

# Combinatorial Optimization

Lecture notes, WS 2010/11, TU Munich

Prof. Dr. Raymond Hemmecke

Version of February 9, 2011



# Contents

<b>1</b>	<b>The knapsack problem</b>	<b>1</b>
1.1	Complete enumeration . . . . .	1
1.2	Dynamic programming approach . . . . .	2
1.3	FPTAS for the knapsack problem . . . . .	3
1.4	Related problems . . . . .	4
<b>2</b>	<b>Main solution ideas for ILPs</b>	<b>7</b>
2.1	General remarks . . . . .	7
2.2	Cutting plane methods . . . . .	7
2.3	Branch-and-bound algorithm . . . . .	8
2.4	Branch-and-cut algorithm . . . . .	9
2.5	Primal methods . . . . .	9
2.6	Goals for the rest of the course . . . . .	9
<b>3</b>	<b>Methods to strengthen an IP formulation</b>	<b>11</b>
3.1	Repetition: Concepts from polyhedral geometry . . . . .	11
3.2	Methods to generate valid inequalities . . . . .	12
3.2.1	Rounding . . . . .	12
3.2.2	Superadditivity . . . . .	13
3.2.3	Modular arithmetic . . . . .	14
3.2.4	Disjunctions . . . . .	14
3.3	Methods to generate facet defining inequalities . . . . .	16
3.3.1	Using the definition . . . . .	16
3.3.2	Lifting a valid inequality . . . . .	17
3.4	Lift-and-Project . . . . .	20

---

<b>4</b>	<b>Ellipsoid Method and Separation</b>	<b>25</b>
4.1	Ellipsoid Method . . . . .	25
4.2	Separation . . . . .	26
<b>5</b>	<b>Branching strategies</b>	<b>29</b>
5.1	Selection of a node . . . . .	30
5.2	Branching rules . . . . .	31
<b>6</b>	<b>Column generation</b>	<b>35</b>
6.1	The Cutting Stock Problem (CSP) . . . . .	35
6.1.1	A first IP formulation . . . . .	36
6.1.2	Second IP formulation . . . . .	37
6.2	Column generation . . . . .	37
6.3	Column generation for CSP . . . . .	38
<b>7</b>	<b>Benders decomposition</b>	<b>39</b>
7.1	Formal derivation . . . . .	39
7.2	Solution via Benders decomposition . . . . .	41
7.3	Stochastic Programming . . . . .	41
<b>8</b>	<b>Lagrange relaxation</b>	<b>43</b>
8.1	Solving the Lagrange dual . . . . .	46

# Chapter 1

## The knapsack problem

We are given  $n$  items having positive weights  $a_1, \dots, a_n$  and positive values  $c_1, \dots, c_n$ . Moreover, we are given a knapsack and a bound  $b$  on the weight that can be carried in the knapsack. We wish to put into the knapsack a selection of items of maximum total value. To avoid trivialities, we will assume that  $a_i \leq b$  for all  $i$ , as otherwise the  $i$ -th item could not be carried in the knapsack and we could discard it from the problem.

To model the knapsack problem, we introduce binary variables  $x_i, i = 1, \dots, n$ .  $x_i$  shall be equal to 1 if item  $i$  is chosen, and 0 otherwise. Then our knapsack problem can be modeled as the following integer program:

$$\max \left\{ \sum_{i=1}^n c_i x_i : \sum_{i=1}^n a_i x_i \leq b, x_i \in \{0, 1\} \forall i \right\}.$$

This is already an NP-hard optimization problem! Therefore, unless  $P=NP$ , we can at most hope for a good approximation algorithm. Below we will encounter a best possible such approximation algorithm, a so-called **FPTAS** (short-hand for “**fully polynomial-time approximation scheme**”). First, let us deal with exact solution methods that find an optimal selection of items.

### 1.1 Complete enumeration

The probably simplest, but most naive, method is to enumerate all  $2^n$  possible selections of items, check for each whether it fits into the knapsack and compute the largest total value among those selections that fit into the knapsack.

Clearly, from a complexity point of view, this complete enumeration is not a good idea and should be avoided.

## 1.2 Dynamic programming approach

Let us explain the idea behind the dynamic programming approach by solving our knapsack problem

$$\max \left\{ \sum_{i=1}^n c_i x_i : \sum_{i=1}^n a_i x_i \leq b, x_i \in \{0, 1\} \forall i \right\}.$$

Instead of choosing all items at once, we assume that we choose the items one at a time, starting with item 1, then item 2, and so on. After  $j$  decisions, we have determined the values of  $x_1, \dots, x_j$ .

At this point, the accumulated value is  $\sum_{i=1}^j c_i x_i$  and accumulated weight is  $\sum_{i=1}^j a_i x_i$ .

Let  $C_j(u)$  be the least possible weight that has to be accumulated in order to attain a total value of exactly  $u$  using only (some or all of) the first  $j$  items. If it is not possible to attain a total value of  $u$  using the first  $j$  items, we put  $C_j(u) = +\infty$ . Moreover, we put  $C_0(0) = 0$  and  $C_0(u) = +\infty$  for  $u \neq 0$ . The crucial observation now is that we have the following simple recursion formula

$$C_{j+1}(u) = \min \{ C_j(u), C_j(u - c_{j+1}) + a_{j+1} \}.$$

In words: if we wish to accumulate a total value of  $u$  with least weight using only items from  $\{1, \dots, j+1\}$ , we either do not choose item  $j+1$  and have smallest possible weight  $C_j(u)$  or we do choose item  $j+1$  and have smallest possible weight  $C_j(u - c_{j+1}) + a_{j+1}$ . For both subcases we already use known best solutions. It is this step where we gain a big advantage in running time over the complete enumeration!

Note: Our knapsack problem is equivalent to finding the largest  $u$  for which  $C_n(u) \leq b$ .

What is the largest  $u$  for which we have to compute  $C_n(u)$ ? Clearly, if  $c_{\max} := \max_{i=1, \dots, n} c_i$ , then  $\sum_{i=1}^n c_i x_i \leq n c_{\max}$ . Thus, we only have to compute the values  $C_n(u)$  for  $u = 0, 1, \dots, n c_{\max}$ , to solve our knapsack problem.

In order to do so, we first compute  $C_1(0), \dots, C_n(0)$  (well, they are all 0), then  $C_1(1), \dots, C_n(1)$ , then  $C_1(2), \dots, C_n(2)$ ,  $\dots$ , and finally  $C_1(n c_{\max}), \dots, C_n(n c_{\max})$ . These values can be arranged into a two-dimensional  $n \times n c_{\max}$  array. In order to obtain the actual values  $x_1, \dots, x_n$  of an optimal solution, we only have to back-track the computation of  $C_n(u)$  for the optimal value  $u$ .

Let us have a look at the running time of this dynamic programming approach. If  $a_{\max} := \max_{i=1, \dots, n} a_i$ , then the computation of  $C_{j+1}(u)$  from  $C_j(u)$  and  $C_j(u - c_{j+1}) + a_{j+1}$  takes  $O(n(\langle a_{\max} \rangle + \langle c_{\max} \rangle))$  binary operations. Thus, in total, we have a running time of  $O(n^3 c_{\max}(\langle a_{\max} \rangle + \langle c_{\max} \rangle))$  for this dynamic programming approach. Although  $c_{\max}$  is typically exponential in the encoding length of the input data (**Exercise:** What is the encoding length of a knapsack instance?), we can clearly see that the dynamic programming approach outperforms the complete enumeration of  $2^n$  potential solutions.

**Exercise:** A somewhat more natural dynamic programming algorithm for the solution of the knapsack problem could be obtained by computing values  $V_j(w)$ , encoding the maximum possible accumulated value using some of the first  $j$  items subject to the constraint that their accumulated weight is exactly  $w$ .

### 1.3 FPTAS for the knapsack problem

**Definition 1.3.1** Consider an optimization problem  $\max\{f(\mathbf{x}) : \mathbf{x} \in X\}$  and denote by  $\mathbf{x}^*$  an optimal feasible solution.

- (a) An algorithm  $A$  is an  $\epsilon$ -approximation algorithm if it computes a feasible solution  $\mathbf{x}_0 \in X$  with  $f(\mathbf{x}_0) \geq (1 - \epsilon)f(\mathbf{x}^*)$ .
- (b) A family of algorithms  $A_\epsilon$  is a **polynomial time approximation scheme (PTAS)** if for every error parameter  $\epsilon > 0$ ,  $A_\epsilon$  is an  $\epsilon$ -approximation algorithm and its running time is polynomial in the encoding length of the instance for fixed  $\epsilon$ .
- (c) If  $A_\epsilon$  is an approximation scheme and its running time is polynomial in the encoding length of the instance and in  $1/\epsilon$ , then it is a **fully polynomial time approximation scheme (FPTAS)**.

Let us now state an FPTAS for the solution of the knapsack problem. Be reminded that the running time of the dynamic programming approach above is  $O(n^3 c_{\max}(\langle a_{\max} \rangle + \langle c_{\max} \rangle))$ . Thus, if we reduce  $c_{\max}$ , the resulting instance is faster to solve. For example, instead of solving a knapsack problem with objective function  $94x_1 + 72x_2 + 104x_3$  we could slightly change the objective function to  $90x_1 + 70x_2 + 100x_3$  and hope that the optimal cost does not change too much. The latter objective, however, is equivalent to  $9x_1 + 7x_2 + 10x_3$ , leading to a knapsack that can be solved 10 times faster!

---

#### FPTAS for the knapsack problem

**Input:** vectors  $\mathbf{a}, \mathbf{c} \in \mathbb{Z}_+^n$ , a scalar  $b \in \mathbb{Z}_+$ , an error parameter  $\epsilon > 0$

**Output:** a solution  $\mathbf{x}_0$  to  $\max\{\mathbf{c}^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq b, \mathbf{x} \in \{0, 1\}^n\}$  with  $\mathbf{c}^\top \mathbf{x}_0 \geq (1 - \epsilon)\mathbf{c}^\top \mathbf{x}^*$

1. If  $n/c_{\max} > \epsilon$ , then solve the knapsack problem exactly using the dynamic programming algorithm above.
2. **(Rounding)** If  $n/c_{\max} \leq \epsilon$ , let

$$t = \left\lceil \log_{10} \left( \frac{\epsilon c_{\max}}{n} \right) \right\rceil,$$

$$\bar{c}_i = \left\lfloor \frac{c_i}{10^t} \right\rfloor, i = 1, \dots, n.$$

Solve the knapsack problem  $\max\{\bar{\mathbf{c}}^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq b, \mathbf{x} \in \{0, 1\}^n\}$  exactly using the dynamic programming algorithm above. Let  $\mathbf{x}_0$  be the optimal solution found.

3. Return  $\mathbf{x}_0$ .
- 

The following theorem shows that this is indeed an FPTAS for the knapsack problem.

**Theorem 1.3.2** *This algorithm finds a feasible solution  $\mathbf{x}_0$  to  $\max\{\mathbf{c}^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq b, \mathbf{x} \in \{0, 1\}^n\}$  with  $\mathbf{c}^\top \mathbf{x}_0 \geq (1 - \epsilon)\mathbf{c}^\top \mathbf{x}^*$  in  $O(n^4(\langle a_{\max} \rangle + \langle c_{\max} \rangle)/\epsilon)$  steps.*

**Proof.** If  $n/c_{\max} > \epsilon$ , we solve the problem exactly, that is, the approximation error in this case is 0. The running time is  $O(n^3 c_{\max}(\langle a_{\max} \rangle + \langle c_{\max} \rangle)) = O(n^4(\langle a_{\max} \rangle + \langle c_{\max} \rangle)/\epsilon)$ .

If  $n/c_{\max} \leq \epsilon$ , then  $t$  satisfies

$$\frac{\epsilon}{10} < \frac{n10^t}{c_{\max}} \leq \epsilon. \quad (1.1)$$

Thus, we get

$$\bar{c}_{\max} \leq \frac{c_{\max}}{10^t} \leq \frac{10n}{\epsilon},$$

and the running time is  $O(n^3 \bar{c}_{\max}(\langle a_{\max} \rangle + \langle c_{\max} \rangle)) = O(n^4(\langle a_{\max} \rangle + \langle c_{\max} \rangle)/\epsilon)$ .

Let us now compute the approximation error. Let  $S$  (respectively  $\bar{S}$ ) be a set of items chosen by some optimal solution for  $\max\{\mathbf{c}^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq b, \mathbf{x} \in \{0, 1\}^n\}$  (respectively for  $\max\{\bar{\mathbf{c}}^\top \mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq b, \mathbf{x} \in \{0, 1\}^n\}$ ). Then we have

$$\sum_{i \in S} c_i \geq \sum_{i \in \bar{S}} c_i \geq 10^t \sum_{i \in \bar{S}} \bar{c}_i \geq 10^t \sum_{i \in S} \bar{c}_i \geq \sum_{i \in S} (c_i - 10^t) \geq \sum_{i \in S} c_i - n10^t. \quad (1.2)$$

The first inequality holds because of optimality of  $S$  for the original problem. The second inequality holds, since  $10^t \bar{c}_i \leq c_i$  for all  $i$ . The third inequality holds because of optimality of  $\bar{S}$  for the modified problem. The fourth inequality holds, since  $\bar{c}_i = \lfloor c_i/10^t \rfloor \geq c_i/10^t - 1$  and thus  $10^t \bar{c}_i \geq c_i - 10^t$ . Finally, the last inequality holds, since  $|S| \leq n$ .

Thus, we have

$$\frac{\mathbf{c}^\top \mathbf{x}^* - \mathbf{c}^\top \mathbf{x}_0}{\mathbf{c}^\top \mathbf{x}^*} = \frac{\sum_{i \in S} c_i - \sum_{i \in \bar{S}} c_i}{\sum_{i \in S} c_i} \leq \frac{n10^t}{c_{\max}} \leq \epsilon. \quad (1.3)$$

The first inequality follows from Equation (1.2) and from the fact that  $a_i \leq b$  for all  $i$  and hence  $\sum_{i \in S} c_i \geq c_{\max}$ . The second inequality follows from Equation (1.1).

Rewriting Equation (1.3), we get

$$\mathbf{c}^\top \mathbf{x}_0 \geq (1 - \epsilon)\mathbf{c}^\top \mathbf{x}^*,$$

and the approximation error is sufficiently small.  $\square$

## 1.4 Related problems

The knapsack problem is the simplest discrete optimization (optimization over lattice points in a simplex), but it is already NP-hard. It relates to other interesting problems that, at least, we wish to mention here.

## The Frobenius problem

**Frobenius problem.** Let  $a_1, \dots, a_n \in \mathbb{Z}_+$  with  $\gcd(a_1, \dots, a_n) = 1$ . What is the largest number  $b = f(a_1, \dots, a_n) \in \mathbb{Z}_+$  such that the equation

$$a_1x_1 + \dots + a_nx_n = b$$

has no solution  $x_1, \dots, x_n \in \mathbb{Z}_+$ . This number  $f(a_1, \dots, a_n)$  is called the *Frobenius number* of  $a_1, \dots, a_n$ .

The Frobenius problem is also known as the *Coin exchange problem*: If  $a_1, \dots, a_n \in \mathbb{Z}_+$  denote the values of coins, what is the largest amount that cannot be changed with these coins?

**Exercise.** Why is  $f(a_1, \dots, a_n) < \infty$  whenever  $\gcd(a_1, \dots, a_n) = 1$ ?

The Frobenius number is NP-hard to compute. However, one can again design a dynamic programming framework to compute it. From a theoretical point of view, it is interesting to find explicit formulas for the Frobenius number. For  $n = 2$ , there exists such a closed-form expression:

**Lemma 1.4.1 (Nijenhuis and Wilf, 1972)** For  $a_1, a_2 \in \mathbb{Z}_+$  with  $\gcd(a_1, a_2) = 1$  we have  $f(a_1, a_2) = (a_1 - 1)(a_2 - 1) - 1 = a_1a_2 - (a_1 + a_2)$ .

There is no known closed-form solution for  $n = 3$ , although a semi-explicit solution is known which allows values to be computed quickly.

## The feasibility problem

One fundamental problem in integer programming is the question, whether the set of feasible solutions  $\{\mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} \leq \mathbf{b}\}$  or  $\{\mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  is empty or not.

**Exercise.** In fact, integer linear programs could be solved by polynomially many calls to a feasibility oracle, which tells you for any given polytope, if this polytope contains a lattice point or not. How? (“Polynomially many”: polynomially bounded from above in the encoding length of the input data  $A$  and  $b$ .)

As the knapsack problem is the simplest (non-trivial) discrete optimization problem, it is pretty well studied. To study the general situation, we see that  $\{\mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$  is not empty if and only if  $\mathbf{b} \in \text{monoid}(\mathbf{a}_1, \dots, \mathbf{a}_n)$ , where  $\mathbf{a}_i$  denotes the  $i$ -th column of  $A$ . A necessary condition for  $\mathbf{b} \in \text{monoid}(\mathbf{a}_1, \dots, \mathbf{a}_n)$  is  $\mathbf{b} \in \text{cone}(\mathbf{a}_1, \dots, \mathbf{a}_n) \cap \mathbb{Z}^n$ , which can be checked via linear programming.

One interesting problem is the question whether  $\text{monoid}(\mathbf{a}_1, \dots, \mathbf{a}_n) = \text{cone}(\mathbf{a}_1, \dots, \mathbf{a}_n) \cap \mathbb{Z}^n$ , that is, whether  $\mathbf{a}_1, \dots, \mathbf{a}_n$  form a *Hilbert basis* for the cone they generate. In other words: is there some  $\mathbf{b} \in \mathbb{Z}^n$  that can be represented as a nonnegative *real* linear combination but not as a nonnegative *integer* linear combination of  $\mathbf{a}_1, \dots, \mathbf{a}_n$ . Clearly, this is an NP-hard problem. From a practical perspective, however, it would be great to model this problem as some feasibility or linear/nonlinear optimization problem (that may have good practical algorithms and implementations for their solution).



## Chapter 2

# Main solution ideas for ILPs

### 2.1 General remarks

For any polytope  $P$  we denote by  $P_I$  the convex hull of all lattice points in  $P$ , that is,

$$P_I = \text{conv}(P \cap \mathbb{Z}^n).$$

If we knew a linear description of  $P_I$ , we could apply any method from linear programming to solve the IP

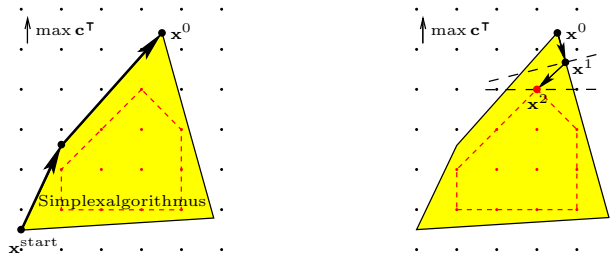
$$\max\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in P \cap \mathbb{Z}^n\} = \max\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in P_I\}.$$

The better  $P$  approximates  $P_I$ , the better the LP relaxation  $\max\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in P\}$  approximates the true IP optimum. Thus, we will be interested in strengthening the inequalities of  $P$  (so that we get a smaller polytope still containing  $P_I$ ) and in adding more inequalities (cutting off some parts of  $P$  that do not belong to  $P_I$ ).

### 2.2 Cutting plane methods

The main idea behind cutting plane methods is the following:

1. Relax the problem and solve the LP relaxation  $\max\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in P\}$ .  $\rightarrow \mathbf{x}^*$  optimal solution
2. If LP relaxation is infeasible, STOP: IP is infeasible.
3. If  $\mathbf{x}^* \in \mathbb{Z}^n$ , we have found an optimal IP solution.  $\rightarrow$  STOP: optimal IP solution found
4. If  $\mathbf{x}^* \notin \mathbb{Z}^n$ , find an inequality  $\mathbf{a}^\top \mathbf{x} \leq \alpha$  that cuts off  $\mathbf{x}^*$ , Let  $P := P \cap \{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq \alpha\}$  and go to 1.

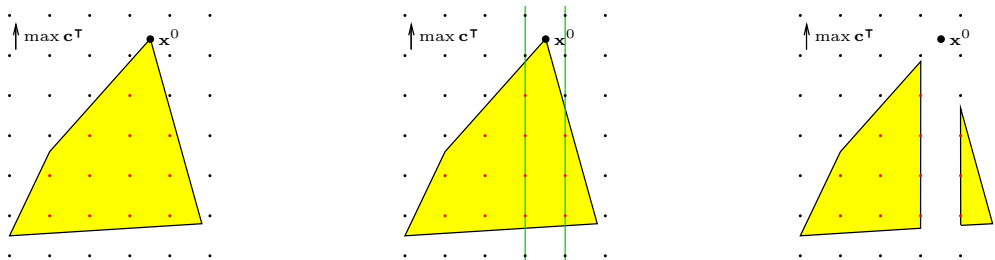


## 2.3 Branch-and-bound algorithm

The main ideas behind the branch-and-bound algorithm are the following.

### Branching

In a branch-and-bound framework, the original IP is iteratively split into smaller and smaller subproblems in some way. For example, one may consider a relaxed problem by dropping integrality or nonnegativity constraints. Then, an optimal solution to this relaxed problem is used to split the given IP into two or more smaller IPs which are then solved individually. In the worst case, this iterative construction leads to a complete enumeration of all lattice points in the original polyhedron. Another way to split the original problem, even without considering any relaxed problem, is to split it according to  $x_i = 0, 1, \dots, u_i$ .



### Bounding

To prune this tree of IP subproblems, one can use the following observations.

- If a subproblem is infeasible, it can be dropped.
- If the optimal solution of a subproblem is integer (or nonnegative or ...), we have found a feasible solution to our original problem. The subproblem need not be split any further and can be removed, as it has been solved to optimality.

- If the optimal value of (the relaxation of) a subproblem is equal or worse than the currently best objective value of a feasible solution found so far, this subproblem can be removed, since it cannot contain a better feasible solution than the currently best known solution.

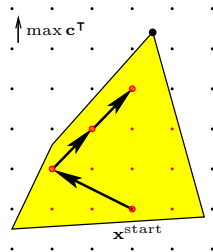
## 2.4 Branch-and-cut algorithm

Branch-and-cut combines the ideas of branch-and-bound and cutting plane methods. In some or all nodes of the branch-and-bound tree, the IP formulation is strengthened by cuts. This strengthening is very expensive computationally. However, far fewer nodes have to be considered. Currently, branch-and-cut is the practically best approach to solve an ILP.

## 2.5 Primal methods

Primal methods start with a feasible solution and iteratively augment it to better and better feasible solutions until an optimal solution is found. Prominent examples are the Ford-Fulkerson algorithm to find a maximum flow in a network.

Heuristics that try to find an augmentation step for a given feasible solution may also help any branch-and-bound or branch-and-cut algorithm, since better feasible solutions help to prune the search tree even further.



## 2.6 Goals for the rest of the course

Among others, we will deal with several aspects of the above algorithms:

- strengthening of inequalities
- finding new valid inequalities and facets for  $P_I$
- separating over exponentially many facets/cuts
- primal and dual heuristics to find good feasible solutions or bounds
- branching rules



## Chapter 3

# Methods to strengthen an IP formulation

### 3.1 Repetition: Concepts from polyhedral geometry

In this section we repeat some notions from polyhedra geometry that will be needed for the remainder of the course.

#### Definition 3.1.1

- A **polyhedron** is the solution set to a system of finitely many linear inequalities  $A\mathbf{x} \leq \mathbf{b}$ .
- A polyhedron is called **rational** if it is the solution set to a system of finitely many linear inequalities  $A\mathbf{x} \leq \mathbf{b}$  for some rational matrix  $A$  and some rational vector  $\mathbf{b}$ .
- A **polyhedral cone** is the solution set to a (homogeneous) system of finitely many linear inequalities  $A\mathbf{x} \leq \mathbf{0}$ .
- A (polyhedral) cone is called **rational** if it is a rational polyhedron.
- A **polytope** is the convex hull of finitely many points.

**Lemma 3.1.2** A set  $P$  is a polytope if and only if  $P$  is a bounded polyhedron.

**Lemma 3.1.3 (Minkowski-Weyl)** A set  $P$  is a rational polyhedron if and only if there exist finite sets  $V \in \mathbb{Q}^n$  and  $E \in \mathbb{Q}^n$  such that  $P = \text{conv}(V) + \text{cone}(E)$ .

**Definition 3.1.4** Let  $P = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \leq \mathbf{b}\}$  be a polyhedron.

- A **valid inequality** for  $P$  is an inequality  $\mathbf{c}^\top \mathbf{x} \leq \gamma$  that is satisfied by all  $\mathbf{x} \in P$ .

- If  $\mathbf{c}^\top \mathbf{x} \leq \gamma$  is valid for  $P$ , then  $P \cap \{\mathbf{x} : \mathbf{c}^\top \mathbf{x} = \gamma\}$  is called a **face** of  $P$ .
- For a face  $F$  of  $P$ , the set  $\text{eq}(F)$  denotes the set of indices of inequalities  $A_i \mathbf{x} \leq b_i$  that are satisfied with equality on  $F$ , that is,  $F = P \cap \{\mathbf{x} \in \mathbb{R}^n : A_{\text{eq}(F)} \mathbf{x} = \mathbf{b}_{\text{eq}(F)}\}$ .
- The **dimension** of a face is the dimension of  $\text{aff}(F) = \{\mathbf{x} \in \mathbb{R}^n : A_{\text{eq}(F)} \mathbf{x} = \mathbf{b}_{\text{eq}(F)}\}$ , that is,  $\dim(F) = n - \text{rank}(A_{\text{eq}(F)})$ .
- Faces of dimensions 0, 1, and  $\dim(P) - 1$  are called **vertices**, **edges**, and **facets**, respectively.

## 3.2 Methods to generate valid inequalities

Let  $P = \{\mathbf{x} \in \mathbb{R}_+^n : A\mathbf{x} \leq \mathbf{b}\}$  with  $A \in \mathbb{Z}^{d \times n}$ . We wish to generate new valid inequalities for  $P_I = \text{conv}(\mathbf{x} : \mathbf{x} \in P \cap \mathbb{Z}^n)$ . Ideally, these inequalities define facets of  $P_I$ .

### 3.2.1 Rounding

We use the following two simple ideas:

- If  $f(\mathbf{x}) \in \mathbb{Z}$  and  $f(\mathbf{x}) \leq \alpha$ , then we can strengthen the inequality to  $f(\mathbf{x}) \leq \lfloor \alpha \rfloor$ .
- If the inequalities  $A\mathbf{x} \leq \mathbf{b}$  are valid for  $P_I$  and if  $\mathbf{u} \geq \mathbf{0}$ , then  $(\mathbf{u}^\top A)\mathbf{x} \leq \mathbf{u}^\top \mathbf{b}$  is valid for  $P_I$ .

With this we can produce new valid inequalities for  $P_I$  as follows:

1. Choose  $\mathbf{u} \in \mathbb{R}_+^d$ . We obtain the valid inequality  $(\mathbf{u}^\top A)\mathbf{x} = \sum_{j=1}^n (\mathbf{u}^\top A_{\cdot j}) x_j \leq \mathbf{u}^\top \mathbf{b}$  for  $P_I$ .
2. Since  $\mathbf{x} \geq \mathbf{0}$ , we can conclude that also  $\sum_{j=1}^n \lfloor \mathbf{u}^\top A_{\cdot j} \rfloor x_j \leq \mathbf{u}^\top \mathbf{b}$  is valid for  $P_I$ .
3. Since  $\mathbf{x} \in \mathbb{Z}^n$ , the inequality  $\sum_{j=1}^n \lfloor \mathbf{u}^\top A_{\cdot j} \rfloor x_j \leq \lfloor \mathbf{u}^\top \mathbf{b} \rfloor$  is valid for  $P_I$ .

The valid inequality  $\sum_{j=1}^n \lfloor \mathbf{u}^\top A_{\cdot j} \rfloor x_j \leq \lfloor \mathbf{u}^\top \mathbf{b} \rfloor$  can be added to  $A\mathbf{x} \leq \mathbf{b}$  to strengthen the IP formulation. Collecting all such inequalities for varying  $\mathbf{u} \geq \mathbf{0}$ , we obtain the **first Chvátal-closure**  $P_1$  of  $P$ :

$$P_1 := \left\{ \mathbf{x} \in \mathbb{R}_+^n : A\mathbf{x} \leq \mathbf{b}, \sum_{j=1}^n \lfloor \mathbf{u}^\top A_{\cdot j} \rfloor x_j \leq \lfloor \mathbf{u}^\top \mathbf{b} \rfloor, \forall \mathbf{u} \in \mathbb{R}_+^d \right\}.$$

It can be shown that  $P_1$  is again a polyhedron, that is, finitely many inequalities suffice to define it. In fact, if  $A\mathbf{x} \leq \mathbf{b}$  is a TDI-description for  $P$ , then  $A\mathbf{x} \leq \lfloor \mathbf{b} \rfloor$  is a finite description for  $P_1$ . If we recursively define

$$P_{t+1} = (P_t)_1,$$

one can show that there is some finite number  $k$ , the so-called Chvátal-rank of  $P$ , such that  $P_k = P_{k+1} = P_I$ , that is, the sequence  $P_1, P_2, \dots$ , always stabilizes at  $P_I$  after finitely many closure operations.

### 3.2.2 Superadditivity

The valid inequality  $\sum_{j=1}^n [u^\top A_{.j}] x_j \leq [\mathbf{u}^\top \mathbf{b}]$  can also be written in the form  $\sum_{j=1}^n F(A_{.j}) x_j \leq F(\mathbf{b})$  for  $F(\mathbf{a}) = [\mathbf{u}^\top \mathbf{a}]$ . This function is a special case of the more general class of nondecreasing superadditive functions.

**Definition 3.2.1** A function  $F : D \subset \mathbb{R}^d \rightarrow \mathbb{R}$  is **superadditive** if  $F(\mathbf{a}_1) + F(\mathbf{a}_2) \leq F(\mathbf{a}_1 + \mathbf{a}_2)$  for all  $\mathbf{a}_1, \mathbf{a}_2 \in D$  such that  $\mathbf{a}_1 + \mathbf{a}_2 \in D$ .

It is **nondecreasing** if  $F(\mathbf{a}_1) \leq F(\mathbf{a}_2)$  whenever  $\mathbf{a}_1 \leq \mathbf{a}_2$ , for all  $\mathbf{a}_1, \mathbf{a}_2 \in D$ .

Nondecreasing superadditive functions yield valid inequalities in general.

**Theorem 3.2.2** If  $F : \mathbb{R}^d \rightarrow \mathbb{R}$  is superadditive and nondecreasing with  $F(\mathbf{0}) = 0$ , the inequality

$$\sum_{j=1}^n F(A_{.j}) x_j \leq F(\mathbf{b})$$

is valid for  $P_I$ , where  $P = \{\mathbf{x} \in \mathbb{R}_+^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ .

**Proof.** Let  $\mathbf{x} \in P_I$ . We first show by induction on  $x_j$  that  $F(A_{.j}) x_j \leq F(A_{.j} x_j)$ . For  $x_j = 0$  the inequality is clearly true. Assume that it is true for  $x_j = k - 1$ . Then we obtain

$$F(A_{.j}) k = F(A_{.j}) + F(A_{.j})(k - 1) \leq F(A_{.j}) + F(A_{.j}(k - 1)) \leq F(A_{.j} + A_{.j}(k - 1)),$$

by superadditivity, and the induction is complete. Thus, we have

$$\sum_{j=1}^n F(A_{.j}) x_j \leq \sum_{j=1}^n F(A_{.j} x_j).$$

By superadditivity we get

$$\sum_{j=1}^n F(A_{.j} x_j) \leq F\left(\sum_{j=1}^n A_{.j} x_j\right) = F(\mathbf{A}\mathbf{x}).$$

Since  $\mathbf{A}\mathbf{x} \leq \mathbf{b}$  and since  $F$  is nondecreasing, we get

$$F(\mathbf{A}\mathbf{x}) \leq F(\mathbf{b}).$$

Putting these inequalities together, we obtain

$$\sum_{j=1}^n F(A_{.j}) x_j \leq F(\mathbf{b}).$$

□

### 3.2.3 Modular arithmetic

Let us consider the polyhedron

$$P = \left\{ \mathbf{x} \in \mathbb{R}_+^n : \sum_{j=1}^n a_j x_j = a_0 \right\}$$

for given  $\mathbf{a}_i \in \mathbb{Z}$ ,  $i = 0, 1, \dots, n$ . Moreover, let  $k \in \mathbb{Z}_+$ . We write  $a_j = b_j + u_j k$ , where  $b_j$  ( $0 \leq b_j < k, b_j \in \mathbb{Z}$ ) is the remainder when  $a_j$  is divided by  $k$ . Then all points in  $P_I$  satisfy

$$\sum_{j=1}^n b_j x_j = b_0 + r k, \quad \text{for some } r \in \mathbb{Z}.$$

Since  $\sum_{j=1}^n b_j x_j \geq 0$  and  $b_0 \leq k$ , we get  $r \geq 0$ . Thus, the inequality

$$\sum_{j=1}^n b_j x_j \geq b_0$$

is valid for  $P_I$ .

**Example.** Let  $P = \{\mathbf{x} \in \mathbb{R}_+^4 : 27x_1 + 17x_2 - 64x_3 + x_4 = 203\}$  and choose  $k = 13$ . Then we obtain the valid inequality  $x_1 + 4x_2 + x_3 + x_4 \geq 8$  for  $P_I$ .  $\square$

An important family of inequalities is obtained when  $k = 1$  and when the  $a_j$  are not all integers. In this case, since  $\mathbf{x} \geq \mathbf{0}$ , we obtain

$$\sum_{j=1}^n [a_j] x_j \leq a_0.$$

As  $\mathbf{x} \in \mathbb{Z}^n$ , we get

$$\sum_{j=1}^n [a_j] x_j \leq [a_0],$$

and consequently,

$$\sum_{j=1}^n (a_j - [a_j]) x_j \geq a_0 - [a_0].$$

These inequalities are called **Gomory cuts** and form the basis of the Gomory cutting plane algorithm.

### 3.2.4 Disjunctions

The idea behind the following construction is to partition  $P_I$  into two sets  $P_1$  and  $P_2$ , obtain valid inequalities for  $P_1$  and  $P_2$  and combine them to get valid inequalities for  $P_I = P_1 \cup P_2$ .

**Lemma 3.2.3** *If the inequality  $\sum_{j=1}^n a_j x_j \leq b$  is valid for  $P_1 \subseteq \mathbb{R}_+^n$ , and the inequality  $\sum_{j=1}^n c_j x_j \leq d$  is valid for  $P_2 \subseteq \mathbb{R}_+^n$ , then the inequality*

$$\sum_{j=1}^n \min(a_j, c_j) x_j \leq \max(b, d)$$

*is valid for  $P_1 \cup P_2$ .*

**Proof.** Let  $\mathbf{x} \in P_1 \cup P_2$ . Without loss of generality, we may assume that  $\mathbf{x} \in P_1$ , and therefore  $\sum_{j=1}^n a_j x_j \leq b$ . As  $\mathbf{x} \geq \mathbf{0}$ , this implies

$$\sum_{j=1}^n \min(a_j, c_j) x_j \leq \sum_{j=1}^n a_j x_j \leq b \leq \max(b, d).$$

□

We now use this lemma to derive valid inequalities for  $P_I$ , where  $P = \{\mathbf{x} \in \mathbb{R}_+^n : \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$ . For this, let  $\alpha \in \mathbb{Z}_+^n$ .

**Lemma 3.2.4** *If the inequality  $\sum_{j=1}^n a_j x_j - s(x_k - \alpha) \leq a$  is valid for  $P_I$  for some  $s \geq 0$ , and the inequality  $\sum_{j=1}^n a_j x_j + t(x_k - \alpha - 1) \leq a$  is valid for  $P_I$  for some  $t \geq 0$ , then the inequality*

$$\sum_{j=1}^n a_j x_j \leq a$$

*is valid for  $P_I$ .*

**Proof.** For  $x_k \leq \alpha$ , we have

$$\sum_{j=1}^n a_j x_j \leq \sum_{j=1}^n a_j x_j - s(x_k - \alpha) \leq a.$$

Hence,  $\sum_{j=1}^n a_j x_j \leq a$  is valid for  $P_1 := P_I \cap \{\mathbf{x} \in \mathbb{R}_+^n : x_k \leq \alpha\}$ . Similarly,  $\sum_{j=1}^n a_j x_j \leq a$  is valid for  $P_2 := P_I \cap \{\mathbf{x} \in \mathbb{R}_+^n : x_k \geq \alpha + 1\}$ . By the lemma above,  $\sum_{j=1}^n a_j x_j \leq a$  is valid for  $P_I = P_1 \cup P_2$ . □

**Example.** Consider the two inequalities  $-x_1 + 2x_2 \leq 4$  and  $-x_1 \leq -1$  and rewrite them as

$$(-x_1 + x_2) + (x_2 - 3) \leq 1 \quad \text{and} \quad (-x_1 + x_2) - (x_2 - 2) \leq 1.$$

Applying the previous lemma with  $\alpha = 2$ , we obtain that the inequality  $-x_1 + x_2 \leq 1$  is valid. □

### 3.3 Methods to generate facet defining inequalities

#### 3.3.1 Using the definition

Assume that we have found a valid inequality  $\mathbf{a}^\top \mathbf{x} \leq \alpha$  for  $P_I$ . We would like to show that this inequality is in fact facet defining for  $P_I$ . First we compute the dimension  $k$  of  $P_I$  (this is the maximum number  $k$  such that there exist  $k + 1$  affinely independent points in  $P_I$ ). Then we try to find  $k$  affinely independent points in  $P_I$  that satisfy  $\mathbf{a}^\top \mathbf{x} = \alpha$ . To show the latter, it is sufficient to show that  $\dim(\{\mathbf{x} \in P_I : \mathbf{a}^\top \mathbf{x} = \alpha\}) = k - 1$ . Let us consider an example.

**Example. (Stable set problem)** Let  $G = (V, E)$  be an undirected graph with weights  $w_i$  for all  $i \in V$ . Then the weighted stable set problem is the problem of finding a stable set  $S$  in  $G$  of maximum total weight. (**Stable set**  $S$ : no two nodes in  $S$  are connected by an edge in  $G$ .)

Let  $n = |V|$  and introduce decision variables  $x_i$ .

$$x_i = \begin{cases} 1, & \text{if node } i \text{ is selected} \\ 0, & \text{otherwise} \end{cases}$$

The problem can then be modeled as follows:

$$\max \left\{ \sum_{i \in V} w_i x_i : x_i + x_j \leq 1 \forall \{i, j\} \in E, x_i \in \{0, 1\} \forall i \in V \right\}.$$

A **clique** of  $G$  is a set  $U \subseteq V$  such that  $\{i, j\} \in E$  for all  $i, j \in U$ . Clearly, if  $U$  is a clique, the following gives a valid inequality:

$$\sum_{i \in U} x_i \leq 1.$$

We now try to find conditions under which this inequality is facet defining. Since  $\mathbf{0} \in P_I$  and since all  $n$  unit vectors  $\mathbf{e}_i \in P_I$ , there are  $n + 1$  affinely independent vectors in  $P_I$ . This implies that  $\dim(P_I) = n$ .

A clique  $U$  is maximal, if for all  $i \in V \setminus U$ , the set  $U \cup \{i\}$  is not a clique. We show that  $\sum_{i \in U} x_i \leq 1$  is facet defining if and only if  $U$  is a maximal clique.

W.l.o.g, assume that  $U = [k]$ . Then, clearly,  $\mathbf{e}_1, \dots, \mathbf{e}_k$  satisfy  $\sum_{i \in U} x_i \leq 1$  with equality. Assume that  $U$  is a maximal clique. That is, for each  $i \notin U$ , there is some node  $r(i) \in U$ , such that  $\{i, r(i)\} \notin E$ . Therefore, the vector  $\mathbf{x}^i$  with the  $i$ -th and the  $r(i)$ -th coordinates equal to 1 and the rest equal to 0 is in  $P_I$  and satisfies  $\sum_{i \in U} x_i \leq 1$  with equality. The vectors  $\mathbf{e}_1, \dots, \mathbf{e}_k, \mathbf{x}^{k+1}, \dots, \mathbf{x}^n$  are linearly independent and thus, they are also affinely independent. Therefore, if  $U$  is a maximal clique,  $\sum_{i \in U} x_i \leq 1$  is facet defining.

Conversely, if  $U$  is not maximal,  $\sum_{i \in U} x_i \leq 1$  is not facet defining: Since  $U$  is not maximal, there is a node  $i \notin U$  such that  $U \cup \{i\}$  is a clique, and thus  $\sum_{j \in U \cup \{i\}} x_j \leq 1$  is valid for  $P_I$ . Since  $\sum_{i \in U} x_i \leq 1$  is the sum of  $-x_i \leq 0$  and  $\sum_{j \in U \cup \{i\}} x_j \leq 1$ , the inequality  $\sum_{j \in U} x_j \leq 1$  cannot be facet defining.  $\square$

**Remark.** Similarly, if we consider a cycle  $C \subseteq E$  and denote by  $V(C)$  the set of nodes incident to an edge in  $C$ . For any such cycle of odd cardinality, there cannot be more than  $(V(C) - 1)/2$  nodes in a stable set. (Otherwise, two nodes in the stable set would be adjacent.) Therefore, the inequality

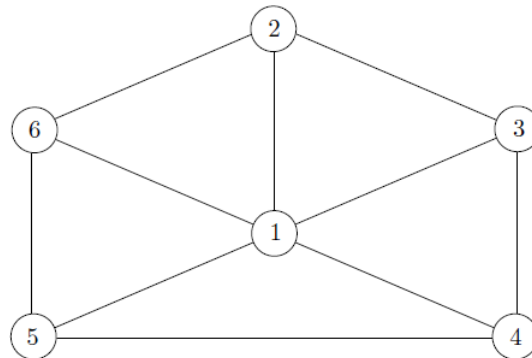
$$\sum_{i \in V(C)} x_i \leq \frac{V(C) - 1}{2},$$

is valid for any cycle  $C$  of odd cardinality.  $\square$

### 3.3.2 Lifting a valid inequality

Geometrically, lifting a valid inequality means to derive a higher-dimensional face from a lower-dimensional face. Let us demonstrate the idea first on an example.

**Example. (Stable set problem, continued)** Consider the maximal clique inequalities on the following graph  $G$ .



$$\begin{array}{rcccccc} x_1 & + & x_2 & + & x_3 & & & \leq & 1 \\ x_1 & & & + & x_3 & + & x_4 & & \leq & 1 \\ x_1 & & & & & + & x_4 & + & x_5 & \leq & 1 \\ x_1 & & & & & & & + & x_5 & + & x_6 & \leq & 1 \\ x_1 & + & x_2 & & & & & & & + & x_6 & \leq & 1 \end{array}$$

The LP maximizing  $\sum_{i=1}^6 x_i$  over the polytope defined by these inequalities and by  $x_i \geq 0$ ,  $i = 1, \dots, 6$ , has as its unique optimal solution  $\mathbf{x}^0 = \frac{1}{2}(0, 1, 1, 1, 1, 1)^\top$ . Thus, the max-clique inequalities are not enough to describe the convex hull of all stable sets.

Note that  $\mathbf{x}^0$  does not satisfy the odd-cycle inequality for  $V(C) = \{2, 3, 4, 5, 6\}$ :

$$x_2 + x_3 + x_4 + x_5 + x_6 \leq 2. \tag{3.1}$$

Let  $\mathcal{F}$  be the set of feasible solutions to the stable set problem on  $G$ . Inequality (3.1) is satisfied with equality by five linearly independent vectors corresponding to the stable sets  $\{2, 4\}$ ,  $\{2, 5\}$ ,

$\{3, 5\}$ ,  $\{3, 6\}$ ,  $\{4, 6\}$ , but it is not facet-defining, as there are no other stable sets that satisfy (3.1) with equality.

However, (3.1) is facet-defining for

$$\text{conv}(\mathcal{F} \cap \{\mathbf{x} \in \{0, 1\}^6 : x_1 = 0\}).$$

We now wish to lift inequality (3.1) that it also defines a facet for  $\text{conv}(\mathcal{F})$ . For this, we consider the inequality

$$\alpha x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 2 \quad (3.2)$$

for some  $\alpha > 0$ . We now try to find a maximal number  $\alpha$  such that (3.2) is still valid for all  $\mathbf{x} \in \mathcal{F}$ , and such that it defines a facet for  $\text{conv}(\mathcal{F})$ .

For  $x_1 = 0$ , inequality (3.2) is valid for all  $\alpha$ . If  $x_1 = 1$ , we get  $\alpha \leq 2 - x_2 - x_3 - x_4 - x_5 - x_6$ . However, since  $x_1 = 1$  implies  $x_2 = x_3 = x_4 = x_5 = x_6 = 0$ , we get  $\alpha \leq 2$ . Thus, inequality (3.2) is valid for  $0 \leq \alpha \leq 2$ . Moreover, for  $\alpha = 2$ , the five vectors above satisfy (3.2) with equality. In addition, also the solution corresponding to the stable set  $\{1\}$  satisfies (3.2) with equality. Since these six vectors are linearly independent, the inequality

$$2x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 2$$

is valid and defines a facet of  $\text{conv}(\mathcal{F})$ . □

The general lifting procedure is as follows.

**Theorem 3.3.1** *Suppose that  $\mathcal{F} \subseteq \{0, 1\}^n$ ,  $\mathcal{F}_i := \mathcal{F} \cap \{\mathbf{x} \in \{0, 1\}^n : x_1 = i\}$ ,  $i = 0, 1$ , and that the inequality*

$$\sum_{j=2}^n a_j x_j \leq a_0$$

*is valid for  $\mathcal{F}_0$ .*

(a) *If  $\mathcal{F}_1 = \emptyset$ , then  $x_1 \leq 0$  is valid for  $\mathcal{F}$ .*

(b) *If  $\mathcal{F}_1 \neq \emptyset$ , then the inequality*

$$a_1 x_1 + \sum_{j=2}^n a_j x_j \leq a_0$$

*is valid for  $\mathcal{F}$  for any  $a_1 \leq a_0 - Z$ , where  $Z$  is the optimal value of  $\max \left\{ \sum_{j=2}^n a_j x_j : \mathbf{x} \in \mathcal{F}_1 \right\}$ .*

(c) *If  $a_1 = a_0 - Z$  and if the inequality  $\sum_{j=2}^n a_j x_j \leq a_0$  defines a face of dimension  $k$  of  $\text{conv}(\mathcal{F}_0)$ ,*

*then the inequality  $a_1 x_1 + \sum_{j=2}^n a_j x_j \leq a_0$  defines a face of  $\mathcal{F}$  of dimension  $k + 1$ .*

**Proof.**

(a) Trivial.

(b) If  $\mathcal{F}_1 \neq \emptyset$ , let  $\mathbf{x} \in \mathcal{F}_0$ . Then

$$a_1x_1 + \sum_{j=2}^n a_jx_j \leq a_0$$

holds. If  $\mathbf{x} \in \mathcal{F}_1$ , then

$$a_1x_1 + \sum_{j=2}^n a_jx_j \leq a_1 + Z \leq a_0.$$

Therefore, the inequality  $a_1x_1 + \sum_{j=2}^n a_jx_j \leq a_0$  is valid for  $\mathcal{F} = \mathcal{F}_0 \cup \mathcal{F}_1$ .

(c) If  $\sum_{j=2}^n a_jx_j \leq a_0$  defines a face of  $\mathcal{F}_0$  of dimension  $k$ , there are  $k+1$  affinely independent points  $\mathbf{x}^1, \dots, \mathbf{x}^{k+1}$  in  $\mathcal{F}_0$  satisfying  $\sum_{j=2}^n a_jx_j \leq a_0$  with equality. Let  $\mathbf{x}^*$  be an optimal solution of the problem of maximizing  $\sum_{j=2}^n a_jx_j$  over  $\mathbf{x} \in \mathcal{F}_1$ . For  $a_1 = a_0 - Z$ , we have

$$a_1x_1^* + \sum_{j=2}^n a_jx_j^* = a_1 + Z = a_0.$$

Thus, the points  $\mathbf{x}^*, \mathbf{x}^1, \dots, \mathbf{x}^{k+1}$  satisfy  $a_1x_1 + \sum_{j=2}^n a_jx_j \leq a_0$  with equality. They are also affinely independent, since  $\mathbf{x}^* \in \mathcal{F}_1$  and  $\mathbf{x}^1, \dots, \mathbf{x}^{k+1} \in \mathcal{F}_0$ . Consequently, the inequality  $a_1x_1 + \sum_{j=2}^n a_jx_j \leq a_0$  defines a face of  $\text{conv}(\mathcal{F})$  of dimension  $k+1$ .

□

Theorem 3.3.1 is meant to be used sequentially. Given  $N_1 \subseteq [n]$  and an inequality  $\sum_{j \in N_1} a_jx_j \leq a_0$  that is valid for  $\mathcal{F} \cap \{\mathbf{x} \in \{0, 1\}^n : x_i = 0, i \notin N_1\}$ , we lift one variable at a time to obtain a valid inequality of the form

$$\sum_{j \notin N_1} \pi_jx_j + \sum_{j \in N_1} a_jx_j \leq a_0.$$

Note that the coefficients  $\pi_j$  depend on the order in which the variables are lifted.

**Example.** Consider the knapsack problem

$$\mathcal{F} = \{\mathbf{x} \in \{0, 1\}^6 : 5x_1 + 5x_2 + 5x_3 + 5x_4 + 3x_5 + 8x_6 \leq 17\}.$$

The inequality  $x_1 + x_2 + x_3 + x_4 \leq 3$  is valid for  $\text{conv}(\mathcal{F} \cap \{\mathbf{x} \in \{0, 1\}^6 : x_5 = x_6 = 0\})$ . Applying lifting first on  $x_5$  and then on  $x_6$  leads to the inequality  $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 \leq 3$ . If we first lift  $x_6$  and then  $x_5$ , we obtain  $x_1 + x_2 + x_3 + x_4 + 2x_6 \leq 3$  instead. □

Clearly, the IP that is to be solved in the lifting phase could be as hard as the original problem. In some cases, however, it is polynomial-time solvable. Even if this IP is hard, we can still use relaxations to obtain lower bounds for  $Z$ . After all, we are still strengthening a known inequality.

### 3.4 Lift-and-Project

In this section, we describe a systematic way to derive an integral formulation for binary optimization problems. The idea of the method is to consider the integer optimization problem in a higher dimensional space (the lifting phase), and then to project back to the original space inequalities found in the higher dimensional space resulting in a tighter formulation (the projection phase).

For a polyhedron  $P = \{(\mathbf{x}, \mathbf{y}) : D\mathbf{x} + B\mathbf{y} \leq \mathbf{d}\}$ , we define the projection of  $P$  onto the set of  $\mathbf{x}$ -variables to be

$$P_{\mathbf{x}} := \{\mathbf{x} : \exists \mathbf{y} \text{ with } (\mathbf{x}, \mathbf{y}) \in P\}.$$

A description of  $P_{\mathbf{x}}$  can be obtained as follows.

**Lemma 3.4.1** *Let  $C = \{\mathbf{u} : \mathbf{u}^T B = \mathbf{0}, \mathbf{u} \geq \mathbf{0}\}$  and  $E$  be the set of extreme rays of  $C$ . Then*

$$P_{\mathbf{x}} = \{\mathbf{x} : (\mathbf{u}^T D)\mathbf{x} \leq \mathbf{u}^T \mathbf{d}, \text{ for all } \mathbf{u} \in E\}. \quad (3.3)$$

**Proof.** Clearly,  $C$  is a polyhedral cone and it suffices to take in (3.3) only vectors  $\mathbf{u}$  that are extreme rays of  $C$ , since  $C = \text{cone}(E)$ . (The “non-extreme ray inequalities” are nonnegative linear combinations of the “extreme ray inequalities”.)

Let  $S = \{\mathbf{x} : (\mathbf{u}^T D)\mathbf{x} \leq \mathbf{u}^T \mathbf{d}, \text{ for all } \mathbf{u} \in E\}$ . If  $\mathbf{x} \in P_{\mathbf{x}}$ , then there exists some  $\mathbf{y}$  such that  $(\mathbf{x}, \mathbf{y}) \in P$ . Hence for all  $\mathbf{u} \in E$ , we have  $(\mathbf{u}^T D)\mathbf{x} \leq \mathbf{u}^T \mathbf{d}$ , that is,  $\mathbf{x} \in S$ . Conversely, if  $\mathbf{x} \in S$ , we have  $(\mathbf{u}^T D)\mathbf{x} \leq \mathbf{u}^T \mathbf{d}$ , for all  $\mathbf{u} \in C$ . By the Farkas Lemma,  $B\mathbf{y} \leq (\mathbf{d} - D\mathbf{x})$  has a solution if and only if for every  $\mathbf{u} \in C$  we have that  $\mathbf{u}^T(\mathbf{d} - D\mathbf{x}) \geq \mathbf{0}$ . This latter condition holds. Thus, there exists some  $\mathbf{y}$  such that  $(\mathbf{x}, \mathbf{y}) \in P$ .  $\square$

**Remark.** Consequently, the problem of obtaining an explicit description of  $P_{\mathbf{x}}$  reduces to the problem of determining the extreme rays of  $C$ .  $\square$

Let

$$\mathcal{F} := \{\mathbf{x} \in \{0, 1\}^n : A\mathbf{x} \leq \mathbf{b}\}$$

with  $A \in \mathbb{Z}^{d \times n}$  and  $\mathbf{b} \in \mathbb{Z}^d$ . Our goal is to derive an explicit description of  $\text{conv}(\mathcal{F})$ . W.l.o.g. we assume that the inequalities  $0 \leq x_i \leq 1$ ,  $i = 1, \dots, n$ , are contained in  $A\mathbf{x} \leq \mathbf{b}$ . Let

$$P = \{\mathbf{x} : A\mathbf{x} \leq \mathbf{b}\}.$$

The lift-and-project method works as follows.

#### Algorithm

**Input:** a matrix  $A \in \mathbb{Z}^{d \times n}$ , a vector  $\mathbf{b} \in \mathbb{Z}^d$ , and an index  $j \in [n]$

**Output:** a polyhedron  $P_j$

- (1) **(Lift)** Multiply  $A\mathbf{x} \leq \mathbf{b}$  by  $x_j$  and by  $(1 - x_j)$ :

$$\begin{aligned} (A\mathbf{x})x_j &\leq \mathbf{b}x_j, \\ (A\mathbf{x})(1 - x_j) &\leq \mathbf{b}(1 - x_j), \end{aligned}$$

and substitute  $y_{ij} := x_i x_j$  for  $i = 1, \dots, n, i \neq j$ , and  $x_j := x_j^2$ . Let  $L_j(P)$  be the resulting polyhedron in  $(\mathbf{x}, \mathbf{y})$ -space.

- (2) **(Project)** Project  $L_j(P)$  back to the original space of the  $\mathbf{x}$ -variables by eliminating all  $\mathbf{y}$ -variables. Let  $P_j = (L_j(P))_{\mathbf{x}}$  be the resulting polyhedron.

We now show that the  $j$ -th component of each vertex of  $P_j$  is either 0 or 1.

**Theorem 3.4.2**

$$P_j = \text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\}).$$

**Proof.** Let  $\bar{\mathbf{x}} \in P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\}$  and  $\bar{y}_{ij} = \bar{x}_i \bar{x}_j$  for  $i \neq j$ . Since  $\bar{x}_j = (\bar{x}_j)^2$  and  $A\bar{\mathbf{x}} \leq \mathbf{b}$ , we have  $(\bar{\mathbf{x}}, \bar{\mathbf{y}}) \in L_j(P)$  and thus  $\bar{\mathbf{x}} \in P_j$ . Consequently,

$$\text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\}) \subseteq P_j.$$

If  $P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j = 0\} = \emptyset$ , then from the Farkas-Lemma there exists  $\mathbf{u} \geq \mathbf{0}$  such that  $\mathbf{u}^\top A = -\mathbf{e}_j$  and  $\mathbf{u}^\top \mathbf{b} = -\epsilon$  with  $\epsilon > 0$ . Thus, for all  $\mathbf{x}$  satisfying

$$\begin{aligned} (A\mathbf{x})x_j &\leq \mathbf{b}x_j, \\ (A\mathbf{x})(1 - x_j) &\leq \mathbf{b}(1 - x_j), \end{aligned}$$

we have

$$\mathbf{u}^\top A\mathbf{x}(1 - x_j) \leq \mathbf{u}^\top \mathbf{b}(1 - x_j).$$

Hence, for all  $\mathbf{x} \in P_j$ , we get

$$-\mathbf{e}_j^\top \mathbf{x}(1 - x_j) = -x_j(1 - x_j) \leq -\epsilon(1 - x_j).$$

Replacing  $x_j^2$  by  $x_j$ , we see that  $-x_j(1 - x_j) = -x_j + x_j^2 = -x_j + x_j = 0$  and thus  $x_j \geq 1$  is valid for  $P_j$ . Since in addition  $P_j \subseteq P$  (Exercise: Why does this inclusion hold?), we obtain

$$P_j \subseteq P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j = 1\} = \text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\}).$$

Similarly we show if  $P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j = 1\} = \emptyset$ , then

$$P_j \subseteq \text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\}).$$

Suppose now that both  $P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j = 0\}$  and  $P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j = 1\}$  are not empty. To show that  $P_j \subseteq \text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\})$ , we prove that all inequalities valid for  $\text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\})$  are also valid for  $P_j$ .

Let  $\mathbf{a}^\top \mathbf{x} \leq \alpha$  be valid for  $\text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\})$  and let  $\mathbf{x} \in P$ .

If  $x_j = 0$ , then for all  $\lambda \in \mathbb{R}$ , we have

$$\mathbf{a}^\top \mathbf{x} + \lambda x_j = \mathbf{a}^\top \mathbf{x} \leq \alpha,$$

since  $\mathbf{a}^\top \mathbf{x} \leq \alpha$  is valid for  $P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j = 0\}$ .

If  $x_j > 0$ , then there exists  $\lambda \leq 0$ , such that for all  $\mathbf{x} \in P$  we have

$$\mathbf{a}^\top \mathbf{x} + \lambda x_j \leq \alpha.$$

Analogously, since  $\mathbf{a}^\top \mathbf{x} \leq \alpha$  is valid for  $P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j = 1\}$ , there exists some  $\nu \leq 0$ , such that for all  $\mathbf{x} \in P$ , we have

$$\mathbf{a}^\top \mathbf{x} + \nu(1 - x_j) \leq \alpha.$$

Thus, for all  $\mathbf{x}$  satisfying

$$\begin{aligned} (A\mathbf{x})x_j &\leq \mathbf{b}x_j, \\ (A\mathbf{x})(1 - x_j) &\leq \mathbf{b}(1 - x_j), \end{aligned}$$

we have

$$\begin{aligned} (1 - x_j)(\mathbf{a}^\top \mathbf{x} + \lambda x_j) &\leq (1 - x_j)\alpha, \\ x_j(\mathbf{a}^\top \mathbf{x} + \nu(1 - x_j)) &\leq x_j\alpha, \end{aligned}$$

and thus, by adding these two inequalities together:

$$\mathbf{a}^\top \mathbf{x} + (\lambda + \nu)(x_j - x_j^2) \leq \alpha.$$

After setting  $x_j^2 = x_j$ , we obtain that for all  $\mathbf{x} \in P_j$  the inequality  $\mathbf{a}^\top \mathbf{x} \leq \alpha$  holds. Consequently,

$$P_j \subseteq \text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\}\}).$$

□

**Example.** Let

$$P = \{(x_1, x_2)^\top : 2x_1 - x_2 \geq 0, 2x_1 + x_2 \leq 2, 0 \leq x_1 \leq 1, 0 \leq x_2 \leq 1\}.$$

Choose  $j = 1$  and apply the algorithm above to obtain  $P_1$ :

$$\begin{aligned} 2x_1^2 - x_1x_2 &\geq 0 \\ 2x_1(1 - x_1) - x_2(1 - x_1) &\geq 0 \\ 2x_1^2 + x_1x_2 &\leq 2x_1 \\ 2x_1(1 - x_1) + x_2(1 - x_1) &\leq 2(1 - x_1) \\ 0 \leq x_1^2 &\leq x_1 \\ 0 \leq x_1(1 - x_1) &\leq 1 - x_1 \\ 0 \leq x_1x_2 &\leq x_1 \\ 0 \leq x_2(1 - x_1) &\leq 1 - x_1 \end{aligned}$$

Setting  $y = x_1x_2$  and  $x_1 = x_1^2$ , we get:

$$\begin{aligned}
2x_1 - y &\geq 0 \\
-x_2 + y &\geq 0 \\
y &\leq 0 \\
x_2 - y &\leq 2 - 2x_1 \\
0 \leq x_1 &\leq x_1 \\
0 \leq 0 &\leq 1 - x_1 \\
0 \leq y &\leq x_1 \\
0 \leq x_2 - y &\leq 1 - x_1
\end{aligned}$$

This implies that  $y = 0$ , which leads to the inequalities:

$$\begin{aligned}
x_1 &\geq 0 \\
-x_2 &\geq 0 \\
x_2 &\leq 2 - 2x_1 \\
0 \leq x_1 &\leq 1 \\
0 \leq x_2 &\leq 1 - x_1
\end{aligned}$$

Thus, we get

$$P_1 = \{(x_1, x_2)^\top : 0 \leq x_1 \leq 1, x_2 = 0\} = \text{conv}(P \cap \{(x_1, x_2)^\top : x_1 \in \{0, 1\}\}).$$

□

The algorithm above can be applied sequentially for every variable  $x_j$ . If we apply it to  $x_{i_1}, \dots, x_{i_t}$  (in this order), we obtain a polyhedron

$$P_{i_1, \dots, i_t} = ((P_{i_1})_{i_2} \dots)_{i_t}.$$

**Theorem 3.4.3** *The polyhedron  $P_{i_1, \dots, i_t}$  satisfies:*

$$P_{i_1, \dots, i_t} = \text{conv}(P \cap \{\mathbf{x} \in \mathbb{R}^n : x_j \in \{0, 1\} : j \in \{i_1, \dots, i_t\}\}).$$

**Proof.** We prove this statement by induction on  $t$ . For  $t = 1$ , the statement follows from Theorem 3.4.2. Let  $t \geq 2$  and assume that the statement is true for any sequence of size less than  $t$ . Consider some sequence  $i_1, \dots, i_t$  of size  $t$  and let

$$Q = \{\mathbf{x} \in \mathbb{R}^n : x_{i_q} \in \{0, 1\} : q = 1, \dots, t-1\}.$$

We have

$$\begin{aligned}
P_{i_1, \dots, i_t} &= (P_{i_1, \dots, i_{t-1}})_{i_t} \\
&= \text{conv}(P \cap Q)_{i_t} && \text{by induction hypothesis} \\
&= \text{conv}(\text{conv}(P \cap Q) \cap \{\mathbf{x} : x_{i_t} \in \{0, 1\}\}) && \text{by induction hypothesis for } k = 1 \\
&= \text{conv}(\text{conv}(P \cap Q) \cap \{\mathbf{x} : x_{i_t} = 0\} \cup \text{conv}(P \cap Q) \cap \{\mathbf{x} : x_{i_t} = 1\}) \\
&= \text{conv}(\text{conv}(P \cap Q \cap \{\mathbf{x} : x_{i_t} = 0\}) \cup \text{conv}(P \cap Q \cap \{\mathbf{x} : x_{i_t} = 1\})) \\
&= \text{conv}((P \cap Q \cap \{\mathbf{x} : x_{i_t} = 0\}) \cup (P \cap Q \cap \{\mathbf{x} : x_{i_t} = 1\})) \\
&= \text{conv}((P \cap \{\mathbf{x} : x_{i_q} \in \{0, 1\}, q = 1 \dots, t\})),
\end{aligned}$$

where the last equation holds because

$$\text{conv}(\text{conv}(S) \cup \text{conv}(T)) = \text{conv}(S \cup T)$$

is true for arbitrary sets  $S, T \in \mathbb{R}^n$ . □

This theorem implies that if we apply the lift-and-project algorithm above to all  $n$  variables, we obtain the convex hull of solutions  $\text{conv}(\mathcal{F})$ , that is,  $P_{1, \dots, n} = \text{conv}(\mathcal{F})$ . Moreover, this theorem implies that the resulting polyhedron does not depend on the order of the variables that are used. Thus, we may write  $P_{\{i_1, \dots, i_t\}}$  instead of  $P_{i_1, \dots, i_t}$ . Finally, it should be noted that the method of lift and project only applies to binary optimization problems.

# Chapter 4

## Ellipsoid Method and Separation

### 4.1 Ellipsoid Method

**Notation.** For a positive definite matrix  $D$  let

$$\text{Ell}(D, \mathbf{z}) = \{\mathbf{x} : (\mathbf{x} - \mathbf{z})^\top D(\mathbf{x} - \mathbf{z}) \leq 1\}$$

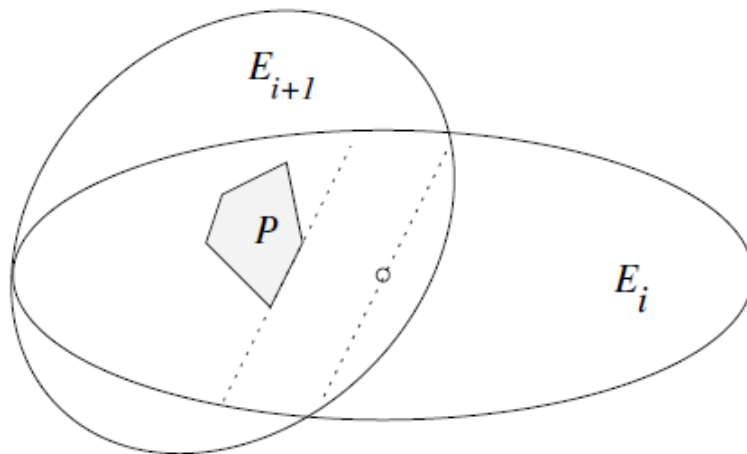
be the ellipsoid described by  $D$ .

**Problem.** Given a polyhedron  $P = \{\mathbf{x} : A\mathbf{x} \leq \mathbf{b}\}$ , find a point in  $P$ .

This problem can be solved via the following algorithm:

**The Ellipsoid Algorithm.**

1. Let  $\gamma$  be the maximum number of bits required to describe a vertex of  $P$ .
2. Set  $r = 2^\gamma$ .
3. Start with the ellipsoid  $E_0 = \text{Ell}(r^2 I_n, \mathbf{0})$ .
4. At the  $i$ -th iteration, check whether  $\mathbf{z}_i$  lies in  $P$ .
  - Yes: Output  $\mathbf{z}_i$  as feasible solution.
  - No: Find a constraint  $\mathbf{a}_k^\top \mathbf{x} \leq b_k$  valid for  $P$ , which is violated by  $\mathbf{z}_i$ . Let  $E_{i+1}$  be the minimum volume ellipsoid containing  $E_i \cap \{\mathbf{x} : \mathbf{a}_k^\top \mathbf{x} \leq \mathbf{a}_k^\top \mathbf{z}_i\}$ , and recurse with  $E_{i+1}$ .



The algorithm stops, when a point in  $P$  has been found. It has to stop, since  $P$  is contained in  $E_i$  and in each iteration the volume of  $E_{i+1}$  has dropped by a certain factor. For some value  $i$ , the volume of  $E_i$  will be smaller than the volume of  $P$ . Clearly, the algorithm must halt before this situation is reached. The ellipsoid method can be implemented in polynomial time!

**Lemma 4.1.1** *The minimum volume ellipsoid containing  $\text{Ell}(D, \mathbf{z}) \cap \{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \leq \mathbf{a}^\top \mathbf{z}\}$  is exactly  $E' = \text{Ell}(D', \mathbf{z}')$ , where*

$$\mathbf{z}' = \mathbf{z} - \frac{1}{n+1} \cdot \frac{D\mathbf{a}}{\sqrt{\mathbf{a}^\top D\mathbf{a}}}$$

and

$$D' = \frac{n^2}{n^2 - 1} \left( D - \frac{2}{n+1} \cdot \frac{D\mathbf{a}\mathbf{a}^\top D}{\mathbf{a}^\top D\mathbf{a}} \right)$$

and

$$\frac{\text{vol}(E')}{\text{vol}(E)} \leq e^{-\frac{1}{2n+2}}.$$

## 4.2 Separation

A separation procedure is a procedure that decides for a given point  $\mathbf{z}$  either that it lies in  $P$  or it finds a valid inequality  $\mathbf{a}^\top \mathbf{x} \leq b$  for  $P$  that is violated for  $\mathbf{z}$ .

Polynomiality of the ellipsoid method means that we can even solve integer programs in polynomial time if we can separate over the facet defining inequalities of  $P_I$ . Even if we cannot separate over the full facet-description of  $P_I$ , it may help a lot in cutting-plane methods if we can at least separate over exponentially many facets of  $P_I$ .

**Example.** Let us consider the maximum stable set problem in a graph  $G = (V, E)$ . This problem can be modeled (and solved) via the following IP:

$$\max \left\{ \sum_i x_i : x_i + x_j \leq 1 \forall \{i, j\} \in E, x_i \in \{0, 1\} \forall i \right\}$$

The LP relaxation

$$\max \left\{ \sum_i x_i : x_i + x_j \leq 1 \ \forall \{i, j\} \in E, 0 \leq x_i \leq 1 \ \forall i \right\}$$

can be pretty bad. For example, for the complete graph, the optimal IP solution has value 1, while the LP relaxation has optimal value  $n/2$  with optimal solution  $x_1 = \dots = x_n = 1/2$ .

Thus, let us add the odd-cycle inequalities to the problem. The new LP is as follows:

$$\max \left\{ \sum_i x_i : x_i + x_j \leq 1 \ \forall \{i, j\} \in E, \sum_{i \in C} x_i \leq \frac{|C| - 1}{2} \ \forall \text{ odd cycles } C, 0 \leq x_i \leq 1 \ \forall i \right\}$$

Note that the optimal values of this LP and of the IP can still be different.

Let us find a separation oracle for these constraints. As the inequalities  $x_i + x_j \leq 1 \ \forall \{i, j\} \in E$  and  $0 \leq x_i \leq 1$  can easily be checked, we only need to find a separation oracle for the odd-cycle inequalities. For this, define  $y_{ij} = 1 - x_i - x_j$  for each edge  $\{i, j\} \in E$ . With this notation, the odd-cycle inequalities are equivalent to

$$\sum_{\{i, j\} \in E(C)} y_{ij} \geq 1.$$

Therefore, in order to find a violated odd-cycle inequality, we need to find an odd cycle in  $G$  with minimum weight  $\sum_{\{i, j\} \in E(C)} y_{ij}$ . If  $\sum_{\{i, j\} \in E(C)} y_{ij} < 1$ , we have found a violated odd-cycle inequality.

If  $\sum_{\{i, j\} \in E(C)} y_{ij} \geq 1$ , none of the odd-cycle inequalities is violated.

Next we show that such a minimum-weight odd cycle in  $G$  can be found in polynomial time. For this, we construct a graph  $G' = (V_1 \cup V_2, E')$  such that there is a node  $i_1 \in V_1$  and a node  $i_2 \in V_2$  for every node  $i \in V$ . For each edge  $\{i, j\} \in E$ , we add two edges  $\{i_1, j_2\}$  and  $\{j_1, i_2\}$  to  $E'$ . Odd cycles in  $G$  now correspond to paths from  $i_1$  to  $i_2$  in  $E'$ . Thus, finding a minimum-weight odd cycle in  $G$  is equivalent to finding shortest paths from  $i_1$  to  $i_2$  in  $E'$ .

Thus, we have constructed a polynomial-time separation procedure.  $\square$



## Chapter 5

# Branching strategies

An important component of branch-and-bound (or branch-and-cut) algorithms is the branching step. The chosen strategy often heavily affects the total running time. The branching strategy includes two types of decisions:

1. How do we decompose a problem into subproblems?
2. Which open subproblem is considered next?

Goals of a branching strategy:

1. strong improvement of dual bound (for a better quality certificate)
2. quick construction of good feasible (primal) solutions (early pruning of branches)

The quality of a chosen strategy can only be checked experimentally and depends heavily on the structure of the problem instances.

In the following, we always consider minimization problems. Let  $p$  be a (sub)problem.

- $\bar{x}_i(p)$  = last LP-value of  $x_i$  in  $p$
- $f_i^-(p) = \bar{x}_i(p) - \lfloor \bar{x}_i(p) \rfloor$ ,  $f_i^+(p) = \lceil \bar{x}_i(p) \rceil - \bar{x}_i(p)$
- $f_i(p) = \min \{f_i^-(p), f_i^+(p)\}$
- $z(p)$  = dual bound in  $p$  (last LP-optimum)

## 5.1 Selection of a node

**Problem:** Given a set of open subproblems, which subproblem should be considered next?

### Breadth-first search

Choose node in branching tree with smallest depth in the tree.

**Advantage:** None.

### Depth-first search

Choose node with smallest depth in the branching tree.

**Advantage:** Finds primal solutions quickly, as many things get fixed.

**Disadvantage:** Dual bound is nearly left unchanged.

### Best-first search

Choose node  $p$  with worst (= smallest) dual bound  $z(p)$ .

**Advantage:** Dual bound potentially changes a lot.

**Disadvantage:** Rarely finds a feasible primal solutions.

### Best-projection search

**Idea:** Search for good primal solutions. One guesses that these solutions can be found in those subproblems, for which the dual bound remains bad.

Choose node  $p$  with minimal value

$$z(p) + \frac{\bar{z} - z(\text{root})}{s(\text{root})} s(p),$$

where  $s(p) = \sum_i f_i(p)$  and where  $\bar{z}$  denotes the currently best feasible primal solution. Moreover,  $\frac{\bar{z} - z(\text{root})}{s(\text{root})}$  is a measure of the expected improvement of the bound for decreasing fractionality.

## Final conclusions about selection of next node

- There are many more strategies.
- Experiments show:
  - Depth-first search is best to find good feasible primal solutions.
  - Best-first and Best-projection yield the best dual bounds (depending on the type of instances).

## 5.2 Branching rules

**Problem:** Given a subproblem  $p$ , divide it into a several new subproblems, where the set of feasible solutions of  $p$  is the union of the feasible solutions of the new subproblems.

There are many possibilities:

- divide into two or more subproblems
- divide with respect to variables or linear constraints
- 
- 
- 

Let us concentrate here on the simplest way of branching: **Branching along variables**

**Idea.** Choose an integer variable  $x_i$  with fractional  $\bar{x}_i(p)$  and generate two new subproblems by adding the constraints

$$x_i \leq \lfloor \bar{x}_i(p) \rfloor \quad \text{or} \quad x_i \geq \lceil \bar{x}_i(p) \rceil$$

In case of binary variables:

$$x_i = 0 \quad \text{or} \quad x_i = 1$$

**Problem:** Which variable should be chosen for branching? → again many strategies

### Most infeasible

Choose variable with maximal  $f_i(p)$  → the least determined variable

**Goal:** Fixing has hopefully a large impact.

**Experiments:** Not much better than random choice of branching variable.

## Pseudo-cost branching

Let  $p_i^-$  and  $p_i^+$  be the subproblems obtained from  $p$  by rounding  $x_i$  down and up, respectively.

Let  $\Delta_i^+(p) := z(p_i^+) - z(p)$ ,  $\Delta_i^-(p) := z(p_i^-) - z(p)$ . Then  $\Delta_i^-(p)/f_i(p)$  measures the gain by rounding  $x_i$  down in  $p$ .

**Downwards-pseudo-costs:**  $\text{pk}^-(x_i) = \text{average of } \Delta_i^-(p)/f_i(p) \text{ over all } p, \text{ in which } p_i^- \text{ has been considered. (If there is none, } \text{pk}^-(x_i) \text{ is undefined/uninitialized.)}$

**Upwards-pseudo-costs:** Analogously.

**Pseudo-costs:**  $\text{pk}(x_i) = \text{score}(\text{pk}^-(x_i), \text{pk}^+(x_i))$ , where

$$\text{score}(a, b) = (1 - \mu) \min(a, b) + \mu \max(a, b), \quad \mu \in [0, 1].$$

Typically,  $\mu = \frac{1}{6}$  is chosen.

Choose fractional variable  $x_i$  with largest value  $\text{pk}(x_i)$  for branching.

**Advantage:** good branching decisions, fast computation

**Disadvantage:** initially pseudo-costs are uninitialized or contain nearly no information

## Strong branching

**Idea:** Look ahead!

Bound  $\Delta_i^-(p)$  and  $\Delta_i^+(p)$  by temporarily adding  $x_i \leq \lfloor \bar{x}_i(p) \rfloor$  and  $x_i \geq \lceil \bar{x}_i(p) \rceil$ , respectively, and by solving both LPs.

Choose variable with largest value of

$$\text{score} \left( \frac{\Delta_i^+(p)}{f_i^+(p)}, \frac{\Delta_i^-(p)}{f_i^-(p)} \right).$$

**Advantage:** good estimation of success of branching using  $x_i$

**Disadvantage:** Computation takes a lot of time! Two LPs per candidate variable!

**In praxis:**

- Test only a selection of variables.
- Perform only a few simplex iterations.
- Still very expensive!

## Mixed approaches

**Goal:** Combine advantages of pseudo-cost and strong branching. First perform strong branching, then use pseudo-cost branching when the information on former branchings become more reliable.

There are a variety of more or less complex approaches:

1. Up to a predetermined depth in the search tree, use strong branching and otherwise pseudo-cost branching.
2. If pseudo-costs (in one direction) are uninitialized, replace them by estimations from strong branching.
3. If pseudo-costs (in one direction) rely on less than  $\nu$  previous branchings, use strong branching. This strategy is called **Reliability branching** and works pretty well in praxis.

$\nu = 0$     pseudo-cost branching

$\nu = \infty$     strong branching



## Chapter 6

# Column generation

Assume we wish to solve an LP

$$\min\{\mathbf{c}^\top \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}\},$$

where  $\mathbf{x} \in \mathbb{R}^n$  with  $n$  very large. Then checking the reduced costs in each iteration of the simplex method is very expensive. The idea of “column generation” is to

- start with a small subset of variables,
- solve the small LP,
- generate one or more missing columns with negative reduced costs,
- add columns to master problem and iterate.

Let us have a look at an example.

### 6.1 The Cutting Stock Problem (CSP)

Consider a paper company that has a supply of large rolls of paper, of width  $W$ . (We assume that  $W$  is a positive integer.) However, customer demand is for smaller widths of paper; in particular  $d_i$  rolls of width  $w_i$ ,  $i = 1, 2, \dots, n$ , need to be produced. We assume that  $w_i \leq W$  for each  $i$ , and that each  $w_i$  is an integer. Smaller rolls are obtained by slicing a large roll in a certain way, called a *cutting pattern*. For example, a large roll of width 70 can be cut into three rolls of width  $w_1 = 17$  and one roll of width  $w_2 = 15$ , with a waste of 4. The goal of the company is to minimize the number of large rolls used while satisfying customer demand.

- CSP is NP-hard.
- if all  $d_i = 1$ : binary CSP

### 6.1.1 A first IP formulation

In general, a cutting pattern is a vector  $\mathbf{q} \in \mathbb{Z}_+^n$ , which encodes the cutting of a large roll of paper into  $q_1$  rolls of width  $w_1$ ,  $q_2$  rolls of width  $w_2$ ,  $\dots$ ,  $q_n$  rolls of width  $w_n$ , plus potentially some waste. Clearly, every feasible cutting pattern satisfies  $\sum_{i=1}^n q_i w_i \leq W$ . Moreover, there is a trivial upper bound for the number of large rolls needed:

$$K := \sum_{i=1}^n d_i.$$

(Simply cut each small roll out of a separate large roll.)

Thus, define binary variables  $y_1, \dots, y_K$  with  $y_k = 1$  if and only if roll  $k$  is used. Hence, we wish to minimize  $\sum_{k=1}^K y_k$ .

To each roll  $k$ , we assign a valid cutting pattern  $(x_1^k, \dots, x_n^k)^\top$ . Validity is encoded in

$$\sum_{i=1}^n w_i x_i^k \leq W y_k.$$

As we also need to satisfy demand, we get the constraints

$$\sum_{k=1}^K x_i^k \geq d_i, \quad \forall i = 1, \dots, n.$$

Summing up, we get:

$$z_{IP_1} = \min \left\{ \begin{array}{ll} \sum_{k=1}^K y_k : & \sum_{k=1}^K x_i^k \geq d_i, \quad \forall i = 1, \dots, n, \\ & \sum_{i=1}^n w_i x_i^k \leq W y_k, \quad \forall k = 1, \dots, K, \\ & y_k \in \{0, 1\}, \quad \forall k = 1, \dots, K, \\ & x_i^k \in \mathbb{Z}_+, \quad \forall k = 1, \dots, K, i = 1, \dots, n \end{array} \right\}$$

This formulation contains only polynomially many rows and columns, which is “good” for Branch-and-Cut. HOWEVER:

**Theorem 6.1.1** *The optimal value of the LP relaxation of  $z_{IP_1}$  is  $\frac{\sum_{i=1}^n w_i d_i}{W}$ .*

**Proof.** Exercise! □

This bound is typically pretty bad:

**Example.** Consider the problem instance with  $n = 1$  and  $w_1 = \lfloor \frac{W}{2} \rfloor + 1$ . Then we need  $z_{IP_1} = d_1$  large rolls. However, the optimal LP solution has value

$$\frac{d_1 w_1}{W} = \frac{d_1 (\lfloor \frac{W}{2} \rfloor + 1)}{W} \approx \frac{d_1}{2},$$

for large  $d_1$ .

Consequently:

- LP-solution gives a bad bound
- in a branch-and-bound tree only a few subproblems can be examined in detail
- typically one has to do a complete enumeration of all solutions

Moreover, the large symmetry in the problem causes even further problems for a branch-and-bound approach.

### 6.1.2 Second IP formulation

Let  $Q$  be the set of valid cutting patterns  $\mathbf{x}^j$ . For  $j = 1, \dots, |Q|$  introduce variables  $\mu^j \in \mathbb{Z}_+$  that count how often cutting pattern  $\mathbf{x}^j \in Q$  is used.

**Goal:** Minimize used rolls:  $\min \sum_{j=1}^{|Q|} \mu_j$ .

**Master problem:**

$$(\text{IP}_2) : \quad \begin{aligned} \sum_{j=1}^{|Q|} x_i^j \mu^j &\geq d_i, \forall i = 1, \dots, n, \\ \mu^j &\in \mathbb{Z}_+ \end{aligned}$$

**Subproblem:** The  $\mathbf{x}^j$  are feasible solutions of

$$\begin{aligned} \sum_{i=1}^n w_i x_i^j &\leq W, \\ \mathbf{x}^j &\in \mathbb{Z}_+^n. \end{aligned}$$

**Problem:** exponentially many ( $= |Q|$ ) variables

**Solution:** column generation/pricing

- general principle for the solution of an LP
- particularly useful for LPs with exponentially many variables

## 6.2 Column generation

Assume we have to solve a (feasible) LP:

$$(\text{LP}) : \quad \max\{\mathbf{c}^\top \mathbf{x} : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$$

Let  $A'$  consist of a subset of  $n'$  columns of  $A$ , but with the same set of rows as  $A$ . W.l.o.g.,  $A'$  consists of the first  $n'$  columns. We consider

$$(\text{LP}') : \quad \max\{\mathbf{c}'^\top \mathbf{x}' : A'\mathbf{x}' \leq \mathbf{b}, \mathbf{x}' \geq \mathbf{0}\},$$

where we assume that  $A'$  is chosen in such a way that  $(LP')$  is feasible.

Let  $\mathbf{x}^{*'}$  be an optimal solution of  $(LP')$  with basis  $(A'_B, A'_N)$ . Then  $(\mathbf{x}^{*'}_B, \mathbf{0})$  is feasible for  $(LP)$  and defines a feasible basis  $(A'_B, A_N)$  (possibly not optimal) for  $(LP)$ .

A column in  $A_N$  with positive reduced costs is a candidate to enter the basis for  $(LP)$ . Such a column is found by solving the so-called *pricing problem*.

**Reduced costs:**  $\bar{\mathbf{c}} = \mathbf{c}_N - \mathbf{c}_B A'_B{}^{-1} A_N$

Let  $\mathbf{y}_B := \mathbf{c}_B A'_B{}^{-1}$  and consider the problem

$$(\text{Pri}) : \quad \max\{c_a - \mathbf{y}_B^\top \mathbf{a} : \mathbf{a} \text{ is column of } A\}.$$

The optimal solution of  $(\text{Pri})$  gives a column of  $A_N$  with largest reduced costs  $\bar{c}_a$  (without necessarily going through all columns of  $A_N$  one by one).

If  $\bar{c}_a \leq 0$ , we have solved  $(LP)$  optimally. Otherwise, we add  $\mathbf{a}$  as a column to  $A'$  and iterate until an optimal solution has been found.

### 6.3 Column generation for CSP

Start with  $(IP_2)$  with a subset of all columns (cutting patterns). Via pricing, we generate additional columns if necessary. The pricing problem searches for the cutting pattern with *smallest* reduced costs. (CSP is a *minimization* problem!)

$$(\text{Pri}) : \quad \min \left\{ 1 - \mathbf{y}^\top \mathbf{x}^j : \sum_{i=1}^n w_i x_i^j \leq W, \mathbf{x}^j \in \mathbb{Z}_+^n \right\}.$$

Herein,  $\mathbf{y} = \mathbf{c}_B A'_B{}^{-1}$ , where  $A'_B$  is an optimal basis of the current reduced master problem.

The pricing problem for CSP is simply a knapsack problem! This is also NP-hard, but can be solved exactly for pretty large  $n$ . (Remember that  $n$  is much smaller than  $|Q|$ .)

**Remark.** Pricing can also be used within a Branch-and-Cut framework.  $\rightarrow$  Branch-Cut&Price

# Chapter 7

## Benders decomposition

### 7.1 Formal derivation

Consider the following problem:

$$\min \{ \mathbf{c}^\top \mathbf{x} + \mathbf{f}^\top \mathbf{y} : A\mathbf{x} + B\mathbf{y} = \mathbf{b}, \mathbf{x} \geq \mathbf{0}, \mathbf{y} \in Y \}, \quad (7.1)$$

where  $\mathbf{x}$  and  $\mathbf{y}$  are vectors of continuous variables having dimensions  $p$  and  $q$ , respectively,  $Y$  is a polyhedron,  $A, B$  are matrices, and  $\mathbf{b}, \mathbf{c}, \mathbf{f}$  are vectors having appropriate dimensions. Suppose that the  $\mathbf{y}$ -variables are “*complicating variables*” in the sense that the problem becomes significantly easier to solve if the  $\mathbf{y}$ -variables are fixed, e.g., due to a special structure inherent in the matrix  $A$ .

Benders decomposition partitions problem (7.1) into two problems:

- a master problem that contains the  $\mathbf{y}$ -variables, and
- a subproblem that contains the  $\mathbf{x}$ -variables.

Note that problem (7.1) can be rewritten as

$$\min \{ \mathbf{f}^\top \mathbf{y} + q(\mathbf{y}) : \mathbf{y} \in Y \}, \quad (7.2)$$

where  $q(\mathbf{y})$  is defined to be the optimal value of

$$\min \{ \mathbf{c}^\top \mathbf{x} : A\mathbf{x} = \mathbf{b} - B\mathbf{y}, \mathbf{x} \geq \mathbf{0} \}. \quad (7.3)$$

#### Observations:

- Problem (7.3) is a linear program for any given value  $\mathbf{y} \in Y$ .
- If (7.3) is unbounded for some  $\mathbf{y} \in Y$ , then also problem (7.2) is unbounded, and thus, also our original problem (7.1).

- Assuming boundedness of (7.3), we can calculate  $q(\mathbf{y})$  by solving the dual problem of (7.3).

Let us associate dual variables  $\boldsymbol{\alpha}$  for the constraints  $A\mathbf{x} = \mathbf{b} - B\mathbf{y}$ , then the dual problem of (7.3) is

$$\max \{ \boldsymbol{\alpha}^\top (\mathbf{b} - B\mathbf{y}) : A^\top \boldsymbol{\alpha} \leq \mathbf{c} \}. \quad (7.4)$$

**Key observation:** The feasible region of the dual formulation does not depend on the value of  $\mathbf{y}$ , which only affects the objective function.

If the feasible region of the dual problem (7.4) is empty, then either

- the primal problem (7.3) is unbounded for some  $\mathbf{y} \in Y$  (and thus also the original problem (7.1) is unbounded), or
- the feasible region of the primal problem (7.3) is also empty for *all*  $\mathbf{y} \in Y$  (and thus also the original problem (7.1) is infeasible).

Thus, assume that the feasible region of the dual problem (7.4) is not empty. Enumerate

- all extreme points  $\boldsymbol{\alpha}_p^1, \dots, \boldsymbol{\alpha}_p^I$  and
- all extreme rays  $\boldsymbol{\alpha}_r^1, \dots, \boldsymbol{\alpha}_r^J$ .

Then, for a given vector  $\bar{\mathbf{y}}$ , the dual problem can be solved by

- checking whether  $(\boldsymbol{\alpha}_r^j)^\top (\mathbf{b} - B\bar{\mathbf{y}}) > 0$  for an extreme ray  $\boldsymbol{\alpha}_r^j$ , in which case the dual formulation is unbounded and the primal formulation is infeasible, and by
- finding an extreme point  $\boldsymbol{\alpha}_p^i$  that maximizes the value of the objective function  $(\boldsymbol{\alpha}_p^i)^\top (\mathbf{b} - B\bar{\mathbf{y}})$ , in which case both primal and dual formulations have finite optimal solutions.

Based on this, the dual problem (7.4) can be reformulated as follows:

$$\min \left\{ q : \begin{array}{ll} (\boldsymbol{\alpha}_r^j)^\top (\mathbf{b} - B\bar{\mathbf{y}}) \leq 0 & \forall j \in 1, \dots, J \\ (\boldsymbol{\alpha}_p^i)^\top (\mathbf{b} - B\bar{\mathbf{y}}) \leq q & \forall i \in 1, \dots, I \\ q \in \mathbb{R} \end{array} \right\}.$$

We can use this to replace  $q(\mathbf{y})$  in (7.2) and obtain a reformulation of the original problem (7.1):

$$\min \left\{ \mathbf{f}^\top \mathbf{y} + q : \begin{array}{ll} (\boldsymbol{\alpha}_r^j)^\top (\mathbf{b} - B\bar{\mathbf{y}}) \leq 0 & \forall j \in 1, \dots, J \\ (\boldsymbol{\alpha}_p^i)^\top (\mathbf{b} - B\bar{\mathbf{y}}) \leq q & \forall i \in 1, \dots, I \\ y \in Y \\ q \in \mathbb{R} \end{array} \right\}. \quad (7.5)$$

## 7.2 Solution via Benders decomposition

- Since there is typically an exponential number of extreme points and extreme rays of the dual formulation (7.4), generating all constraints of (7.5) is not practical.
- Instead, Benders decomposition starts with a subset of these constraints, and solves a *relaxed master problem*, which yields a candidate optimal solution  $(\mathbf{y}^*, q^*)$ .
- It then solves the dual subproblem (7.4) to calculate  $q(\mathbf{y}^*)$ .
- If the subproblem has an optimal solution having  $q(\mathbf{y}^*) = q^*$ , then the algorithm stops.
- Otherwise, if the dual subproblem is unbounded, then a constraint  $(\boldsymbol{\alpha}_r^j)^\top(\mathbf{b} - B\bar{\mathbf{y}}) \leq 0$  is generated and added to the relaxed master problem, which is then re-solved.  
→ **Benders feasibility cuts** (they enforce necessary conditions for feasibility of the primal subproblem (7.3))
- Similarly, if the subproblem has an optimal solution having  $q(\mathbf{y}^*) > q^*$ , then a constraint  $(\boldsymbol{\alpha}_p^i)^\top(\mathbf{b} - B\bar{\mathbf{y}}) \leq q$  is added to the relaxed master problem, and the relaxed master problem is re-solved.  
→ **Benders optimality cuts** (they are based on optimality conditions of the subproblem)
- Since  $I$  and  $J$  are finite, and new feasibility or optimality cuts are generated in each iteration, this method converges to an optimal solution in a finite number of iterations.

### Remarks.

- The master problem need not be a linear program. It could also be an integer linear program, a nonlinear program or a constrained linear program.
- The subproblem need not be a linear program, either. It could be a convex program, since dual multipliers satisfying strong duality conditions can be calculated for such problems.

## 7.3 Stochastic Programming

Let us now consider a problem type where Benders decomposition can be applied. In 2-stage stochastic programming problems, some action needs to be taken in a first stage, which is followed by the occurrence of an uncertain event. After the realization of the uncertain event has been observed, a second-stage decision, a so-called **recourse decision**, can be made. For example, one has to decide which way to drive from  $A$  to  $B$  without knowing the current and future traffic (and the possible delay due to heavy traffic).

Clearly, if we do not know anything about the uncertain event, it is impossible to make a good first-stage decision. (For example: How would you define “good” in the first place?) So, we assume that

we know a probability distribution for this uncertain event. Such knowledge can often be generated from data from the past. Or it has to be guessed/approximated by experts. Then we could ask to minimize the costs for the first-stage decision  $\mathbf{x}$  plus the expected costs for the second-stage recourse decision (**expectation model**). Nowadays, one even models risk, that is, one wishes to minimize first-stage costs plus expected costs for the recourse decision plus the risk that the overall costs exceed a certain threshold.

The expectation model for the 2-stage stochastic programming problem can be written as follows:

$$\min\{\mathbf{c}^\top \mathbf{x} + Q(\mathbf{x}) : \mathbf{x} \in X\},$$

where

$$Q(\mathbf{x}) = \mathbb{E}_\omega (\min\{\mathbf{d}_\omega^\top \mathbf{y} : W\mathbf{y} \geq \mathbf{b}_\omega - T\mathbf{x}, \mathbf{y} \in \mathbb{R}_+^m\}).$$

Theoretically, the expected value in  $Q(\mathbf{x})$  can be computed via symbolic integration. However, this becomes practically intractable already for very few variables. Hence, one typically approximates the probability distribution via  $N$  scenarios (Problem: How should we should them well?) which simplifies the integrals to sums. We obtain the (integer) linear programming equivalent to our stochastic program:

$$\min \left\{ \mathbf{c}^\top \mathbf{x} + \sum_{i=1}^N p^i \mathbf{d}^{i\top} \mathbf{y}^i : \begin{pmatrix} T & W & & \\ \vdots & & \ddots & \\ T & & & W \end{pmatrix} \begin{pmatrix} \mathbf{x} \\ \mathbf{y}^1 \\ \vdots \\ \mathbf{y}^N \end{pmatrix} \geq \begin{pmatrix} \mathbf{b}^1 \\ \vdots \\ \mathbf{b}^N \end{pmatrix}, \mathbf{x} \in X, \mathbf{y}^1, \dots, \mathbf{y}^N \in \mathbb{R}_+^m \right\},$$

in which  $p^i$  denotes the probability of scenario  $i$ . Typically, one has  $X = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} \geq \mathbf{b}^0\}$ . Moreover, one often also puts integrality constraints onto some or all  $\mathbf{x}$ - or  $\mathbf{y}_i$ -variables. Note that the large LP (or IP) decomposes into  $N$  independent smaller LPs (or IPs) once the first-stage decision  $\mathbf{x}$  has been fixed.

If we have no uncertainty in  $\mathbf{d}$ , that is  $\mathbf{d}^1 = \dots = \mathbf{d}^N$ , and if all second-stage variables  $\mathbf{y}$  are continuous, then we can apply Benders decomposition again. Even if there are integrality constraints on  $\mathbf{x}$ ! In this case, one has one master problem in  $\mathbf{x}$  and  $N$  subproblems in  $\mathbf{y}^1, \dots, \mathbf{y}^N$ .

## Chapter 8

# Lagrange relaxation

Let us consider an integer program

$$z_{\text{IP}} = \max\{\mathbf{c}^\top \mathbf{x} : \mathbf{x} \in S\},$$

where  $|S| < \infty$ . Let us assume that we can rewrite the problem as follows:

$$\text{IP}(Q) : \quad \max\{\mathbf{c}^\top \mathbf{x} : \underbrace{A^1 \mathbf{x} \leq \mathbf{b}^1}_{\text{hard}}, \underbrace{\mathbf{x} \in Q}_{\text{easy}}\},$$

where  $Q = \{\mathbf{x} : A^2 \mathbf{x} \leq \mathbf{b}^2, \mathbf{x} \geq \mathbf{0}\}$ . Let  $S = \{\mathbf{x} \in \mathbb{Z}^n : A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\}$ , that is,

$$A = \begin{pmatrix} A^1 \\ A^2 \end{pmatrix} \quad \text{and} \quad \mathbf{b} = \begin{pmatrix} \mathbf{b}^1 \\ \mathbf{b}^2 \end{pmatrix}.$$

Assume that  $\text{IP}(Q)$  is much easier to solve if the constraints  $A^1 \mathbf{x} \leq \mathbf{b}^1$  are dropped. Many problems can be formulated in such a way, for example also two-stage stochastic integer programs.

**Example. (2-stage stochastic integer programming)** Let us introduce copies  $\mathbf{x}^1, \dots, \mathbf{x}^N$  of the first-stage variables  $\mathbf{x}$  and put them all equal:  $\mathbf{x}^1 = \dots = \mathbf{x}^N$ . Thus, after rearranging variables, the two-stage stochastic IP can be written as

$$\max \left\{ \sum_{i=1}^N p^i (\mathbf{c}^\top \mathbf{x}^i + \mathbf{d}^{i\top} \mathbf{y}^i) : \begin{pmatrix} T & W & & & \\ & & \ddots & & \\ & & & T & W \\ - & - & H & - & - \\ - & - & -H & - & - \end{pmatrix} \begin{pmatrix} \mathbf{x}^1 \\ \mathbf{y}^1 \\ \vdots \\ \mathbf{x}^N \\ \mathbf{y}^N \end{pmatrix} \geq \begin{pmatrix} \mathbf{b}^1 \\ \vdots \\ \mathbf{b}^N \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}, \mathbf{x}^i \in X, \mathbf{y}^i \in \mathbb{R}_+^m \right\},$$

in which  $H(\mathbf{x}^1, \mathbf{y}^1, \dots, \mathbf{x}^N, \mathbf{y}^N)^\top = \mathbf{0}$  models the constraints  $\mathbf{x}^1 = \dots = \mathbf{x}^N$ . Relaxing these coupling constraints, the problem decomposes into  $N$  smaller IPs. After solving them, one faces the problem that the first-stage decisions  $\mathbf{x}^1, \dots, \mathbf{x}^N$  may be all different. Still, the optimal value of the relaxed IP gives an upper bound for the 2-stage stochastic IP. If  $\mathbf{x}^1 = \dots = \mathbf{x}^N$  for the relaxed problem, we have found an optimal solution to our stochastic IP.  $\square$

Let us assume that  $A^1 \mathbf{x} \leq \mathbf{b}^1$  contains  $m_1$  inequalities and let  $\boldsymbol{\lambda} \in \mathbb{R}_+^{m_1}$ . We consider the **Lagrange relaxation**  $LR(\boldsymbol{\lambda})$  of  $IP(Q)$ :

$$LR(\boldsymbol{\lambda}) : \quad z_{LR} = \max\{z(\boldsymbol{\lambda}, \mathbf{x}) : \mathbf{x} \in Q\} \quad \text{where} \quad z(\boldsymbol{\lambda}, \mathbf{x}) = \mathbf{c}^\top \mathbf{x} + \boldsymbol{\lambda}^\top (\mathbf{b}^1 - A^1 \mathbf{x}).$$

Since  $\boldsymbol{\lambda} \geq \mathbf{0}$ : If  $\bar{\mathbf{x}} \in \mathbb{R}^n$  violates  $A_i^1 \mathbf{x} \leq \mathbf{b}_i^1$ , we have  $A_i^1 \bar{\mathbf{x}} > \mathbf{b}_i^1$ , that is,  $\mathbf{b}_i^1 - A_i^1 \bar{\mathbf{x}} < 0$ . Thus, for sufficiently large  $\lambda_i$  the inequality  $A_i^1 \mathbf{x} \leq \mathbf{b}_i^1$  is satisfied in an optimal solution.

**Lemma 8.0.1** For all  $\boldsymbol{\lambda} \in \mathbb{R}_+^{m_1}$ ,  $LR(\boldsymbol{\lambda})$  is a relaxation of  $IP(Q)$ , that is,  $z_{IP} \leq z_{LR}$ .

**Proof.** Let  $\bar{\mathbf{x}}$  be feasible for  $IP(Q)$ . Thus  $\bar{\mathbf{x}} \in Q$  and thus  $\bar{\mathbf{x}}$  is feasible for  $LR(\boldsymbol{\lambda})$  for any  $\boldsymbol{\lambda}$ . Moreover, we have

$$z(\boldsymbol{\lambda}, \bar{\mathbf{x}}) = \mathbf{c}^\top \bar{\mathbf{x}} + \underbrace{\boldsymbol{\lambda}}_{\geq \mathbf{0}} \underbrace{(\mathbf{b}^1 - A^1 \bar{\mathbf{x}})}_{\geq \mathbf{0}} \geq \mathbf{c}^\top \bar{\mathbf{x}}.$$

□

Each vector  $\boldsymbol{\lambda} \in \mathbb{R}_+^{m_1}$  defines a relaxation. We are interested in the best (= smallest) upper bound, that is, we are looking for some  $\boldsymbol{\lambda}^*$ , such that  $z_{LR}(\boldsymbol{\lambda}^*)$  is minimal. Let  $\boldsymbol{\lambda}^*$  be an optimal solution to

$$LD : \quad z_{LD} = \min\{z_{LR}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \in \mathbb{R}_+^{m_1}\}.$$

LD is called the **Lagrange dual** of  $IP(Q)$  with respect to  $A^1 \mathbf{x} \leq \mathbf{b}^1$ .

**Example.** Consider the IP:

$$\max\{7x_1 + 2x_2 : -x_1 + 2x_2 \leq 4, \mathbf{x} \in Q\},$$

where

$$Q = \left\{ \mathbf{x} \in \mathbb{Z}_+^2 : \begin{array}{rcl} 5x_1 & + & x_2 \leq 20 \\ -2x_1 & - & 2x_2 \leq -7 \\ -x_1 & & \leq -2 \\ & & x_2 \leq 4 \end{array} \right\}.$$

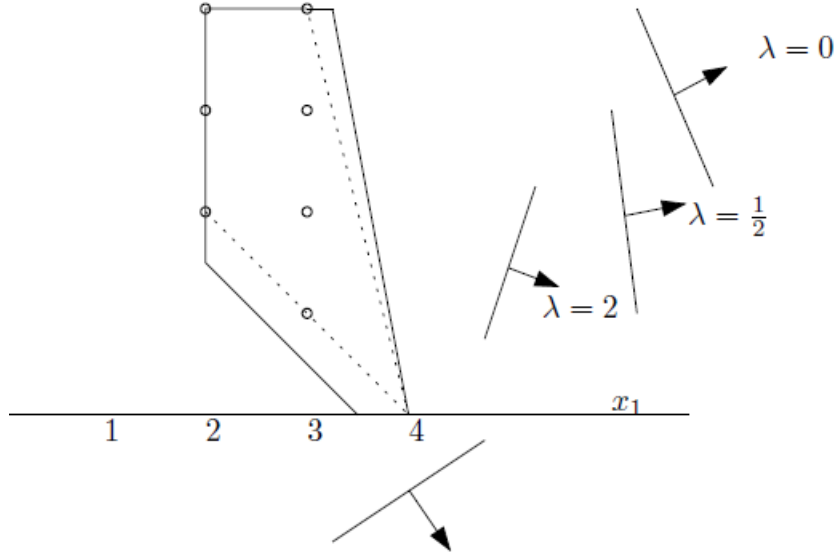
The Lagrange relaxation is:

$$\begin{aligned} z_{LR} &= \max\{7x_1 + 2x_2 + \lambda(4 + x_1 - 2x_2) : \mathbf{x} \in Q\} \\ &= \max\{(7 + \lambda)x_1 + (2 - 2\lambda)x_2 + 4\lambda : \mathbf{x} \in Q\}, \end{aligned}$$

where  $Q = \{(2, 2), (2, 3), (2, 4), (3, 1), (3, 2), (3, 3), (3, 4), (4, 0)\}$ . Let us first consider  $z(\boldsymbol{\lambda}, \mathbf{x})$  for fixed  $\boldsymbol{\lambda}$ :

$$z(\boldsymbol{\lambda}, \mathbf{x}) = \underbrace{(\mathbf{c}^\top - \boldsymbol{\lambda} A^1)}_{=: \text{const}} \mathbf{x} + \underbrace{\boldsymbol{\lambda} \mathbf{b}^1}_{=: \text{const}},$$

and see it as an affine function in  $\mathbf{x}$ .



Thus, we may determine  $z_{\text{LR}}(\boldsymbol{\lambda})$  from

$$z_{\text{LR}}(\boldsymbol{\lambda}) = \max\{z(\boldsymbol{\lambda}, \mathbf{x}) : \mathbf{x} \in \text{conv}(Q)\}.$$

On the other hand, we could fix  $\bar{\mathbf{x}} \in Q$ : For fixed  $\bar{\mathbf{x}} \in Q$ ,

$$z(\boldsymbol{\lambda}, \bar{\mathbf{x}}) = \underbrace{\mathbf{c}^\top \bar{\mathbf{x}}}_{=:\text{const}} + \boldsymbol{\lambda} \underbrace{(b^1 - A^1 \bar{\mathbf{x}})}_{=:\text{const}}$$

is an affine function in  $\boldsymbol{\lambda}$ .

$$\begin{aligned} z_{\text{LR}}(\boldsymbol{\lambda}) &= \min_{\boldsymbol{\lambda} \geq \mathbf{0}} \max_{\bar{\mathbf{x}} \in Q} (\mathbf{c}^\top \bar{\mathbf{x}} + \boldsymbol{\lambda}^\top (b^1 - A^1 \bar{\mathbf{x}})) \\ &= \min\{\omega : \omega \geq z(\boldsymbol{\lambda}, \bar{\mathbf{x}}) \forall \bar{\mathbf{x}} \in Q\} \end{aligned}$$

One can easily show that  $z_{\text{LR}}(\boldsymbol{\lambda})$  is a piece-wise linear and convex function.  $\square$

**Theorem 8.0.2** *The optimal value  $z_{\text{LD}}$  of the Lagrange dual of  $\max\{\mathbf{c}^\top \mathbf{x} : A^1 \mathbf{x} \leq \mathbf{b}^1, \mathbf{x} \in Q\}$  is equal to  $\max\{\mathbf{c}^\top \mathbf{x} : A^1 \mathbf{x} \leq \mathbf{b}^1, \mathbf{x} \in \text{conv}(Q)\}$ .*

**Remark.** This means that the Lagrange dual has the same value as a vector from the convex hull of  $Q$  that satisfies also the complicated inequalities  $A^1 \mathbf{x} \leq \mathbf{b}^1$  and which, among those vectors, maximizes  $\mathbf{c}^\top \mathbf{x}$ .  $\square$

**Proof.** We start by rewriting  $z_{\text{LD}}$ :

$$\begin{aligned} z_{\text{LD}} &= \min\{z_{\text{LR}}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \geq \mathbf{0}\} \\ &= \min_{\boldsymbol{\lambda} \geq \mathbf{0}} \max_{\mathbf{x}_i \in Q} [\mathbf{c}^\top \mathbf{x}_i + \boldsymbol{\lambda}^\top (\mathbf{b}^1 - A^1 \mathbf{x}_i)] \\ &= \min_{\boldsymbol{\lambda} \geq \mathbf{0}} \{\gamma : \gamma \geq \mathbf{c}^\top \mathbf{x}_i + \boldsymbol{\lambda}^\top (\mathbf{b}^1 - A^1 \mathbf{x}_i) \forall \mathbf{x}_i \in Q\} \\ &= \min\{\gamma : \gamma + \boldsymbol{\lambda}^\top (A^1 \mathbf{x}_i - \mathbf{b}^1) \geq \mathbf{c}^\top \mathbf{x}_i \forall \mathbf{x}_i \in Q, \boldsymbol{\lambda} \geq \mathbf{0}, \gamma \in \mathbb{R}\} \end{aligned}$$

Dualizing this last minimization problem, we get

$$\begin{aligned}
z_{\text{LD}} &= \max \left\{ \sum_{\mathbf{x}_i \in Q} (\mathbf{c}^\top \mathbf{x}_i) : \sum_{x_i \in Q} \alpha_i = 1, \sum_{x_i \in Q} \alpha_i (A^1 \mathbf{x}_i - \mathbf{b}^1) \leq \mathbf{0}, \boldsymbol{\alpha} \geq \mathbf{0} \right\} \\
&= \max \left\{ \mathbf{c}^\top \left( \sum_{x_i \in Q} \alpha_i \mathbf{x}_i \right) : \boldsymbol{\alpha} \geq \mathbf{0}, \sum_{x_i \in Q} \alpha_i = 1, A^1 \left( \sum_{x_i \in Q} \alpha_i \mathbf{x}_i \right) \leq \mathbf{b}^1 \right\} \\
&= \max \{ \mathbf{c}^\top \mathbf{y} : \mathbf{y} \in \text{conv}(Q), A^1 \mathbf{y} \leq \mathbf{b}^1 \}.
\end{aligned}$$

□

In general, we have

$$\text{conv}(S) \subseteq \text{conv}(Q) \cap \{ \mathbf{x} : A^1 \mathbf{x} \leq \mathbf{b}^1 \} \subseteq \{ \mathbf{x} : A \mathbf{x} \leq \mathbf{b} \}.$$

Consequently, we get

$$z_{\text{IP}} \leq z_{\text{LD}} \leq z_{\text{LP}}.$$

## 8.1 Solving the Lagrange dual

We wish to solve the following problem:

$$z_{\text{LD}} = \min \{ z_{\text{LR}}(\boldsymbol{\lambda}) : \boldsymbol{\lambda} \geq \mathbf{0} \}.$$

Herein,  $f(\boldsymbol{\lambda}) := z_{\text{LR}}(\boldsymbol{\lambda})$  is a piece-wise linear convex function. If it were differentiable, we could easily apply a steepest descend method: Choose a starting point  $\boldsymbol{\lambda}_1$  and iteratively augment it along the descending direction  $-\nabla f(\boldsymbol{\lambda}_1)$ . However, as in our case  $f(\boldsymbol{\lambda})$  is not differentiable, we need to generalize the notion of a gradient.

**Definition 8.1.1** A vector  $\mathbf{s}$  is called a **subgradient** of  $f$  at  $\boldsymbol{\lambda}^*$  if

$$f(\boldsymbol{\lambda}) \geq f(\boldsymbol{\lambda}^*) + \mathbf{s}^\top (\boldsymbol{\lambda} - \boldsymbol{\lambda}^*)$$

holds for all  $\boldsymbol{\lambda} \in \mathbb{R}^n$ . The set of all subgradients of  $f$  at  $\boldsymbol{\lambda}^*$  is denoted by  $\partial f(\boldsymbol{\lambda}^*)$  and is called the **subdifferential** of  $f$  at  $\boldsymbol{\lambda}^*$ .

**Lemma 8.1.2** A function  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  is convex if and only if for any  $\boldsymbol{\lambda}^*$  there exists a subgradient of  $f$  at  $\boldsymbol{\lambda}^*$ .

If  $f$  is differentiable, one can show that  $\partial f(\boldsymbol{\lambda}^*) = \{ \nabla f(\boldsymbol{\lambda}^*) \}$ .

**Lemma 8.1.3** Let  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  be a convex function. A vector  $\boldsymbol{\lambda}^*$  minimizes  $f$  over  $\mathbb{R}^n$  if and only if  $\mathbf{0} \in \partial f(\boldsymbol{\lambda}^*)$ .

Thus, we get the following subgradient algorithm to find the minimum of a convex function.

- (1) Let  $\boldsymbol{\lambda}_1 \in \mathbb{R}_+^{m_1}$  be arbitrary. Choose an infinite sequence  $\{s_1, s_2, \dots\}$  of positive step-lengths. Set  $k := 1$ .
- (2) Compute a subgradient  $\mathbf{s} \in \partial f(\boldsymbol{\lambda}_k)$ . If  $\mathbf{s} = \mathbf{0}$ , then stop with optimal solution  $\boldsymbol{\lambda}_k$ .
- (3) Use step-length  $s_k$ .
- (4) Set  $\boldsymbol{\lambda}_{k+1} := \boldsymbol{\lambda}_k - s_k \frac{\mathbf{s}}{\|\mathbf{s}\|_1}$ ,  $k := k + 1$ , and go to step (1).

One can show that this approach converges if the step lengths  $s_1, s_2, \dots$  are chosen in such a way that:

- $\lim_{i \rightarrow \infty} s_i = 0$  and
- $\sum_{i=1}^{\infty} s_i = \infty$

hold. In general, a good selection of  $s_k$  is important for the practical convergence. Applying the subgradient approach to the Lagrange dual, we get the following algorithm:

- (1) Let  $\boldsymbol{\lambda}_1 \in \mathbb{R}_+^{m_1}$  be arbitrary.
- (2) In step  $k$  let  $\boldsymbol{\lambda}_k$  be given.
- (3) Compute a vector  $\mathbf{x}_k$  that maximizes

$$\mathbf{c}^\top \mathbf{x} + \boldsymbol{\lambda}_k^\top (\mathbf{b}^1 - A^1 \mathbf{x})$$

over  $Q$ . Note that  $\mathbf{b}^1 - A^1 \mathbf{x}^k$  is a subgradient.

- (4) Set  $\boldsymbol{\lambda}_{k+1} := \max \{ \mathbf{0}, \boldsymbol{\lambda}_k - s_k (\mathbf{b}^1 - A^1 \mathbf{x}^k) \}$ .

The step lengths  $s_k$  have to be chosen carefully. Moreover, if  $\boldsymbol{\lambda}_k = \mathbf{0}$ , stop.

Finally, it could make sense in praxis to fix a maximum number of iterations before the algorithm stops.