap Z$. If additionally $\mathbf{x}$ is a vertex of X, then $\mathbf{x}$ is a vertex of $X \cap Z$.

The proposition is illustrated in Figure 2.4(b).

Proof. Since $\mathbf{x}$ is a vertex of X, there exists $\mathbf{c} \in \mathbb{R}^n$ such that $\mathbf{x}$ is the unique solution of $\min_{\mathbf{x}' \in X} \langle \mathbf{c}, \mathbf{x}' \rangle$. Since $\mathbf{x} \in Z$, it holds also that it is a unique solution of $\min_{\mathbf{x}' \in X \cap Z} \langle \mathbf{c}, \mathbf{x}' \rangle$ (see Proposition 2.13), and, therefore, $\mathbf{x}$ is a vertex of $X \cap Z$. □

2.4 Linear programs

Linear programs (LP) are the most important components of integer linear programs – a unified representation for combinatorial problems. We will see below that although linear programs constitute a continuous, non-discrete object, the set of points, which is sufficient to check to obtain their exact solution, is finite. Moreover, it can be shown that this set grows exponentially with the dimension of the problem in general. In this sense linear programs can be seen as combinatorial problems themselves. The main difference, however, is in their solvability: There exist polynomial algorithms for any linear program, whereas the class of combinatorial problems contains $\mathcal{NP}$-hard problems, like, for example, the considered MAP-inference for graphical models. Interestingly, polynomially solvable combinatorial problems often can be equivalently represented in the form of a linear program, whose size grows polynomially with the size of the combinatorial problem.

In this section we will discuss the most important properties of linear programs.

Definition 2.49. Let P be a polyhedron in $\mathbb{R}^n$ defined by a set of linear constraints. Optimization problems of the form

$$\min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \qquad \text{and} \qquad \max_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle \tag{2.22}$$

are called *linear programs (LP)*.

The following statement is the key to the combinatorial property of linear programs:

Theorem 2.50. For any $\mathbf{c} \in \mathbb{R}^n$ and any finite $X \subset \mathbb{R}^n$ it follows

$$\min_{\mathbf{x}\in X} \langle \mathbf{c}, \mathbf{x} \rangle = \min_{\mathbf{x}\in \operatorname{conv}(X)} \langle \mathbf{c}, \mathbf{x} \rangle \ , \tag{2.23}$$

and

$$\operatorname{conv}\left(\arg\min_{\mathbf{x}\in X} \langle \mathbf{c}, \mathbf{x} \rangle \right) = \arg\min_{\mathbf{x}\in \operatorname{conv}(X)} \langle \mathbf{c}, \mathbf{x} \rangle \ . \tag{2.24}$$

Proof. The proof of the first statement is given by the transformaiton:

$$\begin{aligned} \min_{\mathbf{x}\in \operatorname{conv}(X)} \langle \mathbf{c}, \mathbf{x} \rangle &= \min_{\mathbf{p}\in \Delta^X} \left\langle \mathbf{c}, \sum_{\mathbf{x}\in X} p_{\mathbf{x}} \mathbf{x} \right\rangle \\ &= \min_{\mathbf{p}\in \Delta^X} \sum_{\mathbf{x}\in X} p_{\mathbf{x}} \langle \mathbf{c}, \mathbf{x} \rangle \overset{\text{Lem. 2.43}}{=} \min_{\mathbf{x}\in X} \langle \mathbf{c}, \mathbf{x} \rangle \ . \end{aligned} \tag{2.25}$$

The second statement has a similar, but a more bulkier proof:

$$\begin{aligned} \arg\min_{\mathbf{x}\in \operatorname{conv}(X)} \langle \mathbf{c}, \mathbf{x} \rangle &= \arg\min_{\mathbf{x}= \sum_{\mathbf{x}'\in X} p_{\mathbf{x}'}\mathbf{x}' \,,\ \mathbf{p}\in\Delta^X} \langle \mathbf{c}, \mathbf{x} \rangle \\ &= \left\{ \sum_{\mathbf{x}\in X} p^*_{\mathbf{x}} \mathbf{x} \colon \mathbf{p}^* \in \arg\min_{\mathbf{p}\in\Delta^X} \left\langle \mathbf{c}, \sum_{\mathbf{x}\in X} p_{\mathbf{x}} \mathbf{x} \right\rangle \right\} \\ &= \left\{ \sum_{\mathbf{x}\in X} p^*_{\mathbf{x}} \mathbf{x} \colon \mathbf{p}^* \in \arg\min_{\mathbf{p}\in\Delta^X} \sum_{\mathbf{x}\in X} p_{\mathbf{x}} \langle \mathbf{c}, \mathbf{x} \rangle \right\} \\ &\overset{\text{Lem. 2.43}}{=} \left\{ \sum_{\mathbf{x}\in \hat{X}} p^*_{\mathbf{x}} \mathbf{x} \colon \mathbf{p}^* \in \Delta^{\hat{X}}, \hat{X} := \arg\min_{\mathbf{x}\in X} \langle \mathbf{c}, \mathbf{x} \rangle \right\} \\ &= \operatorname{conv}\left(\arg\min_{\mathbf{x}\in X} \langle \mathbf{c}, \mathbf{x} \rangle \right) \end{aligned} \tag{2.26}$$

□

In general, the claim of Theorem 2.50 holds for any $X \subseteq \mathbb{R}^n$, however, for the purpose of this monograph, considering a finite set is sufficient.

Corollary 2.51 (Combinatorial property of linear programs)**.** For any $\mathbf{c} \in \mathbb{R}^n$ and any polytope P it holds that

$$\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x} \rangle = \min_{\mathbf{x}\in \mathrm{vrtx}(P)} \langle \mathbf{c}, \mathbf{x} \rangle \ . \tag{2.27}$$

and

$$\arg\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x} \rangle = \mathrm{conv}\left(\arg\min_{\mathbf{x}\in \mathrm{vrtx}(P)} \langle \mathbf{c}, \mathbf{x} \rangle \right) . \tag{2.28}$$

Proof. The statement follows directly from the fact that

$$P = \mathrm{conv}(\mathrm{vrtx}(P))$$

(Corollary 2.41) and Theorem 2.50. □

Corollary 2.51 claims that to solve a linear program over a polytope it is sufficient to evaluate the objective on the finite set of vertices of the polytope. This does not mean, that it is impossible that the minimum is attained in a non-vertex point of the polytope. However, it guarantees, that there will always be a vertex corresponding to the minimal value of the objective, see Figure 2.4(a). Solutions which correspond to vertices of a polytope P are called *basic solutions* and can be found with the famous simplex algorithm (see references in Section 2.7).

The second statement of Corollary 2.51 implies

Corollary 2.52. A face of a polytope P is a polytope itself. Vertices of the face are vertices of the polytope itself.

Proof. By definition, a face is the set representable as $\arg\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x} \rangle$ for some $\mathbf{c}$. Due to (2.28) it is representable as a convex hull of a finite set of points, therefore, due to Theorem 2.40 it is a polytope. Since these points are, according to (2.28), the vertices of the polytope P, then, according to Corollary 2.41 and Lemma 2.44, these are the vertices of the considered face. □

2.5 Integer linear programs

The integer linear program (ILP) is a unified format to represent combinatorial problems. The main advantage of formulating combinatorial problems as ILPs, is the unified geometrical (polyhedral) interpretation common to all integer linear programs. This has allowed for significant progress in creating standardized methods to solve them. Below, we will provide a definition of ILPs and show their relation to linear programs.

Definition 2.53. A linear program with additional constraints allowing all the variables to take only values 0 or 1, e.g.

$$\min_{\mathbf{x}\in P\cap\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle \tag{2.29}$$

is called a 0/1 *integer linear program* (further we omit mentioning '0/1' for brevity). Here P is a polyhedron in $\mathbb{R}^n$. Constraints of the form $\mathbf{x} \in \{0,1\}^n$ are called *integrality constraints*.

The class of integer linear programs is $\mathcal{NP}$-hard, as is shown by the following two examples, where we represent two known $\mathcal{NP}$-hard problems as ILPs:

Example 2.54 (Binary knapsack problem)**.** Let $\{1,\dots,n\}$ be indexes of n items, each of which has its volume w_i and its cost c_i. Therefore, we assume two vectors $\mathbf{w}, \mathbf{c} \in \mathbb{R}^n$ to be given. The task is to select a subset of items to maximize their total cost, whereas their total volume should not exceed a given (knapsack) volume W. This can be equivalently formulated as the integer linear program

$$\begin{aligned} &\max_{\mathbf{x}\in\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle \\ &\text{s.t. } \langle \mathbf{w}, \mathbf{x}\rangle \le W\,. \end{aligned} \tag{2.30}$$

Example 2.55 (Max-weight independent set problem)**.** Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a undirected graph with a *node set* $\mathcal{V}$ and an *edge set* $\mathcal{E} \subseteq \binom{\mathcal{V}}{2}$. An *independent* or *stable set* is the subset of mutually

non-adjacent vertices $\mathcal{V}' \subseteq \mathcal{V}$, *e.g.* for any $i, j \in \mathcal{V}'$ it holds $\{i, j\} \notin \mathcal{E}$. Given the costs c_i, $i \in \mathcal{V}$, the *maximum weight independent set (MWIS)* problem consists in finding an independent set with the maximum total cost.

Introduce a binary variable $x_i \in \{0, 1\}$ for each node $i \in \mathcal{V}$. Its value 1 denotes that the respective node belongs to the maximum-weight independent set. This leads to a ILP formulation of the max-weight independent set problem:

$$\begin{aligned} &\max_{\mathbf{x} \in \{0,1\}^{\mathcal{V}}} \langle \mathbf{c}, \mathbf{x} \rangle \\ &\text{s.t. } x_i + x_j \leq 1\,, \ \{i, j\} \in \mathcal{E}\,. \end{aligned} \tag{2.31}$$

Universality of an ILP representation has been studied in a number of works. We give here one of the early results from [13] related to the general *non-binary* integer programming.

Definition 2.56 ([13])**.** We say that a combinatorial problem

$$\min_{\mathbf{x} \in S \subset \mathbb{Z}^n} f(\mathbf{x}) \tag{2.32}$$

is formulated as an integer programming problem

$$\begin{aligned} &\min_{\substack{\mathbf{x} \in \mathbb{Z}^n \\ \mathbf{y} \in \mathbb{Z}^m}} z(\mathbf{x}, \mathbf{y}) = \langle \mathbf{c}, \mathbf{x} \rangle + \langle \mathbf{c}', \mathbf{y} \rangle \\ &\text{s.t. } A \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix} \leq \mathbf{b}\,, \end{aligned} \tag{2.33}$$

if there exist $m \in \mathbb{N} \cup \{0\}$, $p \in \mathbb{N}$, an $p \times (n + m)$ integer matrix A, vectors $\mathbf{c} \in \mathbb{Z}^n$, $\mathbf{c}' \in \mathbb{Z}^m$ and $b \in \mathbb{Z}^p$ such that

(i) $\mathbf{x}' \in S$ if and only if there exists $\mathbf{y}' \in \mathbb{Z}^m$ satifying

$$A \begin{pmatrix} \mathbf{x}' \\ \mathbf{y}' \end{pmatrix} \leq \mathbf{b}\,, \tag{2.34}$$

(ii) $z(\mathbf{x}', \mathbf{y}') = f(\mathbf{x}')$ for any $(\mathbf{x}', \mathbf{y}')$ that satify (2.34).

Theorem 2.57 ([13])**.** Any combinatorial optimization problem (2.32) can be formulated as an integer programming problem (2.33) if S is finite.

Since S if finite, it is bounded. W.l.o.g. we also can assume the set of all vectors $\mathbf{y}'$ that satisfy (2.34) to be bounded. To this end we assume $\mathbf{y}' \in S' \subset \mathbb{Z}^m$ and S' is such that for each $\mathbf{x}' \in S$ there is (at least one) $\mathbf{y}' \in S'$ that fulfills (2.34).

In turn, any bounded integer program can be reduced to a *binary* integer program by considering the binary representation of non-negative integer variables:

$$x = 1 \cdot b_1 + 2 \cdot b_2 + 4 \cdot b_3 + \dots, \quad b_i \in \{0, 1\} \tag{2.35}$$

Boundness of the initial problem guarantees finitness of the representation and linearity of the problem and the representation imply linearity of the resulting binary integer program.

A variable x that make take both negative and positive values can be represented $x = x^+ - x^-$ with $x^+ \geq 0$, $x^- \geq 0$.

ILP as LP Somewhat surprisingly, any ILP can be represented as a linear program by a specially constructed polytope. Note that the feasible set of the problem (2.29) is finite, therefore one can use Theorem 2.50, which implies

$$\min_{\mathbf{x} \in P \cap \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle = \min_{\mathbf{x} \in \operatorname{conv}(P \cap \{0,1\}^n)} \langle \mathbf{c}, \mathbf{x} \rangle , \tag{2.36}$$

and, since, according to Theorem 2.40, the convex hull of a finite set is a polytope, the right-hand side of (2.36) constitutes a linear program. This fact does not mean polynomial solvability of the right-hand-side of (2.36) and, therefore, of (2.29). This is because the polytope $\operatorname{conv}(P \cap \{0,1\}^n)$ may require exponentially many linear (in)equalities to be specified explicitly, and computational complexity of the LP optimization methods grows as a polynomial of this amount. For instance, consider the following straightforward (and, therefore, not necessarily shortest) polyhedral representation of $\mathbf{y} \in \operatorname{conv}(X)$,

where $X = P \cap \{0,1\}^n$:

$$\mathbf{y} = \sum_{\mathbf{x} \in X} p_{\mathbf{x}} \mathbf{x}\,, \quad \sum_{\mathbf{x} \in X} p_{\mathbf{x}} = 1,\ p_{\mathbf{x}} \geq 0\ \forall \mathbf{x} \in X\,. \tag{2.37}$$

The length of constraints and their number grows linearly with the size of the set $X = P \cap \{0,1\}^n$, which is exponentially large for typical ILPs (see e.g. Examples 2.54 and 2.55). Otherwise such problems would be easily solvable by a direct enumeration of these vectors.

We will call the polytope $\operatorname{conv}(P \cap \{0,1\}^n)$ *the integer hull* of P, although this name has a broader meaning as the convex set of all integer points (not only those having coordinates 0 and 1) in P.

Vertices of the integer hull

Lemma 2.58. Let $X \subseteq \{0,1\}^n$. For any $\mathbf{x} \in \operatorname{conv}(X)$ it holds $0 \leq x_i \leq 1$, that is, $\operatorname{conv}(X) \subseteq [0,1]^n$.

Proof. Due to Proposition 2.34 and Example 2.37 it holds that $\operatorname{conv}(X) \subseteq \operatorname{conv}(\{0,1\}^n) = [0,1]^n$. □

Corollary 2.59. For any $\mathbf{x} \in \operatorname{conv}(P \cap \{0,1\}^n)$ it holds that $0 \leq x_i \leq 1$.

Since $P \cap \{0,1\}^n$ is a finite set, it follows from Theorem 2.42 that $\operatorname{vrtx}(\operatorname{conv}(P \cap \{0,1\}^n)) \subseteq P \cap \{0,1\}^n$. The following proposition states that these sets are equal:

Proposition 2.60. For any $X \subseteq \{0,1\}^n$ it holds that $\operatorname{vrtx}(\operatorname{conv}(X)) = X$.

Proof. Let $\mathbf{z} \in X$. To show that it is a vertex of $\operatorname{conv}(X)$, consider the problem $\max\limits_{\mathbf{x} \in \operatorname{conv}(X)} \langle \mathbf{c}, \mathbf{x} \rangle$ with $c_i = 2z_i - 1$. Repeating the proof of Example 2.21 we conclude that

$$\max_{\mathbf{x} \in \operatorname{conv}(X)} \langle \mathbf{c}, \mathbf{x} \rangle = \max_{\mathbf{x} \in \operatorname{conv}(X)} \left(\sum_{\substack{i=1,\dots,n\,: \\ z_i \neq 0}} x_i - \sum_{\substack{i=1,\dots,n\,: \\ z_i = 0}} x_i \right).$$

Due to Lemma 2.58 it follows that this expression equals to $\sum_{i=1}^{n} z_i$, and is attained in the point $\mathbf{z}$ only. This implies that $\mathbf{z}$ is the vertex of $\operatorname{conv}(X)$. □

Corollary 2.61. $\operatorname{vrtx}(\operatorname{conv}(P \cap \{0,1\}^n)) = P \cap \{0,1\}^n$. In other words, each feasible point of a 0/1 ILP is a vertex of its integer hull [3].

Due to Corollary 2.61 we conclude that for any feasible point of the integer linear program $\min_{\mathbf{x} \in P \cap \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle$ such a vector $\mathbf{c} \in \mathbb{R}^n$ exists, that this point becomes the unique solution of the program, see Figure 2.4(c) for an illustration.

2.6 Linear program relaxation

The simplest approximation of an $\mathcal{NP}$-hard integer linear program by a polynomially solvable linear problem can be constructed by omitting integrality constraints. This approximation, known as *linear program relaxation*, or simply *LP relaxation*, is a very important and powerful tool for obtaining approximate and exact solutions of integer linear programs. Below, we investigate some geometric properties of LP relaxations.

Definition 2.62. The linear program

$$\min_{\mathbf{x} \in P \cap [0,1]^n} \langle \mathbf{c}, \mathbf{x} \rangle \tag{2.38}$$

is called *linear programming (LP) relaxation* of the integer linear program (2.29).

Note that

- the LP relaxation is constructed by substituting the integrality constraints $x_i \in \{0,1\}$ with *interval* or *box* constraints $x_i \in [0,1]$.

[3] This claim does not hold for a more general class of integer linear programs without the 0/1 constraint.

Since $x_i \in [0,1]$ is equivalent to $0 \leq x_i \leq 1$, these constraints define a polyhedron. As the intersection of two polyhedra is a polyhedron (see Proposition 2.19), the problem (2.38) is a linear program;

- the LP relaxation is a relaxation in terms of Definition 2.6, since its feasible set is a superset of the one for the non-relaxed problem and their objectives coincide.

The most important properties of LP relaxations read:

Proposition 2.63. Let f^* and $\hat{f}$ be optimal values of the ILP problem (2.29) and its LP relaxation (2.38) respectively. Let also $\hat{\mathbf{x}}$ be a solution of the LP relaxation. Then the following assertions hold:

1. $\hat{f} \leq f^*$.
2. If $\hat{\mathbf{x}} \in \{0,1\}^n$, then $\hat{\mathbf{x}}$ is a solution of the non-relaxed ILP problem (2.29).
3. $P \cap [0,1]^n \supseteq \operatorname{conv}(P \cap \{0,1\}^n)$, that is, the feasible set of the LP relaxation is a superset of the integer hull.

Proof. The first two statements follow directly from Proposition 2.9. The third one follows from the fact that $\operatorname{conv}(P \cap [0,1]^n) = P \cap [0,1]^n$ (since P and $[0,1]^n$ are convex, their intersection is a convex set), $P \cap [0,1]^n \supseteq P \cap \{0,1\}^n$, and Proposition 2.34. □

2.6.1 Vertices of the feasible set of the LP relaxation

While the statement (2) of Proposition 2.63 claims that integrality of a relaxed solution implies its optimality for the non-relaxed ILP problem, it does not say, whether there has to be a solution to the LP relaxation which is integral. Or in other words, whether any solution of the ILP problem can be obtained as an optimal solution of its LP relaxation. The following proposition fills this gap:

Proposition 2.64. For any polyhedron P it holds that $P \cap \{0,1\}^n \subseteq \operatorname{vrtx}(P \cap [0,1]^n)$.

In other words, Proposition 2.64 states that all feasible vectors of an ILP are vertices of the feasible set of its LP relaxation and, therefore, can be reached as a relaxed solution.

Proof. Let us consider the intersection of the polyhedron P and the polytope $[0,1]^n$. According to Proposition 2.48, vertices of $[0,1]^n$ belonging to P are also vertices of $P \cap [0,1]^n$. As shown in Example 2.46, any vertex of the n-dimensional cube $[0,1]^n$ is a vector from the set $\{0,1\}^n$ and vice versa. Therefore, any point from $P \cap \{0,1\}^n$ is a vertex of $P \cap [0,1]^n$. □

The following corollary additionally relates vertices of the integer hull and of the feasible set of the LP relaxation (see Figure 2.4(c)):

Corollary 2.65. $\text{vrtx}(\text{conv}(P \cap \{0,1\}^n)) \subseteq \text{vrtx}(P \cap [0,1]^n)$.

Proof. The statement follows from $\text{vrtx}(\text{conv}\{P \cap \{0,1\}^n\}) = P \cap \{0,1\}^n \subseteq \text{vrtx}(P \cap [0,1]^n)$, where the first relation follows from Corollary 2.61 and the second holds due to Proposition 2.64. □

The last proposition of the chapter states that there is no other binary vector in the feasible set of LP relaxation than those, which are feasible for the non-relaxed integer linear program:

Proposition 2.66. For any polyhedron $P \subset \mathbb{R}^n$ it holds that $\text{vrtx}(P \cap [0,1]^n) \cap \{0,1\}^n = P \cap \{0,1\}^n$.

Proof. Let $\mathbf{x} \in P \cap \{0,1\}^n$. Trivially $\mathbf{x} \in \{0,1\}^n$. According to Proposition 2.64 $\mathbf{x} \in \text{vrtx}(P \cap [0,1]^n)$. Therefore, $P \cap \{0,1\}^n \subseteq \text{vrtx}(P \cap [0,1]^n) \cap \{0,1\}^n$.

The other way around, let $\mathbf{x} \in \text{vrtx}(P \cap [0,1]^n) \cap \{0,1\}^n$. Trivially, $\mathbf{x} \in \text{vrtx}(P \cap [0,1]^n)$ and $\mathbf{x} \in \{0,1\}^n$. Since $\text{vrtx}(P \cap [0,1]^n) \subseteq P \cap [0,1]^n$, it holds $\mathbf{x} \in P$. Therefore, $\mathbf{x} \in P \cap \{0,1\}^n$. □

2.7 Bibliography and further reading

One of the best introductory text-books about linear programming and the simplex method is written by the author of the simplex method, Georg Dantzig [14]. For fundamental results on polyhedra, linear inequalities and linear programming the monograph [15] is a classical reference. It includes also fundamental results on the polynomial solvability of linear programs. We refer to the text-book [6] for the Minkowski theorem and its corollary. This text-book as well as [16] can be recommended for learning the basics of combinatorial optimization and its relation to (integer) linear programming. The Caratheodory's theorem is a fundamental fact in convex analysis, its proof can be found in any textbook, *e.g.*, [17], on the topic.

3 Linearization: From quadratic to linear integer objective

In this chapter we consider linearly constrained integer programs with quadratic objectives such as *labeling* and *quadratic assignment*. Our main goal is to turn them into *linear* integer programs in the most efficient way, where efficiency means the relation between the number of (additional) constraints and tightness of the resulting LP relaxation.

The problems we will deal with have the form

$$\min_{\mathbf{x}\in P\cap\{0,1\}^n} \sum_{i=1}^{n} c_i x_i + \sum_{\{i,j\}\in\mathcal{E}} c_{\{ij\}} x_i x_j \,, \tag{3.1}$$

where the polytope P is defined by a set of linear constraints, $\mathcal{E} \subseteq \binom{[n]}{2}$ and the brackets "$\{\cdot\}$" in $y_{\{ij\}}$ denote that $\{ij\}$ and $\{ji\}$ are the same. Respectively $c_{\{ij\}}$ and $c_{\{ji\}}$ are the same as well.

A general approach to transformation of this problem to ILP format consists of two steps:

- Introduce a new variable $y_{\{ij\}}$ for each $\{i,j\} \in \mathcal{E}$.
- Add constraints

  $$y_{\{ij\}} = x_i x_j, \ \{i,j\} \in \mathcal{E} \tag{3.2}$$

  to the feasible set.

This turns the problem (3.1) into

$$\min_{\substack{\mathbf{x}\in P\cap\{0,1\}^n, \mathbf{y}\in\{0,1\}^{\mathcal{E}} \\ y_{\{ij\}}=x_i x_j, \ ij\in\mathcal{E}}} \sum_{i=1}^{n} c_i x_i + \sum_{\{i,j\}\in\mathcal{E}} c_{\{ij\}} y_{\{ij\}} \,. \tag{3.3}$$

The above process of variable substitution $y_{\{ij\}} = x_i x_j$ is often referred to as *variable lifting* in the literature. Respectively, variables $y_{\{ij\}}$ are called *lifted* and the resulting integer program (3.3) a *lifted* problem.

Although the objective is now linear, the constraints are not anymore. So the main question we deal with in this chapter is how to linearize constraints (3.2) given that $\mathbf{x}$ and $\mathbf{y}$ are binary vectors.

3.1 Fortet's linearization

The first linearization method proposed by Fortet in [18] is the substitution of (3.2) by

$$\begin{aligned} y_{\{ij\}} &\leq x_i \,, \\ y_{\{ij\}} &\leq x_j \,, \\ y_{\{ij\}} &\geq x_i + x_j - 1 \,. \end{aligned} \tag{3.4}$$

for all $\{i, j\} \in \mathcal{E}$.

It is easy to see that for any binary $\mathbf{x}$ and $\mathbf{y}$ constraints (3.4) is equivalent to (3.2).

This adds $3|\mathcal{E}|$ additional non-trivial constraints (in addition to the integrality constraints $\mathbf{y} \in \{0,1\}^{\mathcal{E}}$ and/or, the non-negativity constraints $\mathbf{y} \geq 0$ important for the LP relaxation).

Being very simple this linearization method, is known for its main weakness: The resulting LP relaxation is typically not particularly tight. Therefore, several other linearizations have been proposed. We consider the one known to be tighter than most of others in the following section.

Example 3.1. Consider the problem

$$\begin{aligned} \min_{\mathbf{x}\in\{0,1\}^2} \; & c_1 x_1 + c_2 x_2 + c_{12} x_1 x_2 \\ \text{s.t.} \; & x_1 + x_2 = 1 \,. \end{aligned} \tag{3.5}$$

Due to the simplex constraint the quadratic term is always zero for any integer solution.

Let us now consider linearization of (3.5) according to (3.4):

$$\begin{aligned} \min_{\substack{\mathbf{x}\in\{0,1\}^2 \\ y\in\{0,1\}}} \; & c_1x_1 + c_2x_2 + c_{12}y \\ \text{s.t. } & x_1 + x_2 = 1 \\ & y \leq x_1\,, \\ & y \leq x_2\,, \\ & y \geq x_1 + x_2 - 1\,. \end{aligned} \tag{3.6}$$

Consider now the LP relaxation of (3.6):

$$\begin{aligned} \min_{\substack{\mathbf{x}\in[0,1]^2 \\ y\in[0,1]}} \; & c_1x_1 + c_2x_2 + c_{12}y \\ \text{s.t. } & x_1 + x_2 = 1 \\ & y \leq x_1\,, \\ & y \leq x_2\,, \\ & \cancel{y \geq 0}\,, \end{aligned} \tag{3.7}$$

where we also simplified the last constraint given the first one. Since $y \in [0, 1]$ anyway, we can ignore this constraint line further.

Now, the value of y is constrained by the values of x_1 and x_2 and given the first constraint in (3.7), $y \leq 1/2$ and y attains its maximum value if $x_1 = x_2 = 1/2$. This implies, that if c_{12} is negative and large compared to c_1 and c_2, the solution of the LP relaxation will be $(1/2, 1/2, 1/2)$.

We will get back to this result in the next section.

3.2 Sherali-Adams linearization

Is there a generic mechanism of building valid constraints for $y = x_ix_j$ given the constraints on $\mathbf{x}$?

The answer is given by Sherali and Adams in [19]. Consider the following general idea: If $\langle \mathbf{a}, \mathbf{x} \rangle \leq b$ is a valid constraint, than $x_j(\langle \mathbf{a}, \mathbf{x} \rangle \leq b)$ is a valid constraint also, since $x_j \in \{0, 1\} \geq 0$. Therefore,

$$x_j(\langle \mathbf{a}, \mathbf{x} \rangle \leq b) \Leftrightarrow \sum_{i=1}^{n} a_i \underbrace{x_i x_j}_{y_{\{ij\}}} \leq b x_j \tag{3.8}$$

is a valid constraint too. Since $(1 - x_j) \in \{0, 1\} \geq 0$ as well, feasibility holds also for

$$(1 - x_j)\left(\langle \mathbf{a}, \mathbf{x} \rangle \leq b\right) \Leftrightarrow \langle \mathbf{a}, \mathbf{x} \rangle - \sum_{i=1}^{n} a_i \underbrace{x_i x_j}_{y_{\{ij\}}} \leq b - b x_j \,. \tag{3.9}$$

Note that (3.8) and (3.9) are both valid linear constraints that include the new lifted variable $\mathbf{y}$.

Example 3.2. Consider constraints

$$x_i \geq 0, \quad x_i \leq 1, \ i \in [n] \tag{3.10}$$

After multiplication of them by x_j one obtains

$$x_i x_j \geq 0, \quad x_i x_j \leq x_j, \ i \in [n] \tag{3.11}$$

and after multiplication by $(1 - x_j)$

$$x_i - x_i x_j \geq 0, \quad x_i - x_i x_j \leq 1 - x_j, \ i \in [n] \tag{3.12}$$

Easy to see, that after substitution $y_{\{ij\}} = x_i x_j$ Equations (3.11) and (3.12) result in the Fortet's linearization constraints (3.4) plus the trivial constraint $y_{\{ij\}} \geq 0$.

In other words, Sherali-Adams linearization contains Fortet's linearization "automatically", since w.l.o.g. one can assume that the box constraints $x \in [0, 1]$ are fulfilled for all integer variables.

And what about equality constraints $\langle \mathbf{a}, \mathbf{x} \rangle = b$? How should they be processed? It is easy to see that multiplication of this constraints

by x_j and $(1 - x_j)$ is equivalent:

$$(1 - x_j)(\langle \mathbf{a}, \mathbf{x} \rangle = b) \Leftrightarrow \langle \mathbf{a}, \mathbf{x} \rangle - \sum_{i=1}^{n} a_i \underbrace{x_i x_j}_{y_{\{ij\}}} = b - b x_j \,. \tag{3.13}$$

However, since $\langle \mathbf{a}, \mathbf{x} \rangle = b$, it simplifies to

$$x_j(\langle \mathbf{a}, \mathbf{x} \rangle = b) \Leftrightarrow \sum_{i=1}^{n} a_i \underbrace{x_i x_j}_{y_{\{ij\}}} = b x_j \,. \tag{3.14}$$

Therefore, in the equality case it is sufficient to multiply by x_j for all j.

3.2.1 General Sherali-Adams linearization scheme

This section summarizes the observations above. Consider the following general zero-one quadratic programming problem with linear constraints:

$$\min_{\mathbf{x} \in P \cap \{0,1\}^n} \sum_{i=1}^{n} c_i x_i + \sum_{\{i,j\} \in \mathcal{E}} c_{ij} x_i x_j \,, \tag{3.15}$$

where $\mathcal{E} \subseteq \{(i,j) \colon 1 \leq i < j \leq n\}$ and $P := P_1 \cup P_2 \cup P_3$ is given by:

$$P_1 = \{x \in \mathbb{R}^n \colon \sum_{i=1}^{n} a_{ki} x_i = b_k \text{ for } k \in [K]\} \,, \tag{3.16}$$

$$P_2 = \{x \in \mathbb{R}^n \colon \sum_{i=1}^{n} h_{li} x_i \leq g_l \text{ for } l \in [L]\} \,, \tag{3.17}$$

$$P_3 = \{x \in \mathbb{R}^n \colon 0 \leq x_i \leq 1 \text{ for } i \in [n]\} \,. \tag{3.18}$$

Perform now the following set of operations:

1. Form nK constraints $x_j(\sum_{i=1}^{n} a_{ki} x_i = b_k)$, $k \in [K]$, $j \in [n]$;
2. Form nL constraints $x_j(\sum_{i=1}^{n} h_{li} x_i \leq g_l)$, $l \in [L]$, $j \in [n]$;

3. Form nL constraints $(1 - x_j)(\sum_{i=1}^{n} h_{li}x_i \leq g_l)$, $l \in [L]$, $j \in [n]$;

4. Form $3n(n-1)/2$ constraints
   a) $(1 - x_j)(x_i \geq 0)$, $i \in [n]$, $j \in [n]\backslash\{i\}$,
   b) $(1 - x_j)(x_i \leq 1)$, $i \in [n-1]$, $j = i+1, \ldots, n$.

Then substitute all appearances of $x_i x_j$ with $y_{\{ij\}}$ and x_i^2 with x_i as $x^2 = x$ for any $x \in \{0, 1\}$.
This results in:

$$\min_{\mathbf{x}\in\{0,1\}^n} \sum_{i=1}^{n} c_i x_i + \sum_{\{i,j\}\in\mathcal{E}} c_{ij} y_{\{ij\}} \tag{3.19}$$

$$\text{s.t. } (a_{kj} - b_k)x_j + \sum_{i\in[n]\backslash\{j\}} a_{ki} y_{\{ij\}} = 0\,, \quad \forall (j,k) \in [n] \times [K] \tag{3.20}$$

$$\sum_{i=1}^{n} h_{li}x_i - g_l \leq \sum_{i\in[n]\backslash\{j\}} h_{li} y_{\{ij\}} + (h_{lj} - g_l)x_j \leq 0\,, \quad \forall (j,l) \in [n] \times [L] \tag{3.21}$$

$$y_{\{ij\}} \geq 0\,, \quad \forall \{i,j\} \in \binom{[n]}{2} \tag{3.22}$$

$$x_i - y_{\{ij\}} \geq 0\,, \quad \forall \{i,j\} \in \binom{[n]}{2} \tag{3.23}$$

$$y_{\{ij\}} - x_i - x_j \geq -1\,, \quad \forall \{i,j\} \in \binom{[n]}{2} \tag{3.24}$$

$$\mathbf{x} \in P_1\,, \quad \mathbf{x} \in P_3 \tag{3.25}$$

Note that:

- The initial constraint $\mathbf{x} \in P_2$ is not included explicitly as it is implied by (3.21);
- If the constraints of P_1 and P_2 together imply those in P_3, the Fortet's constraints (3.23)-(3.24) (as well as $\mathbf{x} \in P_3$) can be excluded as they are implied by (3.20)-(3.21).
- Other simplifications are often possible. E.g. if P_1 contains a *simplex* (or *uniqueness* or *set partitioning*) constraint $x_1 + \cdots +$

$x_m = 1$ then $y_{\{ij\}} = 0$ for all $i, j \in [m]$, $i \neq j$, see Example 3.3 below.

- Contrary to the Fortet linearization that introduces new variables $y_{\{ij\}}$ only for $\{i, j\} \in \mathcal{E}$ the Sherali-Adams method introduces them for all $\binom{[n]}{2}$. In case of sparse quadratic terms, *i.e.*, $|\mathcal{E}| \ll |\binom{[n]}{2}|$ this significantly increases the size of the initial problem. In some cases the unnecessary variables and constraints can be ignored without influencing the resulting LP relaxation, see, *e.g.*, Section 3.3. In general, one may consider less tight relaxation by defining only a subset of quadratic variables $y_{\{ij\}}$ and the respective constraints.

Example 3.3. Now let us get back to Example 3.1. Multiplication of the simplex constraint $x_1 + x_2 = 1$ by x_1 results in $x_1^2 + x_1 x_2 = x_1 \Rightarrow x_1 + y = x_1 \Rightarrow y = 0$. The same result is obtained with multiplication by x_2.

Hence the result of linearization of (3.5) reads now

$$\min_{\mathbf{x} \in \{0,1\}^2} c_1 x_1 + c_2 x_2 + c_{12} y \tag{3.26}$$

$$\text{s.t. } x_1 + x_2 = 1\,,$$

$$y = 0\,, \tag{3.27}$$

or, after getting rid of y:

$$\min_{\mathbf{x} \in \{0,1\}^2} c_1 x_1 + c_2 x_2 \tag{3.28}$$

$$\text{s.t. } x_1 + x_2 = 1\,.$$

Note that LP relaxation of this problem is tight, contrary to the relaxation (3.7) with the Fortet's linearization.

Example 3.4 (Uniqueness/simplex constraints)**.** Let $\mathbf{x} \in \mathbb{R}^n$. Consider the *uniqueness* constraint

$$\sum_{i \in I \subseteq [n]} x_i = 1\,. \tag{3.29}$$

After its multiplication by x_j, $j \in I$, we obtain

$$\cancel{x_j^2} + \sum_{i \in I \setminus \{j\}} x_i x_j = \cancel{x_j} \tag{3.30}$$

$$\sum_{i \in I \setminus \{j\}} x_i x_j = 0 \tag{3.31}$$

$$0 \leq y_{\{ij\}} := x_i x_j \;\Rightarrow\; y_{\{ij\}} = 0\,. \tag{3.32}$$

If we multiply the constraint (3.29) by x_j, $j \notin I$ this results in

$$\sum_{i \in I} y_{\{ij\}} = x_j\,. \tag{3.33}$$

The latter constraint is often called *marginalization* or *coupling constraint*.

3.3 Case study: Labeling problem

Consider the labeling problem as defined in (1.18):

$$\min_{\mathbf{y} \in \mathcal{Y}_\mathcal{V}} \sum_{u \in \mathcal{V}} \theta_u(y_u) + \sum_{uv \in \mathcal{E}} \theta_{uv}(y_u, y_v) \tag{3.34}$$

Let $\mathcal{I}_1 := \cup_{u \in \mathcal{V}} \mathcal{Y}_u$ be the set of *all labels in all nodes* of the labeling problem. Introduce a binary vector

$$\mathbb{R}^{\mathcal{I}_1} \ni \boldsymbol{\mu} = (\mu_u(s) \in \{0,1\} \colon u \in \mathcal{V}, s \in \mathcal{Y}_u) \tag{3.35}$$

whose coordinates correspond to labels in different nodes. As usual, the value 1 will denote selection of the respective label.

With this notation the labeling problem (3.34) can be rewritten as a *quadratic integer problem*

$$\min_{\boldsymbol{\mu} \in \{0,1\}^{\mathcal{I}_1}} \sum_{u \in \mathcal{V}} \sum_{s \in \mathcal{Y}_u} \theta_u(s)\mu_u(s) + \sum_{uv \in \mathcal{E}} \sum_{sl \in \mathcal{Y}_{uv}} \theta_{uv}(s,l)\mu_u(s)\mu_v(l) \tag{3.36}$$

$$\text{s.t.} \sum_{s \in \mathcal{Y}_u} \mu_u(s) = 1\,,\;\; u \in \mathcal{V}\,. \tag{3.37}$$

Let us now linearize it with the Sherali-Adams method.

- First of all note that the *uniqueness* (known also as *simplex*) *constraints* (3.37) together with non-negativity of μ imply $\mu \in [0,1]^{\mathcal{I}_1}$. Therefore, we can omit the Fortet's constraints (3.23)-(3.24) as they will be implied by the other constraints of the linearized problem.

- Since the only non-trivial constraints (3.37) are *uniqueness constraints*, according to Example 3.4:
  - $\mu_u(s)\mu_u(l) = 0$ for any $s \neq l$ for all $u \in \mathcal{V}$.
  - By introducing $\mu_{uv}(s,l) := \mu_u(s)\mu_v(l)$ the uniqueness constraints are turned into the marginalization constraint

$$\sum_{s \in \mathcal{Y}_u} \mu_{uv}(s,l) = \mu_v(l)\,,\ u \in \mathcal{V} \tag{3.38}$$

    after multiplication by $\mu_v(l)$, $v \in \mathcal{V}$, $v \neq u$, $l \in \mathcal{Y}_v$. Here and further we assume that $\mu_{uv}(s,l)$ is the same variable as $\mu_{vu}(l,s)$, just like $y_{\{ij\}}$ is the same as $y_{\{ji\}}$ in the general case.

- Note, however, that the only non-zero costs in the original problem (3.36) are given for $\mu_{uv}(\cdot)$ for $uv \in \mathcal{E}$. Therefore, the values of $\mu_{uv} := (\mu_{uv}(s,l) \colon s \in \mathcal{Y}_u,\ l \in \mathcal{Y}_v)$ for $uv \notin \mathcal{E}$ do not influence the objective value. But does their existence additionally constraints other variables, be it binary or relaxed ones? The following statement gives the negative answer to this question:

  Lemma 3.5. The sets

$$P_x = \{\mathbf{x} \in \mathbb{R}^{I \cup J}_{\geq 0} \colon \sum_{i \in I} x_i = 1\,,\ \sum_{j \in J} x_j = 1\} \tag{3.39}$$

  and

$$P_y = \left\{ \mathbf{x} \in P_x \mid \exists \mathbf{y} \in \mathbb{R}^{I \times J}_{\geq 0} \colon \begin{array}{l} \sum\limits_{i \in I} y_{ij} = x_j\,,\ \forall j \in J\,, \\ \sum\limits_{j \in J} y_{ij} = x_i\,,\ \forall i \in I \end{array} \right\} \tag{3.40}$$

are equal, *i.e.*, $P_x = P_y$. Moreover, exchanging $\mathbf{x} \in \mathbb{R}_{\geq 0}^{I \cup J}$ with $\mathbf{x} \in \{0,1\}^{I \cup J}$ in (3.39) implies that $\mathbf{y} \in \mathbb{R}_{\geq 0}^{I \times J}$ can be exchanged with $\mathbf{y} \in \{0,1\}^{I \times J}$ in (3.40).

Proof. It is sufficient to prove that for any $\mathbf{x} \in P_x$ there exists $\mathbf{y} \in \mathbb{R}_{\geq 0}^{I \times J}$ such that the additional two constrains in the definition of P_y are fulfilled. Indeed, consider $y_{ij} = x_i x_j$. Then

$$\sum_{i \in I} y_{ij} = \sum_{i \in I} x_i x_j = x_j \sum_{i \in I} x_i \overset{\mathbf{x} \in P_x \Rightarrow \sum_{i \in I} x_i = 1}{=} x_j \,, \tag{3.41}$$

$$\sum_{j \in J} y_{ij} = \sum_{j \in J} x_i x_j = x_i \sum_{j \in J} x_j \overset{\mathbf{x} \in P_x \Rightarrow \sum_{j \in J} x_j = 1}{=} x_i \,. \tag{3.42}$$

The additional constraint $\mathbf{x} \in \{0,1\}^{I \cup J}$ implies $y_{ij} = x_i x_j \in \{0,1\}$. □

- Putting all together for the labeling problem, we obtain the following linear constraints for the linearized (lifted) labeling problem:

$$\mathcal{L} := \begin{cases} \sum_{l \in \mathcal{Y}_v} \mu_{uv}(s,l) = \mu_u(s), & u \in \mathcal{V},\ v \in \mathcal{N}(u),\ s \in \mathcal{Y}_u \quad \text{(a)} \\ \sum_{l \in \mathcal{Y}_v} \mu_v(l) = 1, & v \in \mathcal{V} \quad \text{(b)} \\ \boldsymbol{\mu} \geq 0\,, & \quad \text{(c)} \end{cases} \tag{3.43}$$

The set $\mathcal{L}$ is known as *local marginal polytope*. For bravity, we will omit the word *marginal* when referring to $\mathcal{L}$ further on.

Introducing the set $\mathcal{I}_2 = \bigcup_{uv \in \mathcal{E}} |\mathcal{Y}_{uv}|$ of the lifted, or *pairwise* variables and denoting $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2$, we obtain the resulting *(lifted) linearized labeling problem*:

$$\min_{\boldsymbol{\mu} \in \mathcal{L} \cap \{0,1\}^{\mathcal{I}}} \underbrace{\sum_{u \in \mathcal{V}} \sum_{s \in \mathcal{Y}_u} \theta_u(s) \mu_u(s) + \sum_{uv \in \mathcal{E}} \sum_{sl \in \mathcal{Y}_{uv}} \theta_{uv}(s,l) \mu_{uv}(s,l)}_{\langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle} \,. \tag{3.44}$$

Note that $\mathcal{L} \cap [0,1]^{\mathcal{I}} = \mathcal{L}$, so the LP relaxation of the linearized problem reads

$$\min_{\boldsymbol{\mu} \in \mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle \ . \tag{3.45}$$

Naturally, this relaxation referred to as the *local polytope relaxation* of the labeling problem.

We will also use the notation $\delta(\mathbf{y})$ for the binary vector corresponding to a labeling $\mathbf{y}$, *i.e.*, $\delta_u(s) = [\![y_u = s]\!]$ and $\delta_{uv}(s,t) = [\![(s,t) = (y_u, y_v)]\!]$.

Example 3.6 (How large-scale is the ILP representation of the labeling problem?)**.** Consider the depth reconstruction Example 1.30 from Chapter 1. Each element of the nodes set $\mathcal{V}$ corresponds to a pixel in an input image. Let the latter be as small as $100 \times 100 = 10^4$. Let also each node be associated with 10 labels only. It implies, there are in total 10^{10^4} labeling.

To estimate the number of variables in the ILP representation (3.44) we count for $10 \times 10^4 = 10^5$ total number of labels (corresponding to the variables $\mu_u(s)$).

The number of edges in the grid graph is $\approx 2 \times 10^4$ and each edge is associated with 10^2 label pairs (and the corresponding variables $\mu_{uv}(s,l)$). Therefore, the total number of label pairs is $\approx 2 \times 10^4 \times 10^2 = 2 \times 10^6$.

This results in $O(10^6)$ variables in th ILP representation (3.44).

As it follows from the definition of the local polytope (3.43), the number of equality constraints grows linearly with the total number of labels and therefore is of order $O(10^5)$. However, since there is the non-negativity constraint $\mu_w(s) \geq 0$ for each variable, the total number of constraints has the same order $O(10^6)$ as the number of variables.

The millions of constraints and variables turn the labeling problem (3.44) to the class of *large-scale optimization* problems. Large size of the problem significantly limits the set of methods, which can be applied to tackle it.

Universality of local polytope The relaxed labeling problem $\min_{\boldsymbol{\mu}\in\mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$ defines a special class of linear programs, therefore, it can be solved with general-purpose LP solvers in polynomial time. A natural question is whether this class of problems is specific enough to allow for specialized LP solvers able to solve it faster even in the worst-case.

Unfortunately, the answer to this question is negative, as stated by the following theorem:

Theorem 3.7 (Universality of the local polytope [20])**.** *Any* linear program reduces to the local polytope relaxation of a certain labeling problem with 3 labels in linear time.

In other words, there are instances of the labeling problem, which are as difficult as the most difficult linear programs. Therefore, the known complexity estimates for linear programs apply to the local polytope relaxation.

However, problems emerging in numerous applications, often do not represent the worst-case scenario and can be approximately or even exactly solved with efficient polynomial algorithms.

3.3.1 ILP properties of the labeling problem

Due to the ILP representation (3.44) of the labeling problem, we can directly make conclusions about its own properties and the properties of its natural LP relaxation:

- The integer hull of the set of feasible binary vectors $\mathcal{L} \cap \{0,1\}^{\mathcal{I}}$ is $\mathcal{M} := \operatorname{conv}(\mathcal{L} \cap \{0,1\}^{\mathcal{I}})$ and is called *marginal polytope*. As it follows from Corollary 2.61 each vertex of the marginal polytope corresponds to a labeling, and the other way around, each labeling corresponds to a vertex of $\mathcal{M}$.

- The *local polytope relaxation* (3.45) (as any other relaxation) constitutes a lower bound, i.e. $E(\mathbf{y};\boldsymbol{\theta}) \geq \min_{\boldsymbol{\mu}\in\mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$ for any (in particular the optimal) labeling $\mathbf{y}$. Cases when this bound is tight, i.e. $\min_{\mathbf{y}\in\mathcal{Y}_{\mathcal{V}}} E(\mathbf{y};\boldsymbol{\theta}) = \min_{\boldsymbol{\mu}\in\mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$, are called *LP-tight*.

- The marginal polytope is a subset of the local polytope, i.e. $\mathcal{M} \subseteq \mathcal{L}$. According to Corollary 2.65, moreover, $\text{vrtx}(\mathcal{M}) \subseteq \text{vrtx}(\mathcal{L})$ holds, that is, all vertices of the marginal polytope are also vertices of the local polytope.

Exercise 3.8. Consider Fortet's linearization of the labeling problem. Show explicitly that the respective LP relaxation is less tight than the local polytope relaxation. To this end show that the respective feasible set containts local polytope as its subset.

3.4 Bibliography and further reading

A detailed description of all facets of the reformulation-linearization technique by Sherali and Adams can be found in their excellently written book [21].

4 Convex functions and their basic properties

Fis section introduces the notion of a convex optimization problem. This type of problems plays a central role in optimization as a whole, since there are good reasons to treat these problems as efficiently solvable. Most of the methods we consider below are based on approximating a difficult non-convex problem by a much easier convex one and then solving the latter.

One such approximation technique known as *Lagrange duality* and its application to integer linear programs are considered in the next chapter.

4.1 Convex optimization problems

4.1.1 Extended value functions

So far, when speaking about the optimization problem

$$\min_{\mathbf{x}\in X\subseteq\mathbb{R}^n} f(\mathbf{x})\,, \tag{4.1}$$

we assumed that the value $f(\mathbf{x})$ is finite for all $\mathbf{x} \in X$. But this may not always hold. Consider for example, that $f(\mathbf{x})$ is defined implicitly, as the result of another optimization, e.g. $f(\mathbf{x}) = \max_{z\in Z} g(\mathbf{x}, z)$ with $g\colon X \times Z \to \mathbb{R}$. The maximization problem may turn out to be unbounded for some $\mathbf{x}$. Loosely speaking, this would mean that $f(\mathbf{x}) = \infty$.

Example 4.1 (Implicitly defined extended value function)**.** Consider the following extended value function, defined implicitly:

$$f(\mathbf{x}) = \max_{z \in \mathbb{R}} (x_1 z + x_2) \ . \tag{4.2}$$

It is easy to see that $f(\mathbf{x}) = \begin{cases} x_2, & x_1 = 0 \\ \infty, & x_1 \neq 0 \, . \end{cases}$. Therefore, f is extended value, although $x_1 z + x_2$ is defined for all $z, \mathbf{x}$ and z is an unconstrained variable.

Let us now take a closer look at the minimization problem

$$\min_{\mathbf{x} \in X} f(\mathbf{x})$$

when f is extended value:

- Let now X' be the maximal subset of X such that the value $f(\mathbf{x})$ is finite for all $\mathbf{x} \in X'$. Then the optimal value of (4.1) is finite as well and can be attained in some $\mathbf{x} \in X'$ only.
- If $f(\mathbf{x})$ is unbounded from above for all $\mathbf{x} \in X$, this implies that its minimal value is unbounded as well. In other words, $\min_{\mathbf{x} \in X} f(\mathbf{x}) = \infty$. This is the same as in the situation, where problem (4.1) is infeasible, since there is no element $\mathbf{x} \in X$ such that $f(\mathbf{x})$ is finite.

With these considerations in mind it makes sense to assume the following definition.

Definition 4.2 (Extended value function)**.** A mapping of the form $f \colon X \to \mathbb{R} \cup \{\infty, -\infty\}$ is called an *extended value function*. The set $\{\mathbf{x} \in X \mid -\infty < f(\mathbf{x}) < \infty\}$ is called *domain* of f and is denoted by $\mathrm{dom} f$.

Let f be an extended value function. The constrained minimization problem $\min_{\mathbf{x} \in X} f(\mathbf{x})$ is defined similarly as in Chapter 2 with the following modifications:

- The feasible set is defined as $\{\mathbf{x} \in X \mid f(\mathbf{x}) < \infty\}$.
- The problem is called unbounded if $\inf_{\mathbf{x}\in X} f(\mathbf{x}) = -\infty$. In particular, it is sufficient that there exists an $\mathbf{x} \in X$ such that $f(\mathbf{x}) = -\infty$ for the problem to be unbounded.

The maximization of extended value functions is treated similarly by substituting min with max, inf with sup, and $-\infty$ with ∞ and the other way around.

Most importantly, with the extended-valued functions one can "integrate" constraints into the objective function. Let $X \subset \mathbb{R}^n$ and let $\iota_X \colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be the extended value *indicator* function

$$\iota_X(\mathbf{x}) = \begin{cases} 0, & \mathbf{x} \in X \\ \infty, & \mathbf{x} \notin X\,. \end{cases} \tag{4.3}$$

Unless f is equal to $-\infty$ for some $\mathbf{x} \in X$, the optimization problem $\min_{\mathbf{x}\in X} f(\mathbf{x})$ can be equivalently written as $\min_{\mathbf{x}\in\mathbb{R}^n} (f(\mathbf{x}) + \iota_X(\mathbf{x}))$. Similarly, $\max_{\mathbf{x}\in X} f(\mathbf{x}) = \max_{\mathbf{x}\in\mathbb{R}^n} (f(\mathbf{x}) - \iota_X(\mathbf{x}))$.

In the following, when speaking about a function, we will assume an *extended value* function, if the opposite is not stated.

4.1.2 Convex functions

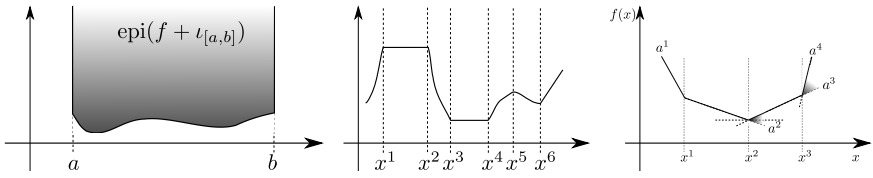
Definition 4.3. An *epigraph* of a function f is defined as the set $\mathrm{epi} f := \{(\mathbf{x}, t) \colon \mathbf{x} \in \mathrm{dom} f \cup \{\mathbf{x} \colon f(\mathbf{x}) = -\infty\},\ t \geq f(\mathbf{x})\}$.

Loosely speaking, epigraph is set of points "above" the plot of a function, as illustrated in Figure 4.1(a).

Example 4.4. The epigraph of a linear function $f \colon \mathbb{R}^n \to \mathbb{R}$ is a half-space in $\mathbb{R}^{n+1}$.

The following proposition simplifies the analysis of a large class of functions:

Proposition 4.5. For any two functions f and g it holds that $\mathrm{epi}\,(\max\{f, g\}) = \mathrm{epi}(f) \cap \mathrm{epi}(g)$.



(a) (b) (c)

Figure 4.1: **(a)** Epigraph of the function $f + \iota_{[a,b]}$; **(b)** Illustration of local and global minima (maxima) of a function. All points in the closed interval $[x^1, x^2]$ are local maxima, points in the open interval (x^1, x^2) are also local minima. Similarly, points $[x^3, x^4]$ are local and global minima, in (x^3, x^4) are also local maxima. Points x^5 and x^6 are local maximum and minimum respectively. **(c)** Illustration of the subgradients of a piecewise linear function. The shadowed sectors denote the range of possible subgradients. In the point x^3 those are all hyperplanes, which lie (are convex combinations of) between a^3 and a^4. The point x^2 contains a horizontal hyperplane, which corresponds to the zero subgradient. Existence of such a hyperplane is a necessary and sufficient optimality condition for convex functions. Note that along with the zero subgradient there are others, that are convex combinations of a^2 and a^3.

Proof. The proof follows directly from the definition of the epigraph of a function: $t \geq f(\mathbf{x})$ and $t \geq g(\mathbf{x})$ together imply $t \geq \max\{f(\mathbf{x}), g(\mathbf{x})\}$, and the other way around. □

Definition 4.6. A function $f\colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is called *convex* if $\operatorname{epi} f$ is a convex set.

Definition 4.7. A function $f\colon \mathbb{R}^n \to \mathbb{R} \cup \{-\infty\}$ is called *concave* if $(-f)$ is convex.

Exercise 4.8. Prove that a concave and convex function f is *linear*, that is, it is representable as $\langle \mathbf{c}, \mathbf{x} \rangle + b$ for some $\mathbf{c} \in \mathbb{R}^n$ and $b \in \mathbb{R}$.

Exercise 4.9. Prove that for any convex function f the set $X = \{\mathbf{x}\colon f(\mathbf{x}) \le 0\}$ is convex.

Proposition 4.10. A function $f\colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is convex if and only if the inequality:

$$f(p\mathbf{x} + (1-p)\mathbf{z}) \le pf(\mathbf{x}) + (1-p)f(\mathbf{z}) \tag{4.4}$$

holds for any $\mathbf{x}, \mathbf{z} \in \mathbb{R}^n$ and $p \in (0,1)$. Here we assume that $p \cdot \infty = \infty$ for any positive p, $\infty + \mathbf{x} = \infty$, $-\infty + \mathbf{x} = -\infty$ for any $\mathbf{x} \in \mathbb{R}$ and $-(-\infty) = \infty$.

Proof. In case either $f(\mathbf{x})$ or $f(\mathbf{z})$ is infinite, the inequality trivially holds. Otherwise, $(\mathbf{x}, f(\mathbf{x})) \in \mathrm{epi} f$ as well as $(\mathbf{z}, f(\mathbf{z})) \in \mathrm{epi} f$. Since $\mathrm{epi} f$ is a convex set, if and only if it contains also $(p\mathbf{x}+(1-p)\mathbf{z}, pf(\mathbf{x})+(1-p)f(\mathbf{z}))$. By the definition of $\mathrm{epi} f$ this is equivalent to $f(p\mathbf{x}+(1-p)\mathbf{z}) \le pf(\mathbf{x}) + (1-p)f(\mathbf{z})$. □

Remark 4.11. The natural addition and multiplication rules for infinite values defined in Proposition 4.10 are considered as defined further in this monograph. Additionally $-\infty \cdot \infty = -\infty$ and the result of $\infty - \infty$ is undefined.

Exercise 4.12. Show that a function $f\colon \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is convex if and only if the inequality

$$f\left(\sum_{i=1}^{N} p_i \mathbf{x}^i\right) \le \sum_{i=1}^{N} p_i f(\mathbf{x}^i) \tag{4.5}$$

holds for any natural N, any N-tuple $\mathbf{x}^i \in \mathbb{R}^n$, $i = 1, \ldots, N$, and any $\mathbf{p} \in \Delta^N$.

The following proposition allows us to construct convex functions with linear operations applied to other convex functions:

Proposition 4.13. Let f and g be convex functions and α and β be non-negative numbers. Then $\alpha f + \beta g$ is convex as well.

Proof. Let $p \in [0,1]$. Then convexity of f and g implies the following sequence of inequalities, which proves the statement of the proposition:

$$\begin{aligned}(\alpha f + \beta g)(p\mathbf{x} + (1-p)\mathbf{z}) &= \alpha f(p\mathbf{x} + (1-p)\mathbf{z}) + \beta g(p\mathbf{x} + (1-p)\mathbf{z}) \\ &\leq \alpha(pf(\mathbf{x}) + (1-p)f(\mathbf{z})) + \beta(pg(\mathbf{x}) + (1-p)g(\mathbf{z})) \\ &= p(\alpha f + \beta g)(\mathbf{x}) + (1-p)(\alpha f + \beta g)(\mathbf{z})\,. \quad (4.6)\end{aligned}$$

□

Definition 4.14. A function $h\colon \mathbb{R}^k \to \mathbb{R}$ is called *non-decreasing*, if $h(x_1, \ldots, x_k) \geq h(z_1, \ldots, z_k)$ when $x_i \geq z_i$ for all $i = 1, \ldots, k$.

A superposition of convex functions is convex under an additional condition:

Proposition 4.15. Let $f(\mathbf{x}) = h(f_1(\mathbf{x}), \ldots, f_k(\mathbf{x}))$ with $f_i\colon \mathbb{R}^n \to \mathbb{R}$ be convex functions, and $h\colon \mathbb{R}^k \to \mathbb{R}$ be convex and non-decreasing. Then f is convex.

Proof. The following inequality holds for any $p \in [0,1]$ and arbitrary $\mathbf{x}$ and $\mathbf{z}$ due to convexity of f_i and the non-decreasing property of h:

$$\begin{aligned}&h(f_1(p\mathbf{x} + (1-p)\mathbf{z}), \ldots, f_k(p\mathbf{x} + (1-p)\mathbf{z})) \\ &\quad \leq h(pf_1(\mathbf{x}) + (1-p)f_1(\mathbf{z}), \ldots, pf_k(\mathbf{x}) + (1-p)f_k(\mathbf{z}))\,. \quad (4.7)\end{aligned}$$

Further, due to convexity of h, the right-hand-side of (4.7) does not exceed

$$ph(f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})) + (1-p)h(f_1(\mathbf{z}), \ldots, f_k(\mathbf{z}))\,, \quad (4.8)$$

that finalizes the proof. □

Proposition 4.16. Let $X \subseteq \mathbb{R}^n$ be a convex set. Then the respective indicator function ι_X as deined by (4.3) is convex.

Proof. Let $\mathbf{x}, \mathbf{z} \in X$. It implies that $(\mathbf{x}, t)$ and $(\mathbf{z}, s)$ belong to $\operatorname{epi}(\iota_X)$ as long as $t, s \geq 0$. For any $0 \leq p \leq 1$, their convex combination $(p\mathbf{x}+(1-p)\mathbf{z}, pt+(1-p)s)$ belongs to $\operatorname{epi}(\iota_X)$, as (i) $p\mathbf{x}+(1-p)\mathbf{z} \in X$ due to convexity of X and (ii) $pt + (1-p)s \geq 0$. □

Since a sum of convex functions is convex according to Proposition 4.13, it implies:

Corollary 4.17. Let f be a convex function defined on $\mathbb{R}^n$ and $X \subset \mathbb{R}^n$ be a convex set. Then the function $f + \iota_X \colon \mathbb{R}^n \to \mathbb{R}$ is convex, where ι_X is defined as in (4.3).

Note that a similar claim holds also for a *concave* function f and a *convex* set X. In this case the function $f - \iota_X$ is concave.

4.1.3 Local and global minima

Let the set $B_\epsilon(\mathbf{x}) = \{\mathbf{x}' \in \mathbb{R}^n \mid \|\mathbf{x} - \mathbf{x}'\| \leq \epsilon\}$ denote the ball of radius $\epsilon \geq 0$ around $\mathbf{x} \in \mathbb{R}^n$.

Definition 4.18 (Local and global minima)**.** Let $f \colon X \to \mathbb{R} \cup \{\infty, -\infty\}$ be an extended value function. A point $\mathbf{x}$ such that $f(\mathbf{x}) < \infty$ is called a *local minimum* of f if there exists $\epsilon > 0$ such that $\mathbf{x} \in \arg\min_{\mathbf{x}' \in B_\epsilon(\mathbf{x})} (f(\mathbf{x}') + \iota_X(\mathbf{x}'))$, with ι_X being the indicator function defined by (4.3). The point $\mathbf{x}$ is called a *global minimum* of f, if $\mathbf{x} \in \arg\min_{\mathbf{x}' \in X} f(\mathbf{x}')$.

Definition 4.18 is illustrated by Figure 4.1(b).

The following proposition states one of the most important properties of convex functions for their minimization:

Proposition 4.19. Any local minimum of a convex function is also its global minimum.

Proof. Let $\mathbf{x}$ be a local, but not a global minimum on $X \subseteq \mathbb{R}^n$. Therefore, there exists $\mathbf{z} \in X$ such that $f(\mathbf{z}) < f(\mathbf{x})$. Since f is convex it holds that $f(p\mathbf{x} + (1-p)\mathbf{z}) \leq pf(\mathbf{x}) + (1-p)f(\mathbf{z}) < f(\mathbf{x})$ for any $p \in (0, 1)$. For any $\epsilon > 0$ there is $p > 0$ such that $(p\mathbf{x}+(1-p)\mathbf{z}) \in B_\epsilon$,

therefore, $f(p\mathbf{x} + (1-p)\mathbf{z}) < f(\mathbf{x})$ implies that $\mathbf{x}$ is not a local minimum. □

Due to Proposition 4.19, it makes sense to speak about *minima* of a convex function, without subdividing them into local and global ones.

Proposition 4.20. The set of minima of a convex function is convex.

Proof. Let $\mathbf{x}$ and $\mathbf{z}$ be two minima and, therefore, $f(\mathbf{x}) = f(\mathbf{z})$. Then for all $p \in (0,1)$ the inequality $f(p\mathbf{x}+(1-p)\mathbf{z}) \leq pf(\mathbf{x})+(1-p)f(\mathbf{z}) = pf(\mathbf{x}) + (1-p)f(\mathbf{x}) = f(\mathbf{x})$ implies that $p\mathbf{x} + (1-p)\mathbf{z}$ is a minimum as well. □

Note that the set of maxima of a concave function is *convex* as well.

The following important proposition essentially states that a maximum of convex functions is a convex function itself. This observation is crucial for properties of Lagrange relaxations, which will be considered later in this chapter.

Proposition 4.21. Let Z be any set and let the function $f\colon \mathbb{R}^n \times Z \to \mathbb{R}$ be convex w.r.t. $\mathbf{x} \in \mathbb{R}^n$ for each fixed $z \in Z$. Then the function $g(\mathbf{x}) = \sup_{z\in Z} f(\mathbf{x}, z)$ is convex.

Proof. For each fixed z the epigraph of $f(\cdot, z)$ is a convex set. The epigraph of the function g is the intersection of the epigraphs of the functions $f(\cdot, z)\colon \mathbb{R}^n \to \mathbb{R}$ for all $z \in Z$. Therefore, according to Lemma 2.28 $\operatorname{epi}(g)$ is a convex set as an intersection of convex sets. So by Definition 4.6 g is convex. □

Corollary 4.22. The function $g(\mathbf{x}) = \inf_{z\in Z} f(\mathbf{x}, z)$ is concave if $f\colon \mathbb{R}^n \times Z \to \mathbb{R}$ is concave w.r.t. $\mathbf{x}$ for each fixed $z \in Z$.

Definition 4.23 (Convex optimization problem)**.** Let f be a convex function and X be a convex set. The problems of the form $\min_{\mathbf{x}\in X} f(\mathbf{x})$ and $\max_{\mathbf{x}\in X}(-f(\mathbf{x}))$ are called *convex optimization problems*. In other words, convex optimization is a minimization of a convex or maximization of a concave function on a convex set.

According to Corollary 4.17, for any convex f and X the problem $\min_{\mathbf{x}\in\mathbb{R}^n}(f(\mathbf{x})+\iota_X(\mathbf{x}))$ is convex and equivalent to $\min_{\mathbf{x}\in X} f(\mathbf{x})$. Therefore, we conclude that the set of optimal solutions of this problem is convex.

Linear programs, as well as the optimization problems from Examples 2.1 and 2.3 are convex.

Definition 4.24. (Convex relaxation) The problem $\min_{\mathbf{x}\in X'} g(\mathbf{x})$ is called a *convex relaxation* of the problem $\min_{\mathbf{x}\in X} f(\mathbf{x})$, if it is both, a relaxation (that is, $X'\subseteq X$ and $g(\mathbf{x})\leq f(\mathbf{x})$ for all $\mathbf{x}\in X$), and a convex optimization problem.

4.2 Subgradient and convex piece-wise linear functions

Most of the functions we will deal with in this monograph will be non-smooth, or, more precisely, non-differentiable. In particular, this implies that they cannot be minimized with such a simple and well-known technique as gradient descent. To circumvent this, we will now introduce a generalization of the notion of gradient to non-differentiable functions. In later chapters we will also consider the corresponding generalizations of the gradient descent algorithm along with other non-smooth optimization methods.

Definition 4.25. Let f be convex. A vector $\mathbf{g}$ is called *a subgradient* of f in the point $\mathbf{x}^0\in\mathrm{dom} f$ if for any $\mathbf{x}\in\mathrm{dom} f$ it holds that

$$f(\mathbf{x})\geq f(\mathbf{x}^0)+\left\langle \mathbf{g},\mathbf{x}-\mathbf{x}^0\right\rangle . \tag{4.9}$$

The set $\partial f(\mathbf{x}^0)$ of all subgradients of f in $\mathbf{x}^0$ is called the *subdifferential* of f in $\mathbf{x}^0$.

Definition 4.25 is illustrated in Figure 4.1(c).

For concave functions the inequality (4.9) changes its sign, i.e. if $\mathbf{g}$ is a subgradient of f it holds that $f(\mathbf{x})\leq f(\mathbf{x}^0)+\left\langle \mathbf{g},\mathbf{x}-\mathbf{x}^0\right\rangle$. Since $-f$ is concave if f is convex, this implies that if $\mathbf{g}$ is a subgradient

of f, then $-\mathbf{g}$ is a subgradient of $-f$. For concave functions the term *supergradient* is also often used instead of subgradient. This reflects the fact that the supergradient defines an upper bound of a concave function contrary to the subgradient defining a lower bound of a convex one.

Proposition 4.26. Let f be a convex function. For any $\mathbf{x} \in \mathrm{dom} f$ the set $\partial f(\mathbf{x})$ is convex.

Proof. Expression (4.9) defines a linear inequality w.r.t. $\mathbf{g}$ for each $\mathbf{x}^0 \in \mathrm{dom} f$. Any linear inequality defines a convex set, see Proposition 2.29. The intersection of these convex sets for all $\mathbf{x} \in \mathrm{dom} f$ is also a convex set according to Lemma 2.28. □

Proposition 4.26 is illustrated in Figure 4.1(c).

In practice, the following proposition is very important for computing subgradients:

Proposition 4.27. If f is convex and differentiable in $\mathbf{x}$, then $\partial f(\mathbf{x}) = \{\nabla f(\mathbf{x})\}$.

We leave out the proof and here provide a suitable reference in Section 4.3.

Since subgradient is defined by a linear inequality, its linear properties follow directly from the definition:

Proposition 4.28 (Linearity of subdifferential)**.** Let f and $\hat{f}$ be convex functions. Then

$$\partial(\alpha f + \beta \hat{f})(\mathbf{x}) = \alpha \partial f(\mathbf{x}) + \beta \partial \hat{f}(\mathbf{x}) \tag{4.10}$$

for any $\alpha, \beta \in \mathbb{R}_+$ with $+$ denoting the Minkowski set sum. In particular, if $\mathbf{g} \in \partial f(\mathbf{x})$ and $\hat{\mathbf{g}} \in \partial \hat{f}(\mathbf{x})$, then $(\alpha \mathbf{g} + \beta \hat{\mathbf{g}}) \in \partial(\alpha f + \beta \hat{f})(\mathbf{x})$.

Subgradient (but not subdifferential in general!) of function composition can be obtained with the same chain rule as gradient:

Proposition 4.29 (Subgradient of function composition)**.** Let $f(\mathbf{x}) = h(f_1(\mathbf{x}), \dots, f_k(\mathbf{x}))$, $h\colon \mathbb{R}^k \to \mathbb{R}$ be convex and non-decreasing and $f_i\colon \mathbb{R}^n \to \mathbb{R}$ be convex functions. Let also

$$\mathbf{g}_i \in \partial f_i(\mathbf{x}^0) \quad \text{and} \quad \mathbf{z} \in \partial h(f_1(\mathbf{x}^0), \dots, f_k(\mathbf{x}^0))\,. \tag{4.11}$$

Then $\mathbf{g} = (\sum_{i=1}^k z_i \mathbf{g}_i) \in \partial f(\mathbf{x}^0)$.

Proof. According to Proposition 4.15 the function f is convex w.r.t. x. The proof of the proposition statement follows from the sequence of (in)equalities:

$$\begin{aligned} f(\mathbf{x}) &= h(f_1(\mathbf{x}), \dots, f_k(\mathbf{x})) \\ &\geq h\left(f_1(\mathbf{x}^0) + \left\langle \mathbf{g}_1, \mathbf{x} - \mathbf{x}^0 \right\rangle, \dots, f_k(\mathbf{x}^0) + \left\langle \mathbf{g}_k, \mathbf{x} - \mathbf{x}^0 \right\rangle\right) \\ &\geq h(f_1(\mathbf{x}^0), \dots, f_k(\mathbf{x}^0)) + \left\langle \mathbf{z}, \left(\left\langle \mathbf{g}_1, \mathbf{x} - \mathbf{x}^0 \right\rangle, \dots, \left\langle \mathbf{g}_k, \mathbf{x} - \mathbf{x}^0 \right\rangle\right)^\top \right\rangle \\ &= f(\mathbf{x}^0) + \left\langle \mathbf{g}, \mathbf{x} - \mathbf{x}^0 \right\rangle\,. \end{aligned} \tag{4.12}$$

□

Therefore, the subgradient can be seen as a generalization of the gradient for convex functions. An important difference is, however, that for non-differentiable functions the subgradient is not unique, i.e. ∂f may contain multiple elements in the points where the function is non-differentiable. Below, we consider an important class of such functions.

Definition 4.30. The function $f\colon \mathbb{R}^n \to \mathbb{R}$ is called *convex piecewise linear*, if there is a finite set I such that for all $\mathbf{x} \in \mathbb{R}^n$ the function is representable as $f(\mathbf{x}) = \max_{i \in I} \left\langle \mathbf{a}^i, \mathbf{x} \right\rangle + b^i$ with $\mathbf{a}^i \in \mathbb{R}^n$, $b^i \in \mathbb{R}$, $i \in I$. By exchanging max with min one obtains *concave piecewise linear* functions.

Note that a convex piecewise linear function is indeed convex as it is a point-wise maximum of convex (here, linear) functions, see Proposition 4.21.

Figure 4.1(c) as well as Example 4.31 below illustrate the notion of piecewise linear functions and their subgradients.

Example 4.31. Let I be a finite set and let f be a convex piecewise linear function $f(\mathbf{x}) = \max_{i\in I} \langle \mathbf{a}^i, \mathbf{x}\rangle + b^i$. Then from $\langle \mathbf{a}^i, \mathbf{x}^0\rangle + b^i = f(\mathbf{x}^0)$ it follows $\mathbf{a}^i \in \partial f(\mathbf{x}^0)$. Indeed,

$$\begin{aligned} f(\mathbf{x}) - f(\mathbf{x}^0) &= \max_{j\in I} \left(\langle \mathbf{a}^j, \mathbf{x}\rangle + b^j\right) - \left(\langle \mathbf{a}^i, \mathbf{x}^0\rangle + b^i\right) \\ &\geq \left(\langle \mathbf{a}^i, \mathbf{x}\rangle + b^i\right) - \left(\langle \mathbf{a}^i, \mathbf{x}^0\rangle + b^i\right) = \langle \mathbf{a}^i, \mathbf{x} - \mathbf{x}^0\rangle \ . \end{aligned} \tag{4.13}$$

The following lemma generalizes Example 4.31:

Lemma 4.32 (Subdifferential of the maximum of a finite number of differentiable convex functions)**.** Let functions $f_i\colon \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ be convex and differentiable. Then the function $f(\mathbf{x}) = \max_{i\in 1,\ldots,m} f_i(\mathbf{x})$ is convex and $\partial f(\mathbf{x}) = \operatorname{conv}\{\nabla f_i(\mathbf{x})\colon i \in I(\mathbf{x})\}$, where $I(\mathbf{x}) := \arg\max_{i\in 1,\ldots,m} f_i(\mathbf{x})$.

Proof. We will here concentrate on showing that

$$\partial f(\mathbf{x}) \supseteq \operatorname{conv}\{\nabla f_i(\mathbf{x})\colon i \in I(\mathbf{x})\}.$$

For the reverse inclusion we refer to the corresponding literature in Section 4.3 as it requires more advanced analysis.

To prove the above inclusion we first show that $\nabla f_i(\mathbf{x}^0) \in \partial f(\mathbf{x}^0)$ for all $i \in I(\mathbf{x}^0)$, $\mathbf{x}^0 \in \mathbb{R}^n$. By Proposition 4.27 we have for all $\mathbf{x} \in \mathbb{R}^n$:

$$f_i(\mathbf{x}) \geq f_i(\mathbf{x}^0) + \langle \nabla f_i(\mathbf{x}^0), \mathbf{x} - \mathbf{x}^0\rangle \ . \tag{4.14}$$

Therefore, by definition of f, it holds that

$$f(\mathbf{x}) \geq f(\mathbf{x}^0) + \langle \nabla f_i(\mathbf{x}^0), \mathbf{x} - \mathbf{x}^0\rangle \ , \tag{4.15}$$

and, hence, $\nabla f_i(\mathbf{x}^0) \in \partial f(\mathbf{x}^0)$.

As $\operatorname{conv}\{\nabla f_i(\mathbf{x})\colon i \in I(\mathbf{x})\}$ is the smallest convex set containing all vectors $\nabla f_i(\mathbf{x})$, $i \in I(\mathbf{x})$, and $\partial f(\mathbf{x})$ is convex by Proposition 4.26, this implies $\partial f(\mathbf{x}) \supseteq \operatorname{conv}\{\nabla f_i(\mathbf{x})\colon i \in I(\mathbf{x})\}$. □

In particular, for a convex piecewise linear function

$$f(\mathbf{x}) = \max_{i\in I} \langle \mathbf{a}^i, \mathbf{x}\rangle + b^i$$

Lemma 4.32 states that $\partial f(\mathbf{x}) = \operatorname{conv}\{\mathbf{a}^j : \langle \mathbf{a}^j, \mathbf{x}\rangle + b^j = f(\mathbf{x}),\ j \in I\}$. In other words, the subdifferential is a convex hull of all vectors $\mathbf{a}^j$ such that $f(\mathbf{x}) = \langle \mathbf{a}^j, \mathbf{x}\rangle + b^j$, see Figure 4.1(c).

Example 4.33. Consider $f(x) = \max\{0, 1 - x\}$.

The subdifferential of f is $\partial f(x) = \begin{cases} -1, & x < 1 \\ [-1, 0], & x = 1 \\ 0, & x > 1 \end{cases}$.

A statement analogous to Lemma 4.32 can be formulated for concave functions by substituting "convex" with "concave" and max with min:

Corollary 4.34. Let functions $f_i \colon \mathbb{R}^n \to \mathbb{R}$, $i = 1, \ldots, m$ be concave and differentiable. Then the function $f(\mathbf{x}) = \min_{i \in 1,\ldots,m} f_i(\mathbf{x})$ is concave and $\partial f(\mathbf{x}) = \operatorname{conv}\{\nabla f_i(\mathbf{x}) \colon i \in I(\mathbf{x})\}$, where $I(\mathbf{x}) := \arg\min_{i \in 1,\ldots,m} f_i(\mathbf{x})$.

Apart from its use in gradient-based optimization methods, the gradient of a differentiable convex function uniquely characterizes its optima. A similar result holds also for the subgradient:

Theorem 4.35. Let f be convex. Then $f(\mathbf{x}^*) = \min_{\mathbf{x} \in \operatorname{dom} f} f(\mathbf{x})$ if and only if $\mathbf{0} \in \partial f(\mathbf{x}^*)$.

Proof. Indeed, if $\mathbf{0} \in \partial f(\mathbf{x}^*)$, then $f(\mathbf{x}) \geq f(\mathbf{x}^*) + \langle \mathbf{0}, \mathbf{x} - \mathbf{x}^* \rangle = f(\mathbf{x}^*)$ for all $\mathbf{x} \in \operatorname{dom} f$. On the other hand, if $f(\mathbf{x}) \geq f(\mathbf{x}^*)$ for all $\mathbf{x} \in \operatorname{dom} f$, then $f(\mathbf{x}) \geq f(\mathbf{x}^*) + \langle \mathbf{0}, \mathbf{x} - \mathbf{x}^* \rangle = f(\mathbf{x}^*)$ and $\mathbf{0}$ is a subgradient of f in $\mathbf{x}^*$. □

Theorem 4.35 is illustrated in Figure 4.1(c).

4.3 Bibliography and further reading

For a more fundamental understanding the notion of convex functions and their subgradients we refer to the textbook [22]. In particular, it contains proofs of Proposition 4.27 and Lemma 4.32. The comprehensive source on the topic is [17]. section 9.2.

5 Lagrange duality

In Chapter 2 we defined a general notion of relaxation and considered an important example thereof, the linear programming relaxations. The latter are constructed by substituting the integrality constraints with box constraints.

Here we will consider another powerful method for constructing relaxations, where some constraints are substituted with a specially constructed function, which is then added to the original objective.

Although these two types of relaxations seem to have nothing in common at the first glance, they are in fact often equivalent. At the end of the chapter we will consider this point and formulate sufficient conditions for this equivalence.

5.1 Dual problem

Let us consider an optimization problem of the form

$$\begin{aligned} &\min_{\mathbf{x}\in P} f(\mathbf{x}) \\ &\text{s.t. } A\mathbf{x} = \mathbf{b}\,, \end{aligned} \tag{5.1}$$

where $P \subseteq \mathbb{R}^n$. We will assume that omitting the equality constraints $A\mathbf{x} = \mathbf{b}$ would make the optimization problem easy (or at least *easier*) to solve. Therefore, we write these constraints separately instead of including them into the set P. Below we consider a powerful technique, which allows us to partially get rid of these constraints to simplify optimization.

Consider the following function of a vector $\boldsymbol{\lambda} \in \mathbb{R}^n$

$$\min_{\mathbf{x}\in P} \left[f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle \right] , \tag{5.2}$$

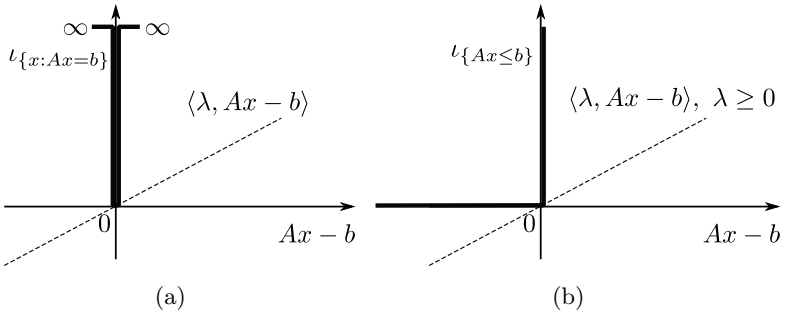


Figure 5.1: Illustration of the Lagrange relaxation for **(a)** the equality constraints $A\mathbf{x} = \mathbf{b}$. Bold lines denote the function $\iota_{\{\mathbf{x}:\, A\mathbf{x}=\mathbf{b}\}}(\mathbf{x})$, see (4.3). The dashed line denotes the linear function $\langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle$, which constitutes a lower bound for $\iota_{\{\mathbf{x}:\, A\mathbf{x}=\mathbf{b}\}}(\mathbf{x})$. **(b)** The same, but when the inequality $A\mathbf{x} \leq \mathbf{b}$ is dualized. Due to the, compared to Figure (a), different function $\iota_{\{\mathbf{x}:\, A\mathbf{x}\leq\mathbf{b}\}}(\mathbf{x})$, only non-negative $\boldsymbol{\lambda}$ must be considered to guarantee the lower bound property of $\langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle$.

which we will call the *Lagrange dual function*. The expression $L(\mathbf{x}, \lambda) := f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle$ is referred to as the *Lagrangean*. One also says that the constraint $A\mathbf{x} = \mathbf{b}$ is *dualized* or *relaxed*. Variables $\boldsymbol{\lambda}$ are called *dual variables*. The dimensionality of $\boldsymbol{\lambda}$ equals the number of rows of the matrix A. In other words, each elementary constraint gets a corresponding dual variable.

Example 5.1. Consider

$$\min_{\mathbf{x}\in[0,1]^2} \langle \mathbf{c}, \mathbf{x} \rangle \tag{5.3}$$

$$\text{s.t. } x_1 = x_2\,. \tag{5.4}$$

Its Lagrange dual function reads

$$\min_{\mathbf{x}\in[0,1]^2} \langle \mathbf{c}, \mathbf{x} \rangle + \lambda(x_1 - x_2) \tag{5.5}$$

with λ being a real number.

Example 5.2. Consider

$$\min_{\mathbf{x}\in P\cap\{0,1\}^3} 3x_1 + 5x_2 + 7x_3 \tag{5.6}$$

$$\text{s.t.} \quad x_1 + 2x_2 = 0 \quad |\lambda_1 \tag{5.7}$$

$$x_1 + x_2 + x_3 = 1 \quad |\lambda_2 \tag{5.8}$$

Lagrange dual is equal to:

$$\min_{\mathbf{x}\in P\cap\{0,1\}^3} 3x_1 + 5x_2 + 7x_3 + \lambda_1(x_1 + 2x_2) + \lambda_2(x_1 + x_2 + x_3 - 1) \tag{5.9}$$

For any $\boldsymbol{\lambda}$ the problem (5.2) is a relaxation of (5.1), because:

- The feasible set of (5.2) is a superset of the one for (5.1).
- The objective values of both problems are equal for any feasible point of (5.1), since in this case $A\mathbf{x} = \mathbf{b}$ holds, and, therefore, $\langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle = 0$.

These kinds of relaxations are called *Lagrange* relaxations.
The properties above imply:

Proposition 5.3. For every $\boldsymbol{\lambda} \in \mathbb{R}$ the optimal value of (5.2) does not exceed the optimal value of (5.1), and they coincide if the constraint $A\mathbf{x}' = \mathbf{b}$ holds in an optimum $\mathbf{x}'$ of (5.2).

Proof. The first claim follows directly from Proposition 2.9. The second one follows from Proposition 2.9 and the fact that the objective functions of (5.1) and (5.2) coincide if $A\mathbf{x}' = \mathbf{b}$. □

Remark 5.4. Note that the problem

$$g(\boldsymbol{\lambda}) = \min_{\mathbf{x}\in P} f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle \tag{5.10}$$

is a Lagrange dual of

$$\min_{\mathbf{x}\in P} f(\mathbf{x}) \tag{5.11}$$

$$\text{s.t. } A\mathbf{x} \leq \mathbf{b} \tag{5.12}$$

for any $\boldsymbol{\lambda} \geq 0$.

In general, Lagrange relaxation is a way to construct a relaxation by dualizing any equality or inequality constraints, not necessarily linear ones. In these cases the constraint is defined as $h(x) = b$ (or $h(x) \leq b$), and is substituted by the term $\lambda(h(x) - b)$ (with $\lambda \geq 0$), which is then added to the objective. In this monograph we make use of the dualization of linear constraints only, therefore, we restrict our attention to this very important special case.

Figure 5.1 gives an intuitive explanation of the relaxation properties of the Lagrange duals. Indeed, let us move the constraint $A\mathbf{x} = \mathbf{b}$ of the problem (5.1) into the objective. Then it becomes equivalent to

$$\min_{\mathbf{x} \in P} f(\mathbf{x}) + \iota_{\{\mathbf{x}\colon A\mathbf{x}=\mathbf{b}\}}(\mathbf{x})\,. \tag{5.13}$$

To obtain the dual (5.2) we only substitute the function $\iota_{\{\mathbf{x}\colon A\mathbf{x}=\mathbf{b}\}}$ with $\langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle$, and, as shown in Figure 5.1(a), the latter is a linear lower bound for the former one. The same reasoning can be used for the inequality case from Remark 5.4 and is illustrated in Figure 5.1(b).

Since it is important to have the tightest possible lower bound, one considers the *Lagrange dual problem*

$$\max_{\boldsymbol{\lambda}} \min_{\mathbf{x} \in P} f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle\ , \tag{5.14}$$

which amounts to maximizing (5.2) w.r.t. λ. One speaks about *full Lagrange dual* if all constraints are dualized and not only a subset of them.

The most important property of the Lagrange dual problem (5.14) is its concavity w.r.t. $\boldsymbol{\lambda}$, which does not depend on the fact whether the primal problem (5.1) is convex or not:

Proposition 5.5. The dual function (5.2) is concave w.r.t. $\boldsymbol{\lambda}$.

Proof. The proof follows from Corollary 4.22 and the fact that the objective of problem (5.2) is linear w.r.t. $\boldsymbol{\lambda}$, and, therefore, concave. □

The following example shows that although the dual function is concave, it may not be differentiable:

Example 5.6 (Piecewise linear Lagrange dual)**.** Consider the Lagrange dual as in (5.1):

$$\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle \tag{5.15}$$

It is a concave piecewise linear function of $\boldsymbol{\lambda}$, when P is

- a finite set, since (5.15) satisfies Definition 4.30 in this case.
- a polytope. In this case

$$\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle \overset{\text{Cor. } 2.51}{=} \min_{\mathbf{x}\in \operatorname{vrtx}(P)} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x}\right\rangle - \langle \boldsymbol{\lambda}, \mathbf{b}\rangle \ , \tag{5.16}$$

  and the right-hand-side satisfies Definition 4.30.

Remark 5.7. In a more general case

$$\min_{\mathbf{x}\in P} f(\mathbf{x}) \tag{5.17}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{5.18}$$

$$B\mathbf{x} \leq \mathbf{d} \tag{5.19}$$

the Lagrange dual problem takes the form

$$\max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}\in P} f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d}\rangle \ . \tag{5.20}$$

All key properties like concavity and piecewise linearity of the Lagrange dual objective hold also in this more general case.

Remark 5.8. For a maximization problem

$$\max_{\mathbf{x}\in P} f(\mathbf{x}) \tag{5.21}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b} \tag{5.22}$$

$$B\mathbf{x} \leq \mathbf{d} \tag{5.23}$$

the Lagrange dual reads

$$\min_{\substack{\boldsymbol{\lambda} \\ \boldsymbol{\mu} \geq 0}} \max_{\mathbf{x} \in P} f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle - \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d} \rangle \ . \tag{5.24}$$

The respective Lagrange dual objective is convex. It is also piecewise linear in the same cases as its minimization counterpart.

Proposition 5.9. Let P be a polytope and the set $P \cap \{\mathbf{x} \colon A\mathbf{x} = \mathbf{b}\}$ be non-empty. Then there exist optimal primal and dual solutions. Moreover, the *strong duality* holds, i.e.

$$\min_{\substack{\mathbf{x} \in P \\ A\mathbf{x} = \mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle = \max_{\boldsymbol{\lambda}} \min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle \ . \tag{5.25}$$

Proof. Let us first show that there exist optimal primal and dual solutions. From P being a polytope it follows that the primal feasible set $P \cap \{\mathbf{x} \colon A\mathbf{x} = \mathbf{b}\}$ is a polytope as well (see Proposition 2.19). Therefore, the optimal primal value p^* is finite and attained in one of its vertices, according to Corollary 2.51.

Let d^* denote the optimal dual value. The following sequence of inequalities, which uses Proposition 5.3,

$$\infty > p^* \geq d^* = \max_{\boldsymbol{\lambda}} \min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle \overset{\boldsymbol{\lambda} = \mathbf{0}}{\geq} \min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle > -\infty \tag{5.26}$$

implies that the dual optimal value d^* is also finite.

Consider the dual objective

$$g(\boldsymbol{\lambda}) := \min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle \overset{\text{Cor. } 2.51}{=} \min_{\mathbf{x} \in \operatorname{vrtx}(P)} \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle \ , \tag{5.27}$$

and the set $\hat{P}(\boldsymbol{\lambda}) := \arg\min_{\mathbf{x} \in \operatorname{vrtx}(P)} \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle$. Since P is non-empty, $\hat{P}(\boldsymbol{\lambda})$ is so as well. The maximal dual value d^* is attained in all vectors $\boldsymbol{\lambda}$ satisfying

$$d^* = \left\langle \mathbf{c}, \mathbf{x}' \right\rangle + \left\langle \boldsymbol{\lambda}, A\mathbf{x}' - \mathbf{b} \right\rangle \ , \tag{5.28}$$

where $\mathbf{x}'$ is some vector from $\hat{P}(\boldsymbol{\lambda})$. The set of $\boldsymbol{\lambda}$ satisfying (5.28) is non-empty as soon as $A\mathbf{x}' - \mathbf{b} = \mathbf{0}$ implies $d^* = \langle \mathbf{c}, \mathbf{x}' \rangle$. This implication is given by the following sequence of equalities:

$$d^* = \max_{\boldsymbol{\lambda}} \min_{\mathbf{x} \in \operatorname{vrtx}(P)} \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle = \max_{\boldsymbol{\lambda}} \left\langle \mathbf{c}, \mathbf{x}' \right\rangle + \left\langle \boldsymbol{\lambda}, A\mathbf{x}' - \mathbf{b} \right\rangle$$
$$\overset{A\mathbf{x}'=\mathbf{b}}{=} \max_{\boldsymbol{\lambda}} \left\langle \mathbf{c}, \mathbf{x}' \right\rangle = \left\langle \mathbf{c}, \mathbf{x}' \right\rangle . \quad (5.29)$$

Therefore, the dual solution exists. Let $\boldsymbol{\lambda}^*$ be such a solution.

According to Lemma 4.32,

$$\partial g(\boldsymbol{\lambda}^*) = \operatorname{conv}\{A\mathbf{x}^* - \mathbf{b} \colon \mathbf{x}^* \in \hat{P}(\boldsymbol{\lambda}^*)\} \overset{\text{Lem. } 2.39}{=} \{A\mathbf{x}^* - \mathbf{b} \colon \mathbf{x}^* \in \operatorname{conv}(\hat{P}(\boldsymbol{\lambda}^*))\} . \quad (5.30)$$

Since $\boldsymbol{\lambda}^*$ is a dual optimal solution, Theorem 4.35 implies that $\partial g(\boldsymbol{\lambda}^*) \ni \mathbf{0}$. In other words, there exists $\mathbf{x}^* \in \operatorname{conv}(\hat{P}(\boldsymbol{\lambda}^*))$ such that

$$A\mathbf{x}^* - \mathbf{b} = \mathbf{0} . \quad (5.31)$$

Observe that $\mathbf{x}^* \in P$, since $\hat{P}(\boldsymbol{\lambda}^*) \subseteq \operatorname{vrtx}(P)$ and, therefore, $\operatorname{conv}(\hat{P}(\boldsymbol{\lambda}^*)) \subseteq \operatorname{conv}(\operatorname{vrtx}(P)) = P$. This yields

$$d^* = \langle \mathbf{c}, \mathbf{x}^* \rangle + \langle \boldsymbol{\lambda}^*, A\mathbf{x}^* - \mathbf{b} \rangle \overset{A\mathbf{x}^*-\mathbf{b}=0}{=} \langle \mathbf{c}, \mathbf{x}^* \rangle \overset{\substack{\mathbf{x}^* \in P \\ A\mathbf{x}^*-\mathbf{b}=\mathbf{0}}}{\geq} \min_{\substack{\mathbf{x} \in P \\ A\mathbf{x}=\mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle = p^* . \quad (5.32)$$

Together with (5.26) this implies $p^* = d^*$. □

Example 5.10. Consider Example 5.1, the primal

$$\min_{\mathbf{x} \in [0,1]^2} \langle \mathbf{c}, \mathbf{x} \rangle \quad (5.33)$$
$$\text{s.t. } x_1 = x_2 , \quad (5.34)$$

and the dual problem:

$$\max_{\lambda} \min_{\mathbf{x} \in [0,1]^2} [\langle \mathbf{c}, \mathbf{x} \rangle + \lambda(x_1 - x_2)]$$
$$= \max_{\lambda} \min_{\mathbf{x} \in [0,1]^2} [(c_1 + \lambda)x_1 + (c_2 - \lambda)x_2] . \quad (5.35)$$

The primal problem can be equivalently rewritten as

$$\min_{x_1\in[0,1]} (c_1+c_2)x_1 = \begin{cases} 0, & (c_1+c_2)\geq 0 \\ (c_1+c_2), & (c_1+c_2)<0\,. \end{cases} \tag{5.36}$$

Consider the dual problem (5.35). It can be equivalently rewritten as

$$\max_{\lambda} \left(\min\{0,(c_1+\lambda)\}+\min\{0,(c_2-\lambda)\}\right)\,. \tag{5.37}$$

The necessary and sufficient optimality condition is the existence of the zero subgradient of the function. According to Lemma 4.32, this condition holds when either

$$c_1+\lambda\geq 0 \text{ and } c_2-\lambda\geq 0\,, \tag{5.38}$$

or

$$c_1+\lambda<0 \text{ and } c_2-\lambda<0\,. \tag{5.39}$$

hold. One pair of these inequalities is always attained when $c_1+\lambda = c_2-\lambda$, therefore, $\lambda=\frac{c_2-c_1}{2}$. Substituting λ with this value in (5.38) we obtain $c_1+c_2\geq 0$. Similarly (5.39) turns into $c_1+c_2<0$.

Summing up,

$$\max_{\lambda} \left(\min\{0,(c_1+\lambda)\}+\min\{0,(c_2-\lambda)\}\right) \tag{5.40}$$

$$= \begin{cases} 0, & (c_1+c_2)\geq 0 \\ (c_1+c_2), & (c_1+c_2)<0\,, \end{cases} \tag{5.41}$$

which coincides with the primal optimal value.

Theorem 5.11. Let P be a polytope and the set $P\cap\{\mathbf{x}\colon A\mathbf{x}=\mathbf{b},\ B\mathbf{x}\leq\mathbf{d}\}$ be non-empty. Then there exist optimal primal and dual solutions. Moreover, the *strong duality* holds, i.e.

$$\min_{\substack{\mathbf{x}\in P\\ A\mathbf{x}=\mathbf{b},\ B\mathbf{x}\leq\mathbf{d}}} \langle\mathbf{c},\mathbf{x}\rangle = \max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}\in P} \langle\mathbf{c},\mathbf{x}\rangle + \langle\boldsymbol{\lambda},A\mathbf{x}-\mathbf{b}\rangle + \langle\boldsymbol{\mu},B\mathbf{x}-\mathbf{d}\rangle\,. \tag{5.42}$$

We provide this theorem here without a proof.

5.2 Lagrange relaxation of integer linear programs

Since we are interested in solving integer linear programs, let us consider the Lagrange relaxation of this kind of problem.

In what follows we will deal with integer linear programs of the form

$$\begin{aligned} \min_{\mathbf{x}\in P\cap\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle \\ \text{s.t. } A\mathbf{x} = \mathbf{b}\,, \\ B\mathbf{x} \leq \mathbf{d}\,, \end{aligned} \tag{5.43}$$

where P is a polytope. The assumption that P is bounded (i.e. it is a polytope and not a general polyhedron) can be made w.l.o.g., since the problem (5.43) can be turned into an equivalent one with a bounded P by adding the constraints $\mathbf{x} \in [0,1]^n$.

As before, we assume that without constraint $A\mathbf{x} = \mathbf{b}$ and $B\mathbf{x} \leq \mathbf{d}$ the problem (5.43) becomes easy or at least easier to solve. Therefore, these constraints are separated out from the polytope P. Dualizing these constraints we obtain the Lagrange dual problem to (5.43):

$$\max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}\in P\cap\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d}\rangle\ . \tag{5.44}$$

Note that objective of the dual problem (5.44) is a concave piecewise linear, therefore, non-differentiable, function.

Example 5.12. Consider (5.43) and assume $P = \mathbb{R}^n$. Then the dual objective in (5.44) can be rewritten as

$$\begin{aligned} g(\boldsymbol{\lambda}, \boldsymbol{\mu}) &= \min_{\mathbf{x}\in\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d}\rangle \\ &= -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle - \langle \boldsymbol{\mu}, \mathbf{d}\rangle + \min_{\mathbf{x}\in\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle + \left\langle A^\top \boldsymbol{\lambda}, \mathbf{x}\right\rangle + \left\langle B^\top \boldsymbol{\mu}, \mathbf{x}\right\rangle \\ &= -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle - \langle \boldsymbol{\mu}, \mathbf{d}\rangle + \min_{\mathbf{x}\in\{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda} + B^\top \boldsymbol{\mu}, \mathbf{x}\right\rangle \\ &= -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle - \langle \boldsymbol{\mu}, \mathbf{d}\rangle + \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda} + B^\top \boldsymbol{\mu}, \mathbf{x}^*\right\rangle\ , \end{aligned} \tag{5.45}$$

with

$$x_i^* = \begin{cases} 0, & (\mathbf{c} + A^\top \boldsymbol{\lambda} + B^\top \boldsymbol{\mu})_i > 0 \\ 1, & (\mathbf{c} + A^\top \boldsymbol{\lambda} + B^\top \boldsymbol{\mu})_i < 0 \\ \in \{0,1\}, & \text{otherwise}\,. \end{cases} \tag{5.46}$$

In this important special case the computation of the dual function can be done independently for each coordinate $i \in [n]$ and is, therefore, particularly easy.

Use Case: Binary Knapsack Problem

Consider the binary knapsack problem from Example 2.54:

$$\max_{\mathbf{x}\in\{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle \tag{5.47}$$

$$\text{s.t. } \langle \mathbf{w}, \mathbf{x} \rangle \leq W\,. \tag{5.48}$$

By dualizing its inequality constraint we obtain the Lagrange dual problem

$$\begin{aligned} \min_{\mu\geq 0} \max_{\mathbf{x}\in\{0,1\}^n} & \langle \mathbf{c}, \mathbf{x} \rangle - \mu(\langle \mathbf{w}, \mathbf{x} \rangle - W) \\ &= \min_{\mu\geq 0} \max_{\mathbf{x}\in\{0,1\}^n} \langle \mathbf{c} - \mu\mathbf{w}, \mathbf{x} \rangle + \mu W \\ &= \min_{\mu\geq 0} \mu W + \max_{\mathbf{x}\in\{0,1\}^n} \langle \mathbf{c} - \mu\mathbf{w}, \mathbf{x} \rangle \\ &= \min_{\mu\geq 0} \mu W + \langle \mathbf{c} - \mu\mathbf{w}, \mathbf{x}^*(\mu) \rangle\ , \end{aligned} \tag{5.49}$$

where

$$x_i^* := x^*(\mu)_i = \begin{cases} 1, & c_i - \mu w_i > 0 \\ 0, & c_i - \mu w_i \leq 0\,. \end{cases} \tag{5.50}$$

Note the "weight-informed cost" $\mathbf{c} - \mu\mathbf{w}$ plays the decisive role in computation of the dual. We will see that similar phenomena appear in, essentially, any Lagrange dual with linear objective and constraints and treat it in a general case in Section 5.2.2.

Assume now W.l.o.g. now assume that $c_i > 0$ and $W \geq w_i > 0$, moreover, $\frac{c_1}{w_1} \geq \frac{c_2}{w_2} \geq \cdots \geq \frac{c_n}{w_n}$. According to (5.49) for $\frac{c_k}{w_k} \geq \mu \geq \frac{c_{k+1}}{w_{k+1}}$ the dual objective takes the form

$$g(\mu) = \mu W + \sum_{i=1}^{k}(c_i - \mu w_i) = \mu(W - \sum_{i=1}^{k} w_i) + \sum_{i=1}^{k} c_i \,. \tag{5.51}$$

We will return to this example after considering optimality conditions for the dual problem.

5.2.1 Primal of the relaxed problem for ILPs

The Lagrange dual problem (5.14) can be treated as a problem of selecting the best relaxation from a given set of relaxations. This class is parametrized with the dual vector $\boldsymbol{\lambda}$. The solution of the dual problem is a value of this parameter. To obtain the corresponding relaxed solution one has to solve the related relaxed problem $\min_{\mathbf{x}\in X} f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle$ with fixed $\boldsymbol{\lambda}$. In this problem, however, both, the objective and the initial feasible sets are changed compared to the primal problem (5.43). This makes analysis of the relaxed solution much more complicated.

This differs from the case of the LP relaxation for integer linear programs, where only the feasible set is increased, but the objective itself remains unchanged. Interestingly, for integer linear programs, one can define a relaxation, which is equivalent to the Lagrange relaxation, by changing only the feasible set.

To do so, let us consider the Lagrange dual (5.44). Theorem 2.50 implies

$$\begin{aligned}\min_{\mathbf{x}\in P\cap\{0,1\}^n} &\langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d}\rangle \\ &= \min_{\mathbf{x}\in \operatorname{conv}(P\cap\{0,1\}^n)} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d}\rangle \,. \end{aligned} \tag{5.52}$$

Consider now the following minimization problem

$$\min_{\substack{\mathbf{x}\in \operatorname{conv}(P\cap\{0,1\}^n)\\ A\mathbf{x}=\mathbf{b},\ B\mathbf{x}\leq\mathbf{d}}} \langle \mathbf{c}, \mathbf{x}\rangle \,. \tag{5.53}$$

Assuming that the problem (5.43) is feasible implies that (5.53) is so, too. Since the feasible set of (5.53) is a polytope, Theorem 5.11 implies that its Lagrange dual is tight:

$$\begin{aligned}&\min_{\substack{\mathbf{x}\in\operatorname{conv}(P\cap\{0,1\}^n)\\ A\mathbf{x}=\mathbf{b},\ B\mathbf{x}\leq\mathbf{d}}} \langle \mathbf{c},\mathbf{x}\rangle \\ &= \max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}\in\operatorname{conv}(P\cap\{0,1\}^n)} \langle \mathbf{c},\mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x}-\mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x}-\mathbf{d}\rangle\ . \end{aligned} \tag{5.54}$$

Comparing it to (5.52) we conclude

Proposition 5.13.

$$\begin{aligned}\max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}\in P\cap\{0,1\}^n} &\langle \mathbf{c},\mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x}-\mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x}-\mathbf{d}\rangle \\ &= \min_{\substack{\mathbf{x}\in\operatorname{conv}(P\cap\{0,1\}^n)\\ A\mathbf{x}=\mathbf{b},\ B\mathbf{x}\leq\mathbf{d}}} \langle \mathbf{c},\mathbf{x}\rangle \end{aligned} \tag{5.55}$$

In other words, the Lagrange dual is tight for the problem (5.53). Therefore, we will call the latter the *primal relaxed* problem for the Lagrange relaxation.

Note that (5.53)) is also a convex relaxation of the ILP problem (5.43), since $\operatorname{conv}(P\cap\{0,1\}^n) \supseteq P\cap\{0,1\}^n$ by definition of the convex hull. Contrary to the Lagrange relaxation, the problem (5.53) modifies only the feasible set of the non-relaxed problem (5.43), and keeps its objective unchanged. Note also that (5.53) is a linear program, since its feasible set is a polytope. It differs from the LP relaxation of (5.43)

$$\min_{\substack{\mathbf{x}\in P\cap[0,1]^n\\ A\mathbf{x}=\mathbf{b},\ B\mathbf{x}\leq\mathbf{d}}} \langle \mathbf{c},\mathbf{x}\rangle \tag{5.56}$$

by the feasible set only. The following statement compares these two linear programs and establishes conditions under which they coincide.

Corollary 5.14. Let $\operatorname{vrtx}(P\cap[0,1]^n)\subseteq\{0,1\}^n$, that is, all vertices of the polytope $P\cap[0,1]^n$ are defined by binary vectors. Then the primal relaxed problem (5.53) is equal to the LP relaxation (5.56). Otherwise, the relaxation (5.53) is tighter.

Since the primal relaxed (5.53) and dual (5.44) problems provide the same bound, one also says that *the Lagrange relaxation is in general tighter than the LP one.*

Proof. According to Proposition 2.64, it holds that $\mathrm{vrtx}(P \cap [0,1]^n) \supseteq P \cap \{0,1\}^n$. According to Proposition 2.66, it holds $\mathrm{vrtx}(P \cap [0,1]^n) \cap \{0,1\}^n = P \cap \{0,1\}^n$. Therefore, $\mathrm{vrtx}(P \cap [0,1]^n) \subseteq \{0,1\}^n$ implies $\mathrm{vrtx}(P \cap [0,1]^n) = P \cap \{0,1\}^n$. From Corollary 2.41 it follows that $\mathrm{conv}(P \cap \{0,1\}^n) = P \cap [0,1]^n$, and, therefore, the considered Lagrange and LP relaxations are equivalent.

In general, however, it holds that $\mathrm{conv}(P \cap \{0,1\}^n) \subseteq P \cap [0,1]^n$ (see Proposition 2.63), which implies that relaxation (5.53) is tighter than the LP one (5.56), as their objective functions coincide. □

5.2.2 Reparametrization

For (integer) linear programs the Lagrange relaxation has a simple and intuitive interpretation, expressed in terms of *reparametrization* or *equivalent transformation* of the cost vector. The optimization of the dual problem can be viewed as finding such an equivalent transformation, such that the dualized constraints are fulfilled for the (relaxed) primal problem.

We will first assume that the dualized constraints have the form $A\mathbf{x} = \mathbf{0}$. Later consider the differences in the general case $A\mathbf{x} = \mathbf{b}$.

Let us consider the Lagrange dual problem (5.44). For $\mathbf{b} = \mathbf{0}$ its objective is

$$\langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} \rangle = \langle \mathbf{c}, \mathbf{x} \rangle + \left\langle A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle = \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle . \tag{5.57}$$

Note that for any feasible $\mathbf{x}$, it holds that $A\mathbf{x} = \mathbf{0}$ and therefore

$$\langle \mathbf{c}, \mathbf{x} \rangle = \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle . \tag{5.58}$$

It implies that the problem

$$\min_{\mathbf{x}\in P\cap\{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle \tag{5.59}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{0} \tag{5.60}$$

is *equivalent* to (5.43) (with $\mathbf{b} = \mathbf{0}$), i.e. for any $\boldsymbol{\lambda}$ and any feasible $\mathbf{x}$ the objectives of both problems have equal values. Clearly, the solutions of both problems are therefore also equal. The transformation $\mathbf{c} \to \mathbf{c} + A^\top \boldsymbol{\lambda}$ is therefore called *an equivalent transformation* or *a reparametrization* of the problem (5.43). The dual problem

$$\max_{\boldsymbol{\lambda}} \left[D(\boldsymbol{\lambda}) := \min_{\mathbf{x}\in P\cap\{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle \right] \tag{5.61}$$

therefore consists in finding an *optimal reparametrization* of the problem.

Example 5.15. In Example 5.10 the reparametrized cost vector has coordinates $c_1 + \lambda$ and $c_2 - \lambda$.

Note that the equality (5.58) implies also that the *relaxed* primal problem, defined by the right-hand-side of (5.55), is equivalent to its reparametrization for any λ:

$$\min_{\substack{\text{conv}(\mathbf{x}\in P\cap\{0,1\}^n)\\ A\mathbf{x}=\mathbf{0}}} \langle \mathbf{c}, \mathbf{x} \rangle = \min_{\substack{\text{conv}(\mathbf{x}\in P\cap\{0,1\}^n)\\ A\mathbf{x}=\mathbf{0}}} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle . \tag{5.62}$$

General case $A\mathbf{x} = \mathbf{b}$ In case $\mathbf{b} \neq \mathbf{0}$

$$\langle \mathbf{c}, \mathbf{x} \rangle = \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle - \langle \mathbf{b}, \boldsymbol{\lambda} \rangle . \tag{5.63}$$

Therefore, the total reparametrization shifts the total cost of each feasible solution by $\langle \mathbf{b}, \boldsymbol{\lambda} \rangle$ contrary to the case $\mathbf{b} = \mathbf{0}$, where the total cost remains the same. The shift, however, preserves the order of solution costs and does not influence their optimality.

The primal problem (5.59) equivalent to (5.43) turns into

$$- \langle \mathbf{b}, \boldsymbol{\lambda} \rangle + \min_{\mathbf{x} \in P \cap \{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle \tag{5.64}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b}\,, \tag{5.65}$$

and the Lagrange dual problem (5.61) reads

$$\max_{\boldsymbol{\lambda}} \left[D(\boldsymbol{\lambda}) := - \langle \mathbf{b}, \boldsymbol{\lambda} \rangle + \min_{\mathbf{x} \in P \cap \{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle \right]. \tag{5.66}$$

Most properties of the dual objective $D(\boldsymbol{\lambda})$ remain the same for any vector $\mathbf{b}$. In particular, the key complexity of its evaluation remains computation of $\min_{\mathbf{x} \in P \cap \{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle$, which is independent of $\mathbf{b}$.

Apart from theoretical, the reparametrized costs have a powerful applied usage: When used with nearly optimal dual vector $\boldsymbol{\lambda}$ they may significantly improve results of multiple primal heuristics, especially those based in the greedy algorithms, see Section 8.2. In Chapter 8 we will see how primal heuristics can profit from being applied to reparametrized costs instead of original ones.

Slack variables and Lagrange dual. Instead of dualizing inequality one may turn it into equality with a *slack* variable and dualize the equality afterwards. Consider problem (5.11) and the respective non-negative slack vector $\mathbf{s}$:

$$\min_{\substack{\mathbf{x} \in P \\ \mathbf{s} \geq 0}} f(\mathbf{x}) \tag{5.67}$$

$$\text{s.t. } A\mathbf{x} + \mathbf{s} = \mathbf{b}\,. \tag{5.68}$$

The dual problem turns into

$$\max_{\boldsymbol{\lambda}} \min_{\substack{\mathbf{x} \in P \\ \mathbf{s} \geq 0}} f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} + \mathbf{s} - \mathbf{b} \rangle \tag{5.69}$$

If $\lambda_i < 0$ for any i, the minimization subproblem is unbounded, as selecting arbitrary large value of s_i makes the values of the dual objective arbitrary small. On the other side, for $\lambda_i \geq 0$ the assignment

$s_i = 0$ is optimal. Therefore, Equation (5.70) can be equivalently rewritten as

$$\max_{\boldsymbol{\lambda}\geq 0} \min_{\mathbf{x}\in P} f(\mathbf{x}) + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle \ , \tag{5.70}$$

thus it coincides with the dual obtained by dualizing the initial inequality.

The above considerations prove the following technical statement:

Lemma 5.16. Lagrange relaxations with and without usage of the slack variables are equivalent, *i.e.*,

$$\begin{aligned}\max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}\in P} &\langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d}\rangle \\ &= \max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}}} \min_{\substack{\mathbf{x}\in P\\ \mathbf{s}\geq 0}} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} + \mathbf{s} - \mathbf{d}\rangle \ . \end{aligned} \tag{5.71}$$

Moreover, the sets of optimal $\boldsymbol{\lambda}$, $\boldsymbol{\mu}$ and $\mathbf{x}$ for both formulations coincide also.

In general, usage of the slack variables for linear f allows to profit from the reparametrization, as in this case the equality constraint is dualized. However, this profit comes with a hitch: Usage of slack variables changes the problem structure and the primal heuristics addressing $\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x}\rangle$ may not be applicable to $\min_{\substack{\mathbf{x}\in P\\ \mathbf{s}\geq 0}} \langle \mathbf{c}, \mathbf{x}\rangle$.

5.2.3 Primal-dual optimality conditions

Consider now a general *not necessarily convex* problem

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \tag{5.72}$$

$$\text{s.t. } h_i(\mathbf{x}) = 0, \ i \in [m_1] \tag{5.73}$$

$$q_i(\mathbf{x}) \leq 0, \ i \in [m_2]\,. \tag{5.74}$$

Its Lagrangean reads:

$$\begin{aligned} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) &= f(\mathbf{x}) + \sum_{i=1}^{m_1} \lambda_i h_i(\mathbf{x}) + \sum_{i=1}^{m_2} \mu_i q_i(\mathbf{x}) \\ &= f(\mathbf{x}) + \langle \boldsymbol{\lambda}, \mathbf{h}(\mathbf{x})\rangle + \langle \boldsymbol{\mu}, \mathbf{q}(\mathbf{x})\rangle \ , \end{aligned} \tag{5.75}$$

where $\mathbf{h}$ and $\mathbf{q}$ are the respective vector-functions.

The corresponding Lagrange dual problem is

$$\max_{\substack{\boldsymbol{\lambda} \\ \boldsymbol{\mu} \geq 0}} \left[g(\boldsymbol{\lambda}, \boldsymbol{\mu}) := \min_{\mathbf{x} \in \mathbb{R}^n} L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \right] . \tag{5.76}$$

In this section we consider optimality conditions and their usage in optimization. Our exposition closely follows [23, Sec. 5.5].

Certificate of optimality and stopping criteria

Any dual feasible solution $(\boldsymbol{\lambda}, \boldsymbol{\mu})$, $\boldsymbol{\mu} \geq 0$, establishes a lower bound for the optimal value p^* of the primal problem (5.72): $p^* \geq g(\boldsymbol{\lambda}, \boldsymbol{\mu})$. One says that $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ provides a *proof* or *certificate* that $p^* \geq g(\boldsymbol{\lambda}, \boldsymbol{\mu})$. Should $p^* = g(\boldsymbol{\lambda}, \boldsymbol{\mu})$, one speaks about *optimality certificate*. Dual suboptimal points allow to estimate how good is the primal solution $\mathbf{x}$ without knowing p^*. For example, $\mathbf{x}$ is ϵ-close to the optimum (in terms of the respective objective), if $f(\mathbf{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \epsilon$ and, therefore:

$$f(\mathbf{x}) - p^* \leq f(\mathbf{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \epsilon \, . \tag{5.77}$$

This condition also proves that the dual solution $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is ϵ-close to the (dual) optimum, as

$$d^* - g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq f(\mathbf{x}) - g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \epsilon \, . \tag{5.78}$$

This and similar conditions can be used as a stopping criteria for optimizaiton.

Complementary slackness

Let $\mathbf{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be primal and dual optima of (5.72) and primal and dual optimal objective values are equal. Then

$$\begin{aligned} f(\mathbf{x}^*) &= g(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) && (5.79) \\ &= \min_{x \in \mathbb{R}^n} \left(f(\mathbf{x}) + \langle \boldsymbol{\lambda}^*, \mathbf{h}(\mathbf{x}) \rangle + \langle \boldsymbol{\mu}^*, \mathbf{q}(\mathbf{x}) \rangle \right) && (5.80) \\ &\leq f(\mathbf{x}^*) + \langle \boldsymbol{\lambda}^*, \mathbf{h}(\mathbf{x}^*) \rangle + \langle \boldsymbol{\mu}^*, \mathbf{q}(\mathbf{x}^*) \rangle && (5.81) \\ &\overset{\mathbf{h}(\mathbf{x}^*)=0,\ \mathbf{q}(\mathbf{x}^*) \leq 0,\ \boldsymbol{\mu}^* \geq 0}{\leq} f(\mathbf{x}^*) && (5.82) \end{aligned}$$

The last inequality follows from feasibility of primal and dual solutions.

We conclude that both inequalities in (5.79) hold with equality and, therefore:

- $\mathbf{x}^*$ is a minimizer of $L(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ over $\mathbf{x}$;
- $\langle \boldsymbol{\mu}^*, \mathbf{q}(\mathbf{x}^*) \rangle = 0$, and since $\mu_i^* \geq 0$ and $q_i(\mathbf{x}^*) \leq 0$ it holds

$$\mu_i^* \cdot q_i(\mathbf{x}^*) = 0\,, \;\; i \in [m_2] \tag{5.83}$$

  It implies that either the respective inequality holds as equality or the corresponding dual variable is zero:

$$\mu_i^* > 0 \Rightarrow q_i(\mathbf{x}^*) = 0\,, \tag{5.84}$$
$$q_i(\mathbf{x}^*) < 0 \Rightarrow \mu_i^* = 0\,. \tag{5.85}$$

Karush-Kuhn-Tucker (KKT) optimality conditions

Now we assume that f, h_i, $i \in [m_1]$ and q_i, $i \in [m_2]$ and, therefore, L are differentiable.

Non-convex case. Let as above $\mathbf{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ be primal and dual optima of (5.72) and primal and dual optimal values are equal. Since $\mathbf{x}^*$ minimizes $L(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ over $\mathbf{x}$, the gradient of L must vanish:

$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{m_1} \lambda_i^* \nabla h_i(\mathbf{x}^*) + \sum_{i=1}^{m_2} \mu_i^* \nabla q_i(\mathbf{x}^*) = 0\,. \tag{5.86}$$

Thus, we have

$$h_i(\mathbf{x}^*) = 0\,, \;\; i \in [m_1] \tag{5.87}$$
$$q_i(\mathbf{x}^*) \leq 0\,, \;\; i \in [m_2] \tag{5.88}$$
$$\mu_i^* \geq 0\,, \;\; i \in [m_2] \tag{5.89}$$
$$\mu_i^* q_i(\mathbf{x}^*) = 0\,, \;\; i \in [m_2] \tag{5.90}$$
$$\nabla f(\mathbf{x}^*) + \sum_{i=1}^{m_1} \lambda_i^* \nabla h_i(\mathbf{x}^*) + \sum_{i=1}^{m_2} \mu_i^* \nabla q_i(\mathbf{x}^*) = 0\,, \tag{5.91}$$

which are called *Karush-Kuhn-Tucker (KKT)* conditions.

To summarize, for *any* optimization problem with differentiable objective and constraint functions for which strong duality holds, any pair of primal and dual optimal points must satisfy the KKT conditions (5.87). In other words, KKT conditions are *necessary for optimality* in this case.

Convex case. When the problem (5.72) is convex, the KKT conditions are not only necessary, but also sufficient for primal-dual optimality *and* strong duality.

Theorem 5.17. Let problem (5.72) be convex, that is h_i is affine and f, q_i are convex, see Exercise 4.9. Let also $\hat{\mathbf{x}}$, $\hat{\boldsymbol{\lambda}}$, $\hat{\boldsymbol{\mu}}$ be any points that satisfy the KKT conditions (5.87) in place of $\mathbf{x}^*$, $\boldsymbol{\lambda}^*$, $\boldsymbol{\mu}^*$ respectively.

Then $\hat{\mathbf{x}}$, $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}})$ are primal and dual optima, with zero duality gap.

Proof. The first two KKT conditions state that $\hat{\mathbf{x}}$ is feasible. Since $\hat{\boldsymbol{\mu}} \geq 0$, $L(\mathbf{x}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}})$ is convex in $\mathbf{x}$. The last KKT condition states that gradient of $L(\mathbf{x}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}})$ w.r.t. $\mathbf{x}$ vanishes at $\hat{\mathbf{x}}$, thus $\hat{\mathbf{x}}$ minimizes $L(\mathbf{x}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}})$ over $\mathbf{x}$. Therefore

$$g(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) = L(\hat{\mathbf{x}}, \hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}}) \tag{5.92}$$

$$= f(\hat{\mathbf{x}}) + \left\langle \hat{\boldsymbol{\lambda}}, \mathbf{h}(\hat{\mathbf{x}}) \right\rangle + \langle \hat{\boldsymbol{\mu}}, \mathbf{q}(\hat{\mathbf{x}}) \rangle \tag{5.93}$$

$$\overset{h_i(\hat{\mathbf{x}})=0,\ \hat{\mu}_i q_i(\hat{\mathbf{x}})=0}{=} f(\hat{\mathbf{x}}) \tag{5.94}$$

This shows that $\hat{\mathbf{x}}$ and $(\hat{\boldsymbol{\lambda}}, \hat{\boldsymbol{\mu}})$ have zero duality gap and, therefore, are primal and dual optima. □

Corollary 5.18 (Dual optimality condition)**.** Vectors $\boldsymbol{\lambda}^*$ and $\boldsymbol{\mu}^* \geq 0$ are an optimum of the dual problem (5.44) if and only if there exists

$$\mathbf{x}^* \in \operatorname*{arg\,min}_{\mathbf{x} \in \operatorname{conv}(P \cap \{0,1\}^n)} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}^* + B^\top \boldsymbol{\mu}^*, \mathbf{x} \right\rangle \tag{5.95}$$

such that $A\mathbf{x}^* = \mathbf{b}$, $B\mathbf{x}^* \leq \mathbf{d}$ and $\langle \boldsymbol{\mu}^*, B\mathbf{x}^* - \mathbf{d} \rangle = 0$. Such an $\mathbf{x}^*$ is a solution of the relaxed primal problem (5.53).

Proof. Denote

$$L(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = \iota_{\mathrm{conv}(P \cap \{0,1\}^n)}(\mathbf{x}) + \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d} \rangle \ , \tag{5.96}$$

where ι is defined as in (4.3). In other words, we consider the convex case with $f(\mathbf{x}) = \iota_{\mathrm{conv}(P \cap \{0,1\}^n)}(\mathbf{x}) + \langle \mathbf{c}, \mathbf{x} \rangle$.

Sufficiency: Condition (5.95) implies that $\mathbf{x}^*$ minimizes $L(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ over $\mathbf{x}$ and applying the same transformation as in (5.92) and taking into account feasibility of $\mathbf{x}^*$, we obtain that $\mathbf{x}^*$ and $(\boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ are primal and dual optimal solutions.

Necessity: Theorem 5.11 implies that there exists an optimal primal solution $\mathbf{x}^*$ of the relaxed primal problem (5.53) such that the strong duality holds. Then from (5.79) it follows that $\mathbf{x}^*$ is a minimizer of $L(\mathbf{x}, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*)$ and, therefore, of (5.95), over $\mathbf{x}$. Optimality of $\mathbf{x}^*$ implies its feasibility for the primal problem (5.53). Additionally, Equation (5.79) implies the complementary slackness. □

Remark 5.19. If $\mathbf{x}^*$ in Corollary 5.18 is binary, *i.e.*, $\mathbf{x}^* \in \{0,1\}^n$, then $\mathbf{x}^*$ is a solution of the non-relaxed ILP problem (5.43). This directly follows from the fact that problem (5.53) is a relaxation of (5.43).

Example 5.20 (Binary knapsack problem)**.** Let us get back to the Lagrange dual (5.49) of the binary knapsack problem and exchange the feasible set $\{0,1\}^n$ with its convex hull $[0,1]^n$, corresponding to $\mathrm{conv}(P \cap \{0,1\}^n)$ in (5.95):

$$\min_{\mu \geq 0} \max_{\mathbf{x} \in [0,1]^n} \langle \mathbf{c}, \mathbf{x} \rangle - \mu(\langle \mathbf{w}, \mathbf{x} \rangle - W) = \min_{\mu \geq 0} \mu W + \langle \mathbf{c} - \mu \mathbf{w}, \mathbf{x}^*(\mu) \rangle \ , \tag{5.97}$$

where

$$x^*(\mu)_i = \begin{cases} 1, & c_i - \mu w_i > 0 \\ 0, & c_i - \mu w_i < 0 \\ [0,1], & c_i - \mu w_i = 0 \, . \end{cases} \tag{5.98}$$

Note that now we separately treat the case $c_i - \mu w_i = 0$, as the relaxed primal variable $x^*(\mu)_i$ may take arbitrary value between zero and one in this case. Considering the same assumptions about c_i and w_i as in (5.51), we now rewrite it for $\mu = \frac{c_k}{w_k}$ as

$$g(\mu) = \mu W + \sum_{i=1}^{k-1}(c_i - \mu w_i) + (c_k - \mu w_k)x_k^* \\ = \mu(W - \sum_{i=1}^{k-1} w_i - w_k x_k^*) + \sum_{i=1}^{k-1} c_i + c_k x_k^* . \quad (5.99)$$

According to Corollary 5.18 $\mu \geq 0$ is optimal, if $\mu(W - \sum_{i=1}^{k-1} w_i - w_k x_k^*) = 0$. In turn, $\mu \neq 0$ implies $W - \sum_{i=1}^{k-1} w_i - w_k x_k^* = 0$. This allows to compute k as the largest one for which $\sum_{i=1}^{k-1} w_i \leq W$ and $x_k^* = (W - \sum_{i=1}^{k-1} w_i)/w_k$.

5.3 Explicit dual constraints, linear program duality

Consider the Lagrange dual (5.76). According to Definition 4.2

$$\operatorname{dom} g = \{(\boldsymbol{\lambda}, \boldsymbol{\mu}) \colon g(\boldsymbol{\lambda}, \boldsymbol{\mu}) > -\infty\} . \quad (5.100)$$

Since the dual g is to be maximized afterwards, points $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ outside $\operatorname{dom} g$ can be explicitly excluded from optimization, as we have already done it in (5.70), when considering dualization of inequalities via usage of slack variables. Below we will treat another important case of a full Lagrange dual for linear programs. The word *full* points out that *all* constraints of the problem are dualized.

Consider

$$\min_{\substack{\mathbf{x}\in\mathbb{R}^n \\ A\mathbf{x}=\mathbf{b} \\ B\mathbf{x}\leq\mathbf{d}}} \langle \mathbf{c}, \mathbf{x}\rangle \geq \max_{\substack{\boldsymbol{\lambda} \\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}\in\mathbb{R}^n} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle + \langle \boldsymbol{\mu}, B\mathbf{x} - \mathbf{d}\rangle \\ = \max_{\substack{\boldsymbol{\lambda} \\ \boldsymbol{\mu}\geq 0}} -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle - \langle \boldsymbol{\mu}, \mathbf{d}\rangle + \min_{\mathbf{x}\in\mathbb{R}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda} + B^\top \boldsymbol{\mu}, \mathbf{x} \right\rangle \quad (5.101)$$

The minimization subproblem above is bounded only if $\mathbf{c} + A^\top \boldsymbol{\lambda} + B^\top \boldsymbol{\mu} = 0$, hence we can add this condition as an explicit constraint:

$$= \max_{\boldsymbol{\lambda},\boldsymbol{\mu}} -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle - \langle \boldsymbol{\mu}, \mathbf{d}\rangle \tag{5.102}$$

$$\text{s.t. } \mathbf{c} + A^\top \boldsymbol{\lambda} + B^\top \boldsymbol{\mu} = 0 \tag{5.103}$$

$$\boldsymbol{\mu} \geq 0 \tag{5.104}$$

The term $(-\langle \boldsymbol{\lambda}, \mathbf{b}\rangle - \langle \boldsymbol{\mu}, \mathbf{d}\rangle)$ is referred, therefore, to as *dual LP objective* and (5.103)-(5.104) as *constraints of the dual LP*. Note that the full dual to a linear program is a linear program itself.

LP in standard form As a special case consider now LP in the *standard* form (see Section 7.4.1 for different forms of linear programs):

$$= \min_{\mathbf{x}\in\mathbb{R}^n} \langle \mathbf{c}, \mathbf{x}\rangle \tag{5.105}$$

$$\text{s.t. } A\mathbf{x} = \mathbf{b}$$

$$\mathbf{x} \geq 0$$

According to (5.102)-(5.104), the respective *dual LP* takes the form

$$\max_{\boldsymbol{\lambda},\boldsymbol{\mu}} -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle \tag{5.106}$$

$$\text{s.t. } \mathbf{c} + A^\top \boldsymbol{\lambda} - \boldsymbol{\mu} = 0$$

$$\boldsymbol{\mu} \geq 0$$

We can additionally eliminate $\boldsymbol{\mu}$ and obtain

$$\max_{\boldsymbol{\lambda}} -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle \tag{5.107}$$

$$\text{s.t. } \mathbf{c} + A^\top \boldsymbol{\lambda} \geq 0\,,$$

which is an LP in the *inequality* form.

LP in the inequality form Similarly, consider the linear program in the inequality form:

$$\begin{aligned} &\min_{\mathbf{x}\in\mathbb{R}^n} \langle \mathbf{c}, \mathbf{x}\rangle \\ &\text{s.t. } B\mathbf{x} \leq \mathbf{d} \end{aligned} \tag{5.108}$$

Comparing to (5.102)-(5.104) we conclude that the dual problem takes the form

$$\begin{aligned} &\max_{\boldsymbol{\mu}} - \langle \boldsymbol{\mu}, \mathbf{d}\rangle \\ &\text{s.t. } B^\top \boldsymbol{\mu} = -\mathbf{c} \\ &\qquad \boldsymbol{\mu} \geq 0\,, \end{aligned} \tag{5.109}$$

which, in turn, is an LP in the standard form.

Note the symetry between forms of the respective primal and dual linear programs: Dual of an LP in the standard form is an LP in the inequality form and vice versa.

Exercise 5.21. Compute the dual LP for (5.107) and (5.109). Show that they are equal to(5.105) and (5.108) respectively, which means that *the dual of the dual LP is the primal LP.*

Example 5.22. Consider

$$\begin{aligned} &\min_{\mathbf{x}} 3x_1 + 5x_2 + 7x_3 \\ &\text{s.t.} \quad x_1 + 2x_2 = 0 \quad |\lambda_1 \\ &\qquad x_1 + x_2 + x_3 = 1 \quad |\lambda_2 \\ &\qquad\qquad x_1 \geq 0 \quad |\mu_1 \\ &\qquad\qquad x_2 \geq 0 \quad |\mu_2 \\ &\qquad\qquad x_3 \geq 0 \quad |\mu_3 \end{aligned} \tag{5.110}$$

The dual problem is equal to:

$$\max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \min_{\mathbf{x}} \Big[3x_1 + 5x_2 + 7x_3 + \lambda_1(x_1 + 2x_2) + \lambda_2(x_1 + x_2 + x_3 - 1) - \mu_1 x_1 - \mu_2 x_2 - \mu_3 x_3\Big]$$

$$= \max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} \Big[-\lambda_2 + \min_{\mathbf{x}} x_1(3 + \lambda_1 + \lambda_2 - \mu_1) + x_2(5 + 2\lambda_1 + \lambda_2 - \mu_2) + x_3(7 + \lambda_2 - \mu_3)\Big]$$

$$= \max_{\substack{\boldsymbol{\lambda}\\ \boldsymbol{\mu}\geq 0}} -\lambda_2$$
$$\text{s.t. } 3 + \lambda_1 + \lambda_2 - \mu_1 = 0 \tag{5.111}$$
$$5 + 2\lambda_1 + \lambda_2 - \mu_2 = 0$$
$$7 + \lambda_2 - \mu_3 = 0$$
$$= \max_{\boldsymbol{\lambda}} -\lambda_2$$
$$\text{s.t. } 3 + \lambda_1 + \lambda_2 \geq 0 \tag{5.112}$$
$$5 + 2\lambda_1 + \lambda_2 \geq 0$$
$$7 + \lambda_2 \geq 0$$

Consider now the dual LP (5.112) and compute the full dual to it:

$$\begin{aligned} \max_{\boldsymbol{\lambda}} & -\lambda_2 \\ \text{s.t. } & 3 + \lambda_1 + \lambda_2 \geq 0 \quad |x_1 \\ & 5 + 2\lambda_1 + \lambda_2 \geq 0 \quad |x_2 \\ & 7 + \lambda_2 \geq 0 \quad |x_3 \end{aligned} \tag{5.113}$$

$$\leq \min_{\mathbf{x}\geq 0} \max_{\boldsymbol{\lambda}} \Big[-\lambda_2 + x_1(3 + \lambda_1 + \lambda_2) + x_2(5 + 2\lambda_1 + \lambda_2) + x_3(7 + \lambda_2)\Big]$$
$$= \min_{\mathbf{x}\geq 0} \Big[3x_1 + 5x_2 + 7x_3 + \max_{\boldsymbol{\lambda}} \lambda_1(x_1 + 2x_2) + \lambda_2(x_1 + x_2 + x_3 - 1)\Big]$$
$$= \min_{\mathbf{x}\geq 0} 3x_1 + 5x_2 + 7x_3$$
$$\text{s.t. } x_1 + 2x_2 = 0 \tag{5.114}$$
$$x_1 + x_2 + x_3 = 1\,, \tag{5.115}$$

which is the initial problem (5.110).

5.4 Bibliography and further reading

For an extended introduction to the duality theory we recommend the excellent textbook on convex optimization [23]. It also details the notion of the extended value functions and their relation to convex analysis and Lagrange duality. More general and deep analysis of the mathematical phenomenon of duality is presented in [24].

5.5 Case study: Lagrange dual of the labeling problem

Below we derive the Lagrangean dual of the local polytope linearization of the labeling problem (3.44). This dual is obtained by relaxing the coupling constraints of the local polytope. Based on the theoretical background provided in Chapter 5 we will analyze properties of the dual.

The second important topic we deal with in this chapter is the optimality conditions for the considered dual problem. We will give exact (necessary and sufficient) as well as approximate (only necessary) optimality conditions. Whereas checking the first ones can be as computationally expensive as optimizing the dual itself, the latter can be verified with a simple and efficient algorithm.

In the last subsection we return to the acyclic labeling problems, and show that the approximate optimality conditions are exact for them.

5.5.1 Reparametrization and Lagrange dual

Recall the local polytope linearization (3.44) of the labeling problem

$$\min_{\boldsymbol{\mu} \in \mathcal{L} \cap \{0,1\}^{\mathcal{I}}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle \tag{5.116}$$

with the local polytope $\mathcal{L}$ defined by (3.43). Note that $\mathcal{L}$ can be rewritten as follows:

$$\mathcal{L} = \begin{cases} \boldsymbol{\mu}_u \in \Delta^{\mathcal{Y}_u}, & \forall u \in \mathcal{V} \\ \boldsymbol{\mu}_{uv} \in \Delta^{\mathcal{Y}_{uv}}, & \forall uv \in \mathcal{E} \\ \sum_{t \in \mathcal{Y}_v} \mu_{uv}(s,t) = \mu_u(s), & \forall u \in \mathcal{V},\ v \in \mathcal{N}(u),\ s \in \mathcal{Y}_u\,. \end{cases} \tag{5.117}$$

As in Section 3.3, the notation $\boldsymbol{\mu}_u$ stands for the vector $(\mu_u(s)\colon s \in \mathcal{Y}_u)$, which encodes the selected label in the node u, and $\boldsymbol{\mu}_{uv} = (\mu_{uv}(s,t)\colon (s,t) \in \mathcal{Y}_{uv})$ encodes the selected label pair in the edge uv. Let us construct the Lagrange dual to (5.116) by relaxing the coupling constraints (the last line in (5.117)). This is done similarly to the general scheme provided in Section 5.1.

Consider the linear term of the objective, which corresponds to dualizing the coupling constraints. In other words, we will specify the term $\langle \boldsymbol{\lambda}, A\mathbf{x} \rangle$ corresponding to the constraints $A\mathbf{x} = 0$, when the role of the latter is played by the coupling constraints. Since for each node $u \in \mathcal{V}$ there is one constraint for each of its labels $s \in \mathcal{Y}_u$ and each neighboring node $v \in \mathcal{N}(u)$, the role of the dual vector $\boldsymbol{\lambda}$ is played by the vector $\boldsymbol{\phi} \in \mathbb{R}^{\mathcal{J}}$ with coordinates $\phi_{u,v}(s)$, where $\mathcal{J} := \{(u,v,s) \mid u \in \mathcal{V},\ v \in \mathcal{N}(u),\ s \in \mathcal{Y}_u\}$. The comma between u and v in the lower index u,v underlines that $\phi_{u,v}(s)$ and $\phi_{v,u}(s)$ are two different values, contrary to $\mu_{uv}(s,t)$ and $\mu_{vu}(t,s)$, where u and v are not separated by the comma.

Given the above notation, the considered linear term corresponding to the coupling constraints reads:

$$\sum_{u \in \mathcal{V}} \sum_{v \in \mathcal{N}(u)} \sum_{s \in \mathcal{Y}_u} \phi_{u,v}(s) \left(\sum_{t \in \mathcal{Y}_v} \mu_{uv}(s,t) - \mu_u(s) \right). \tag{5.118}$$

By adding it to the objective

$$\langle \theta, \mu \rangle = \sum_{u \in \mathcal{V}} \sum_{s \in \mathcal{Y}_u} \theta_u(s)\mu_u(s) + \sum_{uv \in \mathcal{E}} \sum_{(s,t) \in \mathcal{Y}_{uv}} \theta_{uv}(s,t)\mu_{uv}(s,t) \tag{5.119}$$

and regrouping terms one obtains

$$\sum_{u\in\mathcal{V}}\sum_{s\in\mathcal{Y}_u}\mu_u(s)\left(\theta_u(s)-\sum_{v\in\mathcal{N}(u)}\phi_{u,v}(s)\right)$$
$$+\sum_{uv\in\mathcal{E}}\sum_{(s,t)\in\mathcal{Y}_{uv}}\mu_{uv}(s,t)\left(\theta_{uv}(s,t)+\phi_{u,v}(s)+\phi_{v,u}(t)\right)=\left\langle\boldsymbol{\theta}^{\boldsymbol{\phi}},\boldsymbol{\mu}\right\rangle\,, \tag{5.120}$$

where we introduced the *reparametrized costs* $\boldsymbol{\theta}^{\boldsymbol{\phi}}$ defined as

$$\theta^{\phi}_u(s):=\theta_u(s)-\sum_{v\in\mathcal{N}(u)}\phi_{u,v}(s),\ v\in\mathcal{V},\ s\in\mathcal{Y}_u\,, \tag{5.121}$$
$$\theta^{\phi}_{uv}(s,t):=\theta_{uv}(s,t)+\phi_{u,v}(s)+\phi_{v,u}(t)\,,\ uv\in\mathcal{E},\ (s,t)\in\mathcal{Y}_{uv}\,.$$

See Figure 5.2 for an illustration of the reparametrization.

Note that the reparametrized Lagrangean $\left\langle\boldsymbol{\theta}^{\boldsymbol{\phi}},\boldsymbol{\mu}\right\rangle$ is an instance of the reparametrization introduced in Section 5.2.2 for general (integer) linear programs, see Expression (5.61). In other words, the Lagrangian dual constructed by relaxing the coupling constraints consists in finding an optimal reparametrization:

$$\max_{\boldsymbol{\phi}\in\mathbb{R}^{\mathcal{J}}}\ \underbrace{\min_{\substack{\boldsymbol{\mu}\in\{0,1\}^{\mathcal{I}}\\ \boldsymbol{\mu}_u\in\Delta^{\mathcal{Y}_u},u\in\mathcal{V}\\ \boldsymbol{\mu}_{uv}\in\Delta^{\mathcal{Y}_{uv}},uv\in\mathcal{E}}}\left\langle\boldsymbol{\theta}^{\boldsymbol{\phi}},\boldsymbol{\mu}\right\rangle}_{\text{dual function }\mathcal{D}(\boldsymbol{\phi})}\,. \tag{5.122}$$

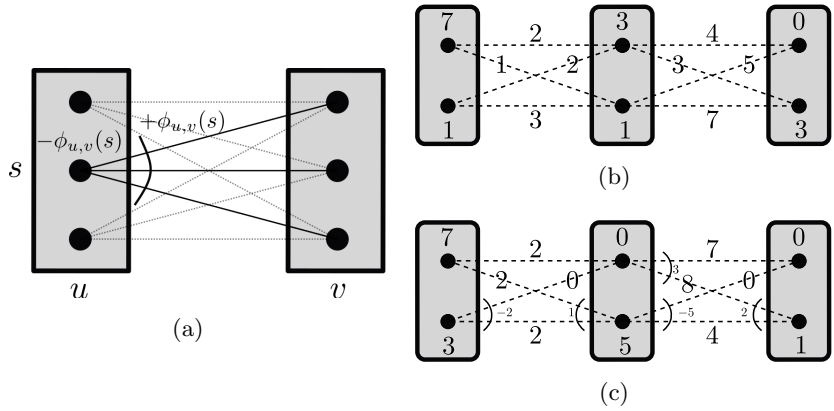


Figure 5.2: **(a)** Illustration of the reparametrization. Energy of all labelings remains the same when the value $\phi_{u,v}(s)$ is subtracted from the cost $\theta_u(s)$ of the label s in the node u and simultaneously added to the costs of all label pairs (s,t), $t \in \mathcal{Y}_v$, associated with one of the edges incident to u. **(b-c)** Example of the reparametrization: (b) initial costs, (c) reparametrized costs. Bigger numbers stand for the costs (initial in (b) and reparametrized in (c)), smaller numbers in (c) define values of the reparametrization vector ϕ with the same meaning as in (a).

Note that the constraints on $\boldsymbol{\mu}$ decouple into the coordinates indexed by u and uv:

$$\begin{aligned}
&\min_{\substack{\boldsymbol{\mu}\in\{0,1\}^{\mathcal{I}} \\ \boldsymbol{\mu}_u\in\Delta^{\mathcal{Y}_u},u\in\mathcal{V} \\ \boldsymbol{\mu}_{uv}\in\Delta^{\mathcal{Y}_{uv}},uv\in\mathcal{E}}} \left[\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}},\boldsymbol{\mu}\right\rangle = \sum_{u\in\mathcal{V}}\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_u,\boldsymbol{\mu}_u\right\rangle + \sum_{uv\in\mathcal{E}}\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_{uv},\boldsymbol{\mu}_{uv}\right\rangle\right] \\
&= \min_{\substack{\boldsymbol{\mu}_u\in\Delta^{\mathcal{Y}_u}\cap\{0,1\}^{\mathcal{Y}_u} \\ u\in\mathcal{V}}} \sum_{u\in\mathcal{V}}\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_u,\boldsymbol{\mu}_u\right\rangle + \min_{\substack{\boldsymbol{\mu}_{uv}\in\Delta^{\mathcal{Y}_{uv}}\cap\{0,1\}^{\mathcal{Y}_{uv}} \\ uv\in\mathcal{E}}} \sum_{uv\in\mathcal{E}}\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_{uv},\boldsymbol{\mu}_{uv}\right\rangle \\
&= \sum_{u\in\mathcal{V}} \min_{\boldsymbol{\mu}_u\in\Delta^{\mathcal{Y}_u}\cap\{0,1\}^{\mathcal{Y}_u}}\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_u,\boldsymbol{\mu}_u\right\rangle + \sum_{uv\in\mathcal{E}} \min_{\boldsymbol{\mu}_{uv}\in\Delta^{\mathcal{Y}_{uv}}\cap\{0,1\}^{\mathcal{Y}_{uv}}}\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_{uv},\boldsymbol{\mu}_{uv}\right\rangle \\
&= \sum_{u\in\mathcal{V}} \min_{s\in\mathcal{Y}_u} \theta^{\boldsymbol{\phi}}_u(s) + \sum_{uv\in\mathcal{E}} \min_{(s,t)\in\mathcal{Y}_{uv}} \theta^{\boldsymbol{\phi}}_{uv}(s,t)\,. \qquad (5.123)
\end{aligned}$$

Therefore, the dual problem (5.122) takes the form

$$\max_{\boldsymbol{\phi}\in\mathbb{R}^{\mathcal{J}}} \mathcal{D}(\boldsymbol{\phi}) := \max_{\boldsymbol{\phi}\in\mathbb{R}^{\mathcal{J}}} \left(\sum_{u\in\mathcal{V}} \min_{s\in\mathcal{Y}_u} \theta_u^{\boldsymbol{\phi}}(s) + \sum_{uv\in\mathcal{E}} \min_{(s,t)\in\mathcal{Y}_{uv}} \theta_{uv}^{\boldsymbol{\phi}}(s,t) \right). \tag{5.124}$$

Note that to compute the value of the dual objective in (5.124) for a fixed $\boldsymbol{\phi}$ one has to select *independently* in each node and each edge an optimal label and an optimal label pair, respectively. Such labels and label pairs will be called *locally optimal* in the following. Note also that these locally optimal labels and label pairs need not be consistent. In other words, if the label s is selected in node u and the label pair (t,t') in edge uv, this does not generally imply that $s=t$. However, as we will show later in this chapter, some kind of relaxed consistency (known as *arc-consistency*) is enforced in the optimum of the dual function.

Example 5.23 (How large-scale is the Lagrangean dual of the labeling problem?)**.** Consider now the Lagrangean dual (5.124). The number of variables is equal to $|\mathcal{J}|$, that grows linearly with the number of labels and the number of edges in a graph. This is in contrast to the primal problem (5.116), where the number of variables grows quadratically with the number of labels.

Consider Example 3.6, where the size of the ILP representation of the labeling problem was estimated for the depth reconstruction problem from Example 1.30. In that case the dual problem has $O(10^5)$ variables, which is an order of magnitude less than the size $O(10^6)$ of the primal problem.

The following proposition summarizes properties of the dual problem, which specialize the general properties of the Lagrange relaxation from Chapter 5.

Proposition 5.24. For the dual problem (5.124) it holds that:

1. $\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu} \rangle = \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$ holds for any $\boldsymbol{\phi} \in \mathbb{R}^{\mathcal{J}}$ and $\boldsymbol{\mu} \in \mathcal{L}$ (in particular for $\boldsymbol{\mu} \in \mathcal{L} \cap \{0,1\}^{\mathcal{I}}$ and $\boldsymbol{\mu} \in \mathcal{M}$). This implies that for any

labeling $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$ it holds that $E(\mathbf{y};\boldsymbol{\theta}) = E(\mathbf{y};\boldsymbol{\theta}^{\boldsymbol{\phi}})$ with E being the energy of $\mathbf{y}$ as defined in (1.18).

2. $\mathcal{D}(\boldsymbol{\phi})$ is a lower bound for optimal energy, i.e. $\mathcal{D}(\boldsymbol{\phi}) \leq \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$ for any $\boldsymbol{\mu} \in \mathcal{L} \cap \{0,1\}^\mathcal{I}$ and $\boldsymbol{\phi} \in \mathbb{R}^\mathcal{J}$.

3. $\mathcal{D}$ is concave piecewise linear non-differentiable function.

4. The primal relaxed problem corresponding to (5.124) is the local polytope relaxation, i.e. $\max_{\boldsymbol{\phi} \in \mathbb{R}^\mathcal{J}} \mathcal{D}(\boldsymbol{\phi}) = \min_{\boldsymbol{\mu} \in \mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$.

5. The optimality condition for the dual $\mathcal{D}(\boldsymbol{\phi})$ reads:

$$\boldsymbol{\phi} \in \operatorname*{arg\,max}_{\boldsymbol{\phi}' \in \mathbb{R}^\mathcal{J}} \mathcal{D}(\boldsymbol{\phi}')$$

if and only if the polytope

$$\mathcal{L}(\boldsymbol{\phi}) := \Big\{ \boldsymbol{\mu} \in \mathcal{L} \colon \mu_w(s) = 0 \text{ if } \theta_w^{\boldsymbol{\phi}}(s) > \min_{s' \in \mathcal{Y}_w} \theta_w^{\boldsymbol{\phi}}(s'), \\ w \in \mathcal{V} \cup \mathcal{E},\ s \in \mathcal{Y}_w \big\} \quad (5.125)$$

is non-empty. In other words, there must be a relaxed labeling $\boldsymbol{\mu} \in \mathcal{L}$ with non-zero coordinates assigned only to locally optimal labels and label pairs. Such a relaxed labeling is also a solution to the local polytope relaxation $\min_{\boldsymbol{\mu} \in \mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$.

6. Tightness of the Lagrange dual: Let the set $\mathcal{L}(\boldsymbol{\phi})$ defined by (5.125) contain an integer labeling, in other words, there is $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$ such that $\delta(\mathbf{y}) \in \mathcal{L}(\boldsymbol{\phi})$. Then $\mathbf{y}$ is the solution of the (non-relaxed) energy minimization problem (1.18). In this case $\mathcal{D}(\boldsymbol{\phi}) = E(\mathbf{y};\boldsymbol{\theta})$. We will call this case *LP-tight*, since the Lagrange dual is equivalent to the local polytope relaxation.
This statement holds also in the opposite direction, that is, $\mathcal{D}(\boldsymbol{\phi}) = E(\mathbf{y};\boldsymbol{\theta})$ if and only if there is $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$ such that $\delta(\mathbf{y}) \in \mathcal{L}(\boldsymbol{\phi})$.

Proof.

1. Equality follows from the fact that for any $\boldsymbol{\mu} \in \mathcal{L}$ the coupling constraints hold, and, therefore, $\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu} \right\rangle$ is a reparametrization of $\langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$, as introduced in Section 5.2.2. The simpler special case of reparametrization for $\boldsymbol{\mu}$ being a labeling, i.e. $\boldsymbol{\mu} = \delta(\mathbf{y})$, $\mathbf{y} \in \mathcal{Y}_{\mathcal{V}}$, is illustrated in Figure 5.2.

2. This follows from Proposition 5.3, since $\mathcal{D}(\boldsymbol{\phi})$ is a Lagrangian relaxation of the labeling problem (5.116).

3. The dual is concave piecewise linear as any Lagrange dual of an integer linear program is. This also can be seen directly from (5.124): The reparametrized costs θ^{ϕ} are linear functions of ϕ according to (5.121), and the dual objective has also the representation (5.122) as a minimum over linear functions of ϕ.

4. It is sufficient to derive the dual from the local polytope relaxation similar to (5.123) and apply the strong duality Theorem 5.11:

$$\begin{aligned}
&\min_{\substack{\boldsymbol{\mu} \in \mathbb{R}^{\mathcal{I}} \\ \boldsymbol{\mu}_u \in \Delta^{\mathcal{Y}_u}, u \in \mathcal{V} \\ \boldsymbol{\mu}_{uv} \in \Delta^{\mathcal{Y}_{uv}}, uv \in \mathcal{E}}} \left[\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu} \right\rangle = \sum_{u \in \mathcal{V}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_u, \boldsymbol{\mu}_u \right\rangle + \sum_{uv \in \mathcal{E}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_{uv}, \boldsymbol{\mu}_{uv} \right\rangle \right] \\
&= \min_{\substack{\boldsymbol{\mu}_u \in \Delta^{\mathcal{Y}_u} \\ u \in \mathcal{V}}} \sum_{u \in \mathcal{V}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_u, \boldsymbol{\mu}_u \right\rangle + \min_{\substack{\boldsymbol{\mu}_{uv} \in \Delta^{\mathcal{Y}_{uv}} \\ uv \in \mathcal{E}}} \sum_{uv \in \mathcal{E}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_{uv}, \boldsymbol{\mu}_{uv} \right\rangle \\
&= \sum_{u \in \mathcal{V}} \min_{\boldsymbol{\mu}_u \in \Delta^{\mathcal{Y}_u}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_u, \boldsymbol{\mu}_u \right\rangle + \sum_{uv \in \mathcal{E}} \min_{\boldsymbol{\mu}_{uv} \in \Delta^{\mathcal{Y}_{uv}}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}_{uv}, \boldsymbol{\mu}_{uv} \right\rangle \\
&= \sum_{u \in \mathcal{V}} \min_{s \in \mathcal{Y}_u} \theta^{\boldsymbol{\phi}}_u(s) + \sum_{uv \in \mathcal{E}} \min_{(s,t) \in \mathcal{Y}_{uv}} \theta^{\boldsymbol{\phi}}_{uv}(s,t) \,. \qquad (5.126)
\end{aligned}$$

   The last equality is implied by Lemma 2.43.

5. The proof follows from Corollary 5.18. The condition (5.95) translates into

$$\boldsymbol{\mu} \in \operatorname*{arg\,min}_{\substack{\boldsymbol{\mu}'_u \in \Delta^{\mathcal{Y}_u}, u \in \mathcal{V} \\ \boldsymbol{\mu}'_{uv} \in \Delta^{\mathcal{Y}_{uv}}, uv \in \mathcal{E}}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu}' \right\rangle \,. \qquad (5.127)$$

Let $\boldsymbol{\phi} \in \mathbb{R}^{\mathcal{J}}$ be a dual optimum. Then according to Corollary 5.18 there exists $\boldsymbol{\mu}$ satisfying both (5.127) and the coupling constraints. It implies $\boldsymbol{\mu} \in \mathcal{L}$. To show that $\boldsymbol{\mu} \in \mathcal{L}(\boldsymbol{\phi})$ it remains to prove that $\mu_w(s) = 0$ if $s \notin \arg\min_{s \in \mathcal{Y}_w} \theta^{\boldsymbol{\phi}}_w(s)$. The latter holds due to

$$\min_{\substack{\boldsymbol{\mu}'_u \in \Delta^{\mathcal{Y}_u}, u \in \mathcal{V} \\ \boldsymbol{\mu}'_{uv} \in \Delta^{\mathcal{Y}_{uv}}, uv \in \mathcal{E}}} \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu}' \right\rangle = \sum_{w \in \mathcal{V} \cup \mathcal{E}} \min_{\boldsymbol{\mu}'_w \in \Delta^{\mathcal{Y}_w}} \sum_{s \in \mathcal{Y}_w} \mu'_w(s) \theta^{\boldsymbol{\phi}}_w(s) \tag{5.128}$$

and Lemma 2.43.

6. From Item 5 it follows that $\delta(\mathbf{y})$ is a minimizer of the local polytope relaxation. Since $\delta(\mathbf{y}) \in \{0,1\}^{\mathcal{I}}$ it also minimizes the non-relaxed problem (5.116) and, therefore, $\mathbf{y}$ is an optimal labeling. This also implies $\mathcal{D}(\boldsymbol{\phi}) = E(\mathbf{y}; \boldsymbol{\theta}) \equiv \langle \boldsymbol{\theta}, \delta(\mathbf{y}) \rangle$.

   To prove the statement in the opposite direction assume that $\mathcal{L}(\boldsymbol{\phi}) \neq \emptyset$, but that it does not contain any integer labeling, i.e. $\delta(\mathbf{y}) \notin \mathcal{L}(\boldsymbol{\phi})$ for any $\mathbf{y} \in \mathcal{Y}$. Due to Item 4 a vector $\boldsymbol{\mu} \in \mathcal{L}$ is an optimum of the local polytope relaxation if and only if $\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu} \right\rangle = \mathcal{D}(\boldsymbol{\phi})$. In turn, according to Item 5, the equality $\left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu} \right\rangle = \mathcal{D}(\boldsymbol{\phi})$ holds only for $\boldsymbol{\mu} \in \mathcal{L}(\boldsymbol{\phi})$. Since $\delta(\mathbf{y}) \notin \mathcal{L}(\boldsymbol{\phi})$, the vector $\delta(\mathbf{y})$ is not a solution of the relaxed problem. Therefore,

$$\mathcal{D}(\boldsymbol{\phi}) = \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \boldsymbol{\mu} \right\rangle < \left\langle \boldsymbol{\theta}^{\boldsymbol{\phi}}, \delta(\mathbf{y}) \right\rangle = E(\mathbf{y}; \boldsymbol{\theta}) \,, \tag{5.129}$$

   which is a contradiction.

□

5.5.2 Necessary optimality condition: Arc-consistency and node-edge agreement

As follows from Proposition 5.24(5), to determine whether a given dual vector $\boldsymbol{\phi}$ is optimal, it suffices to check whether the polytope $\mathcal{L}(\boldsymbol{\phi})$ defined in (5.125) is non-empty. Such kinds of problems, where it is necessary to verify whether a set is empty, are called *feasibility*

problems. In general, feasibility problems for linear programming are as difficult as linear programs themselves. In other words, checking whether a polyhedron is empty can be formulated as a linear program with a different polyhedron.

In practice, a simpler optimality condition is often desirable, which could be checked often enough, for example on each iteration of some iterative algorithm. In this section we formulate such a condition. The price for its simplicity is that it is only necessary for dual optimality and no longer sufficient. Informally, instead of looking for an integer or relaxed labeling consisting of locally optimal labels and label pairs, which would require a *global* reasoning, one may check whether the locally optimal labels and label pairs are *locally* consistent. Below, we give a formal definition of this simpler condition and an algorithm for its verification.

In the following, we will use two functions, which turn real-valued vectors from the primal space $\mathbb{R}^{\mathcal{I}}$ into binary ones:

- For any $\mu \in \mathbb{R}$ let $\operatorname{nz}[\mu] = [\![\mu \neq 0]\!]$ be the indicator function of μ being **n**on-**z**ero. When applied to a vector $\boldsymbol{\mu} \in \mathbb{R}^n$, it acts coordinate-wise, i.e. $\operatorname{nz}[\boldsymbol{\mu}]_i = \operatorname{nz}[\mu_i]$.
- For $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{I}}$ let $\operatorname{mi}[\boldsymbol{\theta}]$ be defined such that locally minimal labels and label pairs (w.r.t. $\boldsymbol{\theta}$) obtain the value 1 and others zero, i.e. $\operatorname{mi}[\boldsymbol{\theta}]_w(x_w) := [\![\theta_w(x_w) = \min_{x_w \in \mathcal{Y}_w} \theta_w(x_w)]\!]$ for $w \in \mathcal{V} \cup \mathcal{E}$. Here mi stands for **mi**n.

Example 5.25.

$$\begin{aligned}
&\operatorname{nz}[\delta(\mathbf{y})] = \delta(\mathbf{y});\\
&\operatorname{nz}[(0, 0.2, 0.8, 0)] = (0, 1, 1, 0);\\
&\operatorname{mi}[(0, -2, -1, -2, 3)] = (0, 1, 0, 1, 0);\\
&\operatorname{mi}[(0, -2), (1, 1, 2, 0), (7, -1)] = (0, 1), (0, 0, 0, 1), (0, 1).
\end{aligned}$$

Definition 5.26. A binary vector $\boldsymbol{\xi} \in \{0, 1\}^{\mathcal{I}}$ is called *arc-consistent* if

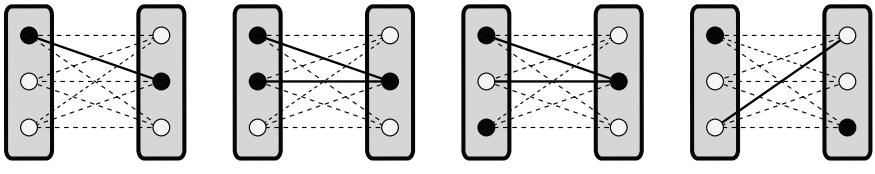


Figure 5.3: Illustration to the notions of arc-consistency and node-edge agreement (non-empty closure). Black circles and solid lines correspond to the non-zero coordinates of $\xi \in \{0,1\}^{\mathcal{I}}$. White circles and dashed lines to the zero-valued coordinates. **(a)** Strictly arc-consistent vector. **(b)** Arc-consistent ξ. **(c)** Not arc-consistent, however the node-edge agreement holds. **(d)** Not arc-consistent, no node-edge agreement.

1. $\xi_{uv}(s,t) = 1$ implies $\xi_u(s) = \xi_v(t) = 1$ for all $uv \in \mathcal{E}$, $(s,t) \in \mathcal{Y}_{uv}$, and
2. $\xi_u(s) = 1$ implies that for any $v \in \mathcal{N}(u)$ there exists $t \in \mathcal{Y}_v$ such that $\xi_{uv}(s,t) = 1$.

The set of arc-consistent vectors from $\{0,1\}^{\mathcal{I}}$ will be denoted as $\mathcal{AC}^{\mathcal{I}}$.

This definition as well as the following one are illustrated in Figure 5.3.

Definition 5.27. $\boldsymbol{\xi} \in \{0,1\}^{\mathcal{I}}$ is *strictly arc-consistent* if it is arc-consistent and $\sum_{s \in \mathcal{Y}_u} \xi_u(s) = 1$ for all $u \in \mathcal{V}$ as well as $\sum_{(s,t) \in \mathcal{Y}_{uv}} \xi_{uv}(s,t) = 1$ for all $uv \in \mathcal{E}$.

In other words, strict arc-consistency means that (i) for a single label in each node and a single label pair in each edge the corresponding coordinate of $\boldsymbol{\xi}$ is equal to 1 and (ii) these labels and label pairs are consistent with each other.

Node-edge agreement as necessary condition for dual optimality. The statements of the following proposition follow directly from the definition of the local polytope $\mathcal{L}$. They show that arc consistency is an intrinsic property of all vectors in $\mathcal{L}$, and that strict arc-consistency is equivalent to the notion of integer labeling.

Proposition 5.28. 1. $\boldsymbol{\mu} \in \mathcal{L}$ implies that $\mathrm{nz}[\boldsymbol{\mu}]$ is arc-consistent.

2. For any $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$, $\delta(\mathbf{y})$ is strictly arc-consistent.

3. For any $\boldsymbol{\theta} \in \mathbb{R}^\mathcal{I}$ the following two statements are equivalent:
   a) $\mathrm{mi}[\boldsymbol{\theta}]$ is strictly arc-consistent;
   b) $\mathrm{mi}[\boldsymbol{\theta}] \in \mathcal{L} \cap \{0,1\}^\mathcal{I}$, i.e. $\mathrm{mi}[\boldsymbol{\theta}]$ is an integer labeling.

Item 1 of Proposition 5.28 defines a *necessary* condition for the dual optimum given in Item 5 of Proposition 5.24. Coordinates of $\boldsymbol{\mu} \in \mathcal{L}(\boldsymbol{\phi})$ may be greater than zero only for locally optimal labels and label pairs. Therefore, for a dual vector $\boldsymbol{\phi}$ to be optimal it is *necessary* that there exists an arc-consistent subset of locally optimal coordinates (corresponding to labels and label pairs) of $\boldsymbol{\theta}^{\boldsymbol{\phi}}$:

Definition 5.29. One says that nodes and edges *agree* (or there is a *node-edge agreement*) for a cost vector $\boldsymbol{\theta} \in \mathbb{R}^\mathcal{I}$, if for each node and each edge $w \in \mathcal{V} \cup \mathcal{E}$ there is a non-empty subset $\mathbb{S}_w \subseteq \arg\min_{s \in \mathcal{Y}_w} \theta_w(s)$ of locally optimal labels and, respectively, label pairs, such that the vector $\boldsymbol{\xi}$ with coordinates $\xi_w(s) = [\![s \in \mathbb{S}_w]\!]$ is arc-consistent.

Verification of the node-edge agreement is an efficiently solvable problem. The respective *relaxation labeling* algorithm is described, *e.g.*, in [3, Sec. 6.2.4].

5.5.3 Dual optimality for acyclic labeling problems

By construction, node-edge agreement is only necessary, but not sufficient dual optimality condition in general. However, there are a few cases, where this condition is sufficient as well. This holds in particular for binary labeling problems, i.e. when only two labels are associated

with each node (see **??** for details), or when the labeling problems are acyclic. The latter case is considered below.

Proposition 5.30. Let $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \theta)$ be a labeling problem with the graph $\mathcal{G}$ being acyclic. Then the node-edge agreement implies $\boldsymbol{\phi}$ is a dual optimum. Moreover, there is an optimal integer labeling $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$ such that $\delta(\mathbf{y}) \leq \boldsymbol{\xi}$, where $\boldsymbol{\xi}$ is the vector defining the node-edge agreement.

Furthermore, for any $(w, s) \in (\mathcal{V} \cup \mathcal{E}) \times \mathcal{Y}_w$ such that $\boldsymbol{\xi} \neq \mathbf{0}$ there is an optimal integer labeling $\mathbf{y}^* \in \mathcal{Y}_\mathcal{V}$ such that $y^*_w = s$.

Proof. Without loss of generality we assume that $\mathcal{G}$ is connected, i.e. is a tree. Otherwise the proof can be done for each connected component separately.

Let $\boldsymbol{\xi}$ be the arc-consistent subset of labels and labels pairs that defines the node-edge agreement. The proof will be done by constructing a labeling $\mathbf{y}$ such that $\delta(\mathbf{y}) \leq \boldsymbol{\xi}$.

We will iteratively build a *connected* subgraph $(\mathcal{V}', \mathcal{E}')$ and assign labels to the nodes of $\mathcal{V}'$ such that if label pair (y_u, y_v) is assigned to the edge $(u, v) \in \mathcal{E}'$ it holds that $\xi_u(y_u) = 1$, $\xi_v(y_v) = 1$ and $\xi_{uv}(y_u, y_v) = 1$. Our procedure finalizes when $\mathcal{V}' = \mathcal{V}$ and $\mathcal{E}' = \mathcal{E}$.

The sets $\mathcal{V}'$ and $\mathcal{E}'$ are empty at the beginning of the procedure. Let $u \in \mathcal{V}$ be any vertex. Consider a label $s \in \mathcal{Y}_u$ for which $\xi_u(s) = 1$. Assign $y_u := s$ and $\mathcal{V}' := \mathcal{V}' \cup \{u\}$.

On each iteration a node $v \in \mathcal{V} \backslash \mathcal{V}'$ is considered, which is incident to some node of $\mathcal{V}'$. Note that any node $v \in \mathcal{V} \backslash \mathcal{V}'$ is incident to *at most* one node in $\mathcal{V}'$. Indeed, if u' and u'' would be two nodes of $\mathcal{V}'$ incident to v, there would be a path p between them in the graph $(\mathcal{V}', \mathcal{E}')$, since the latter is connected. Therefore, there would a cycle u', v, u'', p, u', which would mean a contradiction, as the initial graph is acyclic.

Let therefore $v \in \mathcal{V} \backslash \mathcal{V}'$ be any node connected to some node $u \in \mathcal{V}'$. By definition of node-edge agreement there is a $t \in \mathcal{Y}_v$ such that $\xi_{uv}(y_u, t) = \xi_v(t) = 1$. Assign $y_v := t$ and $\mathcal{V}' := \mathcal{V}' \cup \{v\}$, $\mathcal{E}' := \mathcal{E}' \cup \{uv\}$.

Repeat the process until $\mathcal{V}' = \mathcal{V}$ and therefore $\mathcal{E} = \mathcal{E}'$.

By construction it holds that $\delta(\mathbf{y}) \leq \boldsymbol{\xi}$ for the labeling $\mathbf{y}$, which finalizes the proof of the first two statements.

For the last statement, the labeling $\mathbf{y}^*$ can be constructed with (u, s) being selected at the first step of the procedure. □

Corollary 5.31. Let $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \theta)$ be a labeling problem with the graph $\mathcal{G}$ being acyclic. Let $\mathcal{L}$ and $\mathcal{M}$ be the corresponding local and marginal polytopes. Then $\mathcal{L} = \mathcal{M}$.

Proof. Since $\mathcal{M} \subseteq \mathcal{L}$ if suffices to show that $\mathcal{L} \subseteq \mathcal{M}$. Consider a maximizer $\boldsymbol{\phi}$ of the Lagrange dual $\mathcal{D}$ defined by (5.124) for an acyclic problem with the cost vector $\boldsymbol{\theta}$. Since node-edge agreement is necessary for optimality of $\boldsymbol{\phi}$, Proposition 5.30 implies that $\mathcal{L}(\boldsymbol{\phi})$ contains an integer labeling. In its turns, it implies (see Item 5 of Proposition 5.24) that this labeling is a solution to the local polytope relaxation of the labeling problem.

In other words, Proposition 5.30 implies that for any cost vector $\boldsymbol{\theta}$ there is always an integer solution of the local polytope relaxation of the labeling problem for acyclic graphs. It implies that only for vectors $\delta(\mathbf{y})$, $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$, corresponding to integer labelings, a cost vector $\boldsymbol{\theta}$ may exist such that $\delta(\mathbf{y})$ is the unique solution of the relaxed problem $\min_{\boldsymbol{\mu} \in \mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$.

Due to Definition 2.20 this implies $\operatorname{vrtx}(\mathcal{L}) \subseteq \{\delta(\mathbf{y}) \mid \mathbf{y} \in \mathcal{Y}_\mathcal{V}\}$. Therefore,

$$\mathcal{L} \subseteq \operatorname{conv}\{\delta(\mathbf{y}) \mid \mathbf{y} \in \mathcal{Y}_\mathcal{V}\} = \mathcal{M} \subseteq \mathcal{L}\,, \tag{5.130}$$

which finalizes the proof. □

Corollary 5.32. Let $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \boldsymbol{\theta})$ be a labeling problem with the graph $\mathcal{G}$ being acyclic. Let also $\boldsymbol{\mu}^* \in \mathcal{L}$ be a solution of the local polytope relaxation of the labeling problem. Then from $\mu^*_u(s) > 0$ for some $u \in \mathcal{V}$, $s \in \mathcal{Y}_v$ it follows that there is an optimal integer labeling $\mathbf{y}^*$ such that $y_u = s$.

Proof. The labeling $\mathbf{y}^*$ can be constructed as in the proof of Proposition 5.30 with (u, s) being selected as the first node and label. □

5.5.4 Bibliography and further reading

The local polytope relaxation for labeling problems, its dual and its analysis were first provided by Schlesinger (see [25], [26]). An algorithm equivalent to the relaxation labeling was independently proposed by [27], Schlesinger (see Shlezinger [25]) and revisited by [28].

The comprehensive overview [29] is a standard reference for the relation of the primal and dual formulations of the energy minimization problem. It includes definitions of arc-consistency, the relaxation labeling algorithm and a lot of related notions and facts.

How to estimate a primal relaxed solution during dual optimization of the labeling problem is described in [30].

We refer to [31] and references therein for a definition of constraint satisfaction problems in general. The description of its solvable subclasses was first given in [32], see also [33]. An analysis of solvability of constraint satisfaction problems, where min/max operations instead of $\vee/\wedge$ is given in [34]. A simple generalization of the relaxation labeling algorithm for such problems can be found in [35].

6 Basic scalable convex optimization techniques

The goal of this chapter is to provide a brief overview of fundamental optimization methods suitable for large-scale convex programs – such as, for example, the Lagrange dual of the labeling problem. We focus on three core techniques: gradient descent, the subgradient method, and block-coordinate descent. In addition, we introduce entropy-based smoothing and the related coordinate ascent approach in a simplified form.

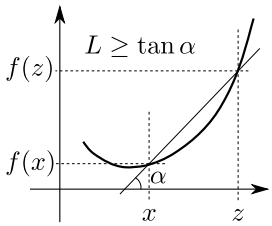
While gradient descent serves primarily as a baseline for comparison, the other methods are illustrated through their application to concrete optimization problems. As in previous chapters, the labeling problem serves as the main illustrative example throughout.

Throughout the chapter we will use the notation $\|\cdot\|$ for the Euclidean norm in $\mathbb{R}^n$.

6.1 Gradient descent

We begin the chapter by considering differentiable functions and introducing the most fundamental algorithm for their optimization: gradient descent. While combinatorial problems and their Lagrangian and LP relaxations are generally non-differentiable, we include key results on gradient descent to establish a baseline for comparison with more general methods.

Lipschitz-continuity We will be interested in differentiable functions with continuous gradient. Its "degree of continuity" is determined by the following definition:



(a)

Figure 6.1: Illustration of the Lipschitz continuity and Lipschitz constant L.

Definition 6.1. A mapping $\mathbf{f}\colon \mathbb{R}^n \to \mathbb{R}^m$ is called *Lipschitz-continuous* on $X \subseteq \mathbb{R}^n$ if there is a constant $L \geq 0$ such that for any $\mathbf{x}, \mathbf{z} \in X$ it holds that

$$\|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{z})\| \leq L\|\mathbf{x} - \mathbf{z}\|\,. \tag{6.1}$$

The smallest value L satisfying the condition above is called the *Lipschitz constant* for f on X.

Figure 6.1 illustrates Definition 6.1. Informally speaking, the Lipschitz constant is an upper bound for the speed of change of the value of the mapping f. The following statement considers the limit case of this inequality and allows us to estimate the value of the Lipschitz constant in simple cases:

Proposition 6.2. If $\mathbf{f}\colon \mathbb{R}^n \to \mathbb{R}^m$ is differentiable, then

$$L = \sup_{\mathbf{x}\in X} \|\nabla\mathbf{f}(\mathbf{x})\|_2,$$

where $\|\cdot\|_2$ is a spectral norm, i.e. the largest eigenvalue of the matrix $\nabla\mathbf{f}(\mathbf{x})$. In a special case, when $m = 1$, it coincides with the Euclidean norm.

Example 6.3. Proposition 6.2 implies that:

- $f(x) = x$ is Lipschitz-continuous on $\mathbb{R}$;

- $f(x) = x^n$, $n = 2, 3, \ldots$ is Lipschitz-continuous on any bounded subset of $\mathbb{R}$, e.g. on $[0, 1]$, but not on $\mathbb{R}$ itself.

We will use the notation $C_L^{1,1}(X)$ for functions $f\colon X \to \mathbb{R}$, which are differentiable on X and whose gradient is Lipschitz-continuous on X with the Lipschitz constant L. In general $C_L^{k,m}(X)$ is a standard notation for the set of k-times differentiable functions such that their mth derivative is Lipschitz-continuous on X with the constant L.

Example 6.4. $C_L^{1,1}(\mathbb{R})$ includes such functions as x and x^2. The functions x^n for $n = 3, 4, \ldots$ belong to $C_L^{1,1}(X)$ for any bounded $X \subset \mathbb{R}$, but not for $X = \mathbb{R}$.

Theorem 6.5. Let $f\colon \mathbb{R}^n \to \mathbb{R}$ be convex and differentiable. Then $\nabla f(\mathbf{x}) = \mathbf{0}$ is the necessary and sufficient condition for $\mathbf{x} \in \mathbb{R}^n$ to be a (global) minimum of f.

Proof. This follows directly from the similar property of the subgradient expressed by Theorem 4.35 together with Proposition 4.27. □

Gradient descent algorithm Let $f\colon \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{x}^0 \in \mathbb{R}^n$ be an initial point. The iterative process of the form

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \alpha^t \nabla f(\mathbf{x}^t) \tag{6.2}$$

for $t = 0, \ldots, \infty$ and some $\alpha^t > 0$ is called *gradient descent*. The value α^t is called the *step-size* of the algorithm.

Let $\mathbf{x}^* \in \arg\min_{\mathbf{x} \in \operatorname{dom} f} f(\mathbf{x})$ and $f^* = f(\mathbf{x}^*)$. The following statement specifies convergence properties of the gradient descent:

Theorem 6.6 ([22], [36])**.** Let $f \in C_L^{1,1}(\mathbb{R}^n)$ be convex. Then with $\alpha^t = \frac{1}{L}$ for the sequence x^t defined by (6.2) it holds that:

$$\|\mathbf{x}^{t+1} - \mathbf{x}^*\|^2 \le \|\mathbf{x}^t - \mathbf{x}^*\|^2 - \frac{1}{L^2}\|\nabla f(\mathbf{x}^t)\|^2, \tag{6.3}$$

$$f(\mathbf{x}^{t+1}) \le f(\mathbf{x}^t) - \frac{1}{2L}\|\nabla f(\mathbf{x}^t)\|^2, \tag{6.4}$$

$$f(\mathbf{x}^t) - f^* \leq \frac{2L\|\mathbf{x}^0 - \mathbf{x}^*\|}{t+4}. \tag{6.5}$$

Let us consider the statement of the theorem in detail. First of all, it considers a *constant* step-size $\alpha := \alpha^t = \frac{1}{L}$, which is inverse-proportional to the speed of change of the gradient ∇f expressed by its Lipschitz constant. In other words, the faster the gradient changes, the smaller step-size must be selected. Therefore, the Lipschitz constant defines the vicinity in which the considered function has "approximately the same" gradient. The larger the constant the smaller is the vicinity and, hence, the smaller the step-size.

Expression (6.3) states that each step of the algorithm produces a solution estimate x^{t+1} which is closer to the optimum x^* than the previous estimate x^t.

Expression (6.4) shows that the algorithm is strictly monotonous, i.e. unless $\nabla f = \bar{0}$, the objective value strictly decreases on each iteration.

Expression (6.5) provides the *convergence rate* of the gradient descent algorithm. The accuracy $\epsilon = f(x^t) - f^*$ is attained after at most $O(\frac{L}{t})$ iterations. This complexity is often formulated in a "dual" fashion, when one says that the algorithm requires $O(\frac{L}{\epsilon})$ iterations to attain a given accuracy ϵ.

Step-size selection for gradient descent One may object that a single Lipschitz constant L can be a too rough estimation for a behavior of a function on its whole domain and, therefore, the constant step-size policy $\alpha = \frac{1}{L}$ may not work well in practice. This is indeed often the case. Consider the function x^3 on the interval $[0, b]$. The Lipschitz constant of its derivative $3x^2$ on this interval is bounded by the value $6b$ according to Proposition 6.2. If $b = 1$ this results in $L = 6$ and if $b = 100$ Proposition 6.2 gives $L = 600$. The step-size $\frac{1}{600}$ would lead to a much slower convergence in the vicinity of $x = 1$ than the step-size $\frac{1}{6}$. In other words, the step-size and the convergence speed depend on the domain on which the estimation of the Lipschitz constant was done. Therefore, a variable step-size quite often works better in practice.

There are three typical ways to define α^t in (6.2):

- $\alpha^t = \frac{1}{L}$ - constant step size. Requires knowing the Lipschitz constant or a good estimate of it. May be very inefficient if $\|\nabla f\|$ significantly varies over the domain of f.
- $\alpha^t = \arg\min_{\alpha \in \mathbb{R}_+} f(\mathbf{x}^t - \alpha \nabla f(\mathbf{x}^t))$, which is the step-size that minimizes f in the negative gradient direction. For efficiency of the gradient descent algorithm as a whole, a closed form solution for α^t is typically required, which is often not available.
- α^t is selected to satisfy

$$f(\mathbf{x}^{t+1}) \leq f(\mathbf{x}^t) - \frac{\alpha^t}{2} \|\nabla f(\mathbf{x}^t)\|^2 . \tag{6.6}$$

  Here $\frac{1}{\alpha^t}$ plays the role of a local estimate for L. To this end α^t is searched in the form $\alpha^t = \frac{2\alpha^{t-1}}{2^n}$ for $n = 0, \ldots, \infty$ until (6.6) is satisfied. In other words, one tries to double the step size, checks condition (6.6) and divides the step-size by two until the condition is not satisfied anymore. The initial value α^0 can be selected arbitrary, as soon as more than just few iterations of the algorithm are used. See the *Goldstein-Armijo* rule in [22], [23] for a substantiation.

Though the practical behavior can differ for the above three cases, their worst-case analysis is similar and for any of these strategies it results in the convergence rate given by Theorem 6.6.

6.2 Sub-gradient method

Unfortunately, gradient descent cannot be applied to non-differentiable functions, such as the Lagrange duals of integer linear programs. To address this, we introduce the *subgradient method*, which generalizes gradient descent to the broader class of non-differentiable convex functions.

Definition 6.7. Let $\mathbf{x}^0 \in \mathbb{R}^n$ be a starting point and $f\colon \mathbb{R}^n \to \mathbb{R}$ be a convex function. The iterative process of the form

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \alpha^t \mathbf{g}^t\,, \tag{6.7}$$

for $t = 0, \ldots, \infty$, some $\alpha^t > 0$ and $\mathbf{g}^t \in \partial f(\mathbf{x}^t)$, is called *a subgradient method*.

Remark 6.8. The subgradient method for maximizing concave functions takes the form

$$\mathbf{x}^{t+1} = \mathbf{x}^t + \alpha^t \mathbf{g}^t\,, \tag{6.8}$$

reflecting the fact that the subgradient changes sign when transitioning from a convex function f to its concave counterpart $-f$, see Section 4.2 for details.

The update rule (6.7) is essentially the same as (6.2). However, when the function f is non-smooth, the step-size sequence α^t must be selected differently than in the smooth case:

Theorem 6.9 ([22], [37])**.** Let f be convex and Lipschitz continuous in a ball $B_R(\mathbf{x}^*) = \{\mathbf{x} \in \mathbb{R}^n\colon \|\mathbf{x}^* - \mathbf{x}\| \le R\}$ with Lipschitz constant M. Let $\mathbf{x}^0 \in B_R(\mathbf{x}^*)$ be the initial point and let the step-size sequence α^t satisfy

$$\alpha^t > 0,\ \alpha^t \xrightarrow{t\to\infty} 0,\ \text{and}\ \sum_{t=1}^{\infty} \alpha^t = \infty\,. \tag{6.9}$$

Then the following holds:

- $f(\mathbf{x}^t) - f^* \xrightarrow{t\to\infty} 0$;
- [37] $0 < \alpha^t < 2\frac{f(\mathbf{x}^t) - f^*}{\|\mathbf{g}^t\|^2}$ implies
$$\|\mathbf{x}^{t+1} - \mathbf{x}^*\| \le \|\mathbf{x}^t - \mathbf{x}^*\|\,; \tag{6.10}$$
- [22] In particular, $\alpha^t = \frac{R}{\|\mathbf{g}^t\|\sqrt{t+1}}$ for all $t = 1, 2\ldots$ implies
$$f(\mathbf{x}^t) - f^* \le \frac{MR}{\sqrt{t+1}}. \tag{6.11}$$

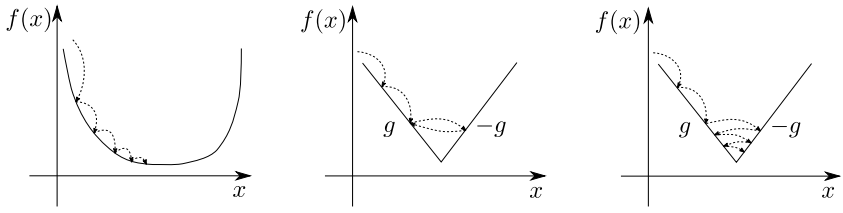


Figure 6.2: Illustration of the step-size rules for minimizing smooth and non-smooth functions. **(a)** The function f is smooth, therefore its gradient vanishes as one gets closer to the optimum. Therefore, the size of the update steps $\alpha^t \nabla f$ will vanish as well, even for a constant step-size $\alpha^t = \alpha$. **(b)** The function f is non-smooth, hence its subgradient does not vanish in general. Therefore, a constant step-size leads to oscillations around the optimum. **(c)** A vanishing step-size allows for convergence to the optimum even for non-smooth functions.

Note the following important properties of the sub-gradient method stated by Theorem 6.9:

- For a non-differentiable function f the convergence is guaranteed only if the step-size α^t satisfies the *diminishing step-size rule* (6.9). This is due to the fact that the subgradient need not vanish in the vicinity of the minimum, contrary to the gradient, see Figure 6.2 for illustration. Since neither $f(\mathbf{x}^t) - f^*$ nor R is typically known, the claims of Theorem 6.9 can be simplified as *there exists* $\{\alpha^t\}_{t=1}^{\infty}$ such that (6.10) holds, as well as *there exists* $\{\alpha^t\}_{t=1}^{\infty}$ such that (6.11) holds. Step-sizes which decrease as $O(\frac{1}{t})$ or $O(\frac{1}{\sqrt{t}})$ are typical choices that satisfy condition (6.9).

- The update rule (6.7) does not guarantee the monotonic improvement of the function value if f is non-differentiable. In other words, it may happen that $f(\mathbf{x}^{t+1}) \geq f(\mathbf{x}^t)$, even for an

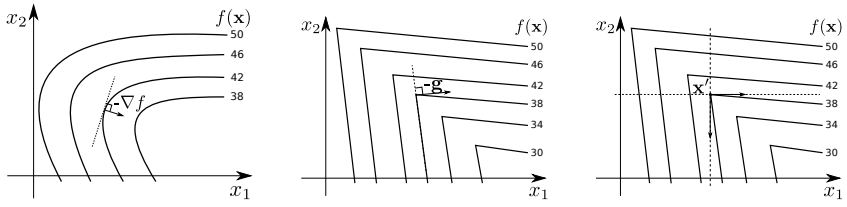


Figure 6.3: Level sets, i.e. lines of a constant function value, of smooth (a) and non-smooth (b-c) convex functions are depicted. **(a)** For differentiable functions the negated gradient always points to the direction of the function decrease, therefore for small enough step-sizes the gradient descent (6.2) is monotonous. **(b)** A negated subgradient of a non-differentiable function may point to the direction in which the function increases. Therefore, the subgradient algorithm (6.7) does not guarantee the monotonous decrease of the function on each iteration. **(c)** The point $\mathbf{x}'$ is the directional minimum of the function f, i.e. when moving along the axes x_1 and x_2 from the point $\mathbf{x}'$, the value of f increases compared to $f(\mathbf{x}')$.

arbitrary small α^t, see Figure 6.3 for illustration. It implies that minimization of f in the direction of a negative subgradient does not make sense, contrary to the case when f is smooth. On the other hand, the distance to the optimum x^* never grows during iterations, according to (6.10).

- The convergence rate defined by (6.11) can be written as $O(\frac{1}{\sqrt{t}})$, or, alternatively, as $O(\frac{1}{\epsilon^2})$. This is significantly slower than the convergence rate $O(\frac{L}{t})$ in the smooth case. For example, to attain the precision $\epsilon = 0.1$ the subgradient algorithm must perform 100 times more iterations than to obtain the precision $\epsilon = 1$. Note that the gradient descent would require only 10

Table 6.1: Difference of (sub)gradient descent algorithms for smooth and non-smooth optimization.

	Smooth f	Non-smooth f
Update rule	$\mathbf{x}^{t+1} = \mathbf{x}^t - \alpha^t \nabla f(\mathbf{x}^t)$	$\mathbf{x}^{t+1} = \mathbf{x}^t - \alpha^t \mathbf{g}^t,\ \mathbf{g}^t \in \partial f(\mathbf{x}^t)$
Step-size $\alpha_t > 0$	$\alpha^t = \frac{1}{L}$	$\alpha^t \xrightarrow{t\to\infty} 0,\ \sum_{t=1}^{\infty} \alpha_t = \infty$
Distance to optimum	$\|\mathbf{x}^{t+1} - \mathbf{x}^*\|^2 \leq \|\mathbf{x}^t - \mathbf{x}^*\|^2 - \frac{1}{L^2}\|\nabla f(\mathbf{x}^t)\|^2$	$\|\mathbf{x}^{t+1} - \mathbf{x}^*\| \leq \|\mathbf{x}^t - \mathbf{x}^*\|$
Monotonicity	$f(\mathbf{x}^{t+1}) \leq f(\mathbf{x}^t) - \frac{1}{2L}\|\nabla f(\mathbf{x}^t)\|^2$	—
Convergence rate	$f(\mathbf{x}^t) - f^* \leq \frac{2L\|\mathbf{x}^0 - \mathbf{x}^*\|}{t+4}$	$f^t - f^* \leq \frac{MR}{\sqrt{t+1}}$

times more iterations given that the function to be optimized is smooth.

Table 6.1 summarizes the difference between gradient- and subgradient descent algorithms, or, more precisely, between the cases where the function f to be optimized is smooth (is in $C_L^{1,1}$) and non-smooth.

Remark 6.10 (Practical step-size rules)**.** Since neither the constant R nor the optimal value f^* are typically known, a variety of different practical step-size rules are used. One of them [37, Sec. 6.3.1] is the approximation of those for (6.10):

$$\alpha^t = \frac{\beta^t (f(\mathbf{x}^t) - \hat{f}^t)}{\|\mathbf{g}^t\|^2}, \tag{6.12}$$

where $\hat{f}^t$ is the approximation of the optimal value and $0 < \beta^t < 2$.

In particular [37, Sec. 6.3.1] suggests two common ways to choose β^t and $\hat{f}^t$:

1. The first option is assign $\hat{f}^t$ to use the best known lower bound for f^*. The latter can often be computed from the dual information. For example, if the primal problem is ILP, the subgradient is used to optimize its convex Lagrange dual, the best lower bound for f^* can be either the best known value of the initial

ILP objective, or, if available, the best known value of the primal of the Lagrange relaxation. Example for computing the latter can be found in, *e.g.*, [30], [38].

The value of β^t is selected then as, *e.g.*,

$$\beta^t = \frac{1+m}{t+m}\,, \tag{6.13}$$

where m is some fixed integer preventing too rapid decrease of the step-size on the first iterations and division by 0 for $t = 0$. Note that β^t selected in this way satisfies the conditions for the diminishing step-size rule (6.9).

2. The second option is to set $\beta^t = 1$ and

$$\hat{f}^t := m^t/(1+\gamma^t) \tag{6.14}$$

with m^t being the best attained value so far, *i.e.*, $m^t := \min_{0\le i\le t} f(\mathbf{x}^i)$. Here $\gamma^t > 0$ and its value is increased by a certain factor (*e.g.*, 1.05) if the previous iteration improves the best attained value so far, and decreased by some other factor otherwise. The method assumes $m^t > 0$.

3. The third option is the modification of the second one, where the updates are performed according to

$$\hat{f}^t := \max\{\widetilde{f}^t, m^t/(1+\gamma^t)\} \tag{6.15}$$

with $\widetilde{f}^t$ being the best available lower bound for f^*.

Note that the formulas above have to be respectively changed if the subgradient is used to maximize a concave function (see [37, Sec. 6.3.1]):

1. Equation (6.12) turns into

$$\alpha^t = \frac{\beta^t(\hat{f}^t - f(\mathbf{x}^t))}{\|\mathbf{g}^t\|^2}\,. \tag{6.16}$$

2. Equation (6.14) turns into

$$\hat{f}^t := (1+\gamma^t)m^t\,,\ \text{where } m^t := \max_{0\le i\le t} f(\mathbf{x}^i)\,. \tag{6.17}$$

3. Equation (6.15) turns into

$$\hat{f}^t := \min\{\widetilde{f}^t, (1+\gamma^t)m^t\}\,. \tag{6.18}$$

6.2.1 Case study: Dual sub-gradient for the relaxed labeling problem

Consider the Lagrange dual of the labeling problem. Recall, it has the form of an unconstrained piecewise linear concave function

$$\mathcal{D}(\boldsymbol{\phi}) = \sum_{u\in\mathcal{V}} \min_{s\in\mathcal{Y}_u} \theta_u^{\boldsymbol{\phi}}(s) + \sum_{uv\in\mathcal{E}} \min_{(s,t)\in\mathcal{Y}_{uv}} \theta_{uv}^{\boldsymbol{\phi}}(s,t)\,, \tag{6.19}$$

with the reparametrized costs $\boldsymbol{\theta}^{\boldsymbol{\phi}}$ defined by (5.121).

The subgradient method (6.7) is the simplest way to optimize concave non-smooth functions in general. To be able to use it we have to compute a subgradient of $\mathcal{D}$.

Consider its single coordinate $\frac{\partial \mathcal{D}}{\partial \phi_{u,v}(s)}$ and the function

$$\mathcal{D}_{u,v}(\phi_{u,v}) := \min_{s'\in\mathcal{Y}_u} \theta_u^{\boldsymbol{\phi}}(s') + \min_{(s',t')\in\mathcal{Y}_{uv}} \theta_{uv}^{\boldsymbol{\phi}}(s',t')\,, \tag{6.20}$$

containing exactly those two terms of $\mathcal{D}$, which depend on $\boldsymbol{\phi}_{u,v}$. Other terms do not contribute to $\frac{\partial \mathcal{D}}{\partial \phi_{u,v}(s)}$, due to the linear properties of the subgradient (Proposition 4.28) and due to the fact that the subgradient of a constant function is identical to zero (since any constant function is differentiable and has gradient zero, which is also its subgradient, see Proposition 4.27). In other words, $\frac{\partial \mathcal{D}}{\partial \phi_{u,v}(s)} = \frac{\partial \mathcal{D}_{u,v}}{\partial \phi_{u,v}(s)}$.

Due to the linearity of the subgradient we can compute it for each term of $\mathcal{D}_{u,v}$ separately. Recall also how $\boldsymbol{\theta}^{\boldsymbol{\phi}}$ depends on $\boldsymbol{\phi}$:

$$\theta_u^{\boldsymbol{\phi}}(s) := \theta_u(s) - \sum_{v\in\mathcal{N}(u)} \phi_{u,v}(s),\ u\in\mathcal{V},\ s\in\mathcal{Y}_u\,, \tag{6.21}$$

$$\theta_{uv}^{\boldsymbol{\phi}}(s,t) := \theta_{uv}(s,t) + \phi_{u,v}(s) + \phi_{v,u}(t)\,,\ uv\in\mathcal{E},\ (s,t)\in\mathcal{Y}_{uv}\,.$$

Consider the first term of $\mathcal{D}_{u,v}$,

$$\min_{s' \in \mathcal{Y}_u} \theta_u^{\boldsymbol{\phi}}(s'). \tag{6.22}$$

Its subgradients are by virtue of Lemma 4.32 equal to the convex hull of the vectors $\frac{\partial \theta_u^{\boldsymbol{\phi}}(s')}{\partial \boldsymbol{\phi}_{u,v}} = \left(\frac{\partial \theta_u^{\boldsymbol{\phi}}(s')}{\partial \phi_{u,v}(s)} \colon s \in \mathcal{Y}_u \right)$ for all minimizers s' of (6.22). Since the numerator is a linear and, therefore, differentiable function of ϕ, we can apply the standard differentiation rules, yielding $\frac{\partial \theta_u^{\boldsymbol{\phi}}(s')}{\partial \phi_{u,v}(s)} = -[\![s = s']\!]$.

Similarly, subgradients of the second term in (6.20) can be computed as $\frac{\partial \theta_{uv}^{\boldsymbol{\phi}}(s'',t'')}{\partial \phi_{u,v}(s)} = [\![s = s'']\!]$ for each minimizer (s'', t'') of the second term. Note that if $s' = s'' = s$ the subgradients of both terms cancel out. We now combine our observations to construct a subgradient of $\mathcal{D}$. Let

$$y'_u \in \operatorname*{arg\,min}_{s \in \mathcal{Y}_u} \theta_u^{\boldsymbol{\phi}}(s) \tag{6.23}$$

and

$$(y''_u, y''_v) \in \operatorname*{arg\,min}_{(s,t) \in \mathcal{Y}_{uv}} \theta_{uv}^{\boldsymbol{\phi}}(s,t) \tag{6.24}$$

be defined for all $u \in \mathcal{V}$ and all $uv \in \mathcal{E}$. The vector $\frac{\partial \mathcal{D}}{\partial \boldsymbol{\phi}}$ with coordinates

$$\frac{\partial \mathcal{D}}{\partial \phi_{u,v}(s)} = \begin{cases} 0, & s \neq y'_u \quad \text{and} \quad s \neq y''_u \\ 0, & s = y'_u \quad \text{and} \quad s = y''_u \\ -1, & s = y'_u \quad \text{and} \quad s \neq y''_u \\ 1, & s \neq y'_u \quad \text{and} \quad s = y''_u, \end{cases} \tag{6.25}$$

is a subgradient of $\mathcal{D}$. Figure 6.4 illustrates all possible combinations occurring in (6.25).

Note that (6.25) defines only one possible subgradient. It is unique only if y'_u and (y''_u, y''_v) are *unique* solutions of the respective local minimization problems (6.23) and (6.24). Multiple solutions imply multiple subgradients of the form (6.25) and the convex combinations thereof.

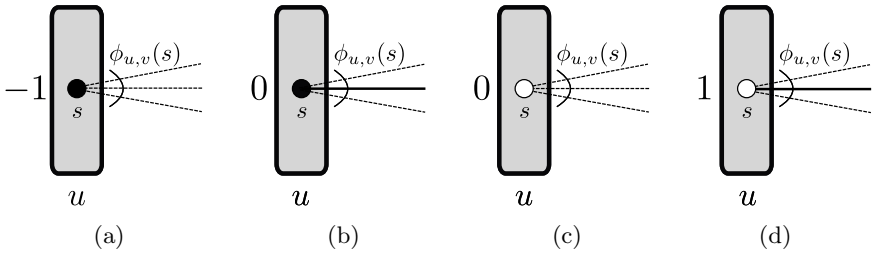


Figure 6.4: Illustration to Equation (6.25). Black circles correspond to y'_u, solid lines to (y'_u, y'_v). White circles and dashed lines are labels and label pairs, which are not equal to y'_u and (y''_u, y''_v) respectively. The number to the left of the rectangles denoting the node u, is the value of the corresponding subgradient coordinate.

However, any of these subgradients can be used in the subgradient method (6.7), which is now defined up to the step-size α^t:

$$\boldsymbol{\phi}^{t+1} = \boldsymbol{\phi}^t + \alpha^t \frac{\partial \mathcal{D}}{\partial \boldsymbol{\phi}}\,. \tag{6.26}$$

We recommend to use the practical step-size rules, as described in Remark 6.10.

Remark 6.11. Note that subgradients computed according to (6.25) are quite sparse. For a given node u and edge uv there are $|\mathcal{Y}_u|$ entries of the subgradient, corresponding to the $|\mathcal{Y}_u|$ dual variables $\phi_{u,v}(s)$, $s \in \mathcal{Y}_u$. According to (6.25) at most two coordinates of the subgradient have non-zero entries. The rest $|\mathcal{Y}_u| - 2$ are zero. This, in turn, implies that only a small number of coordinates of the dual vector $\boldsymbol{\phi}$ are changed on each iteration of the subgradient method. This is one of the reasons, why this algorithm converges quite slowly in practice, especially for problems with a large number of labels $\mathcal{Y}_u$.

Sufficient dual optimality condition Note that Theorem 4.35 provides a necessary and sufficient optimality condition for the Lagrange

dual $\mathcal{D}$. It is the existence of a zero element in $\partial\mathcal{D}(\boldsymbol{\phi})$. However, checking this condition is in general computationally costly, as it would require to check whether the convex polyhedral set $\partial\mathcal{D}$ is non-empty. This is in general as difficult as solving a linear program of the size comparable to the size of the relaxed labeling problem. Therefore, a simplified *sufficient* optimality condition is typically used, where a zero subgradient is searched only in the form given by (6.25). To this end it must be verified, whether $y'_u = y''_u$ for all $u \in \mathcal{V}$, or, in other words, if there is an integer labeling consisting of locally optimal labels and label pairs. Indeed, Item 5 of Proposition 5.24 also implies sufficiency of this condition.

6.2.2 Bibliography and further reading

The gradient and subgradient methods can be found in nearly any text-book dealing with non-smooth optimization. In our exposition we follow [22], [36], [37].

The subgradient method as well as a number of its variants and applications to the non-smooth minimization have been proposed by N. Z. Shor in 1960's, see [39] and references therein.

The dual subgradient algorithm for the labeling problem in the described form was proposed in [40]. Independently, the subgradient method was evaluated for a slightly different (although equivalent) dual problem in [41]–[43].

6.3 Block-Coordinate descent

Like the subgradient method, block-coordinate descent is well-defined for both smooth and non-smooth functions. However, the corresponding convergence guarantees differ substantially.

Definition 6.12. For a function $f\colon \prod_{j=1}^{n} \mathbb{R}^{n_j} \to \mathbb{R}$ the iterative process

$$\mathbf{x}_i^{t+1} = \arg\min_{\boldsymbol{\xi}\in\mathbb{R}^{n_i}} f(\mathbf{x}_1^{t+1}, \ldots, \mathbf{x}_{i-1}^{t+1}, \boldsymbol{\xi}, \mathbf{x}_{i+1}^{t}, \ldots, \mathbf{x}_n^{t}) \tag{6.27}$$

$$i := (i+1) \mod n \tag{6.28}$$

$$t := \left\lfloor \frac{i}{n} \right\rfloor \tag{6.29}$$

is called *cyclic* (or *Gauss-Seidel*) *coordinate descent*. Here $i \mod n$ denotes the remainder from the integer division of i to n and $\lfloor \frac{i}{n} \rfloor$ its integer part.

The method is also known as *alternating minimization*, since one alternates the optimization w.r.t. the different variables. When the dimensionality n_j of individual variables is greater than one, one often speaks also about *block*-coordinate descent.

Note that by definition the coordinate descent is monotonous, i.e. it holds that

$$\underbrace{f(\mathbf{x}_1^{t+1}, \ldots, \mathbf{x}_{i-1}^{t+1}, \mathbf{x}_i^{t}, \mathbf{x}_{i+1}^{t}, \ldots, \mathbf{x}_n^{t})}_{f_{i-1}^{t}} \geq \underbrace{f(\mathbf{x}_1^{t+1}, \ldots, \mathbf{x}_{i-1}^{t+1}, \mathbf{x}_i^{t+1}, \mathbf{x}_{i+1}^{t}, \ldots, \mathbf{x}_n^{t})}_{f_{i-1}^{t+1}} . \tag{6.30}$$

and therefore, the sequence f_i^t is monotonously non-increasing w.r.t. $k = n \cdot t + i \to \infty$. Assuming that the minimal value of f exists, i.e. $f^* > -\infty$, this implies convergence of the sequence f_i^t. To analyze this convergence and the convergence of the argument sequences $\mathbf{x}_i^t$ for each i as $t \to \infty$, we will require the following simple lemma:

Lemma 6.13. Let $f\colon \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be a convex function of two (vector-)variables. Then for any fixed $\mathbf{z}' \in \mathbb{R}^m$ the function $f_{\mathbf{z}'}(\mathbf{x}) := f(\mathbf{x}, \mathbf{z}')$ is convex w.r.t. $\mathbf{x}$ and its optimum is attained in $\mathbf{x} \in \mathbb{R}^n$ if and only if $\mathbf{0} \in \partial f_{\mathbf{z}'}(\mathbf{x})$.

Proof. The epigraph of $f_{\mathbf{z}'}$ is an intersection of the epigraph of f and the linear subspace $\mathbf{z} = \mathbf{z}'$. Therefore, it is convex as an intersection of two convex sets. This proves convexity of $f_{\mathbf{z}'}$. The claim about optimality of $\mathbf{x}$ follows from Theorem 4.35. □

Let us now consider a function $f\colon \prod_{j=1}^{n} \mathbb{R}^{n_j} \to \mathbb{R}$ of n (vector-)variables. Let $f_i(\boldsymbol{\xi}; \mathbf{x}') := f(\mathbf{x}'_1, \dots, \mathbf{x}'_{i-1}, \boldsymbol{\xi}, \mathbf{x}'_{i+1}, \dots, \mathbf{x}'_n)$ be the function of the i-th variable given all others are fixed to the corresponding coordinates of $\mathbf{x}'$. Assume $\mathbf{x}'$ is a fixed point of the algorithm (6.27), i.e. $\mathbf{x}'_i \in \arg\min_{\boldsymbol{\xi} \in \mathbb{R}^{n_i}} f_i(\boldsymbol{\xi}; \mathbf{x}')$. Due to Lemma 6.13 it holds that

$$\mathbf{0} \in \partial f_i(\mathbf{x}'_i; \mathbf{x}') \text{ for all } i = 1, \dots, n\,. \tag{6.31}$$

If f is convex differentiable, then condition (6.31) is equivalent to $\nabla f(\mathbf{x}') = \mathbf{0}$ and, therefore, a fixed point is an optimum.

However, if f is non-differentiable, this implication does not work, as shown in Figure 6.3(c). This is due to the non-uniqueness of the subgradient, which causes this behavior of the fixed point for non-differentiable functions. In other words, there might exist a zero subgradient for each coordinate, but not for all of them together.

Moreover, in general, the algorithm (6.27) does not even guarantee convergence to such a point where the conditions (6.31) are fulfilled. This negative statement holds even if f is differentiable, as shown by [44].

Indeed, to guarantee convergence, an additional *uniqueness* property is required:

Theorem 6.14 ([37])**.** Let f be continuously differentiable on $\mathbb{R}^n$. Furthermore, let for each i and $\mathbf{x} \in \mathbb{R}^n$ the minimum in (6.27) be *uniquely* attained. Then for every limit point $\mathbf{x}^*$ of the sequence $\mathbf{x}^t$, $t = 1, 2, \dots$ defined by (6.27) it holds that $\nabla f(\mathbf{x}^*) = 0$.

If f is convex differentiable the optimality of $\mathbf{x}$ follows from the condition $\nabla f(\mathbf{x}) = \mathbf{0}$. Theorem 6.14 does not claim neither existence nor uniqueness of the limit points. It only proofs their properties if they exist. In particular, if the set $F(\mathbf{x}^0) := \{\mathbf{x}\colon f(\mathbf{x}) \le f(\mathbf{x}^0)\}$ is bounded,

then $\mathbf{x}^t \in F(\mathbf{x}^0)$ due to the monotonicity of the coordinate descent, i.e. $f(\mathbf{x}^t) \leq f(\mathbf{x}^0)$. Hence $\{\mathbf{x}^t\}$ is bounded and has limit points, which all correspond to the optimum of f according to Theorem 6.14.

Convergence rate A little is known about the convergence rate of cyclic coordinate descent (6.27) for general convex differentiable functions. Recently, the convergence rate $O(\frac{1}{t})$ was proven by [45] for the case $n = 2$, i.e. when only two blocks of variables are considered in (6.27).

6.3.1 Case study: Primal coordinate descent for the labeling problem

Iterated Conditional Modes

Iterated Conditional Modes (ICM) is one of the first and simplest algorithms for energy minimization. As we show below, it is an implementation of the coordinate descent (6.27) for the primal non-relaxed energy minimization objective $E(\mathbf{y}; \boldsymbol{\theta})$.

The algorithm starts with some labeling $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$ and iteratively tries to improve it. In its elementary step it minimizes the objective w.r.t. the label of a node $u \in \mathcal{V}$, whereas the labels of other nodes are kept fixed.

Let the notation

$$l(\mathbf{y}, u, s) := \mathbf{y}' \in \mathcal{Y}_\mathcal{V} \quad \text{with} \quad y'_v = \begin{cases} y_v, & v \neq u \\ s, & v = u \end{cases} \tag{6.32}$$

define a labeling where all coordinates but the one corresponding to node u coincide with the labeling $\mathbf{y}$ and the coordinate u is assigned the label s.

Then the elementary step of the algorithm can be written as

$$y_u := \arg\min_{s \in \mathcal{Y}_u} E(l(\mathbf{y}, u, s); \boldsymbol{\theta}) \,. \tag{6.33}$$

Algorithm 7 applies this rule to each node sequentially and iterates this procedure until the labeling $\mathbf{y}$ does not change anymore.

Algorithm 7 ICM: Coordinate Descent for $E(y;\theta)$

1: **Init:** $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$
2: **repeat**
3: **for** $u \in \mathcal{V}$ **do**
4: $y_u := \arg\min_{s \in \mathcal{Y}_u} E(l(\mathbf{y}, u, s); \boldsymbol{\theta})$
5: **end for**
6: **until** the labeling $\mathbf{y}$ does not change anymore

As any coordinate descent, Algorithm 7 guarantees that the objective value of E is monotonically nonincreasing. However, since E as a function of $\mathbf{y}$ is neither convex nor differentiable (moreover, it is defined on a discrete set $\mathcal{Y}_\mathcal{V}$ only), Algorithm 7 does not guarantee attainment of the optimal value of E and its output significantly depends on the initial labeling. In practice Algorithm 7 typically returns labelings with significantly higher energies than many other techniques.

The following example demonstrates that Algorithm 7 may fail to attain the optimum even in pretty simple cases:

Example 6.15. Consider a simple two-node model with two labels in each node as shown in Figure 6.5. Suppose that the unary costs are all zero and the pairwise costs are $\theta_{uv}(0,0) = 0$, $\theta_{uv}(1,1) = 1$, $\theta_{uv}(0,1) = \theta_{uv}(1,0) = 2$. If the initial labeling is $(1,1)$, the ICM updates cannot improve the labeling, since they can switch only to the labelings $(1,0)$ or $(0,1)$, each with total cost 2, while the labeling $(1,1)$ has total cost 1. However, the optimum is attained in labeling $(0,0)$, which has total cost 0.

Block-ICM algorithm

The idea of the ICM algorithm goes beyond Algorithm 7. In general, one has to fix labels in a subset of nodes and optimize with respect to the labels in the remaining nodes. Should this optimization be

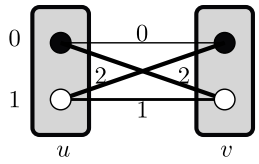


Figure 6.5: The ICM algorithm does not guarantee attainment of the optimum even for a simple binary 2-node model. Suppose that the unary costs are all zero and the pairwise costs are $0, 1, 2$, the thickness of the lines grows with the cost: $\theta_{uv}(0,0) = 0$, $\theta_{uv}(1,1) = 1$, $\theta_{uv}(0,1) = \theta_{uv}(1,0) = 2$. If the initial labeling is $(1,1)$, the ICM updates cannot improve the labeling, see Example 6.15 for a detailed explanation.

tractable then the whole algorithm is as well. Below we describe one such tractable case.

Definition 6.16. A subgraph $(\mathcal{V}', \mathcal{E}')$ of a graph $(\mathcal{V}, \mathcal{E})$ is called *induced* by its set of nodes $\mathcal{V}'$, if $\mathcal{E}' = \{uv \in \mathcal{E} \colon u, v \in \mathcal{V}'\}$, i.e. its set of edges contains exactly those edges from $\mathcal{E}$, that connect nodes from $\mathcal{V}'$.

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph defining a labeling problem and $\mathcal{V}^f \subset \mathcal{V}$ be a set of nodes with fixed labels. The corresponding *partial labeling* will be denoted as $\mathbf{y}^f \in \mathcal{Y}_{\mathcal{V}^f}$.

Introducing $\mathcal{V}' = \mathcal{V} \backslash \mathcal{V}^f$ as complement of $\mathcal{V}^f$ we can generalize the notation defined in (6.32) such that

$$l(\mathbf{y}^f, \mathcal{V}', \mathbf{y}') := \tilde{\mathbf{y}} \in \mathcal{Y}_{\mathcal{V}} \quad \text{with} \quad \tilde{y}_u = \begin{cases} y^f_u, & u \in \mathcal{V}^f \\ y'_u, & u \in \mathcal{V}' \end{cases} \tag{6.34}$$

defines a labeling, where nodes from the set $\mathcal{V}^f$ are labeled with $\mathbf{y}^f$, otherwise with $\mathbf{y}'$.

Let $(\mathcal{V}', \mathcal{E}')$ be the subgraph of $\mathcal{G}$ induced by $\mathcal{V}'$. It turns out that if the subgraph is acyclic, one can optimize *w.r.t.* the labeling on $\mathcal{V}'$,

$$\mathbf{y} := \arg \min_{\mathbf{y}' \in \mathcal{Y}_{\mathcal{V}'}} E(l(\mathbf{y}^f, \mathcal{V}', \mathbf{y}'); \boldsymbol{\theta}) \,, \tag{6.35}$$

using dynamic programming as shown below.[1]

As Example 6.15 in particular shows, Algorithm 7 does not solve the labeling problem on acyclic models in general. Therefore, considering blocks of variables associated with such acyclic models for a block-coordinate descent may result in better approximate solutions.

Below we derive the expression for the unary and pairwise costs of the acyclic model to be optimized over on each iteration of the considered block-ICM algorithm. Let $(\mathcal{V}^f, \mathcal{E}^f)$ be the subgraph of $\mathcal{G}$ induced by $\mathcal{V}^f$ and let $\mathcal{E}'^f = \{uv \in \mathcal{E} \colon u \in \mathcal{V}',\ v \in \mathcal{V}^f\}$ be the set of edges of $\mathcal{G}$ connecting the nodes from $\mathcal{V}'$ and $\mathcal{V}^f$ (see Figure 6.6 for illustration). Consider the energy $E(l(\mathbf{y}^f, \mathcal{V}', \mathbf{y}'); \boldsymbol{\theta})$, defined by (6.34):

$$
\begin{aligned}
&E(l(\mathbf{y}^f, \mathcal{V}', \mathbf{y}'); \boldsymbol{\theta}) \\
&\quad = \sum_{u \in \mathcal{V}'} \theta_u(y'_u) + \sum_{uv \in \mathcal{E}'} \theta_{uv}(y'_u, y'_v) + \sum_{uv \in \mathcal{E}'^f} \theta_{uv}(y'_u, y^f_v) \\
&\qquad + \sum_{u \in \mathcal{V}^f} \theta_u(y^f_u) + \sum_{uv \in \mathcal{E}^f} \theta_{uv}(y^f_u, y^f_v) \\
&\quad = \sum_{u \in \mathcal{V}'} \left(\theta_u(y'_u) + \sum_{v \in \mathcal{N}(u) \cap \mathcal{V}^f} \theta_{uv}(y'_u, y^f_v) \right) \\
&\qquad + \sum_{uv \in \mathcal{E}'} \theta_{uv}(y'_u, y'_v) + \sum_{u \in \mathcal{V}^f} \theta_u(y^f_u) + \sum_{uv \in \mathcal{E}^f} \theta_{uv}(y^f_u, y^f_v)\,. \qquad (6.36)
\end{aligned}
$$

Recall that $\mathbf{y}^f$ is fixed. Therefore, the last expression represents the energy of a problem on the graph $(\mathcal{V}', \mathcal{E}')$ with modified unary costs, plus a constant represented by the last two terms of the expression. It implies that minimization of $E(l(\mathbf{y}^f, \mathcal{V}', \mathbf{y}'); \boldsymbol{\theta})$ w.r.t. $\mathbf{y}'$ can be done efficiently by dynamic programming if the graph $(\mathcal{V}', \mathcal{E}')$ is acyclic.

Algorithm 8 summarizes these observations. At each iteration it generates a subset $\mathcal{V}'$ of nodes over which it minimizes the corresponding auxiliary energy (6.36) with dynamic programming, as described

[1] The dynamic programming algorithm considered in Section 1.5.2 for chain-structured labeling problems can be generalized to cover the whole class of acyclic labeling problems.

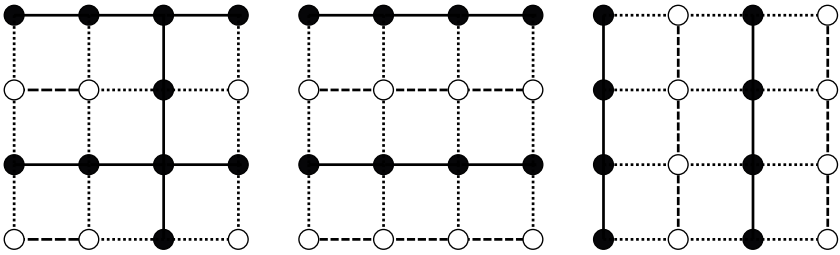


Figure 6.6: A grid graphical model and different acyclic induced subgraphs. Black and white circles stand for graph nodes, solid, dashed and dotted lines for graph edges. White circles and dashed lines denote $\mathcal{V}^f$ and $\mathcal{E}^f$ respectively, i.e. the part of the model with fixed labels. Dotted lines stand for the edges from the set $\mathcal{E}'^f$ connecting nodes from $\mathcal{V}'$ and $\mathcal{V}^f$. The block-ICM algorithm optimizes over the labels in the nodes $\mathcal{V}'$ denoted as black circles and connected by solid lines denoting the set of edges $\mathcal{E}'$. **(b,c)** Note that the subgraphs $(\mathcal{V}', \mathcal{E}')$ consist of several connected components, which can be processed independently in parallel.

in Section 1.5.2, until the current labeling does not change for a certain number of iterations or the total iteration limit is reached.

The way the sets $\mathcal{V}'$ are generated can be either quite generic or depend on a particular graph structure. For grid-graphs a simple and well-parallelizable approach is to consider four graphs $(\mathcal{V}', \mathcal{E}')$ in a cyclic order: columns with even indexes, columns with odd indexes, rows with even indexes and rows with odd indexes. Note that minimization in line 5 of Algorithm 8 can be done for all columns (rows) of each of these subgraphs in parallel, see Figure 6.6(b) and Figure 6.6(c).

Algorithm 8 Block-ICM: Block-Coordinate Descent for $E(\mathbf{y};\boldsymbol{\theta})$

Init: $\mathbf{y} \in \mathcal{Y}_{\mathcal{V}}$
repeat
  Generate $\mathcal{V}' \subseteq \mathcal{V}$ such that $(\mathcal{V}', \mathcal{E}')$ induced by $\mathcal{V}'$ is acyclic
  Define $\mathcal{V}^f := \mathcal{V} \backslash \mathcal{V}'$ and $\mathbf{y}^f := \mathbf{y}|_{\mathcal{V}^f}$
  Compute $\mathbf{y}^* := \arg\min_{\mathbf{y}' \in \mathcal{Y}_{\mathcal{V}'}} E(l(\mathbf{y}^f, \mathcal{V}', \mathbf{y}'); \boldsymbol{\theta})$
  Re-assign the values of $\mathbf{y}$ on the coordinates of $\mathcal{V}'$: $\mathbf{y}|_{\mathcal{V}'} := \mathbf{y}^*$
until some stopping condition holds

6.3.2 Case study: Dual block-coordinate ascent for the labeling problem

The most efficient and scalable methods for optimizing the Lagrange dual (6.19) of the labeling problem are based on the block-coordinate ascent principle. While these methods do not generally guarantee convergence to the dual optimum, they often produce high-quality approximate solutions within a moderate number of iterations.

In this section, we focus on one of the earliest such methods, known in the literature as *min-sum diffusion*. Although it is not the most efficient algorithm of its kind, its simplicity makes it well-suited for illustrating key properties common to block-coordinate ascent methods applied to the Lagrange dual of the labeling problem. Moreover, with relatively minor modifications, it can be transformed into a state-of-the-art block-coordinate ascent algorithm, see [3], [46], [47].

Min-sum diffusion

To construct a block-coordinate ascent algorithm one has to first partition the coordinates into blocks. To get an efficient algorithm, the minimization with respect to each block should be efficient. Ideally, the minimum should have a closed form or be attained by a finite-step algorithm with linear complexity.

One such possibility is to assume that one block consists of variables $(\phi_{u,v}(s) \colon v \in \mathcal{N}(u), s \in \mathcal{Y}_u)$ "attached" to one node of the graph, see Figure 6.7. For a given edge uv and label $s \in \mathcal{Y}_u$ we will refer to the

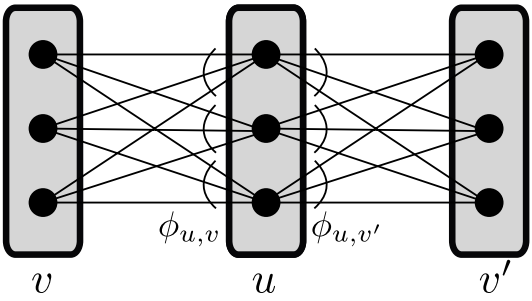


Figure 6.7: One elementary step of the min-sum diffusion algorithm optimizes over dual variables $\phi_{u,v}(s)$, $v \in \mathcal{N}(u)$, $s \in \mathcal{Y}_u$, "attached" to the current node u. The variables are "sitting" on the pencils, associated with each label of the node u and denoted by small arcs.

set of label pairs $\{(s,l),\ l \in \mathcal{Y}_v\}$ as a *pencil*. Each pencil is defined by the triple (s,u,v), where $u \in \mathcal{V}$, $v \in \mathcal{N}(u)$ and $s \in \mathcal{Y}_u$. Pencils (s,u,v), $v \in \mathcal{N}(u)$, will be called *associated* with the label s in node u.

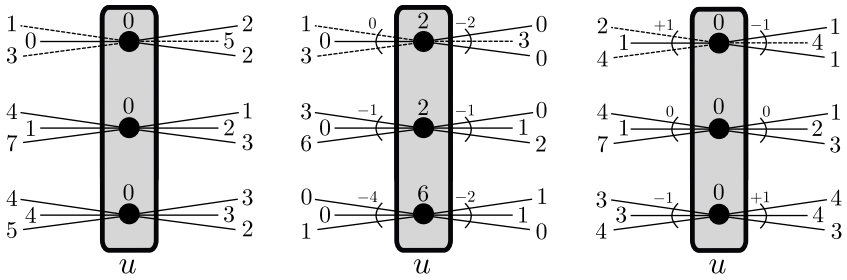
The elementary update step of the diffusion algorithm consists of the following two operations,

$$\forall v \in \mathcal{N}(u)\ \forall s \in \mathcal{Y}_u \quad \phi^{t+1}_{u,v}(s) := \phi^{t}_{u,v}(s) - \min_{l \in \mathcal{Y}_v} \theta^{\phi^t}_{uv}(s,l)\,, \tag{6.37}$$

$$\forall v \in \mathcal{N}(u)\ \forall s \in \mathcal{Y}_u \quad \phi^{t+2}_{u,v}(s) := \phi^{t+1}_{u,v}(s) + \frac{\theta^{\phi^{t+1}}_{u}(s)}{|\mathcal{N}(u)|}\,, \tag{6.38}$$

and is illustrated in Figure 6.8. Loosely speaking, executing one elementary update step for a node u redistributes the minimum costs evenly between all pencils of a given label while moving all costs away from the node itself.

For fixed s and v the first operation (6.37) computes the minimal cost in the pencil (s,u,v), subtracts it from the costs of all label pairs of the pencil and adds it to the cost of the label s. As a result, the minimal cost in each pencil associated with the label s becomes zero.



(a) (b) (c)

Figure 6.8: Example of an elementary update step of the diffusion algorithm. Larger numbers stand for (reparametrized) costs, smaller numbers for the corresponding coordinates of the dual vector ϕ, assuming that initially $\phi = 0$. Solid and dashed lines denote minimal and non-minimal label pairs in each pencil respectively. **(a)** Initial costs. **(b)** Costs and dual variables after (6.37). **(c)** Costs and dual variables after (6.38). Note that unary costs are zero and minimal label pairs in each pencil get equal weights, as stated in (6.39) and (6.40) respectively.

The second operation then redistributes the reparametrized unary costs $\theta_u^\phi(s)$ now corresponding to the label s equally between all pencils (s, u, v), $v \in \mathcal{N}(u)$. As a result, the unary cost of the label s becomes zero and the minimal pairwise costs in all pencils associated with s become equal to each other.

In other words, after operations (6.37) and (6.38) have been executed for node $u \in \mathcal{V}$ the following holds:

$$\theta_u^\phi(s) = 0, \ \forall s \in \mathcal{Y}_u\,, \tag{6.39}$$

$$\min_{t \in \mathcal{Y}_v} \theta_{uv}^\phi(s,t) = \min_{t \in \mathcal{Y}_{v'}} \theta_{uv'}^\phi(s,t), \ \forall v, v' \in \mathcal{N}(u), \ s \in \mathcal{Y}_u\,. \tag{6.40}$$

Proposition 6.17. Operations (6.37)-(6.38) maximize the Lagrange dual $\mathcal{D}$ w.r.t. the block of variables $(\phi_{u,v}(s) \colon v \in \mathcal{N}(u), s \in \mathcal{Y}_u)$.

Proof. By virtue of Lemma 6.13 it is sufficient to prove the existence of a zero subgradient in a subdifferential of $\mathcal{D}$ when the latter is treated as a function of only the considered block of variables.

We will search for a zero subgradient in the form (6.25). Since only the costs in the node u and its incident edges uv, $v \in \mathcal{N}(u)$, are dependent on the considered variable block, it is sufficient to show that

$$\exists y_u \in \mathcal{Y}_u \; \forall v \in \mathcal{N}(u) \; \exists y_v \in \mathcal{Y}_v, : \\ y_u \in \arg\min_{s \in \mathcal{Y}_u} \theta^{\phi}_u(s) \text{ and } (y_u, y_v) \in \arg\min_{(s,l) \in \mathcal{Y}_{uv}} \theta^{\phi}_{uv}(s,l)\,. \quad (6.41)$$

See Figure 6.4(b) for illustration.

Note that $y_u \in \arg\min_{s \in \mathcal{Y}_u} \theta^{\phi}_u(s)$ holds for any $y_u \in \mathcal{Y}_u$ since all labels in node u have the same cost after the elementary diffusion step due to (6.39), namely cost 0. Let us, therefore, assign y_u such that $(y_u, y_{v'})$ is a minimizer of $\theta^{\phi}_{uv'}$ for some $v' \in \mathcal{N}(u)$. Condition (6.40) implies that it then also minimizes θ^{ϕ}_{uv} for all $v \in \mathcal{N}(u)$. This finalizes the proof. □

Remark 6.18. If the block-optimality condition (6.41) holds, the elementary step (6.37)-(6.38) of the diffusion algorithm applied to node u does not change the dual value, since the dual vector is already optimal with respect to the considered block of variables.

The min-sum diffusion algorithm consists in a cyclic repetition of the operations (6.37)-(6.38) for all $u \in \mathcal{V}$, which corresponds to the cyclic block-coordinate ascent method (6.27), up to substitution of the convex function f by the concave function $\mathcal{D}$ and minimization w.r.t. each coordinate block by maximization.

Since $\mathcal{D}$ is neither differentiable nor does it have a unique optimum, the min-sum diffusion is not guaranteed to optimize the dual $\mathcal{D}(\phi)$. Its convergence properties are analyzed below.

Let the function $F_u \colon \mathbb{R}^{\mathcal{I}} \to \mathbb{R}^{\mathcal{I}}$ define the transformation of costs performed by a single elementary update step (6.37)-(6.38) applied to

the node u. In other words, F_u transforms a cost vector $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{I}}$ to another cost vector $F_u(\boldsymbol{\theta}) \in \mathbb{R}^{\mathcal{I}}$.

Then the following holds:

Theorem 6.19. Let $\boldsymbol{\theta}^{\phi} \in \mathbb{R}^{\mathcal{I}}$ satisfy node-edge agreement. Then $F(\boldsymbol{\theta}^{\phi})$ does so as well. Moreover, in this case the transformation F does not change the dual value, i.e. $\mathcal{D}(\boldsymbol{\phi}') = \mathcal{D}(\boldsymbol{\phi})$, where $\boldsymbol{\theta}^{\phi'} = F(\boldsymbol{\theta}^{\phi})$.

Proof. For the first part we refer to [3, Ch.8]

To prove the second one, note that according to the definition, node-edge agreement implies the block-optimality condition (6.41). Therefore, as noted in Remark 6.18, $\mathcal{D}(\phi') = \mathcal{D}(\phi)$. □

An inverse statement falls even stronger:

Theorem 6.20. Let $\boldsymbol{\theta}^{\phi}$ be a fix-point of the diffusion algorithm. Then $\boldsymbol{\theta}^{\phi}$ is arc-consistent.

Proof. Consider a binary vector $\boldsymbol{\xi} \in \mathbb{R}^{\mathcal{I}}$ such that $\boldsymbol{\xi}_{uv}(s,t) = \boldsymbol{\xi}_u(s) = \boldsymbol{\xi}_v(t) := 1$ if (s,t) is locally optimal label pair for the edge $uv \in \mathcal{E}$. Assign other coordinates of $\boldsymbol{\xi}$ to zero. The first condition of Definition 5.26 holds by construction. The second condition holds due to (6.40) similarly as in proof of Proposition 6.17. □

Theorem 6.19 guarantees that as soon as at some iteration of the min-sun diffusion algorithm the node-edge agreement holds, it holds for further iterations as well. In general, it is known that the min-sum diffusion converges to the node-edge agreement [48], [49].

Rounding for min-sum diffusion To obtain an integer labeling one has to apply some rounding method. Unfortunately, the naïve dual rounding, consisting in selection of a locally optimal label in each node, leads to very poor results when used with the min-sum diffusion, since it takes into account only the reparametrized unary costs and those are all equal to zero in the diffusion algorithm. One way to deal with that is to remember the locally optimal label after each "collection

step" (6.37) of the diffusion algorithm, when the pairwise costs are shifted to the unary costs.

Another one, which typically gives similar result when the algorithm converged, is to "equally" split the pairwise costs between corresponding unary factors, i.e. to perform

$$\forall v \in \mathcal{N}(u)\ \forall s \in \mathcal{Y}_u \quad \phi_{u,v}(s) := \phi_{u,v}(s) - \frac{1}{2} \min_{l \in \mathcal{Y}_v} \theta^{\phi}_{uv}(s,l)\,, \qquad (6.42)$$

for all nodes $u \in \mathcal{V}$. Afterwards the naïve dual rounding can be applied.

6.3.3 Bibliography and further reading

Although coordinate descent belongs to one of the earliest algorithms, the results about its convergence and the corresponding convergence rates in general are still missing. Earlier works on this topic mostly considered specializations to different function subclasses e.g. [50]–[53]. The most general convergence results for variants of the coordinate descent were recently given in [45].

A recent series of publications [54]–[57] considers theoretical properties and application of block-coordinate ascent to (relaxations of) the combinatorial problems. A unified analysis of the convergence properties of block-coordinate descent algorithms for non-smooth convex functions was given in [58].

The ICM algorithm [59] is among the first methods addressing the labeling problem. There are several publications suggesting different variants of the block-ICM method with tree-structured subproblems. Among the recent ones are [60] and [61], [62], where its parallelization properties were used to achieve faster convergence.

Another efficient block-ICM scheme for optimizing over small, but arbitrary structured subproblems is proposed in [63] and known as *Lazy Flipper*.

Although the min-sum diffusion algorithm is known at least since the 70's, its detailed convergence analysis was given only recently

in [48] and later in. The very recent work [49] has improved the analysis and provided a convergence rate of the algorithm. Our exposition is a shortcut of the results presented in [3].

A generalization known as anisotropic diffusion was introduced in the work [46] (see also [3]), along with a unified analysis and evaluation of different dual coordinate ascent algorithms including the CMP (Convex Message Passing) algorithm [64] and MPLP (Message Passing for Linear Programming) [65]. The latter was recently significantly improved in [66].

A min-sum diffusion-like algorithm for general convex piecewise linear functions was suggested in [67]. A taxonomy of dual block-coordinate ascent methods for the labeling problem has been proposed in [68].

There is an alternative method that guarantees the monotonic improvement of the dual objective as well as the convergence to the coordinate-wise fxed point (node-edge agreement for labeling problem). Its representative for the labeling problems is the *Augmenting DAG (directed acyclic graph)* algorithm, proposed by [26] and recently reviewed in [29]. A generalization of this algorithm to higher order models was suggested in [69]. This algorithm can be seen as an instance of the combinatorial primal-dual method [70]. The latter covers in particular such famous techniques as the algorithm of Ford-Fulkerson for max-flow and the Hungarian method for the linear assignment problem. The work [56] describes an intriguing relation of the primal-dual method and the block-coordiinate ascent.

6.4 Smoothing technique

6.4.1 Smoothed LP relaxation

Consider an LP relaxation of some ILP problem

$$\min_{\substack{\mathbf{x}\in[0,1]^n \\ A\mathbf{x}=\mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle \ . \tag{6.43}$$

To keep our further exposition simple, we will need the following

Assumption 6.21. Conditions $A\mathbf{x} = \mathbf{b}$ and $\mathbf{x} \geq 0$ imply $\mathbf{x} \in [0,1]^n$.

Assumption 6.21 holds, for instance, for the local polytope relaxation of the labeling problem (3.45) and for the slack-based LP relaxation of the max-weight independent set problem (see Theoretical Exercise Sheet 2).

Consider the primal-dual pair

$$\begin{aligned}\min_{\substack{\mathbf{x}\in[0,1]^n \\ A\mathbf{x}=\mathbf{b}}} \langle \mathbf{c}, \mathbf{x}\rangle = \min_{\substack{\mathbf{x}\geq 0 \\ A\mathbf{x}=\mathbf{b}}} \langle \mathbf{c}, \mathbf{x}\rangle &= \max_{\boldsymbol{\lambda}} \min_{\mathbf{x}\geq 0} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle \\ &= \max_{\boldsymbol{\lambda}} \underbrace{-\langle \boldsymbol{\lambda}, \mathbf{b}\rangle + \min_{\mathbf{x}\geq 0} \left\langle \overbrace{\mathbf{c} + A^\top \boldsymbol{\lambda}}^{\mathbf{c}^{\boldsymbol{\lambda}}}, \mathbf{x} \right\rangle}_{g(\boldsymbol{\lambda})} \end{aligned} \tag{6.44}$$

As shown in Example 5.6, the dual objective $g(\boldsymbol{\lambda})$ is concave, but non-differentiable. This significantly restricts the set of methods that can be used for its optimization. In this section we consider one technique to approximate $g(\boldsymbol{\lambda})$ with a parametrized set of convex smooth functions. Under proper selection of the *smoothing parameter* defining the accuracy of approximation one can attain a required optimization accuracy for $g(\boldsymbol{\lambda})$ by optimizing its smooth approximation.

To this end consider $H(\mathbf{x}) := -(\sum_{i=1}^n x_i \log x_i - x_i)$ - the entropy function "shifted" by the vector $\mathbf{x}$. This function is concave, hence $-H$ is convex and can be efficiently minimized. The function

$$\begin{aligned} g_T(\boldsymbol{\lambda}) = \min_{\mathbf{x}\geq 0} \langle \mathbf{c}, \mathbf{x}\rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b}\rangle - TH(\mathbf{x}) \\ = -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle + \min_{\mathbf{x}\geq 0} \left\langle \mathbf{c}^{\boldsymbol{\lambda}}, \mathbf{x} \right\rangle - TH(\mathbf{x}) \\ = -\langle \boldsymbol{\lambda}, \mathbf{b}\rangle + \sum_{i=1}^n \min_{x_i \geq 0} \underbrace{c_i^{\boldsymbol{\lambda}} x_i + T(x_i \log x_i - x_i)}_{l_i(x_i)} \end{aligned} \tag{6.45}$$

is concave, as any Lagrange dual of a minimizaiton problem. Here $l_i(x)$ is convex as a sum of linear and convex $(x \log x)$ functions. Consider

the unconstrained optimality condition for the function l_i:

$$0 = \frac{\partial l_i}{\partial x_i} = c_i^{\boldsymbol{\lambda}} + T(\log x_i + x_i \frac{1}{x_i} - 1) = c_i^{\boldsymbol{\lambda}} + T \log x_i \,. \tag{6.46}$$

This implies $x_i[\boldsymbol{\lambda}] := e^{-c_i^{\boldsymbol{\lambda}}/T}$ and since $x_i[\boldsymbol{\lambda}] \geq 0$, it holds $x_i[\boldsymbol{\lambda}] \in \arg\min_{x_i \geq 0} l_i(x_i)$. Notation $\mathbf{x}[\boldsymbol{\lambda}]$ will stand for the vector with coordinates $x_i[\boldsymbol{\lambda}]$ defined as above.

Consider now

$$\begin{aligned} g_T(\boldsymbol{\lambda}) &= -\langle \boldsymbol{\lambda}, \mathbf{b} \rangle + \left\langle \mathbf{c}^{\boldsymbol{\lambda}}, \mathbf{x}[\boldsymbol{\lambda}] \right\rangle - TH(\mathbf{x}[\boldsymbol{\lambda}]) \\ &= -\langle \boldsymbol{\lambda}, \mathbf{b} \rangle + \sum_{i=1}^{n} c_i^{\boldsymbol{\lambda}} e^{-c_i^{\boldsymbol{\lambda}}/T} + T\left(e^{-c_i^{\boldsymbol{\lambda}}/T}(-c_i^{\boldsymbol{\lambda}}/T) - e^{-c_i^{\boldsymbol{\lambda}}/T}\right) \\ &= -\langle \boldsymbol{\lambda}, \mathbf{b} \rangle - T \sum_{i=1}^{n} e^{-c_i^{\boldsymbol{\lambda}}/T} \,. \end{aligned} \tag{6.47}$$

Hence, the function $g_T(\boldsymbol{\lambda})$ is not only concave, but also infinitely continuously differentiable.

Strong duality holds not only for bounded linear programs, but for a much broader class. Since it plays an important role for the smoothing approach, we give the following theorem here.

Theorem 6.22.

$$\begin{aligned} &\min_{\substack{\mathbf{x} \geq 0 \\ A\mathbf{x} = \mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle - TH(\mathbf{x}) \\ &\qquad = \max_{\boldsymbol{\lambda}} \min_{\mathbf{x} \geq 0} \langle \mathbf{c}, \mathbf{x} \rangle + \langle \boldsymbol{\lambda}, A\mathbf{x} - \mathbf{b} \rangle - TH(\mathbf{x}) \end{aligned} \tag{6.48}$$

We refer to *e.g.*, [23] for a proof.

Note that given the strong duality, KKT conditions (5.87)-(5.91) together with Assumption 6.21 imply that for an optimal dual solution $\boldsymbol{\lambda}^*$ and the optimal primal solution $\mathbf{x}^* = e^{\mathbf{c}^{\boldsymbol{\lambda}^*}/T} \geq 0$ it holds $A\mathbf{x}^* = \mathbf{b}$ and, therefore, $\mathbf{x}^* \in [0, 1]^n$.

Smooth approximation bounds Note that for any $x \in [0,1]$ it holds $-1 \le x \log x - x \le 0$ and, therefore, $0 \le -H(\mathbf{x}) \le n$ for $\mathbf{x} \in [0,1]^n$. In turn, due to Assumption 6.21 it implies

$$\left(\min_{\substack{\mathbf{x} \ge 0 \\ A\mathbf{x} = \mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle \right) - Tn \le \min_{\substack{\mathbf{x} \ge 0 \\ A\mathbf{x} = \mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle - TH(\mathbf{x}) \le \min_{\substack{\mathbf{x} \ge 0 \\ A\mathbf{x} = \mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle \,. \quad (6.49)$$

The same in terms of the dual function:

$$g(\boldsymbol{\lambda}^*) - Tn \le g_T(\hat{\boldsymbol{\lambda}}^*) \le g(\boldsymbol{\lambda}^*) \,, \quad (6.50)$$

where $\boldsymbol{\lambda}^*$ and $\hat{\boldsymbol{\lambda}}^*$ are maximizers of the original (6.44) and smoothed (6.45) dual problems.

Moreover, even stronger condition hold, if the used optimization algorithm maintains $\mathbf{x}[\boldsymbol{\lambda}] \in [0,1]^n$ on all its iterations. In this case

$$g(\boldsymbol{\lambda}) - Tn \le g_T(\boldsymbol{\lambda}) \le g(\boldsymbol{\lambda}) \,. \quad (6.51)$$

All these conditions substantiate the intuitive assumption that lower values of the *smoothing parameter* T (sometimes called *temperature*) lead to a better approximation of the original function.

6.4.2 Smoothed dual coordinate ascent

Consider a coordinate ascent optimization of the smoothed dual $g_T(\boldsymbol{\lambda})$. Let at iteration $t+1$ one fixes all dual variables $\lambda_{j'}$, $j' \ne j$ and optimizes on λ_j only. Assume that

$$\lambda_{j'}^{t+1} = \begin{cases} \lambda_{j'}^t, & j' \ne j \,, \\ \lambda_j^t + \nu, & j' = j \,. \end{cases} \quad (6.52)$$

Then we can derive the following update rule for $\mathbf{x}[\boldsymbol{\lambda}^t]$:

$$\begin{aligned} x_i[\boldsymbol{\lambda}^{t+1}] &= e^{-\mathbf{c}_i \boldsymbol{\lambda}^{t+1}/T} = e^{-\left[\mathbf{c} + A^\top \boldsymbol{\lambda}^{t+1}\right]_i / T} \\ &= e^{-\left[c_i + \sum_{j'=1}^m a_{j'i} \lambda_{j'}^{t+1}\right]/T} = e^{-\left[c_i + \sum_{j'=1}^m a_{j'i} \lambda_{j'}^t + a_{ji}\nu\right]/T} \\ &= x_i[\boldsymbol{\lambda}^t] e^{-a_{ji}\nu/T} \,. \end{aligned} \quad (6.53)$$

The value of ν can be found from the optimality condition for λ_j:

$$\frac{\partial g_T}{\partial \lambda_j} \overset{(6.45)}{=} \left[A\mathbf{x}[\boldsymbol{\lambda}^{t+1}] - \mathbf{b}\right]_j = 0 \quad \Rightarrow \sum_{i=1}^{n} a_{ji} x_i[\boldsymbol{\lambda}^{t+1}] = b_j \,. \tag{6.54}$$

By plugging in (6.53) we obtain equation

$$\sum_{i=1}^{n} a_{ji} x_i[\boldsymbol{\lambda}^t] e^{-a_{ji}\nu/T} = b_j \tag{6.55}$$

that must be solved *w.r.t.* ν. A unique solution to this equation together with continuous differentiability of g_T would imply that the conditions of Theorem 6.14 hold and any limit point $\boldsymbol{\lambda}^*$ of the sequence $\boldsymbol{\lambda}^t$ is an optimal dual solution. The strong duality in turn implies that $x[\boldsymbol{\lambda}^*]$ is an optimal primal relaxed solution.

Note that due to one-to-one correspondence between $\boldsymbol{\lambda}$ and $\mathbf{x}[\boldsymbol{\lambda}]$ the algorithm can explicitly update either the former or the latter.

Example 6.23 (Smooth coordinate minimization for uniqueness constraints)**.** Let $\left[A\mathbf{x}[\boldsymbol{\lambda}^{t+1}]\right]_j = b_j$ have a form of a uniqueness constraint

$$\sum_{i \in K \subseteq [n]} x_i = 1 \,. \tag{6.56}$$

Then (6.55) turns into

$$e^{-\nu/T} \sum_{i \in K} x_i[\boldsymbol{\lambda}^t] = 1 \,, \tag{6.57}$$

which implies

$$e^{-\nu/T} = \frac{1}{\sum_{i \in K} x_i[\boldsymbol{\lambda}^t]} \tag{6.58}$$

and

$$\begin{aligned} \nu = -T \log \frac{1}{\sum_{i \in K} x_i[\boldsymbol{\lambda}^t]} = T \log \sum_{i \in K} x_i[\boldsymbol{\lambda}^t] = T \log \sum_{i \in K} e^{-c_i^{\boldsymbol{\lambda}^t}/T} \\ = -c_{i^*}^{\boldsymbol{\lambda}^t} + T \log \sum_{i \in K} e^{-(c_i^{\boldsymbol{\lambda}^t} - c_{i^*}^{\boldsymbol{\lambda}^t})/T} \,, \end{aligned} \tag{6.59}$$

where $i^* \in \arg\min_{i\in[n]} c_i^{\boldsymbol{\lambda}^t}$. This lead us to the mathematically equivalent primal and dual update rules

$$\lambda_{j'}^{t+1} = \begin{cases} \lambda_{j'}^t, & j' \neq j \\ \lambda_j^t - c_{i^*}^{\boldsymbol{\lambda}^t} + T \log \sum_{i\in K} e^{-(c_i^{\boldsymbol{\lambda}^t} - c_{i^*}^{\boldsymbol{\lambda}^t})/T}, & j' = j \end{cases} \tag{6.60}$$

$$x_i^{t+1} = \begin{cases} x_i^t, & i \notin K \\ \frac{x_i^t}{\sum_{i\in K} x_i^t}, & i \in K\,. \end{cases} \tag{6.61}$$

Both rules have their advantages and disadvantages: The primal update rule (6.61) is significantly ($5-10$ times) faster than the dual rule (6.60) because the latter contains an extensive usage of the (slow) exponentiation operations.

On the negative side, the primal rule may result in division by 0, usually for low T , whereas the dual rule is numerically stable for any T and $\boldsymbol{\lambda}$ due to the fact that $-(c_i^{\boldsymbol{\lambda}^t} - c_{i^*}^{\boldsymbol{\lambda}^t})/T < 0$, hence $e^{-(c_i^{\boldsymbol{\lambda}^t} - c_{i^*}^{\boldsymbol{\lambda}^t})/T} \in [0,1]$ and $\sum_{i\in K} e^{-(c_i^{\boldsymbol{\lambda}^t} - c_{i^*}^{\boldsymbol{\lambda}^t})/T} \geq 1$ since $e^{-(c_{i^*}^{\boldsymbol{\lambda}^t} - c_{i^*}^{\boldsymbol{\lambda}^t})/T} = 1$.

6.4.3 Case study: Linear assignment problem

Consider the linear assignment problem

$$\min_{\mathbf{x}\in\{0,1\}^{n\times n}} \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \tag{6.62}$$

$$\text{s.t.} \sum_{i=1}^n x_{ij} = 1,\ \forall j \in [n] \tag{6.63}$$

$$\sum_{j=1}^n x_{ij} = 1,\ \forall i \in [n] \tag{6.64}$$

Its natural relaxation satisfies Assumption 6.21 and is known to be LP-tight, hence we address its entropy-smoothed approximation:

$$\min_{\mathbf{x}\geq 0} \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} + T\sum_{i=1}^{n}\sum_{j=1}^{n}(x_{ij}\log x_{ij} - x_{ij}) \tag{6.65}$$

$$\text{s.t.} \sum_{i=1}^{n} x_{ij} = 1,\ \forall j \in [n] \tag{6.66}$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ \forall i \in [n] \tag{6.67}$$

Using the primal update rule (6.53) from Example 6.23 we obtain Algorithm 9, which is a famous Sinkhorn algorithm widely used to approximately solve the linear assignment problem in neural networks.

Algorithm 9 Sinkhorn algorithm for linear assignment.

Initialization: $x_{ij} = \exp(-c_{ij}/T),\ i,j \in [n]$
while stopping criterion not fulfilled **do**
  Normalize all rows:
  for $i = 1$ **to** n **do**
    $s := \sum_{j=1}^{n} x_{ij}$
    $x_{ij} := \frac{x_{ij}}{s},\ \forall i \in [n]$
  end for
  Normalize all columns:
  for $j = 1$ **to** n **do**
    $s := \sum_{i=1}^{n} x_{ij}$
    $x_{ij} := \frac{x_{ij}}{s},\ \forall j \in [n]$
  end for
end while
return $\mathbf{x}$

Recently, this algorithm has been applied to the LP relaxation of the max-weight independent set problem [38].

6.4.4 Temperature scheduling

The lower the temperature T is, the better the smoothed dual (6.45) approximates its non-smoothed counterpart (6.44), see (6.49)-(6.51) and [71] for a detailed analysis. Given a required approximation level, one could imagine two strategies of temperature selection. The first one is the *fixed* temperature. Although being very simple, it leads to very slow algorithms, because the first-order methds like gradient or coordinate ascent make only small progress for low temperatures.

Hence, a practical solution is the *temperature scheduling*, *i.e.*, starting with a high value of T and decreasing it as the computation progresses. The most popular are two methods for the temperature scheduling below.

Method 1: Feasibility-based.

The method is based on the observation that the primal solution $\mathbf{x}[\lambda]$ is feasible only in the optimum, which follows from the convexity of the smoothed primal-dual pair (6.44), strong duality (6.48) and KKT conditions (5.87)-(5.91). The latter are necessary and sufficient for optimality for convex problems, when strong duality holds, see Theorem 5.17. Hence one optimizes the smoothed dual (6.45) for a given fixed temperture until the primal relaxed solution $\mathbf{x}[\boldsymbol{\lambda}]$ satisties the feasibility constraints approximately, *i.e.*,

$$|A\mathbf{x} - \mathbf{b}| \leq \epsilon\,, \tag{6.68}$$

for some predefined *feasibility threshold* $\epsilon > 0$. Should the constraints (6.68) be fulfilled, one drops the temperature by the predefined factor $0 < \tau < 1$, *i.e.* $T := \tau \cdot T$ and repeats the procedure. Typically, $\tau = 1/2$.

The main disadvantage of this strategy is its strong dependence of the value of the feasibility schreshold ϵ: Small values require more iterations in the inner optimization loop (for a fixed T). Large values of ϵ lead to too fast decrease of temperature, which significantly slows down the inner loop after several temperature drops. As a result

- the overall algorithm gets numerically stuck at some temperature value and does not converge to the optimum of the initial non-smooth problem (6.44) anymore.

6.4.5 Method 2: Duality gap-based

Duality gap is an alternative way to control the distance to the optimum when dual variables $\boldsymbol{\lambda}$ are computed explicitly, *i.e.*, in (6.60). Let $\hat{\mathbf{x}}$ and $\mathbf{x}^*$ be a *feasible* and an *optimal* primal relaxed solutions of the non-smoothed problem (6.44) respectively. Let also $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}^*$ be a *feasible* and an *optimal* solutions to its smoothed dual problem with the objective (6.45) to be maximized. The strong duality implies

$$\langle \mathbf{c}, \hat{\mathbf{x}} \rangle - T\,\mathcal{H}(\hat{\mathbf{x}}) \geq \langle \mathbf{c}, \mathbf{x}^* \rangle - T\,\mathcal{H}(\mathbf{x}^*) = g(\boldsymbol{\lambda}^*) \geq g(\boldsymbol{\lambda})\,. \tag{6.69}$$

A progress in optimization for a given temperature is assured if the inequalities above hold strictly, *i.e.*, $T < \frac{\langle \mathbf{c}, \hat{\mathbf{x}} \rangle - g(\boldsymbol{\lambda})}{\mathcal{H}(\hat{\mathbf{x}})}$. This inequality turns into equality in the optimum of the smoothed problems (6.48). Hence, the respective temperature update condition, for a suitable predefined $0 < \tau < 1$, may look like

$$\text{if} \quad T > \tau \cdot \frac{\langle \mathbf{c}, \hat{\mathbf{x}} \rangle - g(\boldsymbol{\lambda})}{\mathcal{H}(\hat{\mathbf{x}})} \quad \text{update } T\,. \tag{6.70}$$

To ensure monotonicity of the temperature, one updates it by setting to

$$T := \min \left\{ T,\ \tau \cdot \frac{\langle \mathbf{c}, \hat{\mathbf{x}} \rangle - g(\boldsymbol{\lambda})}{\mathcal{H}(\hat{\mathbf{x}})} \right\}. \tag{6.71}$$

For the role of the feasible $\boldsymbol{\lambda}$ one may use a current iterate of the dual vector. It is more difficult to obtain the feasible primal solution $\hat{\mathbf{x}}$. In general, one has to project the infeasible primal iterate $\mathbf{x}[\boldsymbol{\lambda}]$ onto the feasible set $\{\mathbf{x} \geq 0 \colon A\mathbf{x} = \mathbf{b}\}$. In general, complexity of such a projection can be comparable to the complexity of the relaxed problem itself, *e.g.*, the Euclidean projection would require solving the quadratic problem

$$\hat{\mathbf{x}} \in \arg\min_{\substack{\mathbf{x} \geq 0 \\ A\mathbf{x} = \mathbf{b}}} (\mathbf{x}[\boldsymbol{\lambda}] - \mathbf{x})^2\,. \tag{6.72}$$

In some cases, however, alternative projection methods can be used. For example, there exist efficient projections to the Birkhoff polytope [72] of the transportation problem [73], local polytope [30] of the relaxed labeling problem and clique covering polytope [38] of the max-weight independent problem.

6.4.6 Bibliography and further reading

A good example for the smoothing technique usage with compasion of feasibility- and duality-based smoothing is [38]. TBD.

7 Overview: How off-the-shelf ILP solvers work

7.1 Systematic search: Branch-and-Bound

Example 7.1 (Binary knapsack problem, due to Prof. E. Vitercik)**.** Consider the binary knapsack problem and the respective branching tree in Figure 7.1. A general algorithm for a minimization (maximization) branch-and-bound approach looks as follows:

1. Solve the **basis LP relaxation**; Initialize a feasible integer solution $S = \infty(-\infty)$
2. **Branch:**
   a) **Select a** non-terminated **leaf**
   b) ...and **branch** on a fractional variable.
   c) Solve the respective LP relaxation.
3. **Evaluate** the results:
   a) Is the obtained lower (upper) LP **bound** f^{LP} higher (lower) than S? If yes - **terminate** the branch (no branching anymore on this leaf) and go to Item 2. This includes also the case when the respective problem is infeasible and the bound is equal to ∞ $(-\infty)$.
   b) Is solution **integral**? Yes - update S and terminate the branch.
   c) Go to Item 2 unless all branches are terminated.

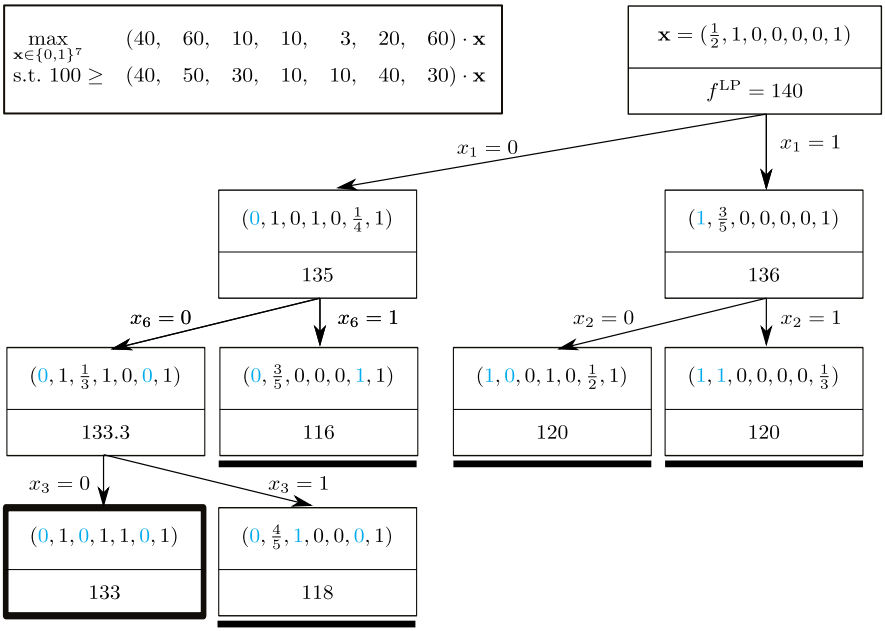


Figure 7.1: Illustration to the branch-and-bound Example 7.1 for the binary knapsack problem. Rectangles stand for the nodes of the branching tree. A relaxed solution is shown in the upper part of each rectangle and the respective relaxed objective value is given in the lower part. Fixed (branching) variables are highlighted with blue. Bold rectangle denotes an optimal solution, rectangles with a bold solid line below - terminated sub-branches as the upper bound provided by them is lower than the best available integer solution (in this case - the optimal one in the bold rectangle).

4. **Output** S.

Essentially, the branch-and-bound algorithm recursively partitions the search space. For each part, one of the following three outcomes occurs:

1. **Optimality or Infeasibility:** An optimal solution is found for the subproblem, either because the LP relaxation is tight (yielding an integer solution), or because the subproblem is infeasible. No further branching is needed.

2. **Pruning by Bound:** The LP relaxation provides a bound worse than the value of the best known integer solution S. This implies that no better solution can be found in this part, so it is pruned.

3. **Further Branching Needed:** Neither of the above conditions applies, and the algorithm proceeds by branching further.

Note that the first condition is always eventually satisfied once all coordinates of $\mathbf{x}$ have been fixed through branching. Hence, the algorithm is guaranteed to terminate after a finite number of steps – although this number may be exponential in the worst case – and return an optimal solution.

Remarks to existing implementations of the branch-and-bound methods:

- Points in which different branching methods may differ are:
  - Selection of the (fractional) variable to branch on. E.g., for the variable selection one may "select one with its value close to 0.5 that has a large coefficient in the objective function".
  - Selection of the leaf for branching. The depth-first and breadth-first are two out of many possible strategies.

  For the practically used heuristics for both items above and their empirical evaluation see, *e.g.*, [74].

- In general the branching tree grows exponentially large and, therefore, good bounds and primal heuristics are vitally important for the practical efficiency of the method.

- As follows from Item 3a, better primal solutions may improve performance of the algorithm since they allow to terminate non-productive branches earlier. Typically primal heuristics are used to obtain better primal solutions as fast as possible, see *e.g.*, [74] and Chapter 8.

- The algorithm for solving the underlying LP relaxation must allow efficient fixing of variables and "warm-start" from the relaxation of the upper level, see Section 7.4.

7.2 Cutting plain method: Tightening the bound

An orthogonal way to address ILP problems is to tighten the feasible set and, as a result, the relaxation it defines. Let P be the integer hull of the ILP of interest in the definition below.

Definition 7.2 (Separation problem, see Figure 7.2)**.** Let P be a convex polytope in $\mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^n$. The *separation problem* consists in answering the question whether $\mathbf{y} \in P$. Should it be that $\mathbf{y} \notin P$ a *separation* inequality $\langle \mathbf{a}, \mathbf{x} \rangle \leq b$ has to be found such that $\langle \mathbf{a}, \mathbf{y} \rangle > b$ and at the same time $\langle \mathbf{a}, \mathbf{x} \rangle \leq b$ for all $\mathbf{x} \in P$.

The role of the convex polytope in Definition 7.2 is usually played by the integer hull of the problem, the point $\mathbf{y}$ is a solution of the current LP relaxation, the separation inequality is the cutting plane that tightens the relaxation, see Figure 7.2. Intuitively "the best tightening" is achieved if the resulting inequality if *facet-defining*, see Figure 7.2 and Section 2.3.1, as these "can not be pushed further" into the feasible set without cutting off at least one feasible integer point.

The following *cutting plane* algorithm can be used to address the general-form ILP $\min_{\mathbf{x} \in P^0 \cap \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle$:

1. Set $t = 0$.

2. **Solve** the relaxed problem $\mathbf{y} = \arg\min_{\mathbf{x} \in P^t \cap [0,1]^n} \langle \mathbf{c}, \mathbf{x} \rangle$.

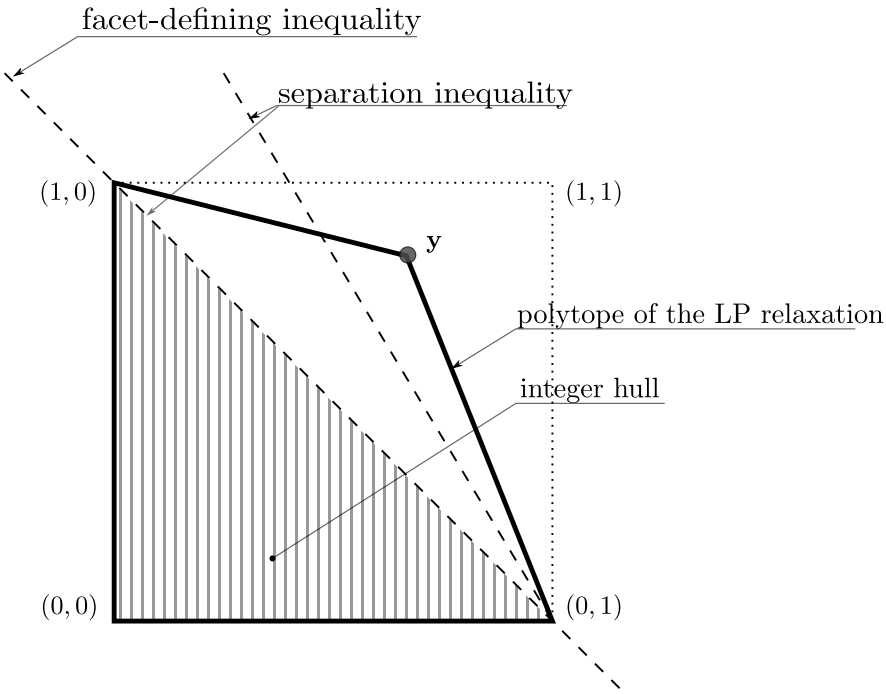


Figure 7.2: Separation problem and inequalities

3. If $\mathbf{y} \notin \operatorname{conv}(P^0 \cap \{0,1\}^n)$
   a) Find a **separation** inequality $p^t := (\langle \mathbf{a^t}, \mathbf{x} \rangle \leq b^t)$,
   b) Add it to the current feasible set: $P^{t+1} := P^t \cup \{p^t\}$.
   c) Set $t := t + 1$ and go to Item 2.

4. **Return y**.

Veryfying whether $\mathbf{y} \notin \operatorname{conv}(P^0 \cap \{0,1\}^n)$ in Item 3 is generally hard in practice. Therefore, it is commonly replaced by the simpler condition $\mathbf{y} \notin \{0,1\}^n$ and usage of LP algorithms that guarantee $\mathbf{y} \in \operatorname{vrtx}(P)$ in Item 2. Corollaries 2.61 and 2.65 justify this replacement.

Theoretical power of the cutting plane method is well-described by the following theorem:

Theorem 7.3 ([75])**.** Let $P = \{\mathbf{x} \in \mathbb{R}^n \colon A\mathbf{x} \leq \mathbf{b}\}$ be a polyhedron such that the length of encoding of each inequality in $A\mathbf{x} \leq \mathbf{b}$ is at most l.[1] Then the optimization problem $\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x} \rangle$ can be solved in polynomial time (in l and coding length of c) if an only if the separation problem with respect to P can be solved in polynomial time (in l and coding length of $\mathbf{y}$) for any $\mathbf{y} \in \mathbb{R}^n$.

In other words, given that a polynomial separation oracle exists, $\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x} \rangle$ is solvable in polynomial time even if the number of inequalities defining P is exponential in n. Important is, however, that the length of each inequality is only polynomial in n. Note that the explicit representation of an arbitrary integer hull (2.37) does not satisfy this condition, as it contains an exponentially long equality.

On the other side, NP-hard ILP problems cannot be solved with the cutting plane method alone in polynomial time. Although there is a general polynomial method, known as Gomory-cuts [16], [76], for generating separation cuts for any ILP, the number of cuts required to solve an ILP problem grows exponentially with n in general. In practice it means that the found cuts lead to smaller and smaller improvements, as the algorithm progresses.

A much more practically efficient way to explore the cutting plane idea for ILPs is the branch-and-cut method described below.

7.2.1 Branch-and-cut method

The branch-and-cut method assumes that there is a (usually exponentially large) set of potential additional inequalities (cutting planes) that can be efficiently computed to separate any relaxed solution. Should the relaxed solution satisfy all these inequalities or their computation becomes too expensive with respect to the attained improvement of the LP bound, one resorts to branching:

[1] To have a finite length all items in A and $\mathbf{b}$ must be rational

1. Solve the **basis LP relaxation**; Initialize a feasible integer solution, let $S = \infty$(for maximization $-\infty$) be its total cost.
2. **Branch:**
   a) **Select a** non-terminated **leaf**
   b) ...and **branch** on a fractional variable.
   c) Solve the respective LP relaxation **without additional separation inequalities**.
3. **Evaluate** the results:
   a) Is the obtained lower (upper) **bound** higher (lower) than S? If yes - **terminate** the branch (no branching anymore on this leaf) and go to Item 2. This includes also the case when the respective problem is infeasible and the bound is equal to ∞ ($-\infty$).
   b) Is solution **integral**? Yes - update S and terminate the branch.
   c) Can an additional **separation inequality** be found? If yes - add it to the feasible set and go to Item 3a. Otherwise:
   d) Go to Item 2 unless all branches are terminated.
4. **Output** S.

As can be seen, the algorithm differs from the original branch-and-bound only by Item 3c, which iteratively tightens the underlying relaxation as along as it can be done with a given class of considered separation inequalities. Although a separation inequality always exists (unless the found solution is integral), finding them may get computationally expensive or the resulting tightening of the feasible set only gets very small. This is essentially the result of Theorem 7.3. Therefore, one restricts the separation routine to the specific class(es) of inequalities that can be efficiently checked. When none of the inequalities is violated, one resorts to branching. Other, usually heuristic, strategies to keep the balance between branching and cutting are

used as well. For example, one may restrict the number of cutting planes that can be added to the initial LP in each leaf. To speed-up search for the violated inequalities one often stores those that have been found so far in a separate table and checks them first in the separation step.

7.2.2 Two "off-the-shelf" constraint classes for $0/1$ ILPs

A number of separation inequality classes are used by standard ILP solvers. For instance, according to [77] the SCIP [78] open source mixed integer programming solver "features separators for SCIP features separators for knapsack cover cuts [79], complemented mixed integer rounding cuts [80] (see also [74] and references therein), Gomory mixed integer cuts [76], strong Chvátal-Gomory cuts[81], flow cover cuts [82], implied bound cuts [83], and clique cuts [83], [84]".

We review here two classes that address $0/1$ integer programs: Knapsack cover cuts [79] and clique cuts [83], [84].

We will not speak about probably the most famous subclass of separation inequalities, the Gomory cuts, as their primary application domain are non-binary integer problems.

Knapsack cover cuts In the practice of off-the-shelf solvers knapsack cuts are generated to tighten each single (binary) *knapsack inequality* $\langle \mathbf{a}, \mathbf{x} \rangle \leq b$ where $\mathbf{a} \geq 0$, $b \geq 0$ and $\mathbf{x} \in \{0,1\}^N$. In our review we follow the exposition of [74] and [85].

Definition 7.4. Subset of variable indices $C \subset N$ is called *cover*, if $\sum_{i \in C} a_i > b$. The cover is *minimal*, if $\sum_{i \in C \setminus \{k\}} a_i \leq b$ for all $k \in C$.

Observe that the *cover* inequality defined as

$$\sum_{i \in C} x_i \leq |C| - 1\,, \tag{7.1}$$

is valid for the knapsack polytope $P^K := \mathrm{conv}(X^K)$ with $X^K := \{\mathbf{x} \in \{0,1\}^N \colon \langle \mathbf{a}, \mathbf{x} \rangle \leq b\}$. It is known [86] that if C is minimal cover, this

constraint is facet-defining for $P^{K'} := \mathrm{conv}(X^K \cap \{\mathbf{x} \in \{0,1\}^N : x_i = 0 \ \forall i \in N \backslash C\})$.

To separate on cover inequalities rewrite (7.1) as

$$\sum_{i \in C} (1 - x_i) \geq 1 \,. \tag{7.2}$$

Then an LP solution x^* satisfies all the cover inequalities with respect to the given knapsack if and only if

$$\sum_{i \in C} (1 - x_i^*) \geq 1 \,. \tag{7.3}$$

for any cover $C \subset N$. This can be equivalently rewritten as

$$\min_{\mathbf{z} \in \{0,1\}^N} \left\{ \sum_{i \in N} (1 - x_i^*) z_i : \text{s.t. } \sum_{i \in N} a_i z_i > b \right\} \geq 1 \,. \tag{7.4}$$

Here one minimizes over all possible covers C defined by the binary vectors $\mathbf{z}$.

Note that the (7.4) is a knapsack problem itself (see Remark 7.5) and is usually solved exactly by dynamic programming (for integer $\mathbf{a}, b$ and small N) or approximately by a greedy method (for large N), see Section 8.2. .

In practice one uses tighter *lifted* cover inequalities that may correspond to the facets of P^K and not only of $P^{K'}$. We refer to [74], [85] and references therein.

Remark 7.5. The minimization binary knapsack problem

$$\min_{\mathbf{x} \in \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle \tag{7.5}$$

$$\text{s.t. } \langle \mathbf{a}, \mathbf{x} \rangle \geq b \,. \tag{7.6}$$

can be formulated as the (maximization) binary knapsack problem

$$\max_{\mathbf{x} \in \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle \tag{7.7}$$

$$\text{s.t. } \langle \mathbf{a}, \mathbf{x} \rangle \leq \sum_{i=1}^{n} a_i - b \,. \tag{7.8}$$

by maximizing the profit of the items *not* contained in the knapsack under condition that their total weight is at most $\sum_{i=1}^{n} a_i - b$. Consequently, if $\mathbf{x}^*$ is the solution of the minimization knapsack, then $\mathbf{1} - \mathbf{x}^*$ is the solution of the maximization knapsack.

Clique cuts We say that two variables x_i and x_j are *conflicting*, if $x_i + x_j \leq 1$. Conflicting variables can be found during presolving with the *probing* procedure, see Section 7.3.2. All pairs of the conflicting variables together are stored usually in a form of a *conflict graph* $(\mathcal{V}, \mathcal{E})$. Convex hull of all binary vectors satisfying all conflicts form the *stable set polytope*: $\operatorname{conv}\{\mathbf{x} \in [0,1]^{\mathcal{V}} \colon x_i + x_j \leq 1,\ \{i,j\} \in \mathcal{E}\}$.

The most practical class of facet-defining constraints for this polytope are *clique inequalities*

$$\sum_{i \in C} x_i \leq 1\,, \tag{7.9}$$

where C is a maximal clique in $(\mathcal{V}, \mathcal{E})$, see [87].

For the separation of violated clique inequalities one uses the LP values of the variables as weights for the nodes $\mathcal{V}$ and searches for the maximum-weight clique or at least a clique with the weight greater than 1. Since the *maximum weighted clique* problem is NP-hard [4], one has to resort to heuristics in order to efficiently separate the clique cuts. Note that one cannot just add all maximal cliques as inequalities to the input problem, as the number of maximal cliques grows exponentially with the graph size in general [88].

7.3 (I)LP Presolve: Domain propagation and probing

Domain propagation and probing are used as a preprocessing, before branching as well as on different levels of the branching tree.

7.3.1 Domain propagation

Consider the convex polyhedron $P := \{\mathbf{x} \in \mathbb{R}^m \colon A\mathbf{x} \leq \mathbf{b}\}$. The goal of *domain propagation* or *bound strengthening* is to find $\mathbf{b}' \leq \mathbf{b}$ such that $P = \{\mathbf{x} \in \mathbb{R}^m \colon A\mathbf{x} \leq \mathbf{b}'\}$. In our exposition we base on [74], [83], [85], [89].

Example 7.6 (Variable's upper bound)**.** Consider

$$P := \begin{cases} 2x_1 + x_2 \leq 3 \\ 0 \leq x_1 \leq 3 \\ 0 \leq x_2 \end{cases} \tag{7.10}$$

Observe that to fulfill the first inequality x_1 should not exceed 1.5. Hence, one can rewrite P as

$$P = \begin{cases} 2x_1 + x_2 \leq 3 \\ 0 \leq x_1 \leq 1.5 \\ 0 \leq x_2 \end{cases} \tag{7.11}$$

Example 7.7 (Constraint's upper bound)**.** Consider

$$P := \begin{cases} x_1 + x_2 \leq 10 \\ 0 \leq x_1 \leq 2 \\ 0 \leq x_2 \leq 1 \end{cases} \tag{7.12}$$

Observe that due to bounds on x_1 and x_2 the first inequality can be tightened and one can rewrite P as

$$P = \begin{cases} x_1 + x_2 \leq 3 \\ 0 \leq x_1 \leq 2 \\ 0 \leq x_2 \leq 1 \end{cases} \tag{7.13}$$

Moreover, the first constraint can be completely eliminated as it does not pose any additional restrictions:

$$P = \begin{cases} 0 \leq x_1 \leq 2 \\ 0 \leq x_2 \leq 1 \end{cases} \tag{7.14}$$

The goal of domain propagation is

- Fixing some variables: if their upper and lower bounds coincide. In case of an integer variable $l \le x \le u$, $x \in \mathbb{Z}$, it is sufficient that $u - l \le 1$ to conclude $x = \lfloor u \rfloor$ if $\lfloor u \rfloor = \lceil l \rceil$ or the problem is infeasible if $\lfloor u \rfloor < \lceil l \rceil$.
- Detect infeasibility if for $l \le x \le u$ it turns that $u \le l$ after domain propagation.
- Eliminating constraints as in Example 7.7.

Domain propagation procedures can be performed in all nodes of the branch-and-bound tree,

Below we consider the formalization of the domain propagation. In our exposition we follow [83]. Let $\mathbf{x} \in \mathbb{R}^n$ and $A := (\mathbf{a_j})_{j \in [m]}$. Consider the constraint set

$$A\mathbf{x} \le \mathbf{b} \tag{7.15}$$

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \tag{7.16}$$

Denote $A^j\mathbf{x} \le \mathbf{b}^j$ the set of constraints obtained from the initial one by deleting the j-th row $\langle \mathbf{a_j}, \mathbf{x} \rangle \le b_j$. For the sake of notation we will assume that

$$\langle \mathbf{a_j}, \mathbf{x} \rangle = \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \,, \tag{7.17}$$

where C^+ and C^- denote variables with positive and negative coefficients respectively (variables with zero coefficients can be ignored) and all $a_{ji} > 0$ therefore.

Identification of infeasibility Consider the following LP:

$$z = \min \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \tag{7.18}$$

$$\text{s.t.} \quad A^j\mathbf{x} \le \mathbf{b}^j \tag{7.19}$$

$$\mathbf{l} \le \mathbf{x} \le \mathbf{u} \tag{7.20}$$

Observe that if $z > b_j$, the feasible region is empty. However, the above problem is nearly as difficult as the initial LP, hence for the sake of efficiency we consider its relaxation that gives us a lower bound z_{LB} for z: $z \geq z_{\mathrm{LB}} > b_j$. The simplest (and the weakest) lower bound is obtained by completely discarding the system $A^j\mathbf{x} \leq \mathbf{b}^j$. In that case we can conclude that the system is infeasible if

$$z_{\mathrm{LB}} = \sum_{i \in C^+} a_{ji} l_i - \sum_{i \in C^-} a_{ji} u_i > b_j \tag{7.21}$$

Identification of redundancy Consider the following LP:

$$z = \max \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \tag{7.22}$$

$$\text{s.t.} \quad A^j\mathbf{x} \leq \mathbf{b}^j \tag{7.23}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{7.24}$$

If $z \leq b_j$, the inequality $\langle \mathbf{a_j}, \mathbf{x} \rangle \leq b_j$ is redundant. Similarly to the previous case, by discarding $A^j\mathbf{x} \leq \mathbf{b}^j$, we obtain an upper bound z_{UB} and the sufficient condition $z \leq z_{\mathrm{UB}} \leq b_j$. In that case we can conclude that the inequality $\langle \mathbf{a_j}, \mathbf{x} \rangle \leq b_j$ is redundant if

$$z_{\mathrm{UB}} = \sum_{i \in C^+} a_{ji} u_i - \sum_{i \in C^-} a_{ji} l_i \leq b_j \,. \tag{7.25}$$

Improving an upper bound Consider the variable x_k and the following LP:

$$z = \min \sum_{i \in C^+ \setminus \{k\}} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \tag{7.26}$$

$$\text{s.t.} \quad A^j\mathbf{x} \leq \mathbf{b}^j \tag{7.27}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{7.28}$$

Observe that $z_k + a_{jk} x_k \leq b_j$ and, therefore, $x_k \leq (b_j - z_k)/a_{jk}$. Hence the upper bound u_k can be improved by setting to the latter value if

$(b_j - z_k)/a_{jk} < u_k$. Again, by discarding $A^j\mathbf{x} \leq \mathbf{b}^j$ we can conclude that u_k can be improved if

$$\underbrace{\left(b_j - \sum_{i \in C^+ \setminus \{k\}} a_{ji} l_i + \sum_{i \in C^-} a_{ji} u_i\right) / a_{jk}}_{\text{new value of } u_k} < u_k \,. \tag{7.29}$$

Improving a lower bound Consider the variable x_k and the following LP:

$$z = \min \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^- \setminus \{k\}} a_{ji} x_i \tag{7.30}$$

$$\text{s.t.} \quad A^j\mathbf{x} \leq \mathbf{b}^j \tag{7.31}$$

$$\mathbf{l} \leq \mathbf{x} \leq \mathbf{u} \tag{7.32}$$

Observe that $z_k - a_{jk}x_k \leq b_j$ and, therefore, $x_k \geq (z_k - b_j)/a_{jk}$. Hence the lower bound l_k can be improved by setting to the latter value if $(z_k - b_j)/a_{jk} > l_k$. Again, by discarding $A^j\mathbf{x} \leq \mathbf{b}^j$ we can conclude that l_k can be improved if

$$\underbrace{\left(\sum_{i \in C^+} a_{ji} l_i - \sum_{i \in C^- \setminus \{k\}} a_{ji} u_i - b_j\right) / a_{jk}}_{\text{new value of } l_k} > l_k \,. \tag{7.33}$$

Iterative domain propagation Domain propagation can be run iteratively over all constraints until no significant change is happening. The following example shows that the iterative improvement can be infinite:

Example 7.8 ([89])**.** For $0 < a < 1$ consider the constraints

$$x_1 - a x_2 \leq 0 \tag{7.34}$$

$$a x_1 - x_2 \geq 0 \tag{7.35}$$

$$0 \leq x_1, x_2 \leq 1 \tag{7.36}$$

Improving the bounds of x_1 using the first constraint yields $x_1 \leq ax_2 \leq a$. Exploiting this new upper bound on x_1, the second constraint implies $x_2 \leq ax_1 \leq a^2$. If we iterate and process each constraint t times, we reach $x_1 \leq a^{2t-1}$ and $x_2 \leq a^{2t}$. Thus we find an infinite sequence of bound reducitons.

Therefore one must use an improvement threshold to stop domain propagation.

7.3.2 Probing

Probing is the technique mostly useful with binary integer variables. Probing means temporarily setting a variable $x_k \in \{0, 1\}$ to 0 (or 1) and then performing the domain propagation.

Possible outcomes of such operation are[2]

- *The problem becomes infeasible*: We can fix the value of x_k to 0 (1 resp.). For example, $5x_1 + 3x_2 \geq 4$ becomes infeasible if x_1 is set to 0. We conclude that $x_1 = 1$.

- *Some constraint becomes redundant*: It can be tightened by *coefficient reduction* or *improvement*. For example, $2x_1 + x_2 \geq 1$ becomes strictly redundant when x_1 is set to 1. Therefore, the constraint can be tightened to $x_1 + x_2 \geq 1$. Note that $(0.5, 0)$ is no longer feasible to the tightened constraint, whereas all feasible integer solutions are preserved.

- *Some other binary variable $x_j \in \{0, 1\}$ gets fixed to one of it values*. An additional linear constraint can be added that tightens the relaxation. In particular, the conflict graph, see Section 7.2.2, can be constructed. As example consider $5x_1 + 4x_2 \leq 8$. If x_1 is set to 1, x_2 must be set to 0. This implies $x_1 + x_2 \leq 1$. Adding this inequality tightens the LP relaxation, as $(1, 0.75)$ is feasible for the original inequality, but is infeasible for the implication inequality.

[2] Examples are due to [85]

Consider now the system

$$\begin{aligned} A\mathbf{x} &\leq \mathbf{b} \\ \mathbf{x} &\in \{0,1\}^n \end{aligned} \tag{7.37}$$

i.e., comparing to the problem treated in Section 7.3.1, the variable $\mathbf{x}$ is now binary integer instead of being continuous. Otherwise we preserve notation introduced in Section 7.3.1 and follow [83] in our exposition.

Fixing variables

1. Consider x_k, $k \in C^+$, $j \in [m]$, and the following ILP:

$$\begin{aligned} z_k = \min & \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i && (7.38) \\ \text{s.t.} \quad & A^j \mathbf{x} \leq \mathbf{b}^j && (7.39) \\ & \mathbf{x} \in \{0,1\}^n && (7.40) \\ & x_k = 1 && (7.41) \end{aligned}$$

   If $z_k > b_j$, the set described by constraints (7.37) and additionally $x_k = 1$ is empty. It implies that $x_k \neq 1$ in any feasible solution of (7.37) and can be set to 0. When we discard the system $A^j \mathbf{x} \leq \mathbf{b}^j$, we can fix the variable x_k to 0 if

$$z_k = \left(a_{jk} - \sum_{i \in C^-} a_{ji} \right) > b_j \,. \tag{7.42}$$

2. Consider x_k, $k \in C^-$, $j \in [m]$, and the following ILP:

$$\begin{aligned} z_k = \min & \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i && (7.43) \\ \text{s.t.} \quad & A^j \mathbf{x} \leq \mathbf{b}^j && (7.44) \\ & \mathbf{x} \in \{0,1\}^n && (7.45) \\ & x_k = 0 && (7.46) \end{aligned}$$

If $z_k > b_j$, the set described by constraints (7.37) and additionally $x_k = 0$ is empty. It implies that $x_k \neq 0$ in any feasible solution of (7.37) and can be set to 1. When we discard the system $A^j\mathbf{x} \leq \mathbf{b}^j$, we can fix the variable x_k to 1 if

$$z_k = \left(- \sum_{i \in C^- \setminus \{k\}} a_{ji} \right) > b_j \,. \tag{7.47}$$

Improving coefficients

1. Consider x_k, $k \in C^+$, $j \in [m]$, and the following ILP:

$$\begin{aligned} z_k = & \max \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i && (7.48) \\ \text{s.t.} \quad & A^j \mathbf{x} \leq \mathbf{b}^j && (7.49) \\ & \mathbf{x} \in \{0,1\}^n && (7.50) \\ & x_k = 0 && (7.51) \end{aligned}$$

   If $z_k \leq b_j$, then the equality $\langle \mathbf{a_j}, \mathbf{x} \rangle \leq b_j$ is redundant under condition $x_k = 0$. This implies:

   Proposition 7.9. In the above setting the set of feasible solutions of (7.37) does not change if we re-assign $a_{jk} := a_{jk} - \delta$ and $b_j := b_j - \delta$ with $\delta = b_j - z_k \geq 0$.

   Proof. Indeed, if $x_k = 0$ the value of a_{jk} does not play any role and the value of b_j is essentially set to z_k.

To prove the implication for $x_k = 1$ consider

$$\sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \leq b_j \tag{7.52}$$

$$a_{jk} x_k + \sum_{i \in C^+ \setminus \{k\}} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \leq b_j \tag{7.53}$$

$$a_{jk} x_k - \delta x_k + \sum_{i \in C^+ \setminus \{k\}} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \leq b_j - \delta x_k \tag{7.54}$$

$$(a_{jk} - \delta) x_k + \sum_{i \in C^+ \setminus \{k\}} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \leq b_j - \delta \,. \tag{7.55}$$

□

In total, when we discard the system $A^j \mathbf{x} \leq \mathbf{b}^j$, we can re-assign a_{jk} and b_j if

$$\sum_{i \in C^+ \setminus \{k\}} a_{ji} < b_j \,. \tag{7.56}$$

2. Consider x_k, $k \in C^-$, $j \in [m]$, and the following ILP:

$$z_k = \max \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i \tag{7.57}$$

$$\text{s.t.} \quad A^j \mathbf{x} \leq \mathbf{b}^j \tag{7.58}$$

$$\mathbf{x} \in \{0,1\}^n \tag{7.59}$$

$$x_k = 1 \tag{7.60}$$

If $z_k \leq b_j$, then the equality $\langle \mathbf{a_j}, \mathbf{x} \rangle \leq b_j$ is redundant under condition $x_k = 1$. Consequently, given $x_k = 1$ we re-assign $a_{jk} := a_{jk} + (b_j - z_k)$ without changing the set of feasible solutions. Furthermore, also when $x = 0$ we can re-assign $a_{jk} := a_{jk} + (b_j - z_k)$ without changing the set of feasible solutions.

In total, we can re-assign $a_{jk} := a_{jk} + (b_j - z_k)$ without changing the set of feasible solutions. When we discard the system $A^j \mathbf{x} \leq$

$\mathbf{b}^j$, we can increase a_{jk} if

$$\sum_{i \in C^+} a_{ji} - a_{jk} < b_j \,. \tag{7.61}$$

Logical implications As noted above, probing in combination with domain propagation can be used to efficiently derive logical implications between variables.

Consider x_{k_1}, x_{k_2}, $k1, k_2 \in [n]$, and the following ILP:

$$\begin{aligned} z = \min & \sum_{i \in C^+} a_{ji} x_i - \sum_{i \in C^-} a_{ji} x_i && (7.62)\\ \text{s.t.} \quad & A^j \mathbf{x} \leq \mathbf{b}^j && (7.63)\\ & \mathbf{x} \in \{0,1\}^n && (7.64)\\ & x_{k_1} = 1 && (7.65)\\ & x_{k_2} = 1 && (7.66) \end{aligned}$$

If $z > b_j$ then the feasible set of (7.37) under condition of $x_{k_1} = x_{k_2} = 1$ is empty. In that case we can imply

$$\begin{aligned} x_{k_1} = 1 \quad &\Rightarrow \quad x_{k_2} = 0, && (7.67)\\ x_{k_2} = 1 \quad &\Rightarrow \quad x_{k_1} = 0\,, && (7.68)\\ & && (7.69) \end{aligned}$$

which can be expressed as $x_{k_1} + x_{k_2} \leq 1$ for binary variables. Similarly the following logical implications can be identified:

$$\begin{aligned} x_{k_1} = 0 \quad &\Rightarrow \quad x_{k_2} = 0, && (7.70)\\ x_{k_1} = 0 \quad &\Rightarrow \quad x_{k_2} = 1\,, && (7.71)\\ x_{k_1} = 1 \quad &\Rightarrow \quad x_{k_2} = 1\,, && (7.72)\\ x_{k_2} = 0 \quad &\Rightarrow \quad x_{k_1} = 0\,, && (7.73)\\ x_{k_2} = 0 \quad &\Rightarrow \quad x_{k_1} = 1\,, && (7.74)\\ x_{k_2} = 1 \quad &\Rightarrow \quad x_{k_1} = 1 && (7.75)\\ & && (7.76) \end{aligned}$$

The identification of logical implications proceeds in two phases. Suppose, w.l.o.g., that we want to identify logical implications associated with $x_{k_1} = 1$. In the first phase we reduce the initial system $A\mathbf{x} \leq \mathbf{b}$ by substituting $x_1 = 1$ throughout. In the second phase, each of the inequalities of the reduced system is analyzed in turn, using the above probing and domain propagation techniques to modify bounds and to fix variables, to establish whether any variable can be fixed. If so, logical implications have been identified.

Logical functions can be used to strengthen various functions embedded in an ILP solver, such as knapsack cover and clique constraints-based separation (Section 7.2.2) and primal heuristic computation.

7.4 Solving LP relaxation

7.4.1 Forms of linear programming problem

Linear programs can be represented in different forms, suitable to different algorithms. Below we review these forms and provide transformations between them.

Definition 7.10. Let $\mathbf{x} \in \mathbb{R}^n$. We distinguish the following three forms of the linear programs:

General:

$$\min \langle \mathbf{c}, \mathbf{x} \rangle \tag{7.77}$$
$$A\mathbf{x} \leq \mathbf{b} \tag{7.78}$$
$$A'\mathbf{x} = \mathbf{b}' \tag{7.79}$$
$$x_i \geq 0,\ i \in I \subseteq [n] \tag{7.80}$$

Canonical:

$$\min \langle \mathbf{c}, \mathbf{x} \rangle \tag{7.81}$$
$$A\mathbf{x} \leq \mathbf{b} \tag{7.82}$$
$$\mathbf{x} \geq \mathbf{0} \tag{7.83}$$

Standard:

$$\begin{aligned} &\min \langle \mathbf{c}, \mathbf{x} \rangle \\ &A\mathbf{x} = \mathbf{b} \\ &\mathbf{x} \geq \mathbf{0} \end{aligned} \tag{7.84}$$

These forms are equivalent, as any LP can be represented in any of them. The general form is essentially the most general and describes any LP according to Definition 2.49. The following transformations are used to switch between different forms:

1. **Sign flipping:** $A\mathbf{x} \geq \mathbf{b} \Leftrightarrow (-A\mathbf{x} \leq -\mathbf{b})$
2. **Equality-to-inequality:** $A\mathbf{x} = \mathbf{b} \Leftrightarrow \{A\mathbf{x} \leq \mathbf{b}\,, \quad A\mathbf{x} \geq \mathbf{b}\}$
3. **Inequality-to-equality:** $a_j x_i \leq b_j \Rightarrow \{a_j x_i + s = b_j\,, \quad s \geq 0\}$, where the newly introduced *slack*[3] variable s is assigned zero cost.
4. **Unconstrained-to-constrained variables:** unconstrained x_i is substituted with $x_i^+ - x_i^-$ and constrains $\left\{x_i^+ \geq 0\,, x_i^- \geq 0\right\}$.

Below we list the transformations required to switch between these forms of linear programs taking into account that canonical and standard forms are special cases of the general one:

- **general** $\rightarrow$ **canonical**: 2, 1, 4;
- **general** $\rightarrow$ **standard**: 3, 4;
- **canonical** $\rightarrow$ **standard**: 3;
- **standard** $\rightarrow$ **canonical**: 2, 1.

[3]It is also called *surplus* variable for the transformation $ax_i \geq b \Rightarrow a_j x_i - s = b_j$.

Remark 7.11. The names *canonical* and *standard* do not seem to be consistent through the literature. E.g. in [16] the inverse inequality $\geq$ is used for the definition of the canonical form and in [14] the canonical form corresponds to the form of equations corresponding to the iterations of the simplex method as shown in (7.88).

7.4.2 (Vanilla) simplex algorithm

Simplex algorithm is one of the standard methods used by off-the-shelf LP solvers. Below we describe its idea and the basic simplified algorithm on an example.

Definition 7.12. Two vertices of a polytope are called *adjacent*, if there is an edge of the polytope they both belong to. We will also say that these vertices are *connected* by this edge in P.

The idea of the simplex algorithm is the following procedure

- Start with a vertex of the feasible set - the underlying polyhedron (in our case always a polytope)
- On each iteration move to the adjacent vertex with a lower objective value. In degenerate cases one stays in the same vertex and the same objective value. Special procedures are used to avoid cycling if the objective value does not drop over some iterations.

The above monotonicity and a finite number of vertices of a polytope guarantee that the algorithm attains a fixed point in a finite number of iterations. This number of iterations need not be polynomial: There are specially constructed examples that require an exponential number of iterations. However, in most practical applications, the number of simplex iterations grows proportionally to the number of variables.

The following theorem states that the fixed point is an optimum:

Theorem 7.13. Let P be a polytope, $\mathbf{c} \in \mathbb{R}^n$ be a cost vector. Let $\mathbf{x} \in \text{vrtx}(P)$ and $\mathcal{N}_P(x)$ be the set of its adjacent vertices in P. If $\langle \mathbf{c}, \mathbf{x} \rangle \leq \langle \mathbf{c}, \mathbf{z} \rangle$ for all $\mathbf{z} \in \mathcal{N}_P(x)$ then $\mathbf{x} \in \arg\min_{\mathbf{x}' \in P} \langle \mathbf{c}, \mathbf{x}' \rangle$.

The proof is based on the following

Lemma 7.14. Let the conditions of Theorem 7.13 hold. Then $\mathbf{x}$ is the local minimum of $\langle \mathbf{c}, \mathbf{x} \rangle$ on P, *i.e.*, there exists $\epsilon > 0$ such that $\mathbf{x} \in \arg\min_{\mathbf{x}' \in B_\epsilon(\mathbf{x}) \cap P} \langle \mathbf{c}, \mathbf{x}' \rangle$.

Proof. **Sketch of the proof:** Induction on the dimension d of the polytope. For $d = 2$: By definition of the polytope's face (see Chapter 2) an optimum of a LP can be attained only on its face. For $d = 2$ the only faces are vertices and edges. To show local optimality it is sufficient to show optimality *w.r.t.* points on the edges adjacent to $\mathbf{x}$ that belong to the ϵ-ball around $\mathbf{x}$, $B_\epsilon(\mathbf{x})$. It holds due to linearity of the objective (it is also linear along each edge) and the fact that all adjacent vertices has no better objective value than $\mathbf{x}$.

Assume it holds for the dimension d. Consider now $d+1$. A polytope with dimension $d+1$ has faces of dimension of at most d. Each of them is a polytope itself (see Corollary 2.52). In each of these polytopes $\mathbf{x}$ is a local optimum by assumption of the induction. Therefore, it is a local optimum for the intersection of their union with $B_\epsilon(\mathbf{x})$. □

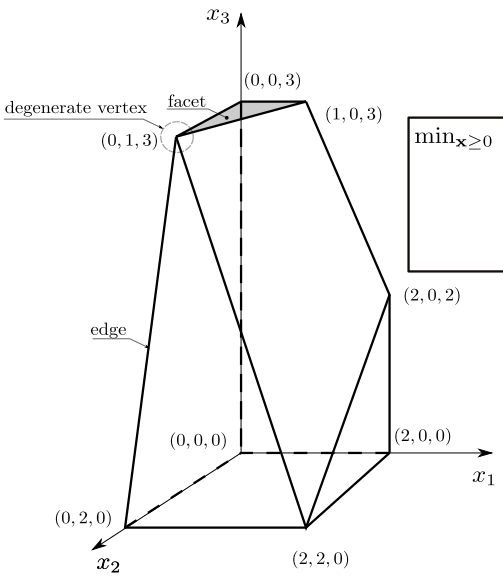
The proof of Theorem 7.13 follows directly from convexity of the problem $\min_{\mathbf{x}' \in P} \langle \mathbf{c}, \mathbf{x}' \rangle$, which implies that each local minimum is a global one, see Proposition 4.19.

We will consider the algorithm on an example, see Figure 7.3.

Example 7.15 (Simplex Method, Example 2.2 in [16])**.**

Consider the linear program in Figure 7.3.

Transformation into standard form: The simplex algorithm requires the LP to be in a standard form. Therefore, we convert inequalities to equalities by introducing slack variables



$$\begin{array}{lrcrcrcl} \min_{\mathbf{x}\geq 0} & -x_1 & - & 14x_2 & - & 6x_3 & & \\ & x_1 & + & x_2 & + & x_3 & \leq & 4 \\ & x_1 & & & & & \leq & 2 \\ & & & & & x_3 & \leq & 3 \\ & & & 3x_2 & + & x_3 & \leq & 6 \end{array}$$

Figure 7.3: Linear program for Example 7.15 and the respective polytope. Interestingly, there is a one-to-one relation between vertices of this polytope and the one obtained by adding slack variables to transform the LP into standard form [16, Lemma 19.2]. We illustrate transition between vertices in the course of the simplex algorithm in the lecture slides.

$$\begin{array}{rcrcrcrcl} x_1 & + & x_2 & + & x_3 & + & s_1 & = & 4 \\ x_1 & & & & & + & s_2 & = & 2 \\ & & & & x_3 & + & s_3 & = & 3 \\ & & 3x_2 & + & x_3 & + & s_4 & = & 6 \end{array} \tag{7.85}$$

Simplex Tableau: Treat objective as equality. Rewrite the problem as the following system of linear equations with the objective z being one of the variables:

$$\begin{array}{rcrcrcrcrcl}
x_1 & + & x_2 & + & x_3 & +s_1 & & & & = & 4 \\
x_1 & & & & & & +s_2 & & & = & 2 \\
& & & & x_3 & & & +s_3 & & = & 3 \\
& & 3x_2 & + & x_3 & & & & +s_4 & = & 6 \\
-z - x_1 & - & 14x_2 & - & 6x_3 & & & & & = & 0
\end{array} \tag{7.86}$$

In a compact form this takes the form of *simplex tableau:*

	x_1	x_2	x_3	s_1	s_2	s_3	s_4	RHS
s_1	1	1	1	1	0	0	0	4
s_2	1	0	0	0	1	0	0	2
s_3	0	0	1	0	0	1	0	3
s_4	0	3	1	0	0	0	1	6
$-z$	-1	-14	-6	0	0	0	0	0

(7.87)

Note that we already have a feasible solution to the problem: $x_i = 0$ for $i = 1, 2, 3$, $s_1 = 4$, $s_2 = 2$, $s_3 = 3$, $s_4 = 6$. It corresponds to the objective value $z = 0$.

Basis solution The simplex algorithm performs linear operations (*i.e.*, multiplication of constraints by a constant and adding one constraint to another) with the constraints defining simplex tableau such that:

- The number of constraints remains the same.
- These constraints define a linear system *equivalent* to the initial one.
- On each iteration of the simplex method the simplex tableau has the form:

	x_1	x_2	x_3	$x_4 \dots x_n$	RHS
x_1	1	0	0	...	v_1
x_2	0	1	0	...	v_2
x_3	0	0	1	...	v_3
$-z$	...	...	...	...	O

(7.88)

W.l.o.g. we shifted the variables with 0/1 coefficients (here x_1, x_2 and x_3) to the left side of the tableau. The variables x_1, x_2, x_3 take the *non-negative* values v_1, v_2, v_3. In this case the vector $(v_1, v_2, v_3, 0, \dots, 0)$ is the feasible solution corresponding to the objective value $-O$. This solution is called *a basic (feasible) solution* or simply *basis*. Respectively, variables x_1, x_2, x_3 defining this solution are referred to as *basic variables*. The costs in the objective value row except for the RHS, are called *reduced costs*. Essentially in the course of the algorithm we switch between different basic solutions, where m (number of constraints) variables get a (potentially) non-zero value and others - zero. The following proposition states the most important property of the basic solutions:

Proposition 7.16. Let $P = \{A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq 0\}$ define a convex polytope. Then each basic feasible solution is a vertex of this polytope.

Proof. Indeed, let $\mathbf{x} = (v_1, \dots, v_m, 0, \dots, 0)$ be a basic feasible solution. Consider the cost vector $\mathbf{c}$ of the form

$$c_i = \begin{cases} 0, & i \in [m]\,, \\ 1, & i > m\,. \end{cases} \tag{7.89}$$

The objective value corresponding to $\mathbf{x}$ is equal to 0, which implies its minimality, as $\mathbf{c} \geq 0$ and $\mathbf{x} \geq 0$. It remains to show the uniqueness of the solution.

Consider now the equivalent representation of the system $A\mathbf{x} = \mathbf{b}$ that has a form (7.88) *w.r.t.* $\mathbf{x}$. Let $\mathbf{y} \geq 0$ be another solution, *i.e.*, $\langle \mathbf{c}, \mathbf{y} \rangle = 0$. Then $y_i = 0$ for $i > m$. Let $\mathbf{z}^m$ be the first m coordinates of any vector $\mathbf{z}$ and I^m be the $m \times m$ identity matrix. Feasibilty of $\mathbf{y}$ implies $A\mathbf{y} = \mathbf{b}$, and, therefore, $I^m \mathbf{y}^m = \mathbf{v}^m$, in the equivalent matrix

representation (7.88). This implies $y_i = v_i$ for $i \in [m]$. Therefore, $\mathbf{y} = \mathbf{x}$. □

The inverse implication that each vertex is a basic solution, holds as well:

Proposition 7.17. Let A have a full rank. Then each vertex of a non-empty convex polytope $P = \{A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geq 0\}$ is representable as a basic feasible solution in the respective simplex tableau.

The proof can be found, in, *e.g.*, [16, Thm 2.4].

Let us get back to the simplex tableau (7.87). Its last line contains coefficients of the respective variables in the objective function. Negative values of some of these coefficients imply that increasing the value of the respective variable (and possibly decreasing the value of another one to keep feasibility of the solution) will decrease the objective value. In our case such candidate variables are x_1, x_2, x_3.

Identify the variable entering the basis Any of the variables with the negative coefficient in the objective row can be potentially added to the basis. There are several practical rules of selecting one of these variables, although there are no theoretical results claiming that one rule is actually better than another one, see, *e.g.*, [16]. We will follow the rule of selecting the variable with the smallest respective coefficient in the objective row. The smallest value in the objective row is -14, so x_2 is the variable that enter the basis solution. As x_2 grows, the objective value takes the value $z = -0 - 14x_2$. Therefore, it is reasonable to set x_2 as large as possible to maximally decrease z. However, x_2 usually cannot be increased indefinitely, as we should guarantee fulfillment of the constraints and non-negativity of all variables.

The basic variables $s_1, -s_4$ depend on x_2 as follows (given that $x_1 = x_3 = 0$):

$$
\begin{aligned}
s_1 = 4 - x_2 \quad &: \quad s_1 \geq 0 \;\Rightarrow\; x_2 \leq 4/1 = 4 && (7.90)\\
s_2 = 2 \quad &: \quad s_2 \geq 0 \;\Rightarrow\; x_2 \leq 2/0 = \infty && (7.91)\\
s_3 = 3 \quad &: \quad s_3 \geq 0 \;\Rightarrow\; x_2 \leq 3/0 = \infty && (7.92)\\
s_4 = 6 - 3x_2 \quad &: \quad s_4 \geq 0 \;\Rightarrow\; x_2 \leq 6/3 = 2 && (7.93)\\
& && (7.94)
\end{aligned}
$$

We see that if x_2 increases beyond $6/3 = 2$ then s_4 becomes negative and if x_2 increases beyond $4/1 = 4$ then also s_2 becomes negative. To keep all variables non-negative we are allowed to set x_2 to the maximum value of 2. This also implies that s_4 gets the value 0 and leaves the basis.

The value of variable x_i entering the basis is in general defined as the minimal non-negative ratio RHS_j/a_{ji} for all current basis variables x_j. Negative and infinite ratios are ignored as they do not restrict the value of x_i. However, should this ratio be negative or infinite for all basis variables, the initial problem is unbounded and the algorithm can be stopped.

Pivot on x_2 in the fourth (s_4) row Now our goal to bring the tableau to the form (7.88), with x_2 being the new basis variable in place of s_4. First we normalize the (fourth) pivot row by dividing its elements by 3 to obtain the coefficient 1 in front of x_2:

	x_1	x_2	x_3	s_1	s_2	s_3	s_4	RHS
s_1	1	1	1	1	0	0	0	4
s_2	1	0	0	0	1	0	0	2
s_3	0	0	1	0	0	1	0	3
x_2	0	1	$\frac{1}{3}$	0	0	0	$\frac{1}{3}$	2
$-z$	-1	-14	-6	0	0	0	0	0

(7.95)

Afterwards, we eliminate x_2 from all other rows including the objective function row by subtracting it from them with the suitable multipliers (1, 0, 0 and -14 for rows $1, 2, 3, 5$ from top to bottom):

	x_1	x_2	x_3	s_1	s_2	s_3	s_4	RHS
s_1	1	0	$\frac{2}{3}$	1	0	0	$-\frac{1}{3}$	2
s_2	1	0	0	0	1	0	0	2
s_3	0	0	1	0	0	1	0	3
x_2	0	1	$\frac{1}{3}$	0	0	0	$\frac{1}{3}$	2
$-z$	-1	0	$-\frac{4}{3}$	0	0	0	$\frac{14}{3}$	28

(7.96)

Pivoting is exchanging variables in the basis. One can show (see, *e.g.*, [16, Thm. 2.10]) that changing the basis means moving to one of the adjacent vertices of the polytope in a non-degenerate case.

Identify the variable entering the basis: The smallest negative value in the objective row is $-\frac{4}{3}$, so x_3 is the new entering variable.

Identify the variable leaving the basis (minimum ratio test):

$$s_1 = 2 - \frac{2}{3}x_3 \quad : \quad s_1 \geq 0 \Rightarrow x_3 \leq 2/(2/3) = 3 \tag{7.97}$$

$$s_2 = 2 \quad : \quad s_2 \geq 0 \Rightarrow x_3 \leq 2/0 = \infty \tag{7.98}$$

$$s_3 = 3 - x_3 \quad : \quad s_3 \geq 0 \Rightarrow x_3 \leq 3/1 = 3 \tag{7.99}$$

$$x_2 = 2 - \frac{1}{3}x_3 \quad : \quad x_2 \geq 0 \Rightarrow x_3 \leq 2/(1/3) = 6 \tag{7.100}$$

$$\tag{7.101}$$

The minimum ratio is 3, so s_1 or s_3 can is leave the basis. Let us select s_3.

Pivot on x_3 in the third (s_3) row Normalize the (third) pivot row: Nothing to be done, the row is normalized:

	x_1	x_2	x_3	s_1	s_2	s_3	s_4	RHS
s_1	1	0	$\frac{2}{3}$	1	0	0	$-\frac{1}{3}$	2
s_2	1	0	0	0	1	0	0	2
x_3	0	0	1	0	0	1	0	3
x_2	0	1	$\frac{1}{3}$	0	0	0	$\frac{1}{3}$	2
$-z$	-1	0	$-\frac{4}{3}$	0	0	0	$\frac{14}{3}$	28

(7.102)

Eliminate x_3 from the other rows by subtracting it with multipliers 2/3, 0, 1/3 and $-4/3$ respectively:

	x_1	x_2	x_3	s_1	s_2	s_3	s_4	RHS
s_1	1	0	0	1	0	$-\frac{2}{3}$	$-\frac{1}{3}$	0
s_2	1	0	0	0	1	0	0	2
x_3	0	0	1	0	0	1	0	3
x_2	0	1	0	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
$-z$	-1	0	0	0	0	$\frac{4}{3}$	$\frac{14}{3}$	32

(7.103)

Identify the variable entering the basis: The only negative value in the objective row is -1, so x_1 is the new entering variable.

Identify the variable leaving the basis (minimum ratio test):

$$
\begin{aligned}
s_1 = 0 - x_1 \quad &: \quad s_1 \geq 0 \;\Rightarrow\; x_1 \leq 0/1 = 0 && (7.104)\\
s_2 = 2 - x_1 \quad &: \quad s_2 \geq 0 \;\Rightarrow\; x_1 \leq 2/1 = 1 && (7.105)\\
x_3 = 3 \quad &: \quad x_3 \geq 0 \;\Rightarrow\; x_1 \leq 3/0 = \infty && (7.106)\\
x_2 = 1 \quad &: \quad x_2 \geq 0 \Rightarrow\; x_1 \leq 1/0 = \infty && (7.107)\\
& && (7.108)
\end{aligned}
$$

The minimum ratio is 0, so s_1 leaves the basis. Since the ratio is 0, the respective vertex is degenerate.

Degeneracy of basic solutions Should the decisive ratio RHS_j/a_{ji} be equal to 0, the variable x_i enters the basis with the zero value and one speaks about *degenerate* solution. This is precisely the case when no improvement in the objective value is attained after the basis change. It happens when the same vertex of the polytope can be defined by multiple basis solutions (but the same full solution, as in this case we just select a subset of variables with 0 value to belong to the basis). Geometrically it means a degeneracy of the polytope vertex, when it is build by an intersection of more than minimally necessary number of facets (see [16, Sec.2.3.1]). For example, for a 3-dimensional polytope this implies more than 3 hyper-planes, see, *e.g.*, the vertex $(2, 2, 0)$ in Figure 7.3. Special rules to avoid infinite looping are proposed for handling degenerate cases, see, *e.g.*, [14], [16].

Exercise 7.18. Prove that if there is no degeneracy, $c_i < 0$ and $a_{ji} > 0$ for at least one j from the basis, the objective value strictly decreases.

Pivot on x_1 in the first (s_1) row Normalize the (first) pivot row: Nothing to be done, the row is normalized:

	x_1	x_2	x_3	s_1	s_2	s_3	s_4	RHS
x_1	1	0	0	1	0	$-\frac{2}{3}$	$-\frac{1}{3}$	0
s_2	1	0	0	0	1	0	0	2
x_3	0	0	1	0	0	1	0	3
x_2	0	1	0	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
$-z$	-1	0	0	0	0	$\frac{4}{3}$	$\frac{14}{3}$	32

(7.109)

Eliminate x_1 from the other rows by subtracting it with multipliers 1, 0, 0 and -1 respectively:

	x_1	x_2	x_3	s_1	s_2	s_3	s_4	RHS
x_1	1	0	0	1	0	$-\frac{2}{3}$	$-\frac{1}{3}$	0
s_2	0	0	0	-1	1	$\frac{2}{3}$	$\frac{1}{3}$	2
x_3	0	0	1	0	0	1	0	3
x_2	0	1	0	0	0	$-\frac{1}{3}$	$\frac{1}{3}$	1
$-z$	0	0	0	1	0	$\frac{2}{3}$	$\frac{13}{3}$	32

(7.110)

All reduced costs are non-negative, so we stop.

Stopping condition Indeed, considering any positive value of the non-basic variables cannot decrease the objective, if all reduced costs are non-negative. Therefore, the optimum is attained. A formal proof can be found in [14], [16].

Solution:
At this point, all coefficients in the objective function row are non-negative, indicating an optimal solution.
The minimum value of the objective function z is -32, occurring at $(x_1, x_2, x_3) = (0, 1, 3)$. One can see in Figure 7.3 that the vertex $(0, 1, 3)$ is indeed degenerate.

Computational complexity of the simplex method

- The naïve simplex method requires $O(nm)$ operations per iteration for the $A \in \mathbb{R}^{m \times n}$.
- Assuming the average $O(n)$ iterations to attain the optimum results in $O(n^2 m)$ time complexity.
- Most combinatorial problems are very sparse - each constraint contains $\ll n$ non-zero coefficients. To make use of this a *revised simplex method* [14, Sec.3.5.3] is used, its complexity is $O(rnm + m^2)$, where r is fraction of the non-zero coefficients. For the problems with $m \ll n$ this gives nearly linear (that considers only non-zero coefficients in A) complexity of each iteration. Still, even for this method the number of iterations required to attain the optimum or at least get close to it grows proportionally to n.
- In total, simplex method usually requires around $O(n^2)$ operations, which can be prohibiting to large-scale problems.

Reduced costs as reparametrization Consider a linear program in the standard form (7.84). Reduced costs produced by the simplex method correspond to the reparametrized costs $\mathbf{c}^{\boldsymbol{\lambda}} = \mathbf{c} + A^\top \boldsymbol{\lambda}$ of the following Lagrange dual

$$\max_{\boldsymbol{\lambda}} -\langle \boldsymbol{\lambda}, \mathbf{b} \rangle + \min_{\mathbf{x} \geq 0} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle . \tag{7.111}$$

The j-th coordinate λ_j of the dual vector is the negated sum of multipliers applied to the j-th constraint before subtracting it from the cost line of the simplex tableau.

Note that before reaching the optimum, the vector $\mathbf{c}^{\boldsymbol{\lambda}} = \mathbf{c} + A^\top \boldsymbol{\lambda}$ has negative coordinates. It renders the inner minimization is unbounded, so the respective lower bound is trivial (equal to $-\infty$).

Advantages of the simplex method At the same time simplex method has nearly unique advantages that make it an ultimate tool for branching, cutting, *pricing* and learning:

Algorithmic advantages. The following operations are very cheap, efficient or can be done "on the fly":

- "Warm" start: Small changes of costs, *e.g.*, during learning, are likely to change the optimal solution only slightly, so it will be reachable within several iterations from the current one. The respective new reduced costs can be computed from the tracked dual variables.

- Removing columns (variables) is required, *e.g.*, by fixing variable value during branching. The column with a non-basis variable can directly be removed from the simplex tableau and the RHS corrected respectively.

- Adding columns corresponds to adding variables and is an important for *branch-and-price* algorithms that are beyond of scope of this course. Adding a column corresponding to a non-basis variable into the simplex tableau is straightforward.

- Adding constraints is required for cutting plane and branch-and-cut algorithms. By adding a new inequality constraint one automatically adds the respective slack variable to the basis. Transformation of the added inequality to the canonical form of the simplex tableau (7.88) requires $O(m)$ operations.

7.5 Bibliography

For further reading on branch-and-bound, branch-and-cut and cutting plane methods we refer to the text-book [90]. The basic ILP presolve operations described in this chapter were adopted from [83]. A detailed analysis of the presolve routines used in the Gurobi MIP solver [91] can be found in [89]. Another interesting source of information for off-the-shelf MILP solvers is [74]. We mostly followed [16] when describing general, canonical and standard forms of linear programs. Our exposition of the simplex algorithm mostly follows [16] and [14].

8 Primal Heuristics

8.1 Memetic Algorithms

Primal heuristics are algorithms operating in the domain of the primal ILP problem $\min_{\mathbf{x}\in P\cap\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle$, that have a weak theoretical substantiation and do not guarantee to attain the optimum. In this chapter, we overview one of the most powerful and general class of primal heuristics, known as *memetic algorithms*. Memetic algorithms contain three types of subroutines, see Figure 8.1:

1. **Generation** of *feasible* solutions, sometimes also referred to as *solution proposals*.
2. **Local search** to improve the available feasible solutions.
3. **Recombination** (also known as *crossover/fusion/merging*) of the feasible solutions. Here one attempts to build a better feasible solution on a basis of two or more proposals that are *fused/merged* together.

In the most general scenario one *generates* a pool of the feasible solutions that are improved with the *local search*. One selects a pair of the existing solutions, *recombines* them and adds them back to the pool. The most sophisticated scenarios manage several pool "generations" and a specialized selection procedure for the pair of solutions to *recombine*. The simplest approach used in practice is a iterative recombination of the current solution with the newly generated (and improved by local search) solution proposal. The result of the recombination is considered as the new current solution. Depending on the problem at hand, local search and recombination can be more

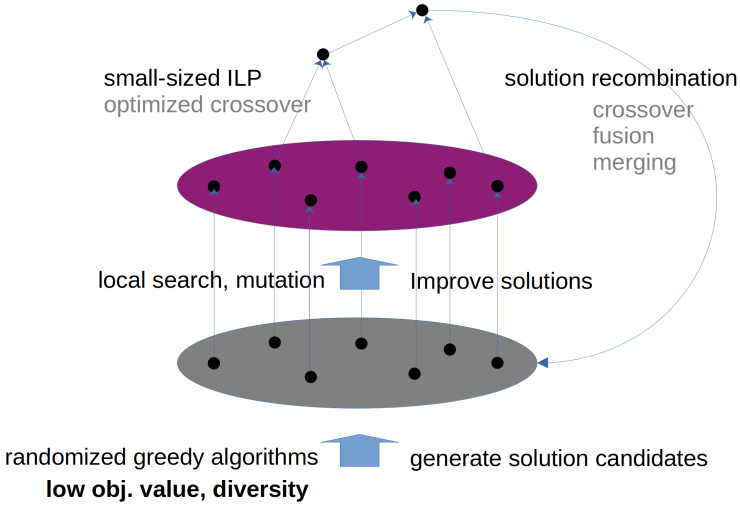


Figure 8.1: Memetic algorithms.

or less computationally expensive and efficient. Therefore, one may employ only one of these two types of algorithms. For example, the initial GRASP method [92] has used only generation and local search operations and the best found solution was chosen as a result. The respective algorithms have been applied to a great variety of combinatorial optimization problems, see [93, Sec. 7]. Another special case, often referred to as *genetic algorithms* [94] often do not contain the local search subroutine, *e.g.*, [38], [95].

Proposal generation should satisfy two criteria:

- **Quality** means that the proposals should have *good* (*low* for minimization problems) objective value to produce possibly good final solutions.
- **Diversity** is required to efficiently search through the solution space.

Often *randomized greedy* algorithms are used to generate proposals, *e.g.*, in the GRASP method [93]. The "greedy" is responsible for the quality and "randomized" for the diversity of the proposals. In Section 8.2 we cover this type of methods in more details.

Local search allows to improve quality of the proposals. The special case of the local search is the proposal *mutation*, when a subset of variables is fixed and one searches for the new feasible solution by optimizing over the rest of them. One searches for a better solution in the given *neighborhood* of a current one and, if the better solution is found, it is considered to be the current one and the process repeats until no better solution is found. In the case of mutation the neighborhood is defined by all possible values of the free, non-fixed variables.

This type of methods is very much problem specific, as, on one side, one has to consider possibly large search neighborhood (*i.e.*, many free variables) for the best results. On the other side, larger neighborhoods are typically computationally expensive. The neighborhood that guarantees attaining an optimal solution in the course of the local search is called *exact*. Although the necessary and sufficient conditions for an exact neighborhood are known [16, Ch. 19], it is often NP-hard to optimize over it. We consider several examples and the theoretical construction of the exact neighborhood in Section 8.3.

Recombination is most often termed *crossover* in the literature, however the latter term is reserved to the recovering of a basic solution from the interior point one (see Chapter 7) in this monograph. Recombination can be done in may different ways, see, *e.g.*, [93], [94]. Here we restrict ourselves to the *optimized recombination*. Its idea is to fix the variables that are common to the two solutions that are merged, and optimize *w.r.t.* the rest. In other words, the optimized recombination is a special case of mutation, when the set of variables to be fixed is defined by two (or more) solution proposals.

Although the recombination problem may take different forms, in a number of cases it can be formulated as a *binary labeling problem*. The respective LP relaxation can be solved very efficiently and has *persistency* property important for optimization. We treat this type of methods in Section 8.4 and Chapter 9.

8.2 Greedy algorithms

Greedy algorithms play a central role in combinatorial optimization due to their simplicity, efficiency, and surprising effectiveness in many practical scenarios. These algorithms make a sequence of locally optimal choice – typically selecting the best immediate option without reconsidering previous decisions – with the hope that this leads to a globally optimal solution. While greedy methods do not always guarantee optimality, especially for NP-hard problems, they often provide good approximations at a fraction of the computational cost of exact methods. Their low overhead and ease of implementation make them a natural starting point for algorithm design, and they are frequently used within more sophisticated frameworks, such as local search, memetic algorithms, or branch-and-bound, to construct initial solutions or guide the search process.

Alongside a general definition of greedy algorithms for integer linear programs, this section explores their integration with the Lagrange dual. In this setting, greedy solutions are constructed using reparametrized costs derived from the dual variables, rather than the original objective coefficients. We show that, when equality constraints are dualized, this costs reweighting can yield theoretical guarantees for the resulting greedy solutions. In practice, this often translates to improved performance, particularly when such dual-informed greedy algorithms are used within memetic methods.

8.2.1 Greedy algorithms for binary ILPs

Consider the ILP

$$\min_{\substack{\mathbf{x}\in P\cap\{0,1\}^n \\ A\mathbf{x}\leqq\mathbf{b}}} \langle \mathbf{c}, \mathbf{x}\rangle \ , \tag{8.1}$$

where the notation $A\mathbf{x} \leqq \mathbf{b}$ stands for an arbitrary set of inequalities and equalities. As in the case with the Lagrange relaxation in Chapter 5, we will assume that the problem without constraints $A\mathbf{x} \leqq \mathbf{b}$ is efficiently solvable.

We partition coordinates of the solution vector $\mathbf{x}$ into k non-intersecting blocks with the sizes $n_1 + n_2 + \cdots + n_k = n$. The respective variables will be denoted as $\mathbf{x}^l \in \{0,1\}^{n_l}$. For the sake of notation we will assume these blocks to be consecutive, *i.e.*, $\mathbf{x}^1 = (x_1, \ldots, x_{n_1})$, $\mathbf{x}^2 = (x_{n_1+1}, \ldots, x_{n_1+n_2})$, $\mathbf{x}^l = (x_{n_1+\cdots+n_{l-1}+1}, \ldots, x_{n_1+\cdots+n_l})$ for $l = 2, \ldots, k$. We will also denote vectors $\mathbf{x}^{l-}$ as a concatenation of $\mathbf{x}^1, \ldots, \mathbf{x}^{l-1}$, and $\mathbf{x}^{l+} := (\mathbf{x}^{l+1}, \ldots, \mathbf{x}^k)$. In the smae way we will also index the respective blocks of the cost vector, *i.e.*, $\mathbf{c}^l \in \mathbb{R}^{n_l}$, $l \in [k]$.

Algorithm 10 Greedy algorithm for a general ILP problem

1: **for** $l \in [k]$ **do**
2: Assign $\hat{\mathbf{c}} := \mathbf{0}$
3: Assign $\hat{\mathbf{c}}^l := \mathbf{c}^l$
4:

$$\text{Find} \quad \mathbf{z} \in \left\{ \operatorname*{arg\,min}_{\substack{\mathbf{x}\in P\cap\{0,1\}^n \\ A\mathbf{x}\leqq\mathbf{b}}} \langle \hat{\mathbf{c}}, \mathbf{x}\rangle \,, \quad \text{s.t.} \quad \mathbf{x}^{l-} = \hat{\mathbf{x}}^{l-} \right\} \tag{8.2}$$

5: Assign $\hat{\mathbf{x}}^l := \mathbf{z}^l$
6: **end for**
7: **Return** $\hat{\mathbf{x}}$

The greedy Algorithm 10 is applicable to the general integer linear programs. Its main step (8.2) effectively optimizes only over the current block of variables $\mathbf{x}^l$. Variables from previous blocks $\mathbf{x}^{l-}$ are fixed

to the values already computed in the greedy solution $\hat{\mathbf{x}}^{l-}$, and variables from the future blocks $\mathbf{x}^{l+}$ do not affect the objective, as their corresponding cost entries in $\hat{\mathbf{c}}$ are set to zero. Yet, their presence in (8.2) guarantees feasibility of the resulting greedy solution $\hat{\mathbf{x}}$.

In general, however, Algorithm 10 may become computationally expensive. In many practical cases, though, the *greedy subproblem* (8.2) can be substantially reduced in size, *i.e.*, if one can omit the variables $\mathbf{x}^{l-}$ and $\mathbf{x}^{l+}$, as well as all constraints not involving $\mathbf{x}^{l}$. The resulting reduced problem has the form

$$\hat{\mathbf{x}}^{l} \in \operatorname*{arg\,min}_{\substack{\mathbf{x} \in P_l \cap \{0,1\}^{n_l} \\ A^l \mathbf{x} \leqq \mathbf{b}^l}} \langle \mathbf{c}^l, \mathbf{x} \rangle \,. \tag{8.3}$$

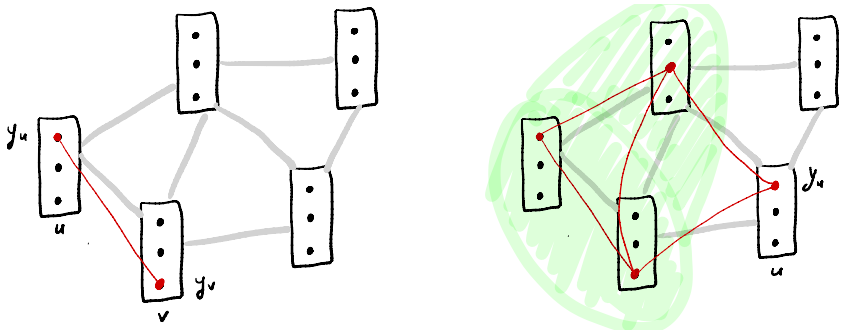
Here, $P_l \subseteq \mathbb{R}^{n_l}$ and $A^l \mathbf{x} \leqq \mathbf{b}^l$ denote a polytope and a subset of explicit constraints derived from the original problem (8.1) by fixing the values of $\mathbf{x}^{l-}$, discarding certain constraints (e.g., those involving only unprocessed variables $\mathbf{x}^{l+}$), and otherwise assigning values to $\mathbf{x}^{l+}$ to guarantee feasibility of the resulting greedy solution (see examples below). The greedy algorithm examples considered in this chapter take the simple form (8.3), hence we will assume it existence below.

The number of subspaces k, the polytopes P_l, $l \in [k]$, as well as their processing order may depend on the actual optimization results of the previous *greedy subproblems* (8.3). In other words, the already computed vectors $\mathbf{x}^s$, $s < l$, can determine the form of the polytope P_{l+1} that will be processed on the next iteration.

Algorithm 10 has no optimality guarantees in general. The special class of problems, known as *weighted matroids* [16, Ch. 12] can be solved with the algorithm exactly, though. This class includes, but not limited to *maximal spanning tree* in a graph and the matrix *rank computation*. We start with examples of the greedy algorithm for three combinatorial probems that do not belong to the above class.

Case study: Labeling problem

Let $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \boldsymbol{\theta})$ be a labeling problem defined on a graph $\mathcal{G}$ with label sets $\mathcal{Y}_u$, $u \in \mathcal{V}$, and the cost vector $\boldsymbol{\theta} \in \mathbb{R}^{\mathcal{I}}$. W.l.o.g., we assume $\mathcal{G}$



(a) (b)

Figure 8.2: Greedy algorithm for the labeling problem. **(a)** Initialization, see (8.4). **(b)** Greedy step, see (8.5)

to be connected. Let $u, v\colon uv \in \mathcal{E}$ be two neighboring nodes. The greedy algorithm starts with finding an optimal labeling in the induced subgraph defined by these nodes, see Figure 8.2(a):

$$(y_u, y_v) \in \operatorname*{arg\,min}_{s \in \mathcal{Y}_u, t \in \mathcal{Y}_v} \left(\theta_u(s) + \theta_v(t) + \theta_{uv}(s,t)\right) . \tag{8.4}$$

The subgraph $\mathcal{G}' := (u, v, uv)$ is called *labelled* if all its nodes already have labels. At each iteration, the algorithm expands the labelled subgraph by adding one new node along with all edges in $\mathcal{G}$ that connect this node to the previously labelled subgraph.

To do so, the algorithm finds a node u that is connected to the already labelled subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ and labels its by optimizing the sum of pairwise costs assigned to the edges between u and $\mathcal{G}'$ as well as the unary cost of u, see Figure 8.2(b):

$$y_u \in \operatorname*{arg\,min}_{s \in \mathcal{Y}_u} \left(\theta_u(s) + \sum_{v \in \mathcal{N}(u) \cap \mathcal{V}'} \theta_{uv}(s, y_v) \right) . \tag{8.5}$$

The labelled subgraph is updated: $\mathcal{V}' := \mathcal{V}' \cup \{u\}$, $\mathcal{E}' := \mathcal{E}' \cup (\mathcal{N}(u) \cap \mathcal{V}')$. The iterations proceed until $\mathcal{G}'$ becomes equal to $\mathcal{G}$.

Let us show that this greedy algorithm is a special case of the general one (8.3): Assign

$$P_1 := \Big\{(\mu_u, \mu_v, \mu_{uv})\colon \mu_u \in \Delta^{\mathcal{Y}_u};\ \mu_v \in \Delta^{\mathcal{Y}_v};\ \mu_{uv} \in \Delta^{\mathcal{Y}_{uv}}\Big\} \tag{8.6}$$

and let $A^1\mathbf{x} \leqq \mathbf{b}^1$ be the respective coupling constraints

$$\sum_{s\in\mathcal{Y}_u} \mu_{uv}(s,t) = \mu_v(t),\ t \in \mathcal{Y}_v;\ \sum_{t\in\mathcal{Y}_v} \mu_{uv}(s,t) = \mu_u(s),\ s \in \mathcal{Y}_u\,. \tag{8.7}$$

On each subsequent iteration, we define

$$P_l := \Big\{\mu_u, \mu_{uv}\colon v \in \mathcal{N}(u) \cap \mathcal{V}'\colon \mu_u \in \Delta^{\mathcal{Y}_u};\ \mu_{uv} \in \Delta^{\mathcal{Y}_{uv}}\Big\} \tag{8.8}$$

and let $A^l\mathbf{x} \leqq \mathbf{b}^l$ denote the corresponding coupling constraints. These constraints reflect that for every neighboring already labelled node $v \in \mathcal{N}(u) \cap \mathcal{V}'$ it holds that

$$\sum_{s\in\mathcal{Y}_u} \mu_{uv}(s,t) = [\![t = y_v]\!],\ t \in \mathcal{Y}_v;\ \sum_{t\in\mathcal{Y}_v} \mu_{uv}(s,t) = \mu_u(s),\ s \in \mathcal{Y}_u\,. \tag{8.9}$$

The greedy algorithm described above always results in a feasible integer labeling. However, this labeling may not be optimal in general, even if the problem graph G is acyclic.

Remark 8.1. Selecting the two-node subgraph for initialization is necessary to make the algorithm robust *w.r.t.* possible problem reparametrizations.

Starting with a single node labeling may results in a poor solution, *e.g.*, in the case when all unary costs are equal to zero. The latter state can be achieved, *e.g.*, by the distribution operation (6.38) of the min-sum diffusion algorithm.

On the other side, optimization of pairwise costs $\boldsymbol{\theta}_{uv}$ associated with a single edge uv is not sufficient for a robust initialization either. This is because any problem can be equivalently represented with the pairwise costs that contain as many as $\max\{|\mathcal{Y}_u, \mathcal{Y}_v|\}$ locally optimal

label pairs (pairwise costs). Such reparametrization can be obtained by,*e.g.*, executing the collection step (6.38) of the min-sum diffusion algorithm.

Remark 8.2. The order of processing nodes by the greedy algorithm can be arbitrary, as long as the connectivity condition $v \in \mathcal{N}(u) \cap \mathcal{V}'$ is satisfied. Different initial subgraphs and different processing order may lead to different resulting labelings.

Case study: Max-weight independent set problem

Consider the max-weight independent set problem as in Example 2.55 and assume all costs to be strictly positive, as nodes with non-positive costs can always be removed from the set without changing the optimal value.

The classical greedy algorithm [96] iteratively

1. Finds a node i with the highest *degree-normalized* cost $\frac{c_i}{|\mathcal{N}(i)|}$ in the *current* graph. Here $\mathcal{N}(i)$ is the set of neighboring nodes to i in the current graph and $|\mathcal{N}(i)|$ is its degree. The best node $i \in \arg\max_{j \in \mathcal{V}} \frac{c_j}{|\mathcal{N}(j)|}$ is labelled as a part of the resulting independent set.

2. Removes the node i from the graph together will all its neighbors.

The iterations proceed until the resulting graph becomes empty. The selected nodes form an independent set by construction.

The degree-normalization takes into account that nodes with many neighbors are less preferable, as they block all their neighbors from being part of the required independent set.

An alternative, *naïve* greedy approach, where on each iteration just a node with the largest cost is selected, performs significantly worse in practice [97] and lacks any guarantees in theory.

Better properties of the greedy algorithm based on the degree-normalized costs have their price: On each iteration of the algorithm

degrees of the nodes are changing, so keeping track of them results in the super-linear complexity of the whole algorithm.

This algorithm fits into the general scheme (8.3) as follows:

- The number of subspaces k is equal to the number of nodes in the initial graph.
- These subspaces fall into two categories: (i) those corresponding to the selected nodes with the highest degree-normalized cost, and (ii) those corresponding to their neighbors in the remaining graph. The associated subproblems are solved in the same sequence as in the described greedy algorithm – namely, iteratively first over a subspace of type (i) for the selected node, followed by the subspaces of type (ii) for its neighboring nodes. The latter subproblems may be processed in any order.
- For the subspaces of the type (i) the greedy subproblem (8.3) takes the form
$$\hat{x}_i \in \arg\max_{x \in \{0,1\}} c_i \cdot x \,, \tag{8.10}$$
which results in $\hat{x}_i = 1$ due to positivity of costs.
- For the subspaces of the type (ii) the greedy subproblem (8.3) takes the form
$$\hat{x}_j \in \arg\max_{\substack{x \in \{0,1\} \\ x \leq 0}} c_j \cdot x \,, \tag{8.11}$$
which results in $\hat{x}_j = 0$ due to the constraint $x \leq 0$. The latter follows from fixing x_i to $\hat{x}_i = 1$ in the constraint $x_i + x_j \leq 1$.

Case study: Binary knapsack problem

Consider the binary knapsack problem as in Example 2.54:

$$\max_{\mathbf{x} \in \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle \tag{8.12}$$

$$\text{s.t. } \langle \mathbf{w}, \mathbf{x} \rangle \leq W \,. \tag{8.13}$$

Disregarding trivial cases, we assume $c_i > 0$, $w_i > 0$ and $w_i \leq W$.

Define *efficiency* of the item i as the ratio of the item cost to its volume: $\frac{c_i}{w_i}$.

The *greedy-knapsack* algorithm [12, Sec. 2.1] starts with the empty knapsack and adds items in the decreasing order of their efficiency if it is not prohibited by the volume constraint.

A small modification of the algorithm guarantees to find a solution with a total cost at least half as large as the optimal solution value [12, Thm. 2.5.4]. The algorithm's complexity is determined by the complexity of sorting the n items accordint to their efficiency and in the most general setting is $O(n \log n)$.

In contrast, a *naïve-greedy-knapsack* algorithm uses costs instead of efficiencies and has no approximation guarantees.

The *greedy-knapsack* algorithm fits the general scheme (8.3) as follows:

- The number of subspaces k is equal to the number of items;
- The subspaces are enumerated in the descreasing order of items efficiencies. W.l.o.g. we will assume items to be enumerated in the same order.
- Each particular subproblem (8.3) takes the form

$$\hat{x}_l \in \operatorname*{arg\,max}_{\substack{x \in \{0,1\} \\ w_l x \leq W^l}} c_l \cdot x \,, \tag{8.14}$$

  where $W^l = b - \sum_{s=1}^{l-1} w_s \hat{x}_s$.

  Since $c_l > 0$ the solution is $\hat{x}_l = [\![w_l \leq W^l]\!]$.

Remark 8.3. A variant of the *greedy-knapsack* algorithm that stops when it fails to add an item for the first time is called the *greedy-split-knapsack* and can be computed in linear time as it does not require sorting of all elements [12, Sec. 3.1]. This algorithm has an intimate relation to the solution of the Lagrange relaxation of the knapsack problem.

Recall Example 5.20: The solution of the Lagrange relaxation reads $x_i = 1$ for $i \in [s-1]$, $x_s = (W - \sum_{i=1}^{s-1} w_i)/w_s$ and $x_t = 0$ for $t > k$, where s is the maximal value such that $\sum_{i=1}^{s-1} w_i < W$.

A simple "rounding" of this solution to an integer one consists in setting x_k to zero if it is not equal to one. Precisely this solution is returned also by the greedy-split-knapsack algorithm.

8.2.2 Leveraging Lagrange relaxation

In this section we restrict ourselves to the dualization of the equality constraints. The case of inequalities is beyond the scope of this monograph.

Consider the Lagrange dual of (8.1):

$$\max_{\boldsymbol{\lambda}} -\langle \boldsymbol{\lambda}, \mathbf{b} \rangle + \min_{\mathbf{x} \in P \cap \{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle . \tag{8.15}$$

As discussed in Section 5.2.2, the ILP

$$\min_{\substack{\mathbf{x} \in P \cap \{0,1\}^n \\ A\mathbf{x} = \mathbf{b}}} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle . \tag{8.16}$$

is equivalent to the original formulation

$$\min_{\substack{\mathbf{x} \in P \cap \{0,1\}^n \\ A\mathbf{x} = \mathbf{b}}} \langle \mathbf{c}, \mathbf{x} \rangle . \tag{8.17}$$

Therefore, the greedy algorithm Algorithm 10 can be equally applied to either problem. However, the solutions produced in the two cases may differ substantially, with typically higher-quality results obtained when $\boldsymbol{\lambda}$ is close to a dual optimum. This phenomenon, observed in multiple previous works, *e.g.*, [38], [95], is explained below.

The dual optimality condition (Corollary 5.18) implies that if there exists

$$\mathbf{x}^* \in \operatorname*{arg\,min}_{\mathbf{x} \in P \cap \{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle \tag{8.18}$$

such that

$$A\mathbf{x}^* = \mathbf{b} , \tag{8.19}$$

then $\mathbf{x}^*$ is the exact solution of the ILP (8.17). Given the separable structure of the set P, the problem (8.18) decomposes into k independent suproblems, one for each $l \in [k]$:

$$\hat{\mathbf{x}}^l \in \operatorname*{arg\,min}_{\mathbf{x} \in P_l \cap \{0,1\}^{n_l}} \left\langle [\mathbf{c}^{\boldsymbol{\lambda}}]^l, \mathbf{x} \right\rangle . \tag{8.20}$$

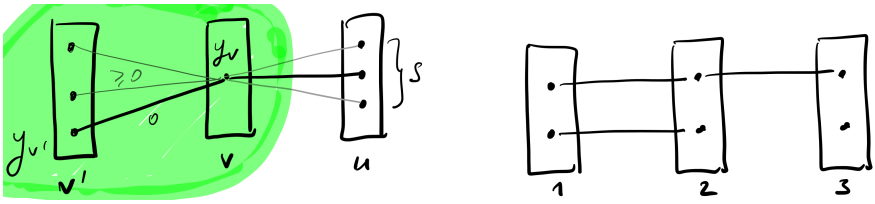
Now we aim to apply the greedy Algorithm 10 in its simplified form (8.3) with the reparametrized costs $\mathbf{c}^{\boldsymbol{\lambda}} = \mathbf{c} + A^\top \boldsymbol{\lambda}$. In this setting, the algorithm sequentially solves the subproblems (8.20), incorporating additional constraints $A^l \mathbf{x} = \mathbf{b}^l$ in order to select solutions of the subproblems that lead to a feasible overall solution $\hat{\mathbf{x}}$ as algorithm terminates. This procedure can be interpretet as an attempt to construct a solution that satisfies the dual optimality conditions (8.18)-(8.19). As these conditions are necessary and sufficient for optimality, this greedy algorithm finds an optimal ILP solution if the solutions $\mathbf{x}^l$ to all greedy subproblems

$$\hat{\mathbf{x}}^l \in \operatorname*{arg\,min}_{\substack{\mathbf{x} \in P_l \cap \{0,1\}^{n_l} \\ A^l \mathbf{x} = \mathbf{b}^l}} \left\langle [\mathbf{c}^{\boldsymbol{\lambda}}]^l, \mathbf{x} \right\rangle \tag{8.21}$$

also satisfy (8.20).

We will refer to the class of algorithms above as *reparametrized greedy algorithms*.

Example 8.4 (Acyclic labeling problem)**.** Let $\mathcal{G}$ be acyclic and the reparametrized costs $\boldsymbol{\theta}^{\boldsymbol{\phi}}$ be the fix-point of the min-sum diffusion algorithm, see Section 6.3.2. Proposition 5.30 together with Theorem 6.20 imply that the dual vector $\boldsymbol{\phi}$ is optimal. All reparametrized unary costs are zero. W.l.o.g. we will assume that also all locally optimal label pairs have zero reparametrized pairwise costs and otherwise all costs are non-negative. The latter can be achieved by subtracting the respective constant from all coordinates of $\boldsymbol{\theta}^{\boldsymbol{\phi}}_{uv}$ for each $uv \in \mathcal{E}$. Apply now the greedy algorithm (8.4)-(8.5) to the above problem while substituting the initial costs $\boldsymbol{\theta}$ with the reparametrized ones $\boldsymbol{\theta}^{\boldsymbol{\phi}}$. Below we will show that this algorithm finds an optimal integer solution if



(a) (b)

Figure 8.3: **(a)** Illustration to the proof of optimality of the reparametrized greedy algorithm for acyclic problems reparametrized with the diffusion algorithm, see Example 8.4 **(b)** All labels and shown label pairs have costs 0, otherwise – strictly positive. The same reparametrized greedy algorithm does not find the optimal labeling, if it starts with the edge $\{1, 2\}$ and selects the lower label pair as initial one and finds it, if it starts with the edge $\{2, 3\}$, see Remark 8.5.

the problem graph is acyclic. It is sufficient to show that the solution corresponds to the zero objective value.

Indeed, at the first step (8.4) the algorithm labels the connected subgraph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}') = (\{u, v\}, uv\})$ for arbitrary $uv \in \mathcal{E}$. The optimal labeling (y_u, y_v) of this subgraph has the total cost zero, since all unary costs are zero, and the optimal pairwise cost is zero by construction. Let now $\mathcal{G}'$ be the connected subgraph labelled at the previous iteration and let the respective labeling $\mathbf{y}_{\mathcal{G}'}$ have the total cost zero. The labelled subgraph obtained by extending it with the operation (8.5) is connected by construction. For the proof of the statement it is sufficient to show that the resulting labeling has zero cost as well.

Indeed, since the graph is acyclic, the set $\mathcal{N}(u) \cap \mathcal{V}'$ contains a single element v, see the proof of Proposition 5.30 for substantiation,

hence (8.5) turns into

$$y_u \in \arg\min_{s \in \mathcal{Y}_u} \left(\theta^{\phi}_u(s) + \theta^{\phi}_{uv}(s, y_v) \right) . \tag{8.22}$$

It holds $\theta^{\phi}_u(s) = 0$ for any $s \in \mathcal{Y}_u$ and $\theta^{\phi}_{uv}(s, y_v) \geq 0$ by construction, see Figure 8.3(a) for illustration. Therefore, it is sufficient to prove that there is $s \in \mathcal{Y}_u$ such that $\theta^{\phi}_{uv}(s, y_v) = 0$. The subgraph $\mathcal{G}'$ is connected, therefore there is $v' \in \mathcal{V}'$ such that $vv' \in \mathcal{E}'$. By assumption $\theta^{\phi}_{vv'}(y_v, y_{v'}) = 0$ and $(y_v, y_{v'})$ is locally optimal for the edge vv'. Therefore, it is locally optimal in the pencil, *i.e.*,

$$\min_{t \in \mathcal{Y}_{v'}} \theta^{\phi}_{vv'}(y_v, t) = 0 . \tag{8.23}$$

For the "redistribution" operation of the min-sum diffusion algorithm (6.38), as well as for its fix point, is holds $\min_{t \in \mathcal{Y}_{v'}} \theta^{\phi}_{vv'}(y_v, t) = \min_{s \in \mathcal{Y}_u} \theta^{\phi}_{uv}(s, y_v)$ for any two neighbors v' and u of v, see (6.40). Taking into account (8.23), we obtain $\min_{s \in \mathcal{Y}_u} \theta^{\phi}_{uv}(u, y_v) = 0$, which finalizes the proof.

Note that the greedy method (8.4)-(8.5) applied to the non-reparametrized costs does not have any optimality guarantee.

Remark 8.5. Performance and optimality guarantees of a particular greedy algorithm depend on the particular dual solution $\boldsymbol{\lambda}$ used for reparametrization. Two different optimal dual solutions may require two different greedy algorithms, with different order of processing variables. The example of acyclic labeling problem in Figure 8.3(b) demonstrates it, as its costs already correspond to the dual optimum ($\boldsymbol{\lambda} = \mathbf{0}$ is an optimal dual solution), but the above greedy approach maintains its optimility guarantee only if considers nodes of the problem in a particular order.

Example 8.6 (Max-weight independent set problem, slack-based clique ILP formulation)**.** Consider clique-cover ILP formulation of the max-weight independent set problem (see Exercise sheets for the related discussion):

$$\max_{\mathbf{x}\in\{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle \tag{8.24}$$

$$\text{s.t.} \sum_{i\in K_j} x_i \leq 1\,, j \in [m]\,. \tag{8.25}$$

Here K_j is the set of cliques covering all edges and nodes. By introducing m *slack* variables $x_{n+1}, \ldots, x_{n+m}$, one for each *clique constraint* (8.25), and assigning the corresponding cost vector coordinates $c_{n+1}, \ldots, c_{n+m}$ to zero, we obtain an equivalent, equality-constrained problem:

$$\max_{\mathbf{x}\in\{0,1\}^{n+m}} \langle \mathbf{c}, \mathbf{x} \rangle \tag{8.26}$$

$$\text{s.t.} \sum_{i\in \bar{K}_j} x_i = 1\,, j \in [m]\,. \tag{8.27}$$

Here $\bar{K}_j = K_j \cup \{x_{n+j}\}$. The slack variables in (8.26) can be interpreted as additional nodes in the graph, adjacent to all nodes of the respective clique.

By dualizing the constraints (8.27), we obtain the dual problem

$$\min_{\boldsymbol{\lambda}} \left[\sum_{j=1}^{m} \lambda_j + \max_{\mathbf{x}\in\{0,1\}^{n+m}} \langle \mathbf{c}^{\boldsymbol{\lambda}}, \mathbf{x} \rangle \right]\,. \tag{8.28}$$

Here the coordinates of the reparametrized cost vector are defined as $\mathbf{c}_i^{\boldsymbol{\lambda}} = \mathbf{c}_i - \sum_{j\in J_i} \lambda_j$ with $J_i \subseteq [m]$ being the set of cliques containing a given node i.

The reparametrized greedy Algorithm 11 considers the graph with additional nodes, one per clique, corresponding to the slack variables. The algorithm (randomly) selects a yet unprocessed clique and adds the node with the lowest cost in the clique to the solution proposal. This node and all adjacent nodes are removed from the graph. The process is repeated until no nodes are left.

In terms of the reparametrized greedy subproblems (8.21) this algorithm can be seen as an iterative repetition of the following two steps:

1. Select a yet unprocessed clique j and solve the reparametrized greedy subproblem

$$(\hat{x}_i)_{i\in\bar{K}_j} \in \operatorname*{arg\,max}_{\mathbf{x}\in\{0,1\}^{\bar{K}_j}} \sum_{i\in\bar{K}_j} c_i^{\boldsymbol{\lambda}} x_i \tag{8.29}$$

$$\text{s.t.} \sum_{i\in\bar{K}_j} x_i = 1 \tag{8.30}$$

$$x_S = \hat{x}_S\,, \tag{8.31}$$

   where the set S denotes all nodes in $\bar{K}_j$ that have been already fixed. Mark $\bar{K}_j$ as *processed*.

2. Let i^* be the selected node, *e.g.*, $\bar{x}_{i^*} = 1$, in the clique $\bar{K}_j$ addressed in Step 1. Then solve (8.29)-(8.31) for all $j \in J_{i^*}$ and mark the respective cliques as *processed*. Due to the constraint (8.31) this step essentially sets to zero all neighbors of i^*, analogously to the step (8.11) in the non-reparametrized greedy algorithm for the MWIS problem.

Note that Algorithm 11 finds an optimal solution, if $|\{i \in \bar{K}_j \colon c_i^{\boldsymbol{\lambda}} \geq 0\}| = 1$ as this solution satisfies the optimality criterion (8.20).

Remark 8.7. It can be shown that there exists an optimal dual solution $\boldsymbol{\lambda}$ such that $\mathbf{c}^{\boldsymbol{\lambda}} \leq 0$. Consequently, every clique contains at least one node whose reparametrized cost is zero. For such nodes, the (reparametrized) degree-normalized cost coincides with the (reparametrized) cost itself. Thus, when the LP relaxation is nearly tight and most nodes in the greedy solution have zero weight, the degree-normalized and naive greedy algorithms yield almost identical results. In this way, reparametrization renders the use of the more computationally expensive degree-normalized costs unnecessary.

Algorithm 11 Reparametrized greedy algorithm for the MWIS problem.

Input: $\mathbf{c}^{\boldsymbol{\lambda}} \in \mathbb{R}^{n+m}$ - reduced costs
Initialize $\mathbf{x} := -\mathbf{1} \in \mathbb{R}^{n+m}$ as undefined
Iterate over all cliques:
for $j \in [m]$ **do**
  Skip cliques with all fixed nodes:
  if $\{i \in \bar{K}_j \mid x_i = 1\} = \emptyset$ **then**
    Find an assignable and locally optimal node
    $i^* := \underset{i \in \bar{K}_j \text{ s.t. } x_i = -1}{\arg\max} c_i^{\boldsymbol{\lambda}}$
    $x_{i^*} := 1$ and set it to one
    and all its graph neighbors to zero:
    for $i \colon \{i^*, i\} \in \mathcal{E}$ **do**
      $x_i := 0$
    end for
  end if
end for
Return $\mathbf{x}$

Identification of an integer solution is NP-complete

Assume the Lagrange relaxation (8.15) is tight and there is an integer $\mathbf{x}^*$ that satisfies conditions (8.18)-(8.19). Note that it does not mean, that finding this $\mathbf{x}^*$ is an easy problem. On the contrary, this *identification* problem is NP-complete in general:

Definition 8.8. The decision problem: *Is there* $\mathbf{x} \in \{0,1\}^n$ *such that* $\sum_{i=1}^n a_i x_i = b$ is called the *subset sum problem.*

Theorem 8.9 (Sec. 8.8 of [98])**.** The subset sum problem is NP-complete.

The subset sum problem reduces to the problem of identification of an integer relaxed solution with $\mathbf{c}^{\boldsymbol{\lambda}} = 0$, $P = \mathbb{R}^n$ and $A\mathbf{x} = \mathbf{b}$ being equal to $\sum_{i=1}^n a_i x_i = b$.

The identification problem is in NP class, as for any $\mathbf{x}' \in \{0,1\}^n$ it can be verified in polynomial time that $A\mathbf{x}' = \mathbf{b}$ and

$$\left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle = \min_{\mathbf{x} \in P \cap \{0,1\}^n} \left\langle \mathbf{c} + A^\top \boldsymbol{\lambda}, \mathbf{x} \right\rangle , \tag{8.32}$$

since the right-hand-side is assumed to be known as the relaxed optimal value.

Due to NP-completeness, the reparametrized greedy algorithm can be seen as a computationally cheap method to find $\mathbf{x}^*$, however, without guarantee in general. To improve performance, its *randomized* versions are used, that allow to efficiently sample the search space around the optimum. We consider these algorithms in Section 8.2.3 below.

Non-optimal dual and non-tight relaxations

Apart from being comparably computationally cheep, the practical advantage of the reparametrized greedy algorithm is its applicability in a significantly broader setting than in the optimum of the tight Lagrange relaxation. Indeed, it remains applicable even when

- the Lagrange relaxation (8.15) is not tight, or
- the attained dual solution $\boldsymbol{\lambda}$ is non-optimal.

The latter is especially typical for large-scale problems, when one is limited in usage to the (non-polynomial) first-order dual optimization methods such as sub-gradient one. These are rarely able to attain the optimum with numerical precision. A relative approximation accuracy of 0.1% of the optimal dual value is often the best one can hope with the first-order techniques in large-scale practical applications.

A (well-defined) reparametrized greedy algorithm typically finds progressively better integer solutions as the dual variables get closer to the relaxed optimum. This effect gets even more pronouncing, when the Lagrange relaxation is nearly tight and the relaxed solutions have *mostly* integer coordinates.

8.2.3 Randomized greedy algorithms

As mentioned in Chapter 8, *randomized* greedy subroutines are a common tool for proposal generation in memetic algorithms. They can be used with original as well as with the reparametrized costs.

Order randomization Randomization may lead to the optimal solution (that satisfies (8.18) and (8.19)), if, *e.g.*, the processing of the (randomized) greedy algorithm iterations (8.3) (resp. (8.21)) is done in the order $\pi(l)$ instead of l with π being a *suitable* randomized permutation.

Example 8.10 (Labeling problem)**.** Consider the greedy algorithm described in Section 8.2.1 and applied to the initial or reparametrized costs. One can randomly select the initial pair of neighboring nodes $uv \in \mathcal{E}$, see (8.4), and on each iteration *randomly* select a neighboring node (node u in (8.5)).

Example 8.11 (Max-weight independent set problem)**.** Consider the greedy algorithm described in Example 8.6. The order of processing of the cliques can be randomly selected.

Cost randomization Another randomization method consists in randomizing the costs. There are several modifications of the technique:

- Instead of selecting the minimal (maximal) in (8.3) or (8.21), one selects elements randomly *w.r.t.* the distribution of their costs. Possible options are, *e.g.*, $\propto \exp(c/T)$ and uniform selection from the predefined number of the best choices [93].

- Adding noise to the costs during generation (see [93]). This can be in particular useful, if the existing greedy algorithm does not allow to other sorts of randomization.

  Adding noise may not be required, if the dual is being optimized with the subgradient method. In this case reparametrizations

computed on different iterations are diverse enough and "naturally randomized" due to the suboptimal length of the subgradient steps, see [95].

Costs randomization allows to explore a larger part of the solution space, although it may require longer to obtain good solutions. Usually it is used in combination with the order randomization and shows best results when the Lagrange relaxation (8.15) is far from being tight and/or the dual solution $\boldsymbol{\lambda}$ is highly non-optimal.

8.3 Local search

8.3.1 Main definitions

Definition 8.12. A mapping $\mathcal{N}\colon X \to 2^X$ from a set X to its powerset is called *neighborhood*.

Definition 8.13. Algorithm 12 is called *local search* for the lowest value of the function $f\colon X \to \mathbb{R}$ with respect to the neighborhood $\mathcal{N}$. Note that the special case of Algorithm 12, with $x^* \in \arg\min_{x \in \mathcal{N}(x^t)} f(x)$, is covered by this definition as well.

Algorithm 12 General local search algorithm

1: Input: Initial solution $x^0 \in X$
2: $t := 0$
3: **while** $\exists x^* \in \mathcal{N}(x^t)\colon f(x^*) < f(x^t)$ **do**
4: $\quad t := t + 1$
5: $\quad x^t := x^*$
6: **end while**
7: Return x^t.

Such algorithms as (block)-ICM (Section 6.3.1) and simplex method (Section 7.4.2) satisfy Definition 8.13. For continuous X, the coordinate- (Section 6.3) and gradient descent (Section 6.1) can be seen as local search.

8.3.2 Mutation

Mutation is the type of local search problems, when a subset of coordinates of a feasible solution $\mathbf{x}$ is *fixed* and the problem is optimized with respect to the rest of *free* coordinates. In case of general ILP solvers the resulting problem is obtained with domain propagation, see Section 7.3.1. Usually the mutation problem belongs to the same problem class as the initial one, but has a much smaller size. Hence in general one has four options how to address it, with

- The total *enumeration* of possible solutions;
- *Branching* (branch-and-bound/cut) as a more scalable way to obtain an exact solution;
- *Heuristic* algorithms (such as greedy ones) to obtain a feasible solution better than the current one;
- *Specialized* exact or approximate algorithms, if the resulting problem belongs to an easier solvable subclass of the more general initial problem.

Whereas mutation problems with a small set of feasible solutions are usually solved with simple and efficient enumeration algorithms, the subproblems with larger feasible set often allow to attain better objective values. The sweet spot are the mutation subproblems with exponentially large feasible set that build a special, efficiently solvable special case of the initial problem.

Example 8.14 (Labeling problem)**.** The ICM algorithm, see Section 6.3.1, is a classical mutation-based method. In the case of the single-node updates, as in Algorithm 7, these are usually computed with the total enumeration of all labels in a given node. In the case of the block-ICM, see Algorithm 8, the resulting subproblem belongs to the efficiently solvable subclass of the acyclic labeling problems and can be solved with the specialized, dynamic programming algorithm.

Example 8.15 (Binary knapsack)**.** Assume $\mathbf{x} \in \{0,1\}^n$ to be a feasible solution of the problem (2.30) and let all coordinates but x_i, $i \in [k]$, be fixed. Than the mutation problem takes the form

$$\max_{\mathbf{x}\in\{0,1\}^k} \sum_{i=1}^{k} c_i x_i \tag{8.33}$$

$$\text{s.t.} \sum_{i=1}^{k} w_i x_i \leq \hat{W} := W - \sum_{j=k+1}^{n} w_j x_j \,, \tag{8.34}$$

which is a binary knapsack problem with k variables and smaller maximal volume parameter $\hat{b}$. Hence it can be either addressed with the dynamic programming (if $\hat{W}$ is small enough), or one of the greedy algorithms (see Section 8.2.1) can be used to find an approximate solution.

Mutation for General ILPs Mutation is also used by off-the-shelf ILP solvers as one of the primal heuristics. In the simplest implementation one fixes a certain amount, (80-90%) of randomly selected variables and solves the problem with respect to the remaining ones. However, fixing of one variable may lead to the implicit fixing of another one, see Example 8.16. In turn, this may result in a very small feasible sets of the mutation problem leading to its overall inefficiency. The problem can be mitigated by interchangeable fixing of the variables and domain propagation until a certain number of *actually* free variables is attained.

Example 8.16 (Implicit variable fixing)**.** Consider the local polytope relaxation of the labeling problem (3.45). Let $uv \in \mathcal{E}$ and the node variables are fixed, *e.g.*, $\mu_u(s) = [\![s = y_u]\!]$ and $\mu_v(l) = [\![l = y_v]\!]$. This implies that all pairwise variables in the edge uv are implicitly fixed as well to $\mu_{uv}(s,l) = [\![s = y_u]\!] \cdot [\![l = y_v]\!]$ that significantly decreases the size of the mutation problem.

8.3.3 Exact neighborhood of binary ILPs

Definition 8.17. Let X be finite. A neighborhood $\mathcal{N}$ is called *exact*, if Algorithm 12 returns an element of $\arg\min_{x\in X} f(x)$.

In this section we formulate necessary and sufficient conditions for a local search neighborhood to be *exact*.

Consider the ILP:

$$\min_{x\in P\cap\{0,1\}^n} \langle \mathbf{c}, \mathbf{x}\rangle \ . \tag{8.35}$$

Recall (see Section 2.5) that the set $M = \mathrm{conv}(P \cap \{0,1\}^n)$ is called its *integer hull*.

For a given $\mathbf{x} \in \mathrm{vrtx}(M) = P \cap \{0,1\}^n$ let $\mathrm{Ad}(\mathbf{x}) \subseteq \mathrm{vrtx}(M)$ be the set of adjacent (see Definition 7.12) vertices of the polytope M.

In the following we are going to prove the following

Theorem 8.18 (Thm. 19.3 in [16])**.** A neighborhood $\mathcal{N}\colon \mathrm{vrtx}(M) \to 2^{\mathrm{vrtx}(M)}$ is exact for any cost vector $\mathbf{c}$ if and only if $\mathcal{N}(\mathbf{x}) \supseteq \mathrm{Ad}(\mathbf{x})$ for any $\mathbf{x} \in \mathrm{vrtx}(M)$.

To prove Theorem 8.18 we will require the following

Lemma 8.19. Let P be a polytope and $\mathrm{vrtx}(P) \in \{0,1\}^n$. Let also $\hat{\mathbf{x}}, \hat{\mathbf{y}} \in \mathrm{vrtx}(P)$. Then $\hat{\mathbf{y}} \in \mathrm{Ad}(\hat{\mathbf{x}})$ if and only if there exists $\mathbf{c} \in \mathbb{R}^n$ such that $\hat{\mathbf{x}}$ is a unique optimal solution and $\hat{\mathbf{y}}$ is the unique second best solution of $\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x}\rangle$.

Proof. **Necessity:** Assume $\hat{\mathbf{y}} \in \mathrm{Ad}(\hat{\mathbf{x}})$, *i.e.*, $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ belong to some edge of the polytope. By the definitions of the edge, there is a cost vector $\mathbf{c}$ such the edge is the set $\arg\min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x}\rangle$. In particular, $\langle \mathbf{c}, \hat{\mathbf{x}}\rangle = \langle \mathbf{c}, \hat{\mathbf{y}}\rangle = c^0 := \min_{\mathbf{x}\in P} \langle \mathbf{c}, \mathbf{x}\rangle$. Below we will slightly modify $\mathbf{c}$ to obtain $\mathbf{c}'$ such that the statement of the lemma holds.

Define $Q := \min_{\mathbf{x}\in \mathrm{vrtx}(P)\setminus\{\hat{\mathbf{x}},\hat{\mathbf{y}}\}} [\langle \mathbf{c}, \mathbf{x}\rangle - c^0] > 0$ as the minimal difference in the objective value between the edge $\hat{\mathbf{x}} - \hat{\mathbf{y}}$ and the rest of vertices of the polytope P.

Since $\hat{\mathbf{x}} \neq \hat{\mathbf{y}}$ there is $j \in [n]$ such that $\hat{y}_j \neq \hat{x}_j$ and only two cases considered below are possible.

Case 1: $\hat{y}_j = 1$, $\hat{x}_j = 0$. Consider vector $\mathbf{c}'$ such that $c'_j = c_j + Q/2$ and $c'_i = c_i$ for $i \in [n]\backslash\{j\}$.

Since $\hat{x}_j = 0$ $\langle \mathbf{c}', \mathbf{x} \rangle = \langle \mathbf{c}, \mathbf{x} \rangle = c^0$. As $\hat{y}_j = 1$ it follows that $\langle \mathbf{c}', \mathbf{y} \rangle = \langle \mathbf{c}', \mathbf{y} \rangle + Q/2 = c^0 + Q/2$.

At the same time for any $\mathbf{x} \in \mathrm{vrtx}(P)\backslash\{\hat{\mathbf{x}}, \hat{\mathbf{y}}\}$ it holds $\langle \mathbf{c}', \mathbf{x} \rangle \geq \langle \mathbf{c}, \mathbf{x} \rangle \geq c^0 + Q$. As a result, $\hat{\mathbf{x}}$ is the unique best and $\hat{\mathbf{y}}$ is the unique second best solution.

Case 2: $\hat{y}_j = 0$, $\hat{x}_j = 1$. Consider vector $\mathbf{c}'$ such that $c'_j = c_j - Q/2$ and $c'_i = c_i$ for $i \in [n]\backslash\{j\}$.

Since $\hat{x}_j = 1$ $\langle \mathbf{c}', \mathbf{x} \rangle = c^0 - Q/2$. As $\hat{y}_j = 0$ it follows that $\langle \mathbf{c}', \mathbf{y} \rangle = \langle \mathbf{c}', \mathbf{y} \rangle = c^0$.

At the same time for any $\mathbf{x} \in \mathrm{vrtx}(P)\backslash\{\hat{\mathbf{x}}, \hat{\mathbf{y}}\}$ it holds $\langle \mathbf{c}', \mathbf{x} \rangle \geq \langle \mathbf{c}, \mathbf{x} \rangle - Q/2 \geq c^0 + Q - Q/2 = c^0 + Q/2$. As a result, $\hat{\mathbf{x}}$ is the unique best and $\hat{\mathbf{y}}$ is the unique second best solution.

Thus, the necessity is proven.

Sufficiency: Assume $\hat{\mathbf{x}}$ is the unique best and $\hat{\mathbf{y}}$ is the second best solution for some cost vector $\mathbf{c}$ and $\hat{\mathbf{x}} \notin \mathrm{Ad}(\hat{\mathbf{y}})$. Then, according to Lemma 7.14, $\hat{\mathbf{y}}$ is the local optimum of the convex optimization problem $\min_{\mathbf{x} \in P} \langle \mathbf{c}, \mathbf{x} \rangle$. However, its unique global optimum is $\hat{\mathbf{x}}$, which is a contradiction, as for convex problems all local optima are global ones (consider the convex function $\langle \mathbf{c}, \mathbf{x} \rangle + \iota_P(x)$ and apply Proposition 4.19). □

Now we are ready to prove Theorem 8.18:

Proof. **Sufficiency:** For the sake of notation denote $X := P \cap \{0,1\}^n = \mathrm{vrtx}(M)$. The neighborhood $\mathcal{N}$ is exact for any cost vector

$\mathbf{c}$, by the simplex algorithm applied to

$$\min_{\substack{\mathbf{y} \\ p_{\mathbf{x}}:\, \mathbf{x}\in X}} \langle \mathbf{c}, \mathbf{y} \rangle \tag{8.36}$$

$$\text{s.t. } y = \sum_{\mathbf{x}\in X} p_{\mathbf{x}} \mathbf{x} \tag{8.37}$$

$$\sum_{\mathbf{x}\in X} p_{\mathbf{x}} = 1 \tag{8.38}$$

$$p_{\mathbf{x}} \geq 0\,,\ \mathbf{x} \in X \tag{8.39}$$

$$\mathbf{y} \geq 0 \tag{8.40}$$

Here we used the representation (2.37) of the polytope M and added the constraint $\mathbf{y} \geq 0$ as all $\mathbf{x}$ are binary integer vectors and, therefore, non-negative. Not that the used representation in an LP in the standard form, so the simplex algorithm is directly applicable.

Necessity: Assume $\hat{\mathbf{x}} \in \mathrm{Ad}(\hat{\mathbf{y}})$ and $\hat{\mathbf{x}} \notin \mathcal{N}(\hat{\mathbf{y}})$. Then, according to Lemma 8.19 applied to the polytope M, there is such a cost vector $\mathbf{c}$ that $\hat{\mathbf{x}}$ is the unique best solution and $\hat{\mathbf{y}}$ is the unique second best one. The respective local search, when started in $\hat{\mathbf{y}}$, gets stuck in this point, as $\langle \mathbf{c}, \hat{\mathbf{y}} \rangle < \langle \mathbf{c}, \mathbf{x} \rangle$ for any $\mathbf{x} \in \mathcal{N}(\hat{\mathbf{y}})$. □

Unfortunately, Theorem 8.18 has mainly a theoretical value, as even answering the question whether two solutions correspond to adjacent vertices of the integer hull is NP-hard in general, see [16, Ch.19].

8.4 Optimal recombination

Consider a general combinatorial optimization problem as given by Definition 1.1:

$$\min_{\mathbf{x}\in S\subseteq\mathbb{Z}^n} f(\mathbf{x})\,. \tag{8.41}$$

Let $\mathbf{x}^1, \mathbf{x}^2 \in S$ be two its feasible solutions.

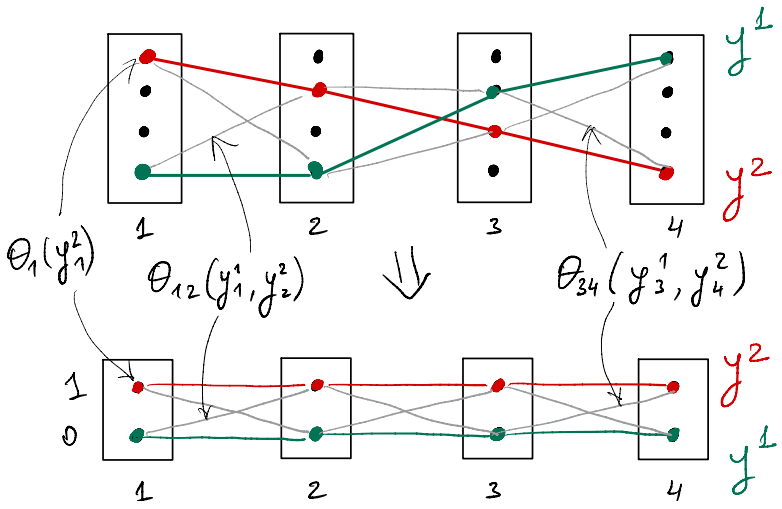


Figure 8.4: Optimal recombination for labeling problem. (Top) An initial problem and two feasible solutions. (Bottom) The resulting recombination problem. Numbers under nodes are node indices, number 0 and 1 to the left of the recombination problem are possible interpretation of the labels y_u^1 and y_u^2 as zero and one. See Example 8.21 for further description.

Definition 8.20. The problem

$$\min_{\mathbf{x} \in S \subseteq \mathbb{Z}^n} f(\mathbf{x}) \tag{8.42}$$

$$\text{s.t. } x_i = x_i^1 \quad \text{whenever } x_i^1 = x_i^2,\ i \in [n] \tag{8.43}$$

is referred to as *optimal recombination* (or *optimal crossover*). Since we restrict attention to this particular type of recombinations throughout this monograph, the qualifier "optimal" will be often omitted in the sequel.

Example 8.21 (Recombination for the labeling problem)**.** Let $(\mathcal{G}, \boldsymbol{\theta}, \mathcal{Y}_\mathcal{V})$ be a multilable labeling problem (see Section 1.5.1) and let $\mathbf{y}^1, \mathbf{y}^2 \in \mathcal{Y}_\mathcal{V}$

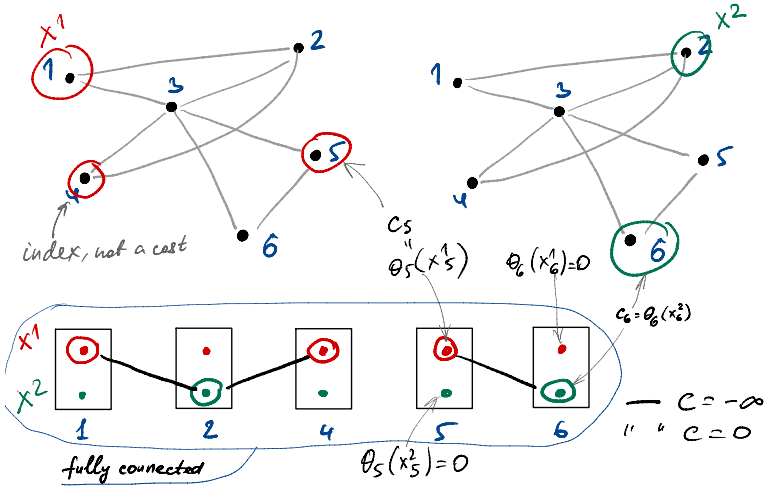


Figure 8.5: Optimal recombination for the max-weight independent set problem. (Top) Two feasible solutions of the same max-weight independent set problem. (Bottom) The resulting recombination problem. Numbers under nodes are node indices. The recombination problem is the binary labeling problem on a fully connected graph with costs $-\infty$ and 0. See Example 8.22 for further description.

be two labelings, see Figure 8.4. The recombination problem consists in finding a labeling $\mathbf{z} \in \mathcal{Y}_\mathcal{V}$ with the minimal total cost such that $z_u \in \{y_u^1, y_u^2\}$ for all $u \in \mathcal{V}$.

The recombination problem can be posed as a labeling problem itself, with $(\mathcal{G}', \boldsymbol{\theta}', \mathcal{Y}'_{\mathcal{V}'})$ with $\mathcal{G}' = \mathcal{G}$, $\mathcal{Y}'_u = \{y_u^1, y_u^2\}$ and $\theta'_u(y_u^i) = \theta_u(y_u^i)$, $\theta'_{uv}(y_u^i, y_u^j) = \theta_{uv}(y_u^i, y_u^j)$, where $i, j \in \{1, 2\}$.

In the above example the labeling problem corresponding to the optimal recombination has two labels in each node. Such labeling problems are called *binary*. Without loss of generality one can assume the two labels to be *zero* and *one*.

Example 8.22 (Recombination for the max-weight independent set problem)**.** Let the graph $(\mathcal{V}, \mathcal{E})$ and costs $\mathbf{c} \in \mathbb{R}^{\mathcal{V}}$ define a max-weight independent set problem, see ex:max-weight-independent-set. And let $\mathbf{x}^1, \mathbf{x}^2 \in \{0,1\}^{\mathcal{V}}$ be two independent sets defined by their indicator vectors, see Figure 8.5. The respective optimal recombination problem consist in finding an independent set $\mathbf{z} \in \{0,1\}^{\mathcal{V}}$ in $\mathcal{G}$ such that for all $i \in [|\mathcal{V}|]$ such that $x_i^1 = x_i^2$ it also holds $z_i = x_i^1$.

This problem can be posed as a binary labeling *maximization* problem $((\mathcal{V}', \mathcal{E}'), \boldsymbol{\theta}', \mathcal{Y}'_{\mathcal{V}'})$ with $\mathcal{V}' = \{i \in \mathcal{V} \colon x_i^1 \neq x_i^2\}$, $\mathcal{E}' = \{ij \in \mathcal{E} \colon i, j \in \mathcal{V}'\}$, $\mathcal{Y}'_i = \{x_i^1, x_i^2\}$, $\boldsymbol{\theta}'_i(x_i^k) = c_i[\![x_i^k = 1]\!]$ for $k = 1, 2$ and

$$\theta'_{ij}(x_i^1, x_j^2) = \begin{cases} -\infty, & x_i^1 = x_j^2 = 1 \\ 0, & \text{otherwise}\,. \end{cases} \tag{8.44}$$

The $-\infty$ value in the pairwise costs assures that the optimization result is an independent set, *e.g.*, there is no edge between selected nodes in $\mathcal{E}$.

Example 8.23 (Recombination for a general ILP)**.** Consider the general ILP problem

$$\max_{\mathbf{x} \in P \cap \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle\ . \tag{8.45}$$

The respective recombination problem by Definition 8.20 takes the form

$$\begin{aligned} &\max_{\mathbf{x} \in P \cap \{0,1\}^n} \langle \mathbf{c}, \mathbf{x} \rangle \\ &\text{s.t. } x_i = x_i^1 \quad \text{whenever } x_i^1 = x_i^2,\ i \in [n]\,. \end{aligned} \tag{8.46}$$

Example 8.24 (Recombination for the binary knapsack)**.** Consider the binary knapsack problem as in Example 2.54. Let $\mathbf{x}^1$ and $\mathbf{x}^2$ be two its feasible solutions. For the sake of notation assume that $x_i^1 \neq x_i^2$ for $i = [k]$ and $x_i^1 = x_i^2$ for $i = k+1, \dots n$.

According to (8.46), the recombination problem can be expressed as

$$\begin{aligned} \max_{\mathbf{x}\in\{0,1\}^n} \ & \langle \mathbf{c}, \mathbf{x} \rangle \\ \text{s.t.} \ & \langle \mathbf{w}, \mathbf{x} \rangle \leq W \,, \\ & x_i = x_i^1, \ i = k+1, \dots n \,. \end{aligned} \tag{8.47}$$

This, in turn, is equivalent to

$$\begin{aligned} \max_{\mathbf{x}\in\{0,1\}^k} \ & \sum_{i=1}^{k} c_i x_i \\ \text{s.t.} \ & \sum_{i=1}^{k} w_i x_i \leq \hat{W} := W - \sum_{i=k+1}^{n} w_i x_i \,. \end{aligned} \tag{8.48}$$

The latter is a knapsack problem itself, although of the smaller size than the initial one.

Optimal recombination is dependent on the actual problem representation:

Example 8.25. Consider the recombination for the local polytope representation (3.44) of the labeling problem, see Figure 8.6 for illustration. Let μ^1 and μ^2 be two feasible solutions. Contrary to the representation used in Example 8.21, the feasible solutions now have coordinates that correspond to both labels $\mu_u(s)$ and label pairs $\mu_{uV}(s,t)$. Consider the example in Figure 8.6. Due to coupling constraints the feasible set of the recombination problem contains only 4 solutions: 00100, 00111, 11100 and 11111, see Figure 8.6(Middle). Compare it to the recombination problem build as in Example 8.21. The latter has in total $2^4 = 16$ points (labelings) in the feasible set, see Figure 8.6(Bottom).

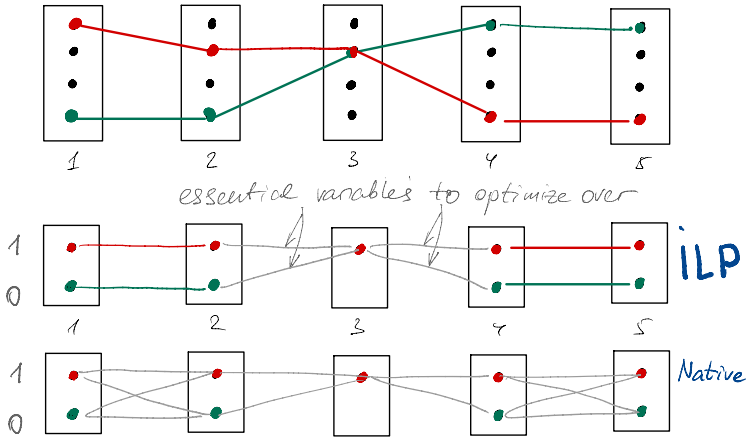


Figure 8.6: Optimal recombination for the ILP representation of the labeling problem. (Top) An initial problem and two feasible solutions. (Middle, "ILP") The resulting recombination problem. (Bottom, "Native") Recombination problem build as in Example 8.21. Numbers under nodes are node indices. See Example 8.25 for further description.

9 Binary Labeling Problems

Although, typically, the optimal recombination problem considered in Section 8.4 reduces to a smaller instance of the original problem, it can often be reformulated as a binary labeling problem, as illustrated by Examples 8.21 and 8.22. While NP-hard in general, this problem exhibits three remarkable properties:

- An important subclass of *submodular* binary labeling problems is reducible to efficiently solvable min-st-cut/max-flow problems, and thus can be addressed with the respective algorithms.
- A natural LP relaxation of the general binary labeling problem also reduces to min-st-cut/max-flow and is therefore efficiently solvable.
- The LP relaxation enjoys the *persistency* property, meaning that every integer variable in an LP solution belongs to some (and the same) optimal integer solution.

These properties enable the design of highly efficient exact and approximate algorithms for binary labeling problems and, consequently, render the corresponding recombination problems efficiently solvable, either exactly or approximately. In what follows, we derive these properties in detail.

9.1 Submodular binary labeling problems

Definition 9.1. A function $f\colon \{0,1\} \times \{0,1\} \to \mathbb{R}$ is called *submodular*, if it satisfies

$$f(0,1) + f(1,0) \geq f(0,0) + f(1,1)\,, \tag{9.1}$$

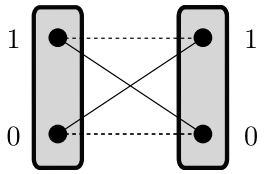


Figure 9.1: Binary pairwise cost function. If the sum of the costs corresponding to the dashed lines does not exceed the sum of the costs associated with the solid lines, the cost function is submodular. Otherwise it is supermodular.

and *supermodular*, if $(-f)$ is submodular, *i.e.*, the opposite inequality holds in (9.1). A *modular* function is super- and submodular at the same time, *i.e.*, the inequality (9.1) holds as equality.

Definition 9.2. A binary labeling problem is called *submodular* (*supermodular*), if all its pairwise costs are submodular (supermodular).

It is also easy to see that in case of the binary labeling problem each pairwise cost is submodular, supermodular or modular, see Figure 9.1 for illustration.

Example 9.3 (Ising model)**.** Let the pairwise costs of a binary labeling problem be defined as $\theta_{uv}(s,t) = \lambda_{uv}[\![s \neq t]\!]$ for some constant λ_{uv}. If $\lambda_{uv} > 0$, the corresponding cost function is submodular, for $\lambda_{uv} < 0$ it is supermodular. It can be shown, see Exercise 9.4, that by reparametrization any pairwise binary energy function can be transformed into the Ising model.

Exercise 9.4. Show that using reparametrization any pairwise binary energy can be transformed into the form of the Ising model (see Example 9.3). Moreover, submodular pairwise costs are transformed into submodular ones with $\lambda_{uv} \geq 0$, and supermodular into supermodular ones with $\lambda_{uv} \leq 0$.

Proposition 9.5. Let the pairwise costs $\boldsymbol{\theta}_{uv}$ be submodular for all $uv \in \mathcal{E}$. Then for any dual vector $\boldsymbol{\phi}$ the reparametrized costs $\boldsymbol{\theta}_{uv}^{\boldsymbol{\phi}}$ are submodular as well.

Proof. Definition 9.1 and the relation $\theta^{\boldsymbol{\phi}}_{uv}(s,t) = \theta_{uv}(s,t) + \phi_{u,v}(s) + \phi_{v,u}(t)$ imply:

$$\begin{aligned} 1 &\leq \theta_{uv}(0,1) + \theta_{uv}(1,0) - \theta_{uv}(0,0) - \theta_{uv}(1,1) \\ &= \theta_{uv}(0,1) + \phi_{u,v}(0) + \phi_{v,u}(1) + \theta_{uv}(1,0) + \phi_{u,v}(1) + \phi_{v,u}(0) \\ &\quad - \theta_{uv}(0,0) - \phi_{u,v}(0) - \phi_{v,u}(0) - \theta_{uv}(1,1) - \phi_{u,v}(1) - \phi_{v,u}(1) \\ &= \theta^{\boldsymbol{\phi}}_{uv}(0,1) + \theta^{\boldsymbol{\phi}}_{uv}(1,0) - \theta^{\boldsymbol{\phi}}_{uv}(0,0) - \theta^{\boldsymbol{\phi}}_{uv}(1,1) \end{aligned} \tag{9.2}$$

□

9.1.1 Submodular set-functions

This subsection relates the binary labeling problems to the whole class of submodular problems, one of the central notions in discrete optimization.

Let X be a finite set and 2^X be its powerset. Mappings of the form $f\colon 2^X \to \mathbb{R}$ are called *set-functions*. Finite sets can be represented by binary indicator vectors with coordinates indexed by elements of X. A binary vector $\xi \in \{0,1\}^X$ corresponds to the set $A \subset X$ if for all $x \in X$ it holds that

$$\xi_x = \begin{cases} 0, & x \notin A\,, \\ 1, & x \in A\,. \end{cases} \tag{9.3}$$

Example 9.6. Let $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \boldsymbol{\theta})$ be a binary graphical model, i.e. $\mathcal{Y}_\mathcal{V} = \{0,1\}^\mathcal{V}$. Then its energy function

$$E(\mathbf{y}) = \sum_{u \in \mathcal{V}} \theta_u(y_u) + \sum_{uv \in \mathcal{E}} \theta_{uv}(y_u, y_v) \tag{9.4}$$

can be seen as a set-function w.r.t. $\mathbf{y}$, since each labeling $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$ is a binary vector.

Definition 9.7. A set-function $f\colon 2^X \to \mathbb{R}$ is called *submodular* if

$$f(A) + f(B) \geq f(A \cap B) + f(A \cup B) \quad \forall A, B \in 2^X\,. \tag{9.5}$$

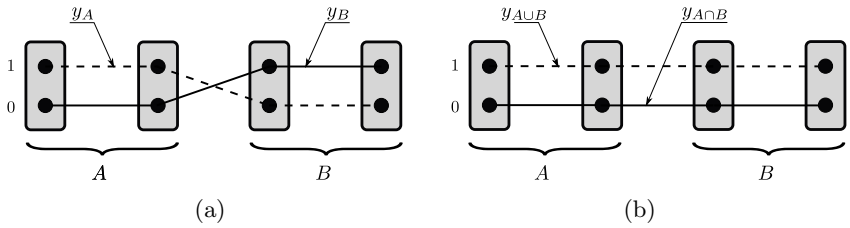


Figure 9.2: Representation of sets intersection and union with binary labelings. The sets A, B, $A \cup B$ and $A \cap B$ are represented with binary labelings $\mathbf{y}_A$, $\mathbf{y}_B$, $\mathbf{y}_{A\cup B} = \mathbf{y}_A \vee y_B$ and $\mathbf{y}_{A\cap B} = \mathbf{y}_A \wedge \mathbf{y}_B$ respectively.

Expressing this definition in terms of the corresponding labeling problem energy, we obtain

$$E(\mathbf{y}) + E(\mathbf{y}') \geq E(\mathbf{y} \wedge \mathbf{y}') + E(\mathbf{y} \vee \mathbf{y}') \quad \forall \mathbf{y}, \mathbf{y}' \in \mathcal{Y}_{\mathcal{V}} \tag{9.6}$$

with logical coordinate-wise *and* $\wedge$ and *or* $\vee$ operations, see Figure 9.2 for illustration.

Similarly to convex/concave functions, a function f is called *supermodular*, if $(-f)$ is submodular.

The *submodular minimization problem* consists in computing

$$A^* = \arg\min_{A \in 2^X} f(A)\,, \tag{9.7}$$

where the set-function f is submodular. This problem is known to be polynomially solvable (assuming that the value of $f(A)$ can be computed in polynomial time for any $A \in X$), although the order of the polynomial is quite high in general. However, different subclasses of submodular functions have lower computational complexity. One of such sub-classes, namely, submodular pairwise binary labeling problems, we will consider below.

The following lemma shows that the set of submodular (supermodular, modular) functions is closed with respect to addition and multiplication by a non-negative constant:

Lemma 9.8. Let f and g be (sub/super)modular and $\alpha, \beta \geq 0$. Then $\alpha f + \beta g$ is (sub/super)modular.

The proof follows trivially from the definition of (sub/super)modularity.

In particular, Lemma 9.8 implies, that adding a modular function to a submodular one results in a submodular function, as any modular function is submodular.

Proposition 9.9. For a binary submodular labelling problem the node-edge agreement implies dual optimality. Moreover, if ϕ is the respective dual vector, then $\mathcal{L}(\phi) \cap \{0,1\}^I \neq 0$ and, therefore, the local polytope relaxation is tight.

Proof. It is sufficient to show that there is a labeling $\mathbf{y}$ consisting of locally optimal labels and label pairs *w.r.t.* the reparametrization defined by $\boldsymbol{\phi}$.

Let $\boldsymbol{\xi} \in \{0,1\}^I$ define an arc-consistent subset of locally optimal labels and label pairs corresponding to the dual vector $\boldsymbol{\phi}$. Construct the labeling $\mathbf{y}$ by selecting the label 1 in each node u whenever $\xi_u(1) = 1$. Otherwise the label 0 is selected (in this case $\xi_u(0) = 1$ due to arc-consistency).

We show now that for any edge uv the label pair (y_u, y_v) is locally optimal. Due to arc-consistency of $\boldsymbol{\xi}$ the only configuration where it may not hold is when $\xi_u(0) = \xi_u(1) = \xi_v(0) = \xi_u(1) = 1$ and $\xi_{uv}(0,1) = \xi_{uv}(1,0) = 1$, see Figure 9.3. Let the respective locally optimal reparametrized costs be equal $\theta^{\boldsymbol{\phi}}_{uv}(0,1) = \theta^{\boldsymbol{\phi}}_{uv}(1,0) = \alpha$. Taking into account submodularity, it implies

$$2\alpha = \theta^{\boldsymbol{\phi}}_{uv}(0,1) + \theta^{\boldsymbol{\phi}}_{uv}(1,0) \geq \underbrace{\theta^{\boldsymbol{\phi}}_{uv}(0,0)}_{\geq \alpha} + \underbrace{\theta^{\boldsymbol{\phi}}_{uv}(1,1)}_{\geq \alpha} \geq 2\alpha\,. \tag{9.8}$$

Hence, the inequalities hold as equalities and $\theta^{\boldsymbol{\phi}}_{uv}(0,0) = \theta^{\boldsymbol{\phi}}_{uv}(1,1) = \alpha$ therefore. In other words, the label pair (y_u, y_v) is locally optimal. □

In particular, the statement of Proposition 9.9 implies that dual block-coordinate ascent algorithms like min-sum diffusion can be used

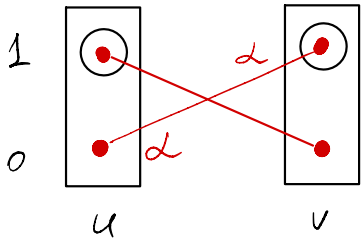


Figure 9.3:

to exactly solve submodular problems. However, more powerful method to do so is described in the following Section 9.2.

9.2 Binary submodular problems as min-st-cut

We start this section with a definition of the min-st-cut problem, which is one of the combinatorial problems having efficient polynomial time algorithms. Later on in this section we will show how submodular MAP-inference problems can be reduced to min-st-cut. Further, in Section 9.3 we will show that the local polytope relaxation of arbitrary binary labeling problems is reducible to min-st-cut as well.

9.2.1 Min-st-cut problem

Definition 9.10. Let $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ be a directed graph, $c\colon \mathcal{E}' \to \mathbb{R}$ be the *weight function of the edges* and $s, t \in \mathcal{V}'$ be two different nodes, which will be called *source* and *target* respectively. An *st-cut* $C = (S, T)$ is a partition of $\mathcal{V}'$ into two sets S and T such that

$$S \cap T = \emptyset \tag{9.9}$$
$$S \cup T = \mathcal{V}' \tag{9.10}$$
$$s \in S,\ t \in T\,. \tag{9.11}$$

In the following we will refer to st-cuts simply as *cuts*.

The weight of the cut C is the total weight of edges leading from S to T:

$$c(S,T) := \sum_{\substack{u\in S, v\in T \\ (u,v)\in\mathcal{E}'}} c_{u,v}\,. \tag{9.12}$$

The *min-st-cut* (or shortly *min-cut*) problem consists in finding a cut with minimal total weight. The min-*st*-cut problem is illustrated in Figure 9.4(a).

As many combinatorial problems, min-*st*-cut can be formulated as an integer linear program, see Example 9.11. In general it is NP-hard, just like other ILPs. However, if all weights $c_{u,v}$ are non-negative, its linear programming relaxation is tight and therefore the problem is polynomially solvable. The corresponding linear programming relaxation has a special structure, which allows us to find its minimum with polynomial finite-step algorithms with the worst-case complexity $O(|\mathcal{V}'|^3)$. Typically, the full Lagrange dual of the problem is solved instead of the primal problem. It constitutes the *max-flow* problem, for which a number of efficient polynomial finite-step solvers exist.

Example 9.11 (ILP representation of the min-*st*-cut and its LP relaxation)**.** The min-*st*-cut problem with non-negative edge weights has the following simple ILP representation:

$$\min_{\mathbf{x}\in\{0,1\}^{\mathcal{V}'\cup\mathcal{E}'}} \sum_{(u,v)\in\mathcal{E}'} c_{u,v}x_{u,v} \tag{9.13}$$

$$\text{s.t. } x_u - x_v + x_{u,v} \geq 0 \quad \forall (u,v)\in\mathcal{E}' \tag{9.14}$$

$$x_s = 0,\ x_t = 1. \tag{9.15}$$

The coordinates of the binary vector x are indexed by vertices and edges of the graph. The constraint (9.14) essentially states that if $u \in S$ and $v \in T$, the edge connecting them is in the cut. The non-negativity of the weights assures that for $u \in T$ and $v \in S$ the edge variable $x_{u,v}$ is set to zero. Indeed, in this case the inequality (9.14) reduces to $1 + x_{u,v} \geq 0$, therefore $x_{u,v}$ may take both values, 0 and 1.

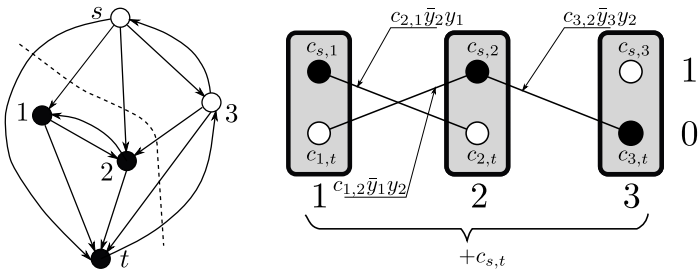


Figure 9.4: **(a)** An example of the min-*st*-cut problem. The shown problem can be equivalently written as minimization of the following pseudo-boolean function represented in the oriented form (see Section 9.2.2): $c_{s,t} + c_{s,1}x_1 + c_{s,2}x_2 + c_{s,3}x_3 + c_{1,t}\bar{x}_1 + c_{2,t}\bar{x}_2 + c_{3,t}\bar{x}_3 + c_{1,2}\bar{x}_1x_2 + c_{2,1}\bar{x}_2x_1 + c_{3,2}\bar{x}_3x_2$. The cut shown by a dashed line corresponds to the partitioning, where white nodes belong to the set S and black ones to the set T. The cut corresponds to the partition $(x_1, x_2, x_3) = (1, 1, 0)$ and its weight is equal to $c_{s,t} + c_{s,1} + c_{s,2} + c_{3,2} + c_{3,t}$. Note that the edge $(t, 3)$ does not belong to the cut. Moreover, the edge $(t, 3)$ does not belong to *any* cut, as well as the edge $(3, s)$. Therefore, these edges can be ignored. **(b)** The labeling problem equivalent to the min-*st*-cut problem in (a). Black circles define the labeling corresponding to the cut shown in (a).

However, if $c_{u,v} > 0$ the value 0 will be selected, as it corresponds to the smaller objective value.

The LP relaxation of (9.13)-(9.15) is obtained as usual, by convexifying the integrality constraints.

9.2.2 Oriented forms

Although the graph form of combinatorial problems is of great importance for the development of the field, their algebraic forms have the advantage of a more compact problem representation and help to give unambiguous proofs. Along with the ILP formulation for the min-*st*-cut problem, we will consider another algebraic form, the *quadratic pseudo-boolean minimization* problem.

For a binary variable $x \in \{0,1\}$, let $\bar{x}$ denote its negation, that is, $\bar{x} = 1 - x$. Functions $f\colon \{0,1\}^n \to \mathbb{R}$ are called *pseudo-boolean*. Note that the class of pseudo-boolean functions coincides with the class of *set-functions* as defined in Section 9.1.1. In the rest of the monography we will prefer the latter term following [99]. An important example of pseudo-boolean functions $f(\mathbf{x})$, $\mathbf{x} \in \{0,1\}^n$, are those representable as a *quadratic* polynomial of x_i and $\bar{x}_i$. A subclass of such functions, tightly related to the min-*st*-cut problem, is defined as follows:

Definition 9.12. Let ν, α_i, β_i and $\gamma_{i,j}$, $i,j = 1,\dots,n$ be arbitrary real numbers. For $\mathbf{x} \in \{0,1\}^n$ a quadratic pseudo-boolean function of the type

$$f(\mathbf{x}) = \nu + \sum_{i=1}^{n} \alpha_i x_i + \sum_{i=1}^{n} \beta_i \bar{x}_i + \sum_{i=1}^{n}\sum_{j=1}^{n} \gamma_{ij} \bar{x}_i x_j \tag{9.16}$$

will be called an *oriented form*.

Contrary to the other terms used in this monography, the term *oriented form* was first introduced in [3] and is not established in the literature yet.

Example 9.13. The following quadratic pseudo-boolean functions are oriented forms: $-5 + 7x_3\bar{x}_1 + 12x_3 + \bar{x}_1$; $2\bar{x}_1x_2 + 3\bar{x}_2x_1 - 6x_3\bar{x}_4$.

Example 9.14. The quadratic pseudo-boolean functions below are not oriented forms: $-5 + 7x_3x_1 + 12x_3 + \bar{x}_1$; $2x_1x_2 + 3\bar{x}_2x_1 - 6x_3\bar{x}_4$.

9.2.3 Min-cut as oriented form

Oriented forms are well suited for representing graph partitions like the one found in the min-*st*-cut problem. Let the coordinates of a binary vector $\mathbf{x} \in \{0,1\}^{\mathcal{V}'}$ be the indicator variables, denoting the subset each vertex belongs to. In other words, if $u \in S$, $x_u = 0$, and, on the other hand, $x_u = 1$ is equivalent to $u \in T$. In this way, the vector $\mathbf{x}$ represents a partition of the graph $\mathcal{G}'$.

Let the monomial $c_{u,v}\bar{x}_u x_v$ correspond to the directed edge (u,v) with the weight $c_{u,v}$. Then the total weight of the cut (9.12) corresponding to the partition associated with the vector $\mathbf{x}$ is equal to

$$\sum_{(u,v)\in\mathcal{E}'} c_{u,v}\bar{x}_u x_v \,. \tag{9.17}$$

Indeed, the items of the sum are non-zero only if $x_u = 0$ and $x_v = 1$, meaning that $u \in S$ and $v \in T$.

Note that since $s \in S$ and $t \in T$ it follows that $x_s = 0$ and $x_t = 1$. Therefore, $c_{s,v}\bar{x}_s x_v = c_{s,v}x_v$, $c_{u,t}\bar{x}_u x_t = c_{u,t}\bar{x}_u$ and $c_{s,t}\bar{x}_s x_t = c_{s,t}$. It also implies that (directed) edges (u,s) and (t,u), $u \in \mathcal{V}$ can be ignored, since they do not belong to any cut and the corresponding monomials $c_{u,s}\bar{x}_u x_s$ and $c_{t,u}\bar{x}_t x_u$ vanish.

In conclusion, each min-*st*-cut problem can be equivalently represented as minimization of the oriented form

$$c_{s,t} + \sum_{v\colon (s,v)\in\mathcal{E}'} c_{s,v}x_v + \sum_{u\colon (u,t)\in\mathcal{E}'} c_{u,t}\bar{x}_u + \sum_{\substack{u,v\in\mathcal{E}' \\ u\neq s,\ v\neq t}} c_{u,v}\bar{x}_u x_v \tag{9.18}$$

with the constant $c_{s,t}$ being zero if the edge (s,t) does not exist. Conversely, minimization of an oriented form can be treated as a min-*st*-cut problem. Figure 9.4(a) illustrates this correspondence. In this representation weights of the graph edges become coefficients of the corresponding monomials. In particular it means that a min-*st*-cut problem with non-negative weights corresponds to an oriented form with non-negative coefficients and the other way around. Table 9.1 summarizes the mapping between the min-*st*-cut problem and minimization of the oriented forms.

Table 9.1: Mapping between the min-st-cut problem, minimization of the oriented forms and the MAP-inference in graphical models.

min-st-cut	oriented form	$E(y;\theta)$
V'	$\mathbf{x} \in \{0,1\}^{\mathcal{V}'}$	$V = V' \backslash \{s,t\}$
$u \in S$	$x_u = 0$	$y_u = 0$
$u \in T$	$x_u = 1$	$y_u = 1$
$c_{u,v}$, $(u,v) \in \mathcal{E}'$	$c_{u,v}\bar{x}_u x_v$	$\theta_{uv}(0,1) = c_{u,v}$
in particular:		
$s \in S$	$x_s = 0$	–
$t \in T$	$x_t = 1$	–
$c_{s,v}$, $(s,v) \in \mathcal{E}'$	$c_{s,v}x_v$	$\theta_v(1) = c_{s,v}$
$c_{u,t}$, $(u,t) \in \mathcal{E}'$	$c_{u,t}\bar{x}_u$	$\theta_u(0) = c_{u,t}$
$c_{s,t}$, $(s,t) \in \mathcal{E}'$	$c_{s,t}$	constant
$c_{u,s}$, $(u,s) \in \mathcal{E}'$	0	–
$c_{t,v}$, $(t,v) \in \mathcal{E}'$	0	–

9.2.4 Binary labeling problem as oriented form

To show a close relation between binary labeling and min-st-cut problems, we show that the former can be equivalently seen as minimization of an oriented form. We will start with an inverse transformation by showing that an oriented form can be treated as a binary labeling problem up to a constant term. The latter does not influence the optimization, and, therefore, can be ignored.

For the beginning, consider Figure 9.5, which illustrates how different quadratic monomials can be represented as pairwise costs of a labeling problem. So, for example, the monomial $c_{u,v}\bar{y}_u y_v$ for binary variables y_u and y_v can be represented as the pairwise cost vector $\boldsymbol{\theta}_{uv}$ with coordinates

$$\theta_{uv}(0,1) = c_{u,v}, \qquad \theta_{uv}(0,0) = \theta_{uv}(1,1) = \theta_{uv}(1,0) = 0\,. \tag{9.19}$$

To specify a general transformation, consider an oriented form (9.16) with n variables. Build the graphical model $(\mathcal{G}, \mathcal{Y}_{\mathcal{V}}, \boldsymbol{\theta})$ with the graph

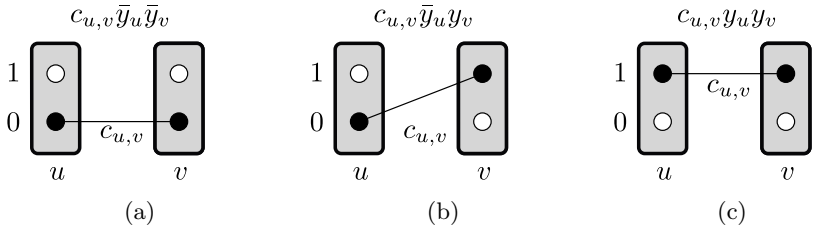


Figure 9.5: Correspondence between quadratic monomials and pairwise costs of binary graphical models. Solid lines corresponding to label pairs denote the only non-zero values of the cost vector of the pairwise factor.

$\mathcal{G}$ consisting of n nodes, $\mathcal{V} = \{1, 2, \ldots, n\}$, where the i-th node corresponds to the i-th variable and vice versa. For each monomial $c_{i,j}\bar{x}_i x_j$ add an edge between nodes i and j to the graph $\mathcal{G}$, should it not exist yet. Set all coordinates of the cost vector $\boldsymbol{\theta}$ initially to zero. For each monomial $\alpha_i x_i$ assign $\theta_i(1) := \alpha_i$, for $\beta_i\bar{x}_i$ assign $\theta_i(0) := \beta_i$ and for $c_{i,j}\bar{x}_i x_j$ assign $\theta_{ij}(0,1) := c_{i,j}$. Table 9.1 summarizes the construction, which is also illustrated in Figures 9.4(b) and 9.5.

To specify the inverse transformation of a graphical model to an oriented form note that only pairwise factors having the structure of (9.19), that is, every entry is zero except one non-zero value on the diagonal, can be given by an oriented form according to the above construction. In the following, we will adopt the name *diagonal form* for such factors. It turns out that any pairwise cost function can be transformed into the diagonal form by reparametrization. Figure 9.6 illustrates this transformation, given by the formulas below:

$$\phi_{u,v}(0) = \theta_{uv}(1,0) - \theta_{uv}(0,0) \tag{9.20}$$

$$\phi_{v,u}(0) = -\theta_{uv}(1,0) \tag{9.21}$$

$$\phi_{v,u}(1) = -\theta_{uv}(1,1) \tag{9.22}$$

Note that after the transformation (9.20)-(9.22) the only non-zero pairwise cost value is equal to

$$\theta^{\phi}_{uv}(0,1) = \theta_{uv}(0,1) + \theta_{uv}(1,0) - \theta_{uv}(1,1) - \theta_{uv}(0,0)\,, \tag{9.23}$$

which is non-negative for submodular pairwise cost functions.

To guarantee that *all* monomials in the oriented form are non-negative, one has to guarantee that the unary costs are non-negative as well. This can easily be achieved by subtracting the minimal value from the unary costs:

$$\forall s \in \mathcal{Y}_u\colon \quad \theta^{\phi}_u(s) := \theta^{\phi}_u(s) - \min_{l \in \mathcal{Y}_u} \theta^{\phi}_u(l)\,. \tag{9.24}$$

Since each labeling contains either the label 0 or 1 in each node, such an operation only subtracts a constant value from the energies of all labelings and, therefore, does not affect the optimal solution of the labeling problem.

9.2.5 Equivalence: binary labeling problem = min-*st*-cut

Summarizing, transformations (9.20)-(9.22) performed for each edge $uv \in \mathcal{E}$ and followed by transformation (9.24) applied to each node $u \in \mathcal{V}$ translate the cost-vector into a form where it can be represented as an oriented form. Moreover, this oriented form has only non-negative coefficients, if the labeling problem is submodular.

Combining this with the results of Section 9.2.3, one obtains the following statement:

Proposition 9.15. The binary labeling problem reduces to the min-*st*-cut problem and the other way around. If the labeling problem is submodular, it can be reduced to the min-*st*-cut problem with non-negative edge weights. Inversely, such min-*st*-cut problems are equivalently representable as binary submodular labeling problems.

Remark 9.16. The transformation (9.20)-(9.22) is asymmetric, since $\theta_{uv}(0,1) \neq 0$ and $\theta_{uv}(1,0) = 0$. A symmetric transformation can also be found, e.g. such that $\theta_{uv}(0,1) = \theta_{uv}(1,0) \neq 0$. Although

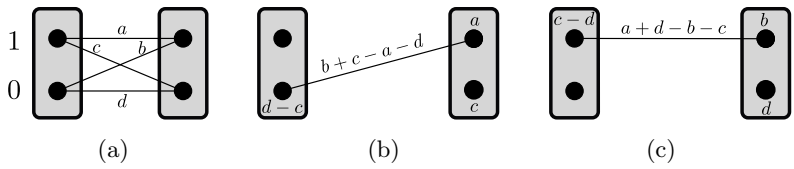


Figure 9.6: Transformation of an arbitrary pairwise cost vector **(a)** to the diagonal **(b)** and parallel **(c)** form. For the use of the latter see Section 9.3.

it can be useful in some cases, the corresponding min-st-cut graph would contain two directed edges (u, v) and (v, u) for such a symmetric representation, whereas the diagonal representation (9.19) requires only a single edge. The latter results in a smaller min-st-cut graph and faster optimization.

9.2.6 Bibliography and further reading

An overview of combinatorial algorithms for general submodular minimization can be found in [100].

The first work formulating the submodular binary labeling problem as min-st-cut was probably [101]. The classical paper, which describes reduction of the binary labeling problem to the min-st-cut problem is the seminal work [102].

The notion of submodularity is also applicable to the multi-label problems, which are also reducible to min-st-cut in this special case. We refer to [3] and references therein for further details.

9.3 Binary LP relaxation as min-st-cut

In Section 9.2 we have shown an equivalence between min-st-cut and binary labeling problems. That is, each binary labeling problem can be reduced to min-st-cut of asymptotically the same size, and the

other way around: Each min-*st*-cut is reducible to a binary labeling problem.

In particular, we have shown that submodular labeling problems correspond to min-*st*-cut problems with non-negative edge weights. Hence, both problems have polynomial complexity and are, moreover, very efficiently solvable.

However, the non-submodular labeling problems can be reduced only to min-*st*-cut with some edge weights being negative. This conforms to the fact that both problems are $\mathcal{NP}$-hard. Therefore, in order to deal with this kind of problems, one has to resort to approximate algorithms such as those addressing its LP relaxation.

Interestingly, the local polytope relaxation of the binary labeling problem can be optimized very efficiently by reducing it to min-*st*-cut with non-negative edge weights. This contrasts with the local polytope relaxation of the multi-label problem, which is known to be as difficult as general linear programs, see Theorem 3.7. In this chapter we will provide the above reduction. Additionally, we will analyze the local polytope corresponding to binary labeling problems. In particular, we will show that it possesses the so called *persistency* or *partial* optimality property, which often allows us to reduce the size of the non-relaxed problem, and, therefore, makes it possible to obtain its exact solution with combinatorial solvers. Another important use of this property are construction of efficient heuristics addressing arbitrary (not necessarily submodular) binary labeling problems and, hence, making the efficient recombination possible.

9.3.1 Half-integrality of local polytope

We start with the most important property of the local polytope of the binary labeling problem (we will refer to it as *binary local polytope*, although it is not a commonly used term in the literature). This is the characterization of its vertices, which all turn out to have *half-integer* coordinates, i.e. for any vertex $\boldsymbol{\mu}$ of the binary local polytope it holds that $\boldsymbol{\mu} \in \{0, 1, \frac{1}{2}\}^{\mathcal{I}}$. Recall that $\mathcal{I}$ indexes all labels in all nodes as well as all label pairs in all edges, see Section 3.3.

For the following statement recall the definition of the set $\mathcal{L}(\boldsymbol{\phi})$ from (5.125):

$$\mathcal{L}(\boldsymbol{\phi}) = \{\boldsymbol{\mu} \in \mathcal{L} \colon \mu_w(s) = 0 \text{ if } \theta^{\boldsymbol{\phi}}_w(s) > \min_{s' \in \mathcal{Y}_w} \theta^{\boldsymbol{\phi}}_w(s'),\ w \in \mathcal{V} \cup \mathcal{E},\ s \in \mathcal{Y}_w\}\,, \tag{9.25}$$

which is the set of all vectors in the local polytope such that their non-zero coordinates correspond only to locally minimal entries of the cost vector.

Theorem 9.17 ([29], [103], [104])**.** Let $(\mathcal{G}, \mathcal{Y}_{\mathcal{V}}, \boldsymbol{\theta})$ be a binary graphical model. Let $\boldsymbol{\phi} \in \mathbb{R}^{\mathcal{J}}$ be the dual vector and $\boldsymbol{\theta}^{\boldsymbol{\phi}}$ be the corresponding reparametrized costs such that the node-edge agreement (see Definition 5.29) holds for it. Then $\mathcal{L}(\boldsymbol{\phi}) \cap \{0, \frac{1}{2}, 1\}^{\mathcal{I}} \neq \emptyset$.

In other words, there is a half-integral point in the local polytope, such that its non-zero coordinates correspond to locally minimal labels and label pairs of the cost vector $\boldsymbol{\theta}^{\boldsymbol{\phi}}$.

Note that due to Proposition 5.24(5), Theorem 9.17 implies that node-edge agreement is sufficient for dual optimality in case of binary problems.

Proof. Let $\boldsymbol{\xi} \in \{0,1\}^{\mathcal{I}}$ an arc-consistent binary vector corresponding to the locally optimal label and labels pairs of the cost vector $\boldsymbol{\theta}^{\boldsymbol{\phi}}$. Such $\boldsymbol{\xi}$ always exists due to Definition 5.29 of node-edge agreement. Denote by n_u the number of non-zero entries in ξ_u, i.e. $n_u = \xi_u(0) + \xi_u(1)$, $u \in \mathcal{V}$. Consider the three factors $\boldsymbol{\xi}_u$, $\boldsymbol{\xi}_{uv}$ and $\boldsymbol{\xi}_v$ for any $uv \in \mathcal{E}$. Up to label flipping they can have only 5 possible configurations, depicted in Figure 9.7, where black circles, solid and dashed lines correspond to coordinates of $\boldsymbol{\xi}$ equal to 1 while the remaining coordinates are 0.

We now construct a half-integer vector $\boldsymbol{\mu}$. Assign $\mu_u(s) = \frac{\xi_u(s)}{n_u}$ for all $u \in \mathcal{V}$. Assign $\mu_{uv}(s,t) = \frac{\xi_{uv}(s,t)}{\max\{n_u, n_v\}}$. It is straightforward to check that such $\boldsymbol{\mu}$ belong to the binary local polytope. It contains only half-integer coordinates, and its non-zero coordinates correspond to non-zero coordinates of $\boldsymbol{\xi}$. Concluding, $\boldsymbol{\mu} \in \mathcal{L}(\boldsymbol{\phi}) \cap \{0, \frac{1}{2}, 1\}^{\mathcal{I}}$, which finalizes the proof. □

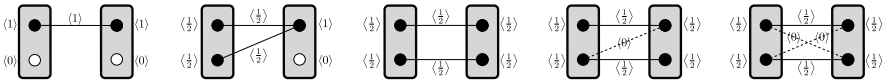


Figure 9.7: Possible configurations (up to label flip) of the arc-consistent subset of locally optimal labels and label pairs for $\boldsymbol{\theta}^{\boldsymbol{\phi}}$ treated as a binary vector $\boldsymbol{\xi} \in \{0,1\}^{\mathcal{I}}$, see proof of Theorem 9.17. Black circles for a label s in a node u corresponds to $\xi_u(s) = 1$, white circle to $\xi_u(s) = 0$. Let n_u be the number of black circles in the node u. Both solid and dashed lines connecting labels s and t in nodes u and v correspond to $\xi_{uv}(s,t) = 1$, otherwise $\xi_{uv}(s,t) = 0$. Angular brackets $\langle\rangle$ are used for coordinates of primal solution $\boldsymbol{\mu}$. To construct a feasible relaxed primal solution $\boldsymbol{\mu}$ assign $\mu_u(s) = \frac{1}{n_u}$ if label s corresponds to a black circle, and $\mu_u(s) = 0$ if to a white one. Similarly, assign $\mu_{uv}(s,t) = \frac{1}{\max\{n_u, n_v\}}$, if (s,t) corresponds to a solid line, otherwise assign $\mu_{uv}(s,t) = 0$.

Corollary 9.18. All vertices of the binary local polytope have half-integral coordinates.

Proof. Let $\boldsymbol{\theta}$ be a cost vector such that the relaxed problem $\min_{\boldsymbol{\mu} \in \mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$ has a unique solution. Let also $\boldsymbol{\phi}$ be an optimal reparametrization, which implies node-agreement for $\boldsymbol{\theta}^{\boldsymbol{\phi}}$. According to Theorem 9.17 the relaxed solution is half-integral. Together with Definition 2.20 this proves the statement. □

9.3.2 LP relaxation as min-*st*-cut

The goal of this section is to reduce the LP relaxation of the binary labeling problem to the min-*st*-cut problem. Such a transformation would imply existence of a polynomial, practically efficient algorithm for the LP relaxation.

Separability of the local polytope As noted in Section 5.5.1, the local polytope for arbitrary, not only binary problems can be written in the following compact form, see (5.117):

$$\mathcal{L} = \begin{cases} \boldsymbol{\mu}_u \in \Delta^{\mathcal{Y}_u}, & \forall u \in \mathcal{V} \\ \boldsymbol{\mu}_{uv} \in \Delta^{\mathcal{Y}_{uv}}, & \forall uv \in \mathcal{E} \\ \sum_{t \in \mathcal{Y}_v} \mu_{uv}(s,t) = \mu_u(s), & \forall u \in \mathcal{V},\ v \in \mathcal{N}(u),\ s \in \mathcal{Y}_u\,. \end{cases} \tag{9.26}$$

Let us fix the "unary" vectors $\boldsymbol{\mu}_u$ for all nodes $u \in \mathcal{V}$. Let also $\mathcal{L}_{uv}(\boldsymbol{\mu}_u, \boldsymbol{\mu}_v)$ be the set of all "pairwise" vectors $\boldsymbol{\mu}_{uv}$, such that the whole vector $\boldsymbol{\mu}$ belongs to $\mathcal{L}$, i.e.:

$$\mathcal{L}_{uv}(\boldsymbol{\mu}_u, \boldsymbol{\mu}_v) = \{\boldsymbol{\mu}_{uv} \in \Delta^{\mathcal{Y}_{uv}} \colon \sum_{t \in \mathcal{Y}_v} \mu_{uv}(s,t) = \mu_u(s) \quad \forall s \in \mathcal{Y}_u\} \tag{9.27}$$

This separable structure implies that given the "unary" vectors $\boldsymbol{\mu}_u$, $u \in \mathcal{V}$, optimization with respect to the "pairwise" ones decomposes into $|\mathcal{E}|$ small independent optimization problems, one for each pairwise factor:

$$\min_{\boldsymbol{\mu} \in \mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle = \min_{\substack{\boldsymbol{\mu}_u \in \Delta^{\mathcal{Y}_u} \\ u \in \mathcal{V}}} \sum_{u \in \mathcal{V}} \langle \boldsymbol{\theta}_u, \boldsymbol{\mu}_u \rangle + \sum_{uv \in \mathcal{E}} \min_{\boldsymbol{\mu}_{uv} \in \mathcal{L}_{uv}(\boldsymbol{\mu}_u, \boldsymbol{\mu}_v)} \langle \boldsymbol{\theta}_{uv}, \boldsymbol{\mu}_{uv} \rangle\ . \tag{9.28}$$

We will use this fact below to reduce the binary local polytope relaxation to the min-st-cut problem.

Idea of the transformation As follows from Section 9.2.4, binary energy can be represented as an oriented form. Submodular pairwise costs $\boldsymbol{\theta}_{uv}$ can be turned into diagonal form (see Figure 9.6(b)) equivalent to a quadratic monomial $c_{u,v}\bar{y}_u y_v$ with non-negative coefficients $c_{u,v}$, whereas for supermodular costs these coefficients are non-positive. Alternatively, supermodular pairwise costs can be turned into the parallel form, as shown in Figure 9.6(c), corresponding to the quadratic monomial $c_{u,v} y_u y_v$, see Figure 9.5(c). Although the coefficient of this monomial is non-negative for supermodular costs, they

cannot be directly represented as edges of the min-st-cut graph. To do so, we will introduce an additional binary vector $\mathbf{z}$ such that $\mathbf{y} = \bar{\mathbf{z}}$ and rewrite $c_{u,v}y_u y_v = \frac{1}{2}c_{u,v}y_u\bar{z}_v + \frac{1}{2}c_{u,v}\bar{z}_u y_v$. The right-hand side is an oriented form, and, therefore, can be represented as edges of a min-st-cut graph.

Binary energy as oriented form with twice the number of variables In general, we will assume that the binary energy is transformed into a quadratic pseudo-boolean function with *non-negative* coefficients. By adding an additional binary vector $\mathbf{z}$ such that $\mathbf{y} = \bar{\mathbf{z}}$ we rewrite each term of the pseudo-boolean function as a symmetric oriented form containing both $\mathbf{y}$ and $\mathbf{z}$ vectors.

- Unary costs are transformed as follows

$$c_u y_u \to \frac{1}{2}c_u y_u + \frac{1}{2}c_u\bar{z}_u\,, \tag{9.29}$$

$$c_u\bar{y}_u \to \frac{1}{2}c_u\bar{y}_u + \frac{1}{2}c_u z_u\,. \tag{9.30}$$

- Submodular pairwise terms are transformed as follows

$$c_{u,v}\bar{y}_u y_v \to \frac{1}{2}c_{u,v}\bar{y}_u y_v + \frac{1}{2}c_{u,v}z_u\bar{z}_v\,. \tag{9.31}$$

- Supermodular pairwise terms are transformed as follows

$$c_{u,v}y_u y_v \to \frac{1}{2}c_{u,v}y_u\bar{z}_v + \frac{1}{2}c_{u,v}\bar{z}_u y_v\,. \tag{9.32}$$

As noted above, we assume all coefficients c_u, c_v and $c_{u,v}$ to be non-negative.

Let E be an energy and $g(\mathbf{y},\mathbf{z})$ be the oriented form, constructed according to (9.29)-(9.32). By construction, $E(\mathbf{y}) = g(\mathbf{y},\bar{\mathbf{z}})$, therefore it holds that:

$$\min_{\mathbf{y},\mathbf{z}\in\{0,1\}^{\mathcal{V}}} g(\mathbf{y},\mathbf{z}) \leq \min_{\substack{\mathbf{y},\mathbf{z}\in\{0,1\}^{\mathcal{V}}\\ \mathbf{z}=\bar{\mathbf{y}}}} g(\mathbf{y},\mathbf{z}) = \min_{\mathbf{y}\in\{0,1\}^{\mathcal{V}}} E(\mathbf{y})\,. \tag{9.33}$$

The following theorem relates the first term in (9.33) to the LP relaxation of the energy minimization:

Theorem 9.19. Let $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \boldsymbol{\theta})$ be a binary labeling problem and its submodular and supermodular pairwise terms be represented in the "diagonal" and "parallel" forms (see Figure 9.6) respectively. Let g be constructed according to (9.29)-(9.32). Then it holds that

$$\min_{\mathbf{y},\mathbf{z}\in\{0,1\}^\mathcal{V}} g(\mathbf{y},\mathbf{z}) = \min_{\boldsymbol{\mu}\in\mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle \ . \tag{9.34}$$

Moreover, a minimizer $\boldsymbol{\mu}'$ of the right-hand-side can be constructed from a minimizer $(\mathbf{y}', \mathbf{z}')$ of the left-hand-side as follows:

$$\mu'_u(1) = \begin{cases} y'_u, & y'_u = \bar{z}'_u \\ \frac{1}{2}, & y'_u \neq \bar{z}'_u \ , \end{cases} \tag{9.35}$$

$$\mu'_u(0) = 1 - \mu'_u(1) \tag{9.36}$$

$$\boldsymbol{\mu}'_{uv} = \operatorname*{arg\,min}_{\boldsymbol{\mu}_{uv}\in\mathcal{L}_{uv}(\boldsymbol{\mu}'_u,\boldsymbol{\mu}'_v)\cap\{0,1,\frac{1}{2}\}^{\mathcal{Y}_{uv}}} \langle \boldsymbol{\theta}_{uv}, \boldsymbol{\mu}_{uv} \rangle \ , \tag{9.37}$$

where $\mathcal{L}_{uv}$ is defined by (9.27).

Proof. The proof of the theorem is quite straightforward: Except for one special case noted separately, for each node and edge $w \in \mathcal{V}\cup\mathcal{E}$ we will show that if $\boldsymbol{\mu}'_w$ is constructed from *arbitrary* (y'_w, z'_w) the relaxed energy term $\langle \boldsymbol{\theta}_w, \boldsymbol{\mu}'_w \rangle$ is equal to the corresponding term of $g(\mathbf{y}', \mathbf{z}')$.

1. If $y_w = \bar{z}_w$ the condition above holds by construction.
2. Unary terms (9.29)-(9.30): If $y_u \neq \bar{z}_u$ then the corresponding term of g is equal to $\frac{1}{2}c_u$, which is equal to $\langle \boldsymbol{\theta}_u, \boldsymbol{\mu}'_u \rangle$.
3. Submodular terms (9.31): Let $y_v = \bar{z}_v$ and $y_u \neq \bar{z}_u$.
   a) If $y_v = \bar{z}_v = 1$ the corresponding term (9.31) of g turns into $\frac{1}{2}c_{u,v}(\bar{y}_u + z_u)$, which is $\frac{1}{2}c_{u,v}$ (independently of the values of $\bar{y}_u \neq z_u$) and is equal to $\langle \boldsymbol{\theta}_{uv}, \boldsymbol{\mu}'_{uv} \rangle$ according to Figure 9.8(a).

b) if $y_v = \bar{z}_v = 0$ the corresponding term (9.31) of g is equal to 0 (independently of the values of $y_u \neq \bar{z}_u$), which is equal to $\langle \boldsymbol{\theta}_{uv}, \boldsymbol{\mu}'_{uv} \rangle$, see Figure 9.8(b) for illustration.

4. Supermodular terms (9.32) with $y_v = \bar{z}_v$ and $y_u \neq \bar{z}_u$ act analogous to the previous case 3 when changing y_v to $\bar{z}_v$.

5. Other submodular (9.31) and supermodular (9.32) terms: Let $y_v \neq \bar{z}_v$ and $y_u \neq \bar{z}_u$. This is the exceptional case mentioned above. It holds that $\langle \boldsymbol{\theta}_{uv}, \boldsymbol{\mu}'_{uv} \rangle = 0$, see Figures 9.8(c) and 9.8(d) for illustration. This is the same 0 value the terms (9.31) and (9.32) have if $y_v = z_v = y_u = z_u = 0$. There are combinations of values of y_v and z_u (e.g. $y_v = z_v = 0$ and $y_u = z_u = 1$ for (9.31)), corresponding to greater values of the considered terms, however they are never optimal. This is due to the fact that as soon as $y_u \neq \bar{z}_u$ the change from $y_u = z_u = 1$ to $y_u = z_u = 0$ can either decrease the value of the corresponding components of $g(\mathbf{x}, \mathbf{y})$ (as in the current case) or leave it unchanged (as in all other situations described above). This means that considering $y_u = z_u = 0$ a soon as $y_u \neq \bar{z}_u$ gives smaller or equal value of g. Hence, we can conclude that $g(\mathbf{y}', \mathbf{z}') = \langle \boldsymbol{\theta}, \boldsymbol{\mu}' \rangle$.

□

9.3.3 Use case: Max-weight independent set problem

The max/weight independent set problem (2.31) is tightly connected to the binary labeling one. One connection has been explored in Example 8.22, showing that the respective recombination problem can be represented in a form of binary labeling problem. Moreover, this recombination problem can be efficiently exactly solved, according to the following statement:

Proposition 9.20. The recombination problem for max-weight independent set, as defined in Example 8.22, is a supermodular binary maximization labeling problem and, therefore, reducible to min-st-cut.

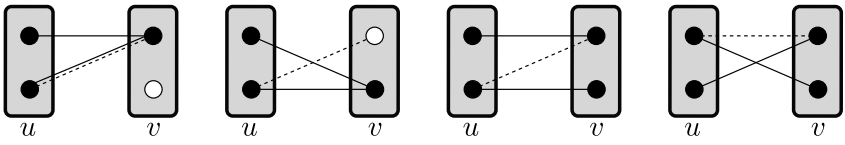


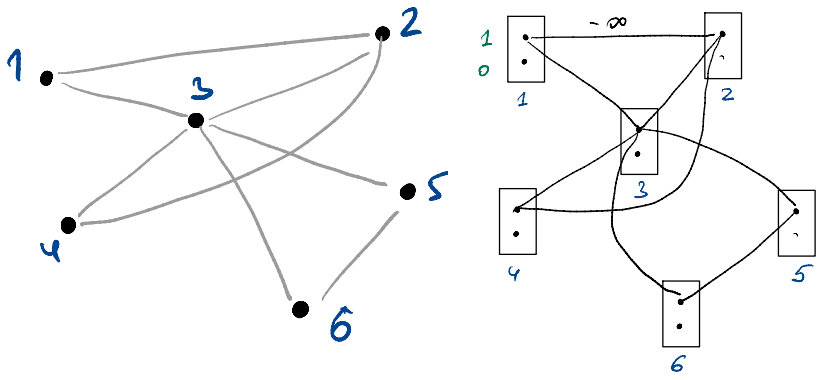
Figure 9.8: Optimal primal solutions for pairwise factors, see the proof of Theorem 9.19. Solid lines correspond to the value $\frac{1}{2}$ of the primal variable $\boldsymbol{\mu}$, black labels - either to 1 (in case of a single black label per node) or to $\frac{1}{2}$ (for two black labels per node). Dashed lines correspond to the diagonal and parallel forms of the submodular and supermodular pairwise factors respectively.

Proof. The submodularity property can be verified by definition: the only non-zero pairwise costs are equal to $-\infty$ and are always "diagonal", *e.g.*, connect labels 0 and 1. The latter is due to the fact that they encode existence of an edge in the original graph between respective nodes. These edges may, however, exist only between nodes belonging to different independent sets and, therefore, between *different* labels in the neighboring nodes of the respective binary labeling problem. □

Below, we reduce the max/weight independent set problem itself to the binary labeling problem and show that the respective LP relaxations of these two problems are equivalent. In particular, it implies that the edge relaxation of the max-weight independent set problem has half-integrality and persistency properties.

Max-weight independent set as binary labeling

Consider example in Figure 9.9. Let a max-weight independent set problem be defined by its graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and a cost vector $\mathbf{c} \in \mathbb{R}^{\mathcal{V}}$. Define the binary maximization labeling problem over the same graph



(a) (b)

Figure 9.9: Max-weight independent set problem (a) reduced to a binary maximization labeling problem (b). Both graphs are equal, corresponding nodes have equal indices. Each node in (a) and (b) is associated with a binary variable, whereas the upper labels in (b) correspond to the value 1 and lower ones to 0. Label pairs shown as solid lines in (b) are prohibited and have the cost $-\infty$ therefore. Not shown label pairs are allowed with the cost 0.

$\mathcal{G}$ and label set $\mathcal{Y}_u = \{0, 1\}$, $u \in \mathcal{V}$. This determines a one-to-one mapping between labelings $\mathbf{y} \in \mathcal{Y}_\mathcal{V}$ and indicator variables $\mathbf{x}$ of the max-weight independent set problem as $x_u = [\![y_u = 1]\!]$.

To ensure that the labelings correspond to independent sets only, we define the pairwise costs as follows: $\theta_{uv}(s, t) = -\infty[\![s = t = 1]\!]$, $uv \in \mathcal{E}$, which implies that selecting the label 1 in two neighboring nodes is prohibited and all other combinations are not penalized.

Finally, we assign the unary costs to $\theta_u(s) = c_i[\![s = 1]\!]$ to ensure that costs are applied only to the nodes labeled as 1.

Edge relaxation and its relation to the binary local polytope

Consider the edge relaxation of the max-weight independent set problem (2.31)

$$\max_{\mathbf{x}\in[0,1]^{\mathcal{V}}} \langle \mathbf{c}, \mathbf{x}\rangle \tag{9.38}$$

$$\text{s.t. } x_i + x_j \leq 1\,,\ \{i,j\} \in \mathcal{E}\,. \tag{9.39}$$

It turn out that this relaxation has half-integrality property, similar to the binary local polytope treated in Section 9.3.1:

Theorem 9.21 ([105])**.** Any basic solution of the max-weight independent set is half-integral. That is, if $\mathbf{x}$ is a vertex of the polytope $\{\mathbf{x} \in [0,1]^{\mathcal{V}} \colon x_i + x_j \leq 1\,,\ \{i,j\} \in \mathcal{E}\}$, then $x_i \in \{0, 1, 1/2\}$.

Proof. Consider the Lagrange dual problem to (9.38)-(9.39), where a dual variable $\lambda_{\{ij\}}$ [1] corresponds to the dualized inequality $x_i + x_j \leq 1$ and the reparametrized cost is defined as $c_i^{\boldsymbol{\lambda}} = c_i - \sum_{j\in\mathcal{N}(i)} \lambda_{\{ij\}}$:

$$\min_{\boldsymbol{\lambda}\in\mathbb{R}^{\mathcal{E}}_{\geq 0}} \left[\sum_{\{i,j\}\in\mathcal{E}} \lambda_{\{ij\}} + \max_{\mathbf{x}\in[0,1]^{\mathcal{V}}} \left\langle \mathbf{c}^{\boldsymbol{\lambda}}, \mathbf{x} \right\rangle \right] \tag{9.40}$$

The dual is tight for the edge relaxation (9.38)-(9.39) according to Theorem 5.11. Hence, the dual optimality condition given by Corollary 5.18 determines also properties of the optimal primal solutions $\mathbf{x}$:

Lemma 9.22. Let $\boldsymbol{\lambda}$ be dual optimal. Then $c_i^{\boldsymbol{\lambda}} > 0$ implies $x_i^* = 1$ in any primal optimal solution $\mathbf{x}^*$ of the relaxed problem (9.38). Likewise, $c_i^{\boldsymbol{\lambda}} < 0$ implies $x_i^* = 0$.

Proof. According to the dual optimality condition (Corollary 5.18), each optimal primal solution $\mathbf{x}^*$ must satisfy $\mathbf{x}^* \in \arg\max_{\mathbf{x}\in[0,1]^{\mathcal{V}}} \left\langle \mathbf{c}^{\boldsymbol{\lambda}}, \mathbf{x} \right\rangle$.

[1] As in Chapter 3 the lower index $\{ij\}$ means that $\lambda_{\{ij\}}$ and $\lambda_{\{ji\}}$ denote the same variable

The latter condition is equivalent to $x_i^* \in \arg\max_{x\in[0,1]} \langle c_i^{\boldsymbol{\lambda}}, x\rangle$ for all $i \in \mathcal{V}$. In turn, $c_i^{\boldsymbol{\lambda}} > 0$ and $x_i^* \in \arg\max_{x\in[0,1]} \langle c_i^{\boldsymbol{\lambda}}, x\rangle$ imply $x_i^* = 1$. Similarly, $c_i^{\boldsymbol{\lambda}} < 0$ and $x_i^* \in \arg\max_{x\in[0,1]} \langle c_i^{\boldsymbol{\lambda}}, x\rangle$ imply $x_i^* = 0$. □

Algorithm 13 Assigning a primal relaxed solution for max-weight independent set problem

Input: $\boldsymbol{\lambda}$ - optimal dual solution, $\mathbf{c}^{\boldsymbol{\lambda}}$ - reparametrized costs
// Enforce $x_i \in \max_{x\in[0,1]} \langle c_i^{\boldsymbol{\lambda}}, x\rangle$:
$\forall i \in \mathcal{V}$: $c_i^{\boldsymbol{\lambda}} > 0$ assign $x_i := 1$
$\forall i \in \mathcal{V}$: $c_i^{\boldsymbol{\lambda}} < 0$ assign $x_i := 0$
repeat
    // Enforce feasibility $x_i + x_j \leq 1$:
    $\forall i \in \mathcal{V}$: $x_i = 1$ assign $x_j := 0$ $\forall j \in \mathcal{N}(i)$
    // Enforce complementary slackness $\lambda_{\{ij\}}(x_i + x_j - 1) = 0$:
    $\forall \{i,j\} \in \mathcal{E}$: $x_i = 0$ and $\lambda_{\{ij\}} > 0$ assign $x_j := 1$
until no new nodes assigned
// if a node had to be assigned both 0 and 1, exit - $\boldsymbol{\lambda}$ is non-optimal
Assign 1/2 to all so far unassigned nodes
return $\mathbf{x}$

Apart from the properties defined by Lemma 9.22, an optimal $\mathbf{x}^*$ must satisfy feasibility ((9.42)) and complementary slackness (5.90) constraints. The latter takes the form $\lambda_{\{ij\}}(x_i + x_j - 1) = 0$ in the considered case.

Algorithm 13 employs these constraints to construct a half-integral primal relaxed solution given an optimal dual solution $\boldsymbol{\lambda}$. Algorithm starts by fixing coordinates of $\mathbf{x}$ based on the condition $x_i \in \arg\max_{x\in[0,1]} \langle c_i^{\boldsymbol{\lambda}}, x\rangle$. Afterwards the solution is propagated based on the feasibility $x_i + x_j \leq 1$ and complementary slackness $\lambda_{\{ij\}}(x_i + x_j - 1) = 0$ conditions. Should this lead to a conflicting assignment, *i.e.*, some node must be assigned both 0 and 1, this implies that the dual optimality condition provided in Corollary 5.18 does not hold and $\boldsymbol{\lambda}$ is not optimal therefore.

The remaining nodes are assigned values 1/2 that satisfies both feasibility and complementary slackness conditions for any edge $\{i,j\} \in \mathcal{E}$.

Algorithm 13 implies that a half-integral solution exists for any cost vector $\mathbf{c}$. Consider now a cost vector such that the primal relaxed solution is unique. It will constitute a vertex of the polytope $\{\mathbf{x} \in [0,1]^{\mathcal{V}} \colon x_i + x_j \leq 1\,,\ \{i,j\} \in \mathcal{E}\}$. As a half-integral solution always exists, this vertex has to have half-integral coordinates, which finalizes the proof. □

Consider now the reduction of the max-weight independent set problem to the binary labeling as given above. Let $\boldsymbol{\theta}$ be the respective costs of the labeling problem and $\mathcal{L}$ – the respective binary local polytope. Then the following holds:

Theorem 9.23.

$$\max_{\mathbf{x} \in [0,1]^{\mathcal{V}}} \langle \mathbf{c}, \mathbf{x} \rangle \tag{9.41}$$

$$\text{s.t. } x_i + x_j \leq 1\,,\ \{i,j\} \in \mathcal{E}\,. \tag{9.42}$$

is equal to

$$\max_{\boldsymbol{\mu} \in \mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle \tag{9.43}$$

and reducible to the min-*st*-cut problem therefore.

Moreover, a relaxed solution $\boldsymbol{\mu}$ of the labeling problem can be computed from the relaxed solution $\mathbf{x}$ of the max-wieght independent set problem as

$$\begin{aligned} \mu_i(1) = x_i\,;\ \mu_i(0) = 1 - x_i\,,\ i \in \mathcal{V} \\ \mu_{ij}(s,l) \in \operatorname*{arg\,min}_{\boldsymbol{\mu}_{ij} \in \mathcal{L}_{ij}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j)} \langle \boldsymbol{\theta}_{ij}, \boldsymbol{\mu}_{ij} \rangle\ ,\ ij \in \mathcal{E}\,, \end{aligned} \tag{9.44}$$

where $\mathcal{L}_{ij}(\boldsymbol{\mu}_i, \boldsymbol{\mu}_j)$ is defined as in (9.27).

Proof. It is sufficient to show that objective values of corresponding basic solutions for both problems coincide, since any other relaxed solution is a convex combination of the basic ones. Due to Theorems 9.17 and 9.21 it is sufficient to show this for all half-integral solutions, as the latter include all basic solutions.

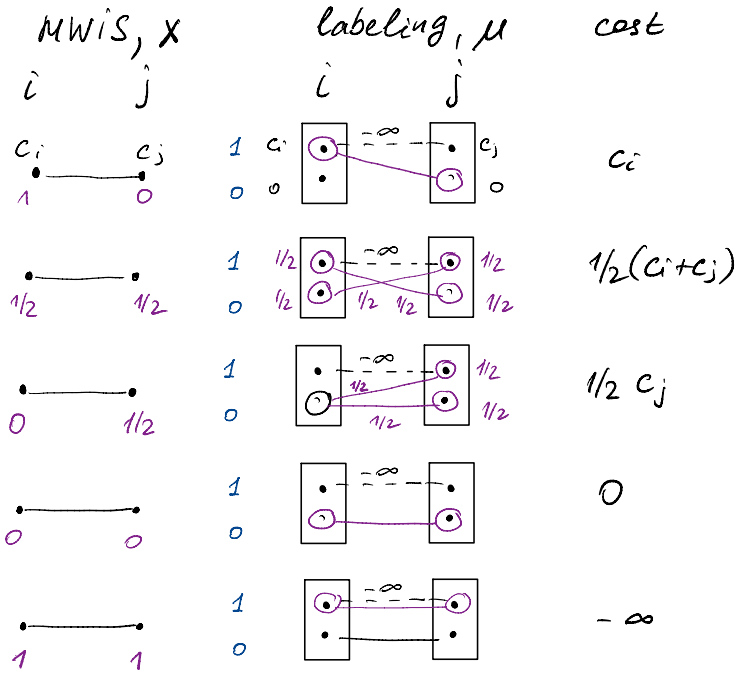


Figure 9.10: Edge-wise transformation of a solution $\mathbf{x}$ of the edge relaxation of the max-weight independent set problem to a solution $\boldsymbol{\mu}$ of the binary local polytope relaxation of the maximization labeling problem according to (9.44).

Consider Figure 9.10. It shows an edge-wise transformation between $\mathbf{x}$ and $\boldsymbol{\mu}$ in accordance with (9.44).

The respective terms have exactly the same costs with an exception that the solution including $x_i = x_j = 1$ if infeasible for $\{i, j\} \in \mathcal{E}$, but for the respective $\boldsymbol{\mu}_i, \boldsymbol{\mu}_j, \boldsymbol{\mu}_{ij}$ it holds $\langle \boldsymbol{\theta}_i, \boldsymbol{\mu}_i \rangle + \langle \boldsymbol{\theta}_j, \boldsymbol{\mu}_j \rangle + \langle \boldsymbol{\theta}_{ij}, \boldsymbol{\mu}_{ij} \rangle = -\infty$ and hence they can never be optimal.

This finalizes the proof, as implies that objective values of corresponding basic solutions for both problems coincide. □

9.3.4 Persistency of the binary local polytope

In this section we get back to the question of how integer coordinates of a relaxed solution are related to an exact non-relaxed solution. In general (see, *e.g.*, [3, Ch. 4] and Exercise Sheet 1), one cannot assume that the integer coordinates are kept unchanged in some exact non-relaxed solution. However, for the binary local polytope this property indeed holds, as we show below.

First, we will give the definition of the considered property and a sufficient condition for its existence for general graphical models. Afterwards we concentrate on the binary labeling problem and show that the sufficient condition always holds in this case.

Persistency definition and criterion for general graphical models

Definition 9.24 (Persistency of an integer coordinate)**.** Let

$$\boldsymbol{\mu}' \in \arg\min_{\boldsymbol{\mu}\in\mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle \tag{9.45}$$

be a relaxed solution of the labeling problem for a graphical model $(\mathcal{G}, \mathcal{Y}_\mathcal{V}, \boldsymbol{\theta})$. For any $u \in \mathcal{V}$ its coordinate $\mu'_u(s)$ is called *(weakly) persistent* or *(weakly) partially optimal*, if (i) $\mu'_u(s) \in \{0, 1\}$ and (ii) there exists an exact integer solution

$$\boldsymbol{\mu}^* \in \arg\min_{\boldsymbol{\mu}\in\mathcal{L}\cap\{0,1\}^\mathcal{I}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle \tag{9.46}$$

such that $\mu^*_u(s) = \mu'_u(s)$. It is *strongly persistent* if this property holds for *all* exact integer solutions $\boldsymbol{\mu}^*$.

A somewhat more general definition of persistency can also be given without relating it to any relaxation.

Definition 9.25 (Persistency of a partial labeling)**.** A partial labeling $\mathbf{y}' \in \mathcal{Y}_\mathcal{A}$ on a subset $\mathcal{A} \subseteq \mathcal{V}$ is *(weakly) partially optimal* or *(weakly) persistent* if $\mathbf{y}' = \mathbf{y}^*_\mathcal{A}$ for some $\mathbf{y}^* \in \arg\min_{\mathbf{y}\in\mathcal{Y}_\mathcal{V}} \langle \boldsymbol{\theta}, \delta(\mathbf{y}) \rangle$.

The following proposition formulates a sufficient persistency condition:

Proposition 9.26 (Persistency criterion [106])**.** A partial labeling $\mathbf{y}' \in \mathcal{Y}_\mathcal{A}$ is persistent, if for all $\tilde{\mathbf{y}} \in \mathcal{Y}_{\mathcal{V}\setminus\mathcal{A}}$ the following holds

$$\mathbf{y}' \in \arg\min_{\mathbf{y}\in\mathcal{Y}_\mathcal{A}} E((\mathbf{y}, \tilde{\mathbf{y}}))\,, \tag{9.47}$$

where $(\cdot,\cdot)$ stands for the labeling concatenation, i.e.

$$(\mathbf{y}, \tilde{\mathbf{y}})_u = \begin{cases} y_u, & u \in \mathcal{A}\,, \\ \tilde{y}_u, & \mathcal{V}\setminus\mathcal{A}\,. \end{cases} \tag{9.48}$$

Proof. Consider the equation

$$\min_{\mathbf{y}\in\mathcal{Y}_\mathcal{V}} E(\mathbf{y}) = \min_{\tilde{\mathbf{y}}\in\mathcal{Y}_{\mathcal{V}\setminus\mathcal{A}}} \min_{\mathbf{y}\in\mathcal{Y}_\mathcal{A}} E((\mathbf{y}, \tilde{\mathbf{y}}))\,. \tag{9.49}$$

Let $\tilde{\mathbf{y}} \in \mathcal{Y}_{\mathcal{V}\setminus\mathcal{A}}$ be such that it leads to a minimal value on the right hand side of (9.49). Then $\tilde{\mathbf{y}}$ is part of an optimal solution. By the assumption (9.47), $\mathbf{y}'$ is an optimal solution to the inner minimization problem of (9.49), hence $(\mathbf{y}', \tilde{\mathbf{y}})$ minimizes E. □

Persistency of the binary local polytope

Let $\mathcal{I}$ be the set of indexes associated with possible labels and label pairs of a binary graphical model, see Section 3.3. For an arbitrary vector $\boldsymbol{\mu} \in \mathbb{R}^\mathcal{I}$ let $\mathrm{Int}(\boldsymbol{\mu}) = \{u \in \mathcal{V} \colon \boldsymbol{\mu}_u \in \{0,1\}^2\}$ be the set of nodes corresponding to integer coordinates of $\boldsymbol{\mu}$.

Theorem 9.27. Let $\boldsymbol{\mu}' \in \arg\min_{\boldsymbol{\mu}\in\mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu}\rangle$ be a relaxed solution of a binary labeling problem. Then $\boldsymbol{\mu}'_{\mathrm{Int}(\boldsymbol{\mu}')}$ is persistent.

Proof. The proof is illustrated in Figure 9.11. Let $\mathcal{A} := \mathrm{Int}(\boldsymbol{\mu}')$. Let also $\mathbf{y}' \in \mathcal{X}_\mathcal{A}$ be selected such that $\delta(\mathbf{y}') = \boldsymbol{\mu}'_\mathcal{A}$. The criterion of Proposition 9.26 is independent of a reparametrization of $\boldsymbol{\theta}$. W.l.o.g. we will assume the latter to be (dual) optimal and such that for all

$w \in \mathcal{V} \cup \mathcal{E}$ it holds that

$$\min_{s \in \mathcal{Y}_w} \theta_w(s) = 0\,. \tag{9.50}$$

Let us consider the persistency criterion (9.47). For a fixed $\tilde{\mathbf{y}} \in \mathcal{Y}_{\mathcal{V}\setminus\mathcal{A}}$ it holds that

$$\arg\min_{\mathbf{y} \in \mathcal{Y}_{\mathcal{A}}} E((\mathbf{y}, \tilde{\mathbf{y}})) = \arg\min_{\mathbf{y} \in \mathcal{Y}_{\mathcal{A}}} \underbrace{\left(E_{\mathcal{A}}(\mathbf{y}) + \sum_{\substack{uv \in \mathcal{E}\,:\\ u \in \mathcal{A}, v \in \mathcal{V}\setminus\mathcal{A}}} \theta_{uv}(y_u, \tilde{y}_v) \right)}_{\geq 0}\,, \tag{9.51}$$

where the energy $E_{\mathcal{A}}$ is restricted to the induced subgraph of $\mathcal{A}$ and the inequality holds due to (9.50).

Since $\boldsymbol{\mu}'$ is primal relaxed optimal and $\boldsymbol{\theta}$ is dual optimal, it holds that $E_{\mathcal{A}}(\mathbf{y}') = 0$ due to the strong duality, see Proposition 5.24(4). Moreover, since the solution $\boldsymbol{\mu}'$ outside $\mathcal{A}$ is non-integer, it holds that $\mu'_u(s) > 0$ for all $s \in \mathcal{Y}_u$, $u \in \mathcal{V}\setminus\mathcal{A}$. Therefore, due to the local polytope constraints, $\mu_{uv}(y'_u, y_v) > 0$ for all $u \in \mathcal{A}$, $v \in \mathcal{V}\setminus\mathcal{A}$ with $uv \in \mathcal{E}$ and $y_v \in \mathcal{Y}_v$. Statement 5 of Proposition 5.24 implies $\theta_{uv}(y'_u, y_v) = 0$, in particular $\theta_{uv}(y'_u, \tilde{y}_v) = 0$. Hence, we obtain

$$\left(E_{\mathcal{A}}(\mathbf{y}') + \sum_{\substack{uv \in \mathcal{E}_{\partial\mathcal{A}}\,:\\ u \in \mathcal{A}}} \theta_{uv}(y'_u, \tilde{y}_v) \right) = 0\,. \tag{9.52}$$

Comparing to (9.51) we obtain $\mathbf{y}' \in \arg\min_{\mathbf{y} \in \mathcal{Y}_{\mathcal{A}}} E((\mathbf{y}, \tilde{\mathbf{y}}))$, which finalizes the proof. □

Remark 9.28. An *improving* property of $\mathbf{y}'$ follows from the fact that it minimizes (9.51). It reads

$$E((\mathbf{y}, \tilde{\mathbf{y}})) \geq E((\mathbf{y}', \tilde{\mathbf{y}})), \quad \forall \mathbf{y} \in \mathcal{Y}_{\mathrm{Int}(\boldsymbol{\mu})} \text{ and } \tilde{\mathbf{y}} \in \mathcal{Y}_{\mathcal{V}\setminus\mathrm{Int}(\boldsymbol{\mu})}\,, \tag{9.53}$$

where $\boldsymbol{\mu}$ is the relaxed solution. This property constitutes a basis for a practical approximate algorithm for aolving arbitrary binary labeling problems considered in Section 9.3.5.

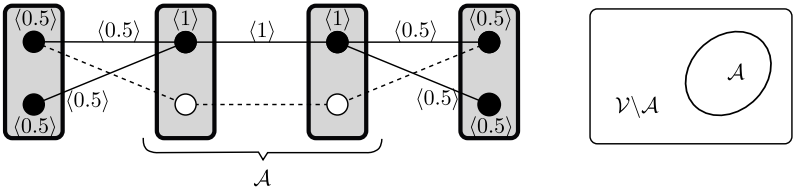


Figure 9.11: Illustration to the proof of Theorem 9.27. Black labels and solid lines correspond to the non-zero values of the primal variables of a given relaxed solution $\boldsymbol{\mu}'$. An example of such values is written in angular brackets $\langle\rangle$. These non-zero values correspond to locally optimal labels and label pairs in an optimal reparametrization $\boldsymbol{\theta}$. We assume these locally optimal labels and label pairs to have zero weight. Dashed lines correspond to an arbitrary selected alternative labeling. Since it may contain locally non-minimal labels and label pairs, it is not minimal in general.

9.3.5 Practical recombination algorithms: QPBO and QPBO-I

QPBO Algorithm Consider now the case when recombination reduces to the binary labeling problem $(\mathcal{G}, \boldsymbol{\theta}, \{0,1\}^{\mathcal{V}})$ and $\mathbf{y}^1 = (1, 1, \ldots, 1)$ and $\mathbf{y}^2 = (0, 0, \ldots, 0)$ be the two labelings to be merged through recombination. By solving the binary local polytope relaxation $\boldsymbol{\mu}^* \in \arg\min_{\boldsymbol{\mu}\in\mathcal{L}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle$ via reduction to min-*st*-cut, we obtain the relaxed solution $\boldsymbol{\mu}^*$. Let $\mathcal{A} := \mathrm{Int}(\boldsymbol{\mu}^*)$ be the integer, persistent part of the solution. Then we can substitute the respective coordinates of $\mathbf{y}^1$ and $\mathbf{y}^2$ with $\boldsymbol{\mu}_{\mathcal{A}}$ obtaining $\hat{\mathbf{y}}^1$ and $\hat{\mathbf{y}}^2$ respectively. We will denote this substitution operation as $\mathrm{FUSE}(\mathbf{y}, \boldsymbol{\mu})$, which is, $\hat{\mathbf{y}}^i = \mathrm{FUSE}(\mathbf{y}^i, \boldsymbol{\mu}^*)$ for $i = 1, 2$. See Figure 9.12 for illustration.

According to (9.53), $E(\hat{\mathbf{y}}^i) \leq E(\mathbf{y}^i)$, $i = 1, 2$. So, the result of the recombination can be computed as $\hat{\mathbf{y}} := \arg\min_{\mathbf{y}\in\{\hat{\mathbf{y}}^1, \hat{\mathbf{y}}^2\}} E(\mathbf{y})$. This algorithm is often referred to as *QPBO*, which means no more than *quadratic pseudo-boolean optimization*.

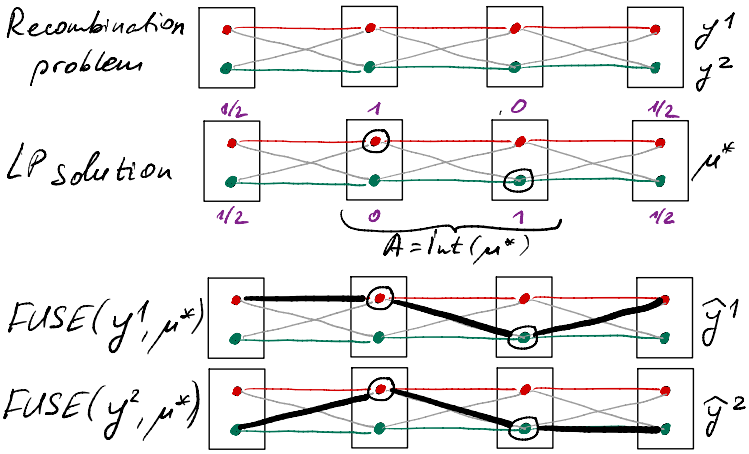


Figure 9.12: Usage of the improving persistency property (9.53) for improving feasible solutions for the binary labeling (recombination) problem.

However, in many practically important cases the set $\mathcal{A}$ is either empty or very small, so there is either no solution improvement or it is very small. One way to deal with this situation is to apply any primal heuristic to the binary labeling problem and return the best labeling out of $\mathbf{y}^1$, $\mathbf{y}^2$ and the solution provided by the heuristic. Below we will describe a particular heuristic, called *QPBO-I* (*QPBO-Improving*) that has two important properties:

- QPBO-I takes a labeling $\mathbf{y}$ as an input and outputs a labeling $\hat{\mathbf{y}}$ such that $E(\hat{\mathbf{y}}) \leq E(\mathbf{y})$.
- QPBO-I leverages the persistency property of the binary local polytope to obtain better solutions.

QPBO-I Algorithm. The idea of the QPBO-I algorithm is the following:

- Let the set of integer labelled nodes $\mathcal{A}$ be empty. Then fixing a label in some of the nodes and recomputing an optimal relaxed solution for the rest of the nodes may lead to a non-empty $\mathcal{A}$. So, the solution can be then be fixed in $\mathcal{A}$ and the process repeated.
- The input solution $\mathbf{y}$ is used to fix label(ing) in some node(s), that is, if i is the node index, it obtains the label y_i. Together with the previous step this guarantees that the obtained labeling is at least as good as $\mathbf{y}$, see Lemma 9.29 below.

To formalize the above procedure we introduce the following operation: given a labeling $\mathbf{y}$ fix its labels in a given node subset $S \subseteq \mathcal{V}$ and compute a relaxed solution $\boldsymbol{\mu}^*$ on the graph induced by $\mathcal{V}\backslash S$ under condition of the fixed $\mathbf{y}_S$:

$$\boldsymbol{\mu}^* := \mathrm{LP}(\mathbf{y}, S) \in \operatorname*{arg\,min}_{\substack{\boldsymbol{\mu}\in\mathcal{L} \\ \boldsymbol{\mu}_S=\delta(\mathbf{y})_S}} \langle \boldsymbol{\theta}, \boldsymbol{\mu} \rangle \ . \tag{9.54}$$

This problem coincides with the local polytope relaxation on the graph induced by nodes $\mathcal{V}\backslash S$, where the pairwise costs of the edges connecting $\mathbf{y}_S$ and $\mathcal{V}\backslash S$ are added to the costs of incident labels in $\mathcal{V}\backslash S$, see Figure 9.13.

Algorithm 14 Algorithm QPBO-I

```
Given: binary graphical model ((V, E), θ, {0,1}^V)
  V = {1, ..., n} - set of nodes is ordered,
  y ∈ {0,1}^V - initial solution
Initialize: μ := LP(y, ∅); ŷ := FUSE(y, μ); S := Int(μ)
for i = 1 to n do
    if i ∉ S then
        μ := LP(ŷ, S ∪ {i}); ŷ := FUSE(ŷ, μ); S := Int(μ)
    end if
end for
return ŷ
```

Algorithm 14 rigorously defines QPBO-I. Its improving character, *i.e.*, $E(\text{QPBO-I}(\hat{\mathbf{y}})) \leq E(\mathbf{y})$ directly follows from an iterative application of

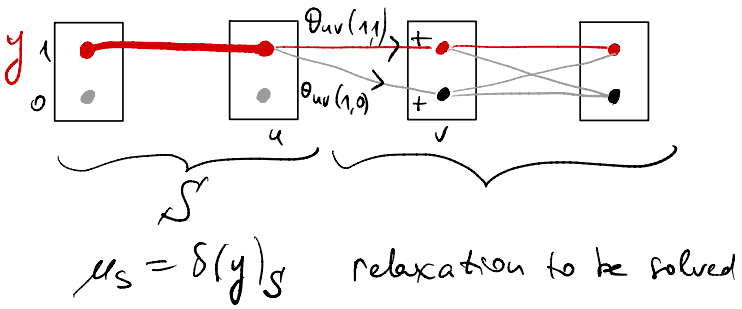


Figure 9.13: Usage of the improving persistency property (9.53) for improving feasible solutions for the binary labeling (recombination) problem.

Lemma 9.29. For any $\mathbf{y} \in \{0,1\}^{\mathcal{V}}$ and any $S \subseteq \mathcal{V}$ it holds $E(\hat{\mathbf{y}}) \leq E(\mathbf{y})$, where $\hat{\mathbf{y}}$ is defined as $\hat{\mathbf{y}} = \text{FUSE}(\mathbf{y}, \text{LP}(\mathbf{y}, S))$.

Proof. The considered operation effectively reduces the set of allowed labels in nodes from S to those from $\mathbf{y}$. Afterwards the required decreasing (non-increasing) of the energy directly follows from the improving property of the persistent labeling (9.53). □

To solve the recombination problem with two input labelings $\mathbf{y}^1$ and $\mathbf{y}^2$ one can compute $\hat{\mathbf{y}} := \text{QPBO-I}(\arg\min_{\mathbf{y} \in \{\mathbf{y}^1, \mathbf{y}^2\}})$.

9.3.6 Bibliography and further reading

extend with results on MWIS Although there is a fair amount of literature on quadratic pseudo-boolean optimization and its relation to binary labeling problems and min-*st*-cut, the presentation of the chapter is quite new. Typically the proof of Theorem 9.19 is done for dual objectives, which is more practical for applications. The primal exposition, like the one in this monograph, seems to be more illustrative and intuitive. Moreover, it is more straightforwardly related to persistency. The typical proof for the persistency property is done by reducing the problem to the vertex cover problem and operating by

known results in that domain. It turned out that this result can easily be proven directly, see Theorem 9.27, although similar proofs exist in the literature, e.g. [104].

Related works on quadratic pseudo-boolean optimization are [99], [107], [108]. Extensions and applications of the persistency property of the binary local polytope are discussed in [109]–[111].

In a recent line of research [106], [112]–[118] persistency of multi-label problems has been studied and efficient polynomial algorithms for its computation were proposed.

The recombination for labeling problems known as *fusion moves* was proposed and discussed in [119]–[121].

Bibliography

[1] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, ACM, 1971, pp. 151–158.

[2] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, Springer, 1972.

[3] B. Savchynskyy, "Discrete graphical models – an optimization perspective," *Foundations and Trends in Computer Graphics and Vision*, 2019.

[4] M. R. Garey and D. S. Johnson, *Computers and intractability*. WH Freeman New York, 2002, vol. 29.

[5] S. Arora and B. Barak, *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[6] B. Korte, J. Vygen, B. Korte, and J. Vygen, *Combinatorial optimization*. Springer, 2002.

[7] R. Bellman and S. Dreyfus, *Applied dynamic programming*. Princeton University Press, 1962.

[8] A. Viterbi, "Convolutional codes and their performance in communication systems," *IEEE Transactions on Communication Technology*, vol. 19, no. 5, pp. 751–772, Oct. 1971. DOI: 10.1109/TCOM.1971.1090700.

[9] J. Pearl, *Probabilistic reasoning in intelligent systems*. Palo Alto: Morgan Kaufmann, 1988.

[10] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE Transactions on information theory*, vol. 47, no. 2, pp. 498–519, 2001.

[11] M. Schlesinger and V. Hlavác, *Ten lectures on statistical and structural pattern recognition*. Springer Science & Business Media, 2013, vol. 24.

[12] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer, 2004, ISBN: 978-3-540-40286-2. DOI: 10.1007/978-3-540-24777-7. [Online]. Available: https://doi.org/10.1007/978-3-540-24777-7.

[13] T. Ibaraki, "Integer programming formulation of combinatorial optimization problems," *Discrete Mathematics*, vol. 16, no. 1, pp. 39–52, 1976.

[14] G. B. Dantzig and M. N. Thapa, *Linear programming 1: Introduction*. Springer Science & Business Media, 2006.

[15] A. Schrijver, *Theory of linear and integer programming*. John Wiley & Sons, 1998.

[16] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: Algorithms and complexity*. Courier Corporation, 1982.

[17] R. T. Rockafellar, *Convex analysis*. Princeton university press, 2015.

[18] R. Fortet, "Applications de l'algebre de boole en recherche opérationelle," *Revue Française de Recherche Opérationelle*, vol. 4, no. 14, pp. 17–26, 1960.

[19] W. P. Adams and H. D. Sherali, "A tight linearization and an algorithm for zero-one quadratic programming problems," *Management Science*, vol. 32, no. 10, pp. 1274–1290, 1986.

[20] D. Prusa and T. Werner, "Universality of the local marginal polytope.," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 4, p. 898, 2015.

[21] H. D. Sherali and W. P. Adams, *A reformulation-linearization technique for solving discrete and continuous nonconvex problems*. Springer Science & Business Media, 2013, vol. 31.

[22] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[23] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2004.

[24] R. T. Rockafellar, *Conjugate duality and optimization*. SIAM, 1974, vol. 16.

[25] M. Shlezinger, "Syntactic analysis of two-dimensional visual signals in the presence of noise," *Cybernetics and systems analysis*, vol. 12, no. 4, pp. 612–628, 1976.

[26] V. K. Koval' and M. I. Schlesinger, "Dvumernoe programmirovanie v zadachakh analiza izobrazheniy (Two-dimensional programming in image analysis problems)," *Avtomatika i Telemekhanika*, no. 8, pp. 149–168, 1976.

[27] D. L. Waltz, "Generating semantic description from drawings of scenes with shadows," MIT Artificial Intelligence Laboratory, Tech. Rep. AI271, 1972.

[28] A. Rosenfeld, R. A. Hummel, and S. W. Zucker, "Scene labeling by relaxation operations," *IEEE Transactions on Systems, Man and Cybernetics*, no. 6, pp. 420–433, 1976.

[29] T. Werner, "A linear programming approach to max-sum problem: A review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 7, pp. 1165–1179, 2007.

[30] B. Savchynskyy and S. Schmidt, "Getting feasible variable estimates from infeasible ones: MRF local polytope study," in *Advanced Structured Prediction*, MIT Press, 2014.

[31] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[32] A. Bulatov, P. Jeavons, and A. Krokhin, "Classifying the complexity of constraints using finite algebras," *SIAM Journal on Computing*, vol. 34, no. 3, pp. 720–742, 2005.

[33] D. Cohen and P. Jeavons, "Chapter 8: The complexity of constraint languages," in *Handbook of constraint programming*, Elsevier, 2006.

[34] E. V. Vodolazskii, B. Flach, and M. I. Schlesinger, "Minimax problems of discrete optimization invariant under majority operators," *Computational Mathematics and Mathematical Physics*, vol. 54, no. 8, pp. 1327–1336, 2014.

[35] M. Schlesinger and B. Flach, "Some solvable subclasses of structural recognition problems," in *Czech Pattern Recognition Workshop*, vol. 2000, 2000, pp. 55–62.

[36] B. T. Polyak, *Introduction to optimization*. Optimization Software New York, 1987.

[37] D. P. Bertsekas, *Nonlinear programming, second edition*. Athena scientific, 1999.

[38] S. Haller and B. Savchynskyy, "A Bregman-Sinkhorn algorithm for the maximum weight independent set problem," *arXiv preprint arXiv:2408.02086*, 2024.

[39] N. Z. Shor, *Minimization methods for non-differentiable functions*. Springer Science & Business Media, 2012, vol. 3.

[40] M. I. Schlesinger and V. V. Giginyak, "Solution to structural recognition (max,+)-problems by their equivalent transformations. in 2 Parts," *Control Systems and Computers*, no. 1-2, 2007.

[41] G. Storvik and G. Dahl, "Lagrangian-based methods for finding MAP solutions for MRF models," *IEEE Transactions on Image Processing*, vol. 9, no. 3, pp. 469–479, 2000.

[42] N. Komodakis, N. Paragios, and G. Tziritas, "MRF optimization via dual decomposition: Message-passing revisited," in *ICCV*, 2007.

[43] N. Komodakis, N. Paragios, and G. Tziritas, "MRF energy minimization and beyond via dual decomposition," *IEEE Trans. PAMI*, vol. 33, pp. 531–552, 3 Mar. 2011. DOI: http://dx.doi.org/10.1109/TPAMI.2010.108.

[44] M. J. Powell, "On search directions for minimization algorithms," *Mathematical Programming*, vol. 4, no. 1, pp. 193–201, 1973.

[45] A. Beck and L. Tetruashvili, "On the convergence of block coordinate descent type methods," *SIAM journal on Optimization*, vol. 23, no. 4, pp. 2037–2060, 2013.

[46] V. Kolmogorov, "A new look at reweighted message passing," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 37, no. 5, pp. 919–930, 2015.

[47] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE transactions on pattern analysis and machine intelligence*, vol. 28, no. 10, pp. 1568–1583, 2006.

[48] M. Schlesinger and K. Antoniuk, "Diffusion algorithms and structural recognition optimization problems," *Cybernetics and Systems Analysis*, vol. 47, no. 2, pp. 175–192, 2011.

[49] V. Voracek and T. Werner, "Convergence of some convex message passing algorithms to a fixed point," in *ICML*, 2024.

[50] Z.-Q. Luo and P. Tseng, "On the convergence of the coordinate descent method for convex differentiable minimization," *Journal of Optimization Theory and Applications*, vol. 72, no. 1, pp. 7–35, 1992.

[51] P. Tseng, "Dual coordinate ascent methods for non-strictly convex minimization," *Mathematical programming*, vol. 59, no. 1-3, pp. 231–247, 1993.

[52] P. Tseng and S. Yun, "Block-coordinate gradient descent method for linearly constrained nonsmooth separable optimization," *Journal of optimization theory and applications*, vol. 140, no. 3, p. 513, 2009.

[53] P. Tseng, "Convergence of a block coordinate descent method for nondifferentiable minimization," *Journal of optimization theory and applications*, vol. 109, no. 3, pp. 475–494, 2001.

[54] T. Werner, "On coordinate minimization of piecewise-affine functions," Dept. of Cybernetics, Fac. of Electrical Eng., Czech Technical Univ. in Prague, Tech. Rep. CTU-CMP-2017-05, Sep. 2017.

[55] T. Werner, D. Průša, and T. Dlask, "Relative interior rule in block-coordinate descent," in *Conf. on Computer Vision and Pattern Recognition*, Jun. 2020, pp. 7559–7567.

[56] T. Dlask and T. Werner, "On relation between constraint propagation and block-coordinate descent in linear programs," in *26th Intl. Conf. on Principles and Practice of Constraint Programming*, 2020, pp. 194–210.

[57] T. Dlask and T. Werner, "Classes of linear programs solvable by coordinate-wise minimization," *Annals of Mathematics and Artif. Intell.*, vol. 90, pp. 777–807, 2022.

[58] T. Werner, D. Prusa, and T. Dlask, "Relative interior rule in block-coordinate descent," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7559–7567.

[59] J. Besag, "Spatial interaction and the statistical analysis of lattice systems," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 192–236, 1974.

[60] B. M. Kelm, N. Mueller, B. H. Menze, and F. A. Hamprecht, "Bayesian estimation of smooth parameter maps for dynamic contrast-enhanced MR images with block-ICM," in *Computer Vision and Pattern Recognition Workshop, CVPRW'06. Conference on*, IEEE, 2006, pp. 96–96.

[61] Q. Chen and V. Koltun, "Fast MRF optimization with application to depth reconstruction," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 3914–3921.

[62] D. Thuerck, M. Waechter, S. Widmer, *et al.*, "A fast, massively parallel solver for large, irregular pairwise Markov random fields.," in *High Performance Graphics*, 2016, pp. 173–183.

[63] B. Andres, J. H. Kappes, T. Beier, U. Köthe, and F. A. Hamprecht, "The lazy flipper: Efficient depth-limited exhaustive search in discrete graphical models," in *European Conference on Computer Vision*, Springer, 2012, pp. 154–166.

[64] T. Hazan and A. Shashua, "Norm-product belief propagation: Primal-dual message-passing for approximate inference," *IEEE Transactions on Information Theory*, vol. 56, no. 12, pp. 6294–6316, 2010.

[65] A. Globerson and T. S. Jaakkola, "Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations," in *Advances in Neural Information Processing Systems 20*, 2008.

[66] S. Tourani, A. Shekhovtsov, C. Rother, and B. Savchynskyy, "MPLP++: Fast, parallel dual block-coordinate ascent for dense graphical models," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 251–267.

[67] T. Werner, "On coordinate minimization of convex piecewise-affine functions," *arXiv preprint arXiv:1709.04989*, 2017.

[68] S. Tourani, A. Shekhovtsov, C. Rother, and B. Savchynskyy, "Taxonomy of dual block-coordinate ascent methods for discrete energy minimization," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2020, pp. 2775–2785.

[69] M. C. Cooper, S. De Givry, M. Sánchez, T. Schiex, M. Zytnicki, and T. Werner, "Soft arc consistency revisited," *Artificial Intelligence*, vol. 174, no. 7-8, pp. 449–478, 2010.

[70] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Prentice Hall Inc., 1982.

[71] J. Weed, "An explicit analysis of the entropic penalty in linear programming," in *Conference On Learning Theory*, PMLR, 2018, pp. 1841–1855.

[72] J. Altschuler, J. Niles-Weed, and P. Rigollet, "Near-linear time approximation algorithms for optimal transport via sinkhorn iteration," *Advances in neural information processing systems*, vol. 30, 2017.

[73] G. Peyré, M. Cuturi, *et al.*, "Computational optimal transport," *Center for Research in Economics and Statistics Working Papers*, no. 2017-86, 2017.

[74] T. Achterberg, "Constraint integer programming," Ph.D. dissertation, TU Berlin, 2007.

[75] M. Grötschel, L. Lovász, and A. Schrijver, "The ellipsoid method and its consequences in combinatorial optimization," *Combinatorica*, vol. 1, no. 2, pp. 169–197, 1981.

[76] R. E. Gomory, "Solving linear programming problems in integers," *Combinatorial analysis*, vol. 10, no. 211-215, p. 25, 1960.

[77] T. Achterberg, T. Berthold, S. Heinz, T. Koch, and K. Wolter, "Constraint integer programming: Techniques and applications," 2008, See https://depositonce.tu-berlin.de/bitstream/11303/1931/2/Dokument_41.pdf.

[78] *SCIP: Solving contraint integer programs*, See https://www.scipopt.org/.

[79] E. Balas, "Facets of the knapsack polytope," *Mathematical programming*, vol. 8, pp. 146–164, 1975.

[80] H. Marchand, "A polyhedral study of the mixed knapsack set and its use to solve mixed integer programs," Ph.D. dissertation, UCL-Université Catholique de Louvain, 1998.

[81] A. N. Letchford and A. Lodi, "Strengthening chvátal–gomory cuts and gomory fractional cuts," *Operations Research Letters*, vol. 30, no. 2, pp. 74–82, 2002.

[82] J. Ostrowski, J. Linderoth, F. Rossi, and S. Smriglio, "Orbital branching," *Mathematical Programming*, vol. 126, pp. 147–178, 2011.

[83] M. W. Savelsbergh, "Preprocessing and probing techniques for mixed integer programming problems," *ORSA Journal on Computing*, vol. 6, no. 4, pp. 445–454, 1994.

[84] E. L. Johnson and M. W. Padberg, "Degree-two inequalities, clique facets, and biperfect graphs," in *North-Holland Mathematics Studies*, vol. 66, Elsevier, 1982, pp. 169–187.

[85] E. L. Johnson, G. L. Nemhauser, and M. W. Savelsbergh, "Progress in linear programming-based algorithms for integer programming: An exposition," *Informs journal on computing*, vol. 12, no. 1, pp. 2–23, 2000.

[86] M. W. Padberg, "A note on zero-one programming," *Operations Research*, vol. 23, no. 4, pp. 833–837, 1975.

[87] A. Schrijver, *Combinatorial optimization: polyhedra and efficiency*. Springer, 2003, vol. 24.

[88] J. W. Moon and L. Moser, "On cliques in graphs," *Israel journal of Mathematics*, vol. 3, pp. 23–28, 1965.

[89] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger, "Presolve reductions in mixed integer programming," *INFORMS Journal on Computing*, vol. 32, no. 2, pp. 473–506, 2020.

[90] M. Conforti, G. Cornuéjols, and G. Zambelli, "Integer programming models," in *Integer Programming*, Springer, 2014, pp. 45–84.

[91] *Gurobi Optimizer: The state-of-the-art mathematical programming solver*, See http://www.gurobi.com/.

[92] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *Journal of global optimization*, vol. 6, pp. 109–133, 1995.

[93] M. G. Resende and C. C. Ribeiro, "Greedy randomized adaptive search procedures: Advances, hybridizations, and applications," *Handbook of metaheuristics*, pp. 283–319, 2010.

[94] K. Sastry, D. Goldberg, and G. Kendall, "Genetic algorithms," *Search methodologies: Introductory tutorials in optimization and decision support techniques*, pp. 97–125, 2005.

[95] L. Hutschenreiter, S. Haller, L. Feineis, C. Rother, D. Kainmüller, and B. Savchynskyy, "Fusion moves for graph matching," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 6270–6279.

[96] A. Kako, T. Ono, T. Hirata, and M. M. Halldórsson, "Approximation algorithms for the weighted independent set problem," in *Graph-Theoretic Concepts in Computer Science: 31st International Workshop, WG 2005, Metz, France, June 23-25, 2005, Revised Selected Papers 31*, Springer, 2005, pp. 341–350.

[97] Y. Dong, A. V. Goldberg, A. Noe, N. Parotsidis, M. G. Resende, and Q. Spaen, "A metaheuristic algorithm for large maximum weight independent set problems," *arXiv preprint arXiv:2203.15805*, 2022.

[98] J. Kleinberg and E. Tardos, *Algorithm design*. Pearson Education India, 2006.

[99] E. Boros and P. L. Hammer, "Pseudo-boolean optimization," *Discrete applied mathematics*, vol. 123, no. 1, pp. 155–225, 2002.

[100] S. T. McCormick, "Submodular function minimization," in *Discrete Optimization*, ser. Handbooks in Operations Research and Management Science, K. Aardal, G. Nemhauser, and W. R., Eds., vol. 12, Elsevier, 2005, pp. 321–391. DOI: https://doi.org/10.1016/S0927-0507(05)12007-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0927050705120076.

[101] D. M. Greig, B. T. Porteous, and A. H. Seheult, "Exact maximum a posteriori estimation for binary images," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 271–279, 1989.

[102] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?" *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 26, no. 2, pp. 147–159, 2004.

[103] M. I. Schlesinger, "Lectures on labeling problems attended by the author," Kyiv, 1999-2002.

[104] V. Kolmogorov and M. Wainwright, "On the optimality of tree-reweighted max-product message-passing," in *Uncertainty in Artificial Intelligence (UAI)*, 2005.

[105] G. L. Nemhauser and L. E. Trotter Jr., "Vertex packings: Structural properties and algorithms," *Mathematical Programming*, vol. 8, no. 1, pp. 232–248, 1975.

[106] P. Swoboda, B. Savchynskyy, J. Kappes, and C. Schnörr, "Partial optimality by pruning for MAP-inference with general graphical models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1170–1177.

[107] E. Boros, P. Hammer, and X. Sun, "Network flows and minimization of quadratic pseudo-boolean functions," Tech. Rep., 1991.

[108] P. L. Hammer, P. Hansen, and B. Simeone, "Roof duality, complementation and persistency in quadratic 0–1 optimization," *Mathematical programming*, vol. 28, no. 2, pp. 121–155, 1984.

[109] C. Rother, V. Kolmogorov, V. Lempitsky, and M. Szummer, "Optimizing binary MRFs via extended roof duality," in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, IEEE, 2007, pp. 1–8.

[110] V. Kolmogorov, "Generalized roof duality and bisubmodular functions," in *Advances in neural information processing systems*, 2010, pp. 1144–1152.

[111] P. Kohli, A. Shekhovtsov, C. Rother, V. Kolmogorov, and P. Torr, "On partial optimality in multi-label MRFs," in *Proceedings of the 25th international conference on Machine learning*, ACM, 2008, pp. 480–487.

[112] P. Swoboda, B. Savchynskyy, J. Kappes, and C. Schnörr, "Partial optimality via iterative pruning for the Potts model," in *International Conference on Scale Space and Variational Methods in Computer Vision*, Springer, 2013, pp. 477–488.

[113] A. Shekhovtsov, "Exact and partial energy minimization in computer vision," PhD Thesis CTU–CMP–2013–24, CMP, Czech Technical University in Prague, 2013.

[114] A. Shekhovtsov, "Maximum persistency in energy minimization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1162–1169.

[115] P. Swoboda, A. Shekhovtsov, J. Kappes, C. Schnörr, and B. Savchynskyy, "Partial optimality by pruning for MAP-inference with general graphical models," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 38, no. 7, pp. 1370–1382, Jul. 2016. DOI: 10.1109/TPAMI.2015.2484327.

[116] A. Shekhovtsov, P. Swoboda, and B. Savchynskyy, "Maximum persistency via iterative relaxed inference with graphical models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 521–529.

[117] A. Shekhovtsov, P. Swoboda, and B. Savchynskyy, "Maximum persistency via iterative relaxed inference with graphical models," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, 2017, pp. 1668–1682. DOI: 10.1109/TPAMI.2017.2730884.

[118] A. Shekhovtsov, "Higher order maximum persistency and comparison theorems," *Computer Vision and Image Understanding*, vol. 143, pp. 54–79, 2016.

[119] V. Lempitsky, S. Roth, and C. Rother, "Fusionflow: Discrete-continuous optimization for optical flow estimation," in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, IEEE, 2008, pp. 1–8.

[120] V. Kolmogorov and C. Rother, "Minimizing nonsubmodular functions with graph cuts-a review," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 29, no. 7, pp. 1274–1279, 2007.

[121] V. Lempitsky, C. Rother, S. Roth, and A. Blake, "Fusion moves for Markov random field optimization," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 32, no. 8, pp. 1392–1405, 2010.