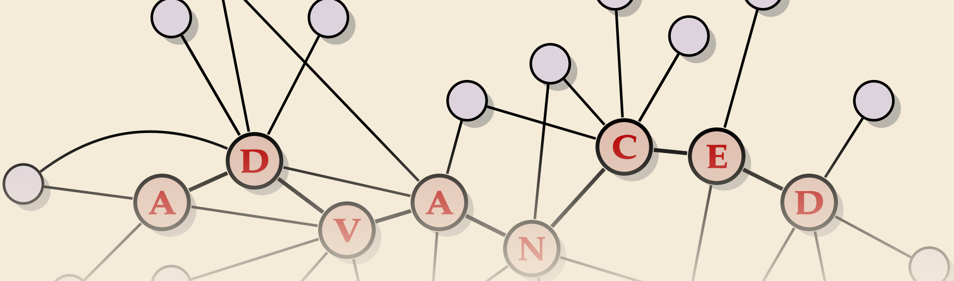


6 Advanced Parameterized Ideas

3 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version `2025-06-03 23:20`

Outline

6 Advanced Parameterized Ideas

6.1 Linear Programs – A Mighty Blackbox Tool

Linear Programs

- ***Linear programs (LPs)*** are a class of optimization problems of **continuous** (numerical) variables
- can be exactly solved in worst case polytime ($\textsc{LinearProgramming} \in \mathsf{P}$)
  - interior-point methods, Ellipsoid method
- routinely solved in practice to optimality with millions of variables and constraints
  - Simplex algorithm, interior-point methods
  - many existing solvers, commercial and open source (e. g., HiGHS)

Hessy James's Apple Farm

- Hessy tries to maximize the profit of his apple farm
  - He is committed to promote regional Hessian heirloom varieties, so he only grows *"Sossenheimer Roter"* and *"Korbacher Edelrenette"*
  - each tree of *"Sossenheimer Roter"* yields apples worth €195 per year
  - each tree of *"Korbacher Edelrenette"* yields applies worth €255 per year
  - He has an orchard of 5 000 m^2
  - each tree needs 4 m^2 of orchard space
  - each tree of *"Sossenheimer Roter"* needs 6 kg of organic fertilizer and 1 h harvest effort per year
  - each tree of *"Korbacher Edelrenette"* needs 4.5 kg of organic fertilizer and 3 h harvest effort per year
  - Hessy can only afford 3000 kg of fertilizer and 1700 h of harvester time per year

Hessy James's Apple Farm

- Hessy tries to maximize the profit of his apple farm
  - He is committed to promote regional Hessian heirloom varieties, so he only grows *"Sossenheimer Roter"* and *"Korbacher Edelrenette"*
  - each tree of *"Sossenheimer Roter"* yields apples worth € 195 per year
  - each tree of *"Korbacher Edelrenette"* yields applies worth € 255 per year
  - He has an orchard of 5 000 m^2
  - each tree needs 4 m^2 of orchard space
  - each tree of *"Sossenheimer Roter"* needs 6 kg of organic fertilizer and 1 h harvest effort per year
  - each tree of *"Korbacher Edelrenette"* needs 4.5 kg of organic fertilizer and 3 h harvest effort per year
  - Hessy can only afford 3000 kg of fertilizer and 1700 h of harvester time per year

⇝ How many trees of each variety should Hessy plant?
  - What will constrain us most? Space? Fertilizer? Harvest hours?
  - What profit can Hessy expect?

Formal Linear Program for Hessy James's Apple Farm

- Classic application of linear programming in *operations research (OR)*
- We formally write LPs as follows:

optimization goal, objective function, constraint, name of the LP

$$
\begin{array}{lrcrcrl}
\textbf{Maximize:} & 195s + 255k & & & & & \\
\textbf{Subject to:} & 4s + 4k & \leq & 5000 & & \text{(Orchard constraint)} & \\
& 6s + 4.5k & \leq & 3000 & & \text{(Fertilizer constraint)} & \\
& 1s + 3k & \leq & 1700 & & \text{(Harvest constraint)} & (P) \\
& s & \geq & 0 & & \text{(Non-negativity)} & \\
& k & \geq & 0 & & \text{(Non-negativity)} &
\end{array}
$$

Formal Linear Program for Hessy James's Apple Farm

- Classic application of linear programming in *operations research (OR)*
- We formally write LPs as follows:

optimization goal

objective function

constraint

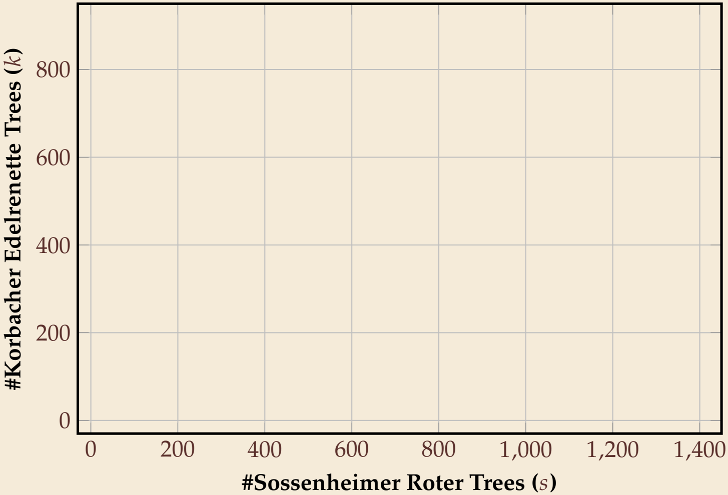
name of the LP

$$
\begin{array}{lrcll}
\textbf{Maximize:} & 195s + 255k & & & \\
\textbf{Subject to:} & 4s + 4k & \le & 5000 & \text{(Orchard constraint)} \\
 & 6s + 4.5k & \le & 3000 & \text{(Fertilizer constraint)} \\
 & 1s + 3k & \le & 1700 & \text{(Harvest constraint)} \\
 & s & \ge & 0 & \text{(Non-negativity)} \\
 & k & \ge & 0 & \text{(Non-negativity)}
\end{array}
\qquad (P)
$$

- **Terminology:**
  - s and k are the two *variables* of the problem; these are always real numbers.
  - A vector $(s, k) \in \mathbb{R}^2$ is a *feasible solution* for the LP if it satisfied all constraints.
  - The largest value of the objective function (over all feasible solutions) is the *(optimal) value* z^* of the LP
  - A feasible solution $(s^*, k^*) \in R^2$ with optimal objective value z^* is called an *optimal solution*

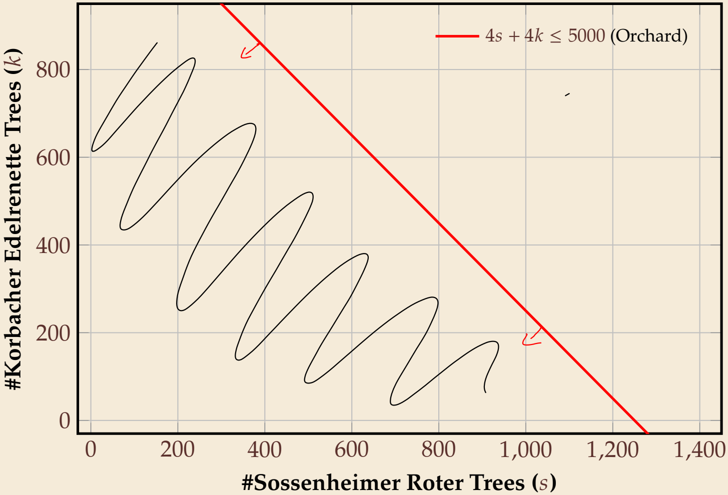
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



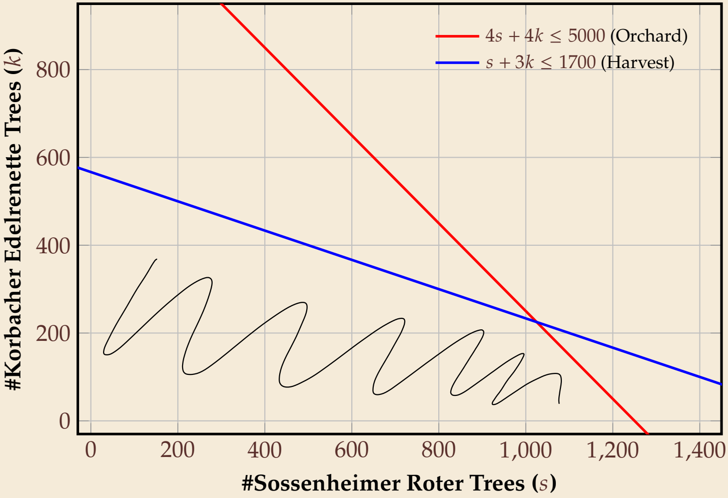
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



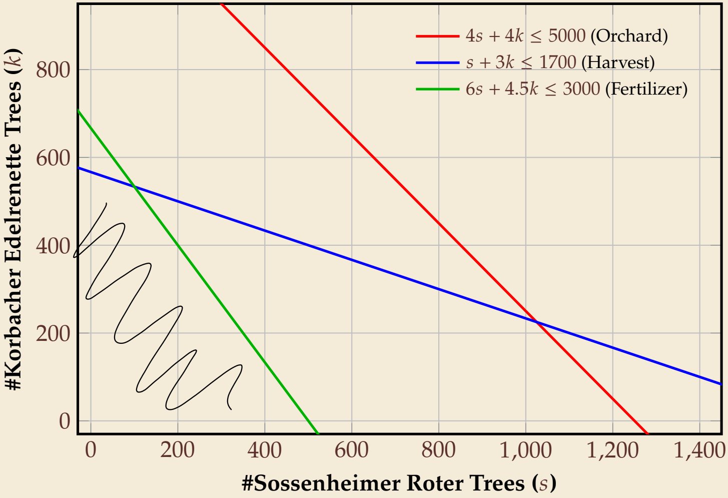
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



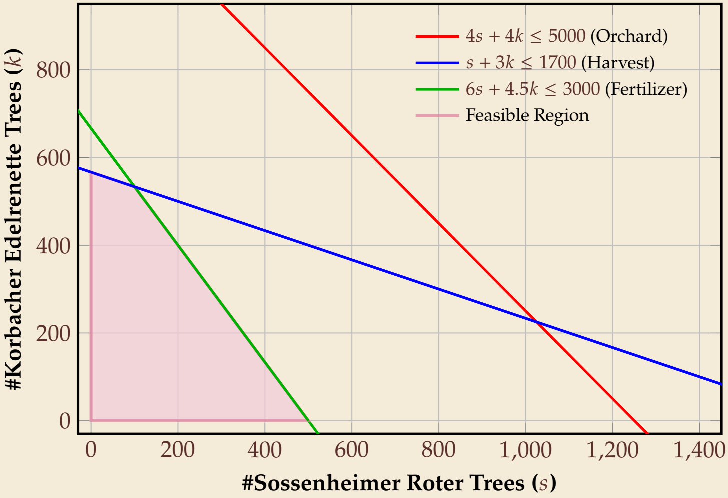
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



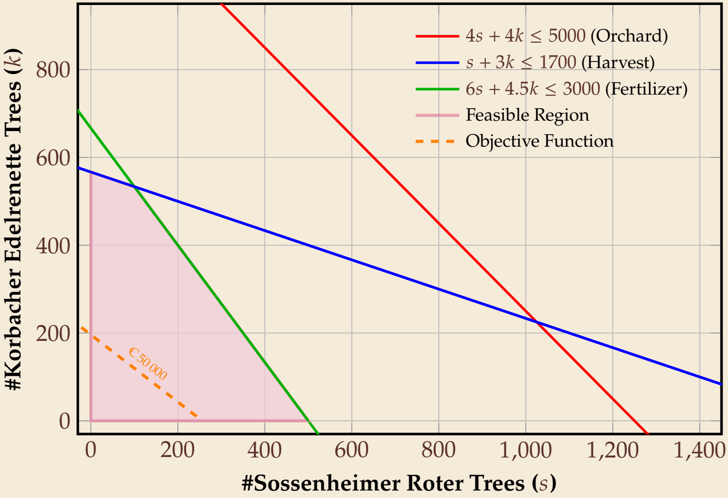
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



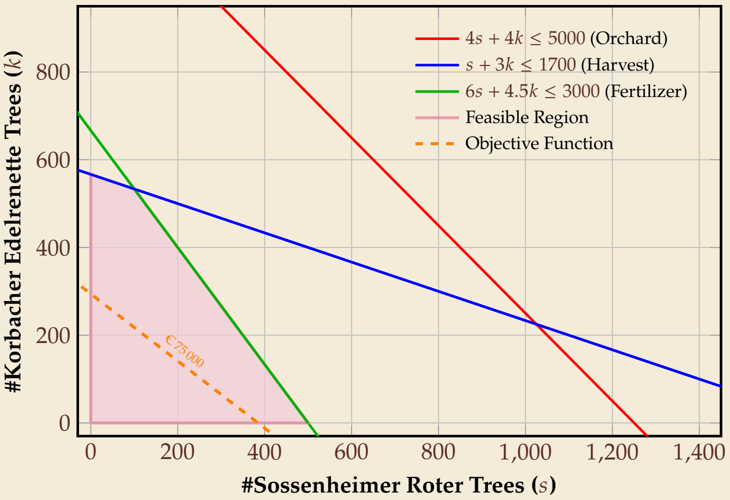
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



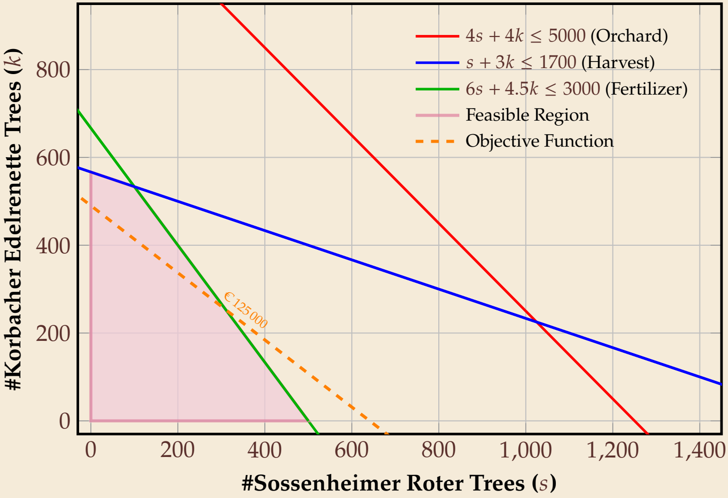
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



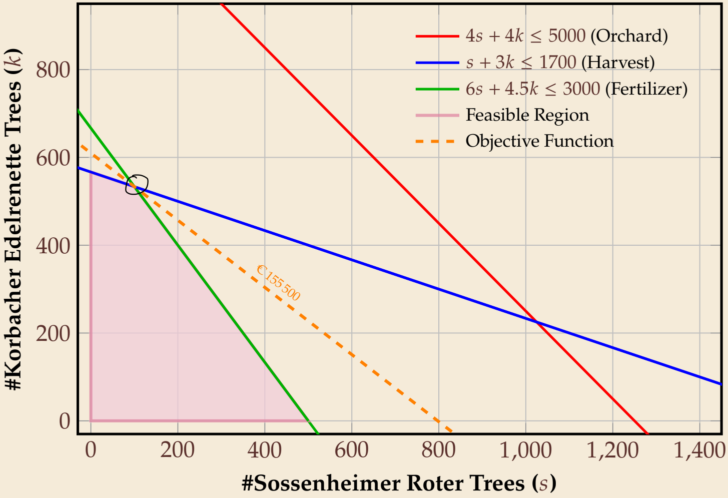
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



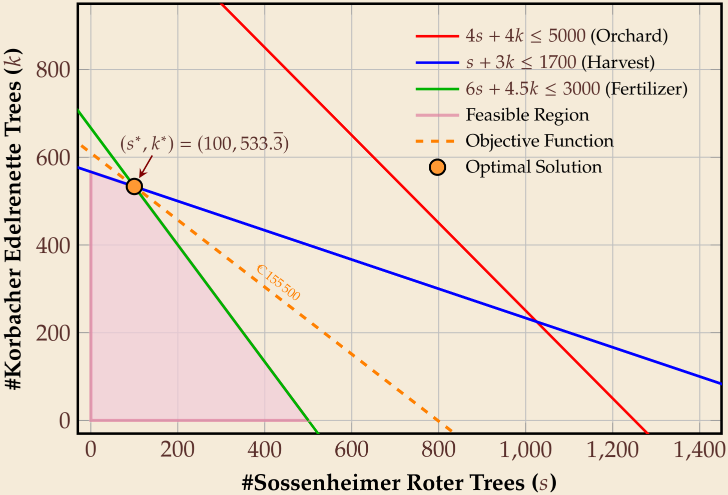
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



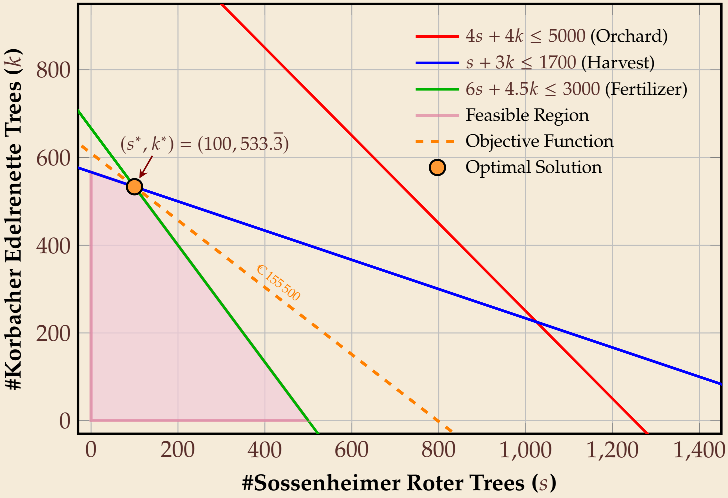
2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



2D LPs – Graphical Solution

LPs with **two** variables can be solved graphically



$\rightsquigarrow$ Hessy should plant

- 100 *Sossenheimer Roter* trees and hmm . . .
- $533+\frac{1}{3}$ *Korbacher Edelrenette* trees
- Harvest **and** fertilizer *tight*
- orchard space isn't

$\rightsquigarrow$ know what to change

LPs – The General Case

- General LP:

$$
\begin{array}{lrcll}
\min & c_1 x_1 + \cdots + \;\; c_n x_n & & & \\
\text{s. t.} & a_{i,1} x_1 + \cdots + a_{i,n} x_n & = & b_i & (\text{for } i = 1, \ldots, p) \\
 & a_{i,1} x_1 + \cdots + a_{i,n} x_n & \leq & b_i & (\text{for } i = p+1, \ldots, q) \\
 & a_{i,1} x_1 + \cdots + a_{i,n} x_n & \geq & b_i & (\text{for } i = q+1, \ldots, m) \\
 & x_j & \geq & 0 & (\text{for } j = 1 \ldots, r) \\
 & x_j & \lesseqgtr & 0 & (\text{for } j = r+1 \ldots, n)
\end{array}
$$

"don't care" (just to make it explicit)

  - arbitrary **linear** objective function
  - arbitrary **linear** constraints, of type "=", "≤" or "≥"
  - variables with non-negativity constraint and unconstrained variables

LPs – The General Case

- General LP:

$$\begin{array}{lrcll} \min & c_1x_1 + \cdots + \; c_nx_n & & & \\ \text{s. t.} & a_{i,1}x_1 + \cdots + a_{i,n}x_n & = & b_i & (\text{for } i = 1, \ldots, p) \\ & a_{i,1}x_1 + \cdots + a_{i,n}x_n & \leq & b_i & (\text{for } i = p+1, \ldots, q) \\ & a_{i,1}x_1 + \cdots + a_{i,n}x_n & \geq & b_i & (\text{for } i = q+1, \ldots, m) \\ & x_j & \geq & 0 & (\text{for } j = 1 \ldots, r) \\ & x_j & \lessgtr & 0 & (\text{for } j = r+1 \ldots, n) \end{array}$$

"don't care" (just to make it explicit)

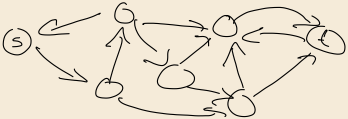
  - arbitrary **linear** objective function
  - arbitrary **linear** constraints, of type "=", "≤" or "≥"
  - variables with non-negativity constraint and unconstrained variables

- In general, an LP can
  (a) have a *finite* optimal *objective value*
  (b) be *infeasible* (contradictory constraints / empty feasibility region), or
  (c) be *unbounded* (allow arbitrarily small objective values "$-\infty$")

⇝ in polytime, can detect which case applies **and** compute optimal solution in case (a)

Classic Modeling Example – Max Flow

- The maximum-s-t-flow problem in a graph $G = (V, E)$ can be reduced to an LP (Flow)
  - variable f_e for each edge $e \in E$
  - maximize flow value F = flow out of s
  - constraint for edge capacity $C(e)$ at each edge
  - constraint for flow conservation at each vertex v (except s and t)



$$
\begin{aligned}
\max \quad & F \\
\text{s.t.} \quad & F = \sum_{v \in V} f_{sv} - \sum_{v \in V} f_{vs} \\
& f_{vw} \le C(vw) && (\text{for } vw \in E) && \text{(Flow)} \\
& \sum_{w \in V} f_{wv} = \sum_{w \in V} f_{vw} && (\text{for } v \in V \setminus \{s, t\}) \\
& f_e \ge 0 && (\text{for } e \in E)
\end{aligned}
$$

$v \to w$ (v, w)

6.2 Linear Programs – Reformulation Tricks

How to solve an LP?

- Our focus will be on using LPs as a tool
  - in theory: reducing problem to an LP means polytime solvable
  - in practice: call good solver!

How to solve an LP?

- Our focus will be on using LPs as a tool
  - in theory: reducing problem to an LP means polytime solvable
  - in practice: call good solver!

- *But as with any good tool, it helps to gave an idea of* ***how*** *it works to effectively use it*

$\rightsquigarrow$ We will briefly visit the conceptual ideas of the simplex algorithm

Recall: General Form of LPs

- General LP:

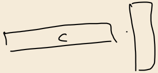
$$
\begin{array}{rrcll}
\min & c_1 x_1 + \cdots + c_n x_n & & & \\
\text{s. t.} & a_{i,1} x_1 + \cdots + a_{i,n} x_n & = & b_i & (\text{for } i = 1, \ldots, p) \\
& a_{i,1} x_1 + \cdots + a_{i,n} x_n & \leq & b_i & (\text{for } i = p+1, \ldots, q) \\
& a_{i,1} x_1 + \cdots + a_{i,n} x_n & \geq & b_i & (\text{for } i = q+1, \ldots, m) \\
& x_j & \geq & 0 & (\text{for } j = 1 \ldots, r) \\
& x_j & \lesseqgtr & 0 & (\text{for } j = r+1 \ldots, n)
\end{array}
$$

  - linear objective function and constraints ("$=$", "$\leq$", or "$\geq$")
  - variables with non-negativity constraint and unconstrained variables

- **Conventions:**
  - n variables (always called x_j)
  - m constraints (coefficients always called $a_{i,j}$, right-hand sides b_i)
  - minimize objective ("<u>c</u>ost"), coefficients c_j; objective value $z = c_1 x_1 + \cdots c_n x_n$

Enter Linear Algebra

$$\begin{array}{llcll} \min & c_1x_1 + \cdots + \ c_nx_n & & & \\ \text{s.t.} & a_{i,1}x_1 + \cdots + a_{i,n}x_n & = & b_i & (\text{for } i = 1,\ldots,p) \\ & a_{i,1}x_1 + \cdots + a_{i,n}x_n & \leq & b_i & (\text{for } i = p+1,\ldots,q) \\ & a_{i,1}x_1 + \cdots + a_{i,n}x_n & \geq & b_i & (\text{for } i = q+1,\ldots,m) \\ & x_j & \geq & 0 & (\text{for } j = 1\ldots,r) \\ & x_j & \lesseqgtr & 0 & (\text{for } j = r+1\ldots,n) \end{array}$$



- Spelling out all those linear combinations is cumbersome

⇝ Concise notation via **matrix and vector products**

- We write
  - **variables** $\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ **cost** coefficients $\boldsymbol{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n$

    bold ⇝ vector/matrix

    ⇝ **objective**: $\min \boldsymbol{c}^T \cdot \boldsymbol{x}$

Enter Linear Algebra

$$\begin{array}{llcll} \min & c_1x_1+\cdots+\ c_nx_n & & & \\ \text{s.t.} & a_{i,1}x_1+\cdots+a_{i,n}x_n & = & b_i & (\text{for } i=1,\ldots,p) \\ & a_{i,1}x_1+\cdots+a_{i,n}x_n & \leq & b_i & (\text{for } i=p+1,\ldots,q) \\ & a_{i,1}x_1+\cdots+a_{i,n}x_n & \geq & b_i & (\text{for } i=q+1,\ldots,m) \\ & x_j & \geq & 0 & (\text{for } j=1\ldots,r) \\ & x_j & \lesseqgtr & 0 & (\text{for } j=r+1\ldots,n) \end{array}$$

- Spelling out all those linear combinations is cumbersome

⇝ Concise notation via **matrix and vector products**

- We write
  - **variables** $\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ **cost** coefficients $\boldsymbol{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n$ ⇝ **objective**: $\min \boldsymbol{c}^T \cdot \boldsymbol{x}$

bold ⇝ vector/matrix

transpose

dot product / scalar product

Enter Linear Algebra

$$
\begin{array}{lrcll}
\min & c_1x_1 + \cdots + c_nx_n & & & \\
\text{s.t.} & a_{i,1}x_1 + \cdots + a_{i,n}x_n & = & b_i & (\text{for } i = 1,\ldots,p) \\
& a_{i,1}x_1 + \cdots + a_{i,n}x_n & \le & b_i & (\text{for } i = p+1,\ldots,q) \\
& a_{i,1}x_1 + \cdots + a_{i,n}x_n & \ge & b_i & (\text{for } i = q+1,\ldots,m) \\
& x_j & \ge & 0 & (\text{for } j = 1\ldots,r) \\
& x_j & \lessgtr & 0 & (\text{for } j = r+1\ldots,n)
\end{array}
$$

- Spelling out all those linear combinations is cumbersome

$\rightsquigarrow$ Concise notation via **matrix and vector products**

- We write
  - **variables** $\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ **cost** coefficients $\underbrace{\boldsymbol{c}}_{\text{bold} \rightsquigarrow \text{vector/matrix}} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n$ $\rightsquigarrow$ **objective:** $\min \boldsymbol{c}^{T} \cdot \boldsymbol{x}$

    transpose

    dot product / scalar product
  - **"="-constraints**

$$A^{(=)} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1} & a_{p,2} & \cdots & a_{p,n} \end{pmatrix} \in \mathbb{R}^{p\times n} \qquad \boldsymbol{b}^{(=)} = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix} \in \mathbb{R}^p \qquad \rightsquigarrow \quad A^{(=)} \cdot \boldsymbol{x} = \boldsymbol{b}^{(=)}$$

Enter Linear Algebra

$$\begin{array}{lrcll} \min & c_1x_1 + \cdots + \ c_nx_n & & & \\ \text{s. t.} & a_{i,1}x_1 + \cdots + a_{i,n}x_n & = & b_i & (\text{for } i = 1,\ldots,p) \\ & a_{i,1}x_1 + \cdots + a_{i,n}x_n & \leq & b_i & (\text{for } i = p+1,\ldots,q) \\ & a_{i,1}x_1 + \cdots + a_{i,n}x_n & \geq & b_i & (\text{for } i = q+1,\ldots,m) \\ & x_j & \geq & 0 & (\text{for } j = 1\ldots,r) \\ & x_j & \lesseqgtr & 0 & (\text{for } j = r+1\ldots,n) \end{array}$$

- Spelling out all those linear combinations is cumbersome

⇝ Concise notation via **matrix and vector products**

- We write
  - **variables** $\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ bold ⇝ vector/matrix **cost** coefficients $\boldsymbol{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n$ ⇝ **objective:** $\min \boldsymbol{c}^T \cdot \boldsymbol{x}$ transpose; dot product / scalar product
  - **"="-constraints**

$$\boldsymbol{A}^{(=)} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1} & a_{p,2} & \cdots & a_{p,n} \end{pmatrix} \in \mathbb{R}^{p\times n} \qquad \boldsymbol{b}^{(=)} = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix} \in \mathbb{R}^p \qquad \rightsquigarrow \quad \boldsymbol{A}^{(=)} \cdot \boldsymbol{x} = \boldsymbol{b}^{(=)}$$

  - similarly for "≤" and "≥" constraints: elementwise ≤ $\boldsymbol{A}^{(\leq)}\boldsymbol{x} \leq \boldsymbol{b}^{(\leq)}$ and $\boldsymbol{A}^{(\geq)}\boldsymbol{x} \geq \boldsymbol{b}^{(\geq)}$

Enter Linear Algebra

$$
\begin{array}{lrcll}
\min & c_1x_1 + \cdots + c_nx_n & & & \\
\text{s. t.} & a_{i,1}x_1 + \cdots + a_{i,n}x_n & = & b_i & (\text{for } i = 1,\ldots,p) \\
& a_{i,1}x_1 + \cdots + a_{i,n}x_n & \leq & b_i & (\text{for } i = p+1,\ldots,q) \\
& a_{i,1}x_1 + \cdots + a_{i,n}x_n & \geq & b_i & (\text{for } i = q+1,\ldots,m) \\
& x_j & \geq & 0 & (\text{for } j = 1\ldots,r) \\
& x_j & \lessgtr & 0 & (\text{for } j = r+1\ldots,n)
\end{array}
$$

- Spelling out all those linear combinations is cumbersome

$\rightsquigarrow$ Concise notation via **matrix and vector products**

- We write
  - **variables** $\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$ **cost** coefficients $\boldsymbol{c} = \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \in \mathbb{R}^n$ $\rightsquigarrow$ **objective**: $\min \boldsymbol{c}^T \cdot \boldsymbol{x}$

    bold $\rightsquigarrow$ vector/matrix

    transpose

    dot product / scalar product
  - "="**-constraints**

$$
\boldsymbol{A}^{(=)} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p,1} & a_{p,2} & \cdots & a_{p,n} \end{pmatrix} \in \mathbb{R}^{p\times n} \qquad \boldsymbol{b}^{(=)} = \begin{pmatrix} b_1 \\ \vdots \\ b_p \end{pmatrix} \in \mathbb{R}^p \qquad \rightsquigarrow \quad \boldsymbol{A}^{(=)} \cdot \boldsymbol{x} = \boldsymbol{b}^{(=)}
$$

  - similarly for "$\leq$" and "$\geq$" constraints: $\boldsymbol{A}^{(\leq)}\boldsymbol{x} \underset{\text{elementwise }\leq}{\leq} \boldsymbol{b}^{(\leq)}$ and $\boldsymbol{A}^{(\geq)}\boldsymbol{x} \geq \boldsymbol{b}^{(\geq)}$

$\rightsquigarrow$ a **single** constraint i can be written as $\boldsymbol{A}_{i,\bullet}\,\boldsymbol{x} = b_i$ A[i, :]

(generally write $\boldsymbol{A}_{i,\bullet}$ for the ith row of $\boldsymbol{A}$ and $\boldsymbol{A}_{\bullet,j}$ for the jth column)

Reformulations

Tricks of the Trade for working with LPs:

- min suffices: $\max \boldsymbol{c}^T \boldsymbol{x} \;=\; -\min(-\boldsymbol{c})^T \boldsymbol{x}$
- "$\geq$"-constraints: $A_{i,\bullet}\,\boldsymbol{x} \;\geq\; b_i \quad\Longleftrightarrow\quad (-A)_{i,\bullet}\,\boldsymbol{x} \;\leq\; -b_i$

Reformulations

Tricks of the Trade for working with LPs:

- min suffices: $\max \boldsymbol{c}^T \boldsymbol{x} \;=\; -\min(-\boldsymbol{c})^T \boldsymbol{x}$
- "≥"-constraints: $A_{i,\bullet}\boldsymbol{x} \;\geq\; b_i \quad\Longleftrightarrow\quad (-A)_{i,\bullet}\boldsymbol{x} \;\leq\; -b_i$
- *slack variables*: $A_{i,\bullet}\boldsymbol{x} \;\leq\; b_i \quad\Longleftrightarrow\quad A_{i,\bullet}\boldsymbol{x} + \boldsymbol{x_{s_i}} \;=\; b_i \quad \text{and} \quad x_{s_i} \;\geq\; 0$

  (x_{s_i} is a new additional variable)

Reformulations

Tricks of the Trade for working with LPs:

- min suffices: $\max \boldsymbol{c}^T \boldsymbol{x} \;=\; -\min(-\boldsymbol{c})^T \boldsymbol{x}$
- "$\geq$"-constraints: $\boldsymbol{A}_{i,\bullet} \boldsymbol{x} \;\geq\; b_i \iff (-\boldsymbol{A})_{i,\bullet} \boldsymbol{x} \;\leq\; -b_i$
- *slack variables*: $\boldsymbol{A}_{i,\bullet} \boldsymbol{x} \;\leq\; b_i \iff \boldsymbol{A}_{i,\bullet} \boldsymbol{x} + \boldsymbol{x}_{s_i} \;=\; b_i$ and $x_{s_i} \;\geq\; 0$

  (x_{s_i} is a new additional variable)
- *nonnegative*: variable $x_j \;\lesseqgtr\; 0 \iff x_j \;=\; x_{j,+} - x_{j,-}$ and $x_{j,+}, x_{j,-} \;\geq\; 0$

  ($x_{j,+}$ and $x_{j,-}$ are new additional variables)

Reformulations

Tricks of the Trade for working with LPs:

- min suffices: $\max \boldsymbol{c}^T \boldsymbol{x} = -\min(-\boldsymbol{c})^T \boldsymbol{x}$
- "$\geq$"-constraints: $A_{i,\bullet}\boldsymbol{x} \geq b_i \iff (-A)_{i,\bullet}\boldsymbol{x} \leq -b_i$
- *slack variables*: $A_{i,\bullet}\boldsymbol{x} \leq b_i \iff A_{i,\bullet}\boldsymbol{x} + \boldsymbol{x}_{s_i} = b_i$ and $x_{s_i} \geq 0$

  (x_{s_i} is a new additional variable)
- *nonnegative*: variable $x_j \lessgtr 0 \iff x_j = x_{j,+} - x_{j,-}$ and $x_{j,+}, x_{j,-} \geq 0$

  ($x_{j,+}$ and $x_{j,-}$ are new additional variables)

⇝ To solve LPs, can assume one of the following **normal forms**

$$\boxed{\begin{array}{lrcl} \min & \boldsymbol{c}^T\boldsymbol{x} & & \\ \text{s.t.} & A\boldsymbol{x} & \leq & \boldsymbol{b} \\ & \boldsymbol{x} & \geq & \boldsymbol{0} \end{array}} \quad \text{or} \quad \boxed{\begin{array}{lrcl} \min & \boldsymbol{c}^T\boldsymbol{x} & & \\ \text{s.t.} & A\boldsymbol{x} & = & \boldsymbol{b} \\ & \boldsymbol{x} & \geq & \boldsymbol{0} \end{array}}$$

with $A \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$

$m \geq n$ $\qquad$ $m \lessgtr n$

6.3 Linear Programs – The Simplex Algorithm

Simplex – Geometric Intuition

$$\begin{aligned} \min\ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} \le \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \end{aligned}$$

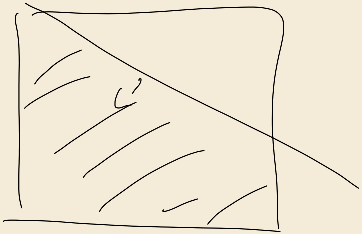
+ *nondegeneracy*

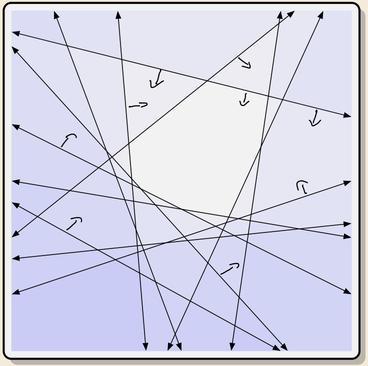
- constraint $A_{i,\bullet}\boldsymbol{x} \le b_i$ defines a *hyperplane*/*halfspace*

$\boxed{n = 2, m = 12}$

$\rightsquigarrow$ $H_i^= = \{\boldsymbol{x} \in \mathbb{R}^n : A_{i,\bullet}\boldsymbol{x} = b_i\}$

$H_i = \{\boldsymbol{x} \in \mathbb{R}^n : A_{i,\bullet}\boldsymbol{x} \le b_i\}$





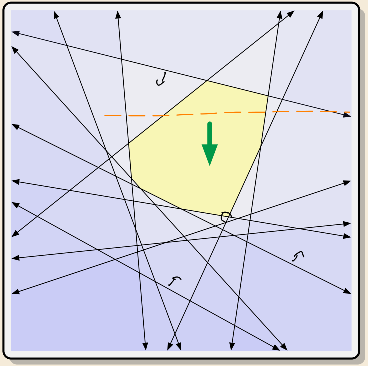
Simplex – Geometric Intuition

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & Ax \le b \\ & x \ge 0 \end{aligned}$$
+ *nondegeneracy*

- constraint $A_{i,\bullet}x \le b_i$ defines a *hyperplane*/*halfspace*

$\boxed{n = 2, m = 12}$

$\rightsquigarrow$ $H_i^= = \{x \in \mathbb{R}^n : A_{i,\bullet}x = b_i\}$
$H_i = \{x \in \mathbb{R}^n : A_{i,\bullet}x \le b_i\}$



- c = **direction** of improvement in $\mathbb{R}^n$ (*normal vector* for hyperplane $\{x \in \mathbb{R}^n : c^T x = 0\}$)
  - *"Roll a ball downhill inside feasible region"*

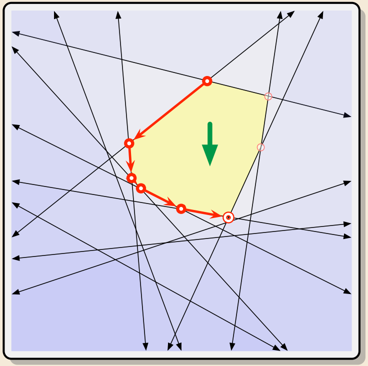
Simplex – Geometric Intuition

$$\begin{array}{rrcl} \min & c^T x & & \\ \text{s.t.} & Ax & \leq & b \\ & x & \geq & \mathbf{0} \end{array}$$
+ *nondegeneracy*

- constraint $A_{i,\bullet} x \leq b_i$ defines a *hyperplane*/*halfspace*

$\boxed{n = 2, m = 12}$

$\rightsquigarrow$ $H_i^= = \{x \in \mathbb{R}^n : A_{i,\bullet} x = b_i\}$
$H_i = \{x \in \mathbb{R}^n : A_{i,\bullet} x \leq b_i\}$



- c = **direction** of improvement in $\mathbb{R}^n$ (*normal vector* for hyperplane $\{x \in \mathbb{R}^n : c^T x = 0\}$)
  - *"Roll a ball downhill inside feasible region"*

  $\rightsquigarrow$ Optimal point x^* must lie on boundary!

  (assuming finite optimal objective value z^*)

Simplex – Geometric Intuition

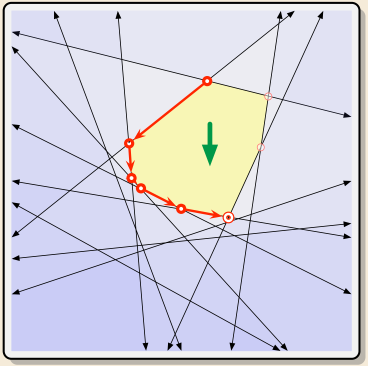
$$\boxed{\begin{aligned} \min \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.} \quad & A\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & + \textit{nondegeneracy} \end{aligned}}$$

- constraint $A_{i,\bullet}\boldsymbol{x} \leq b_i$ defines a *hyperplane*/*halfspace*

$\boxed{n = 2, m = 12}$

$\rightsquigarrow$ $H_i^= = \{\boldsymbol{x} \in \mathbb{R}^n : A_{i,\bullet}\boldsymbol{x} = b_i\}$

$H_i = \{\boldsymbol{x} \in \mathbb{R}^n : A_{i,\bullet}\boldsymbol{x} \leq b_i\}$



- $\boldsymbol{c}$ = **direction** of improvement in $\mathbb{R}^n$ (*normal vector* for hyperplane $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{c}^T\boldsymbol{x} = 0\}$)
  - *"Roll a ball downhill inside feasible region"*

  $\rightsquigarrow$ Optimal point $\boldsymbol{x}^*$ must lie on boundary!

  (assuming finite optimal objective value z^*)

assuming nondegeneracy

- intersection of n hyperplanes $H_i^=$ is unique point

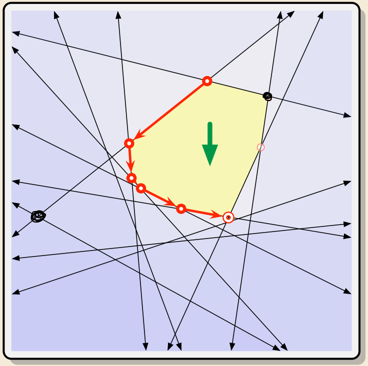
Simplex – Geometric Intuition

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \le b \\ & x \ge 0 \end{aligned}$$
\+ *nondegeneracy*

- constraint $A_{i,\bullet} x \le b_i$ defines a *hyperplane*/*halfspace*

$n = 2, m = 12$

$\rightsquigarrow$ $H_i^= = \{x \in \mathbb{R}^n : A_{i,\bullet} x = b_i\}$
$H_i = \{x \in \mathbb{R}^n : A_{i,\bullet} x \le b_i\}$



- c = **direction** of improvement in $\mathbb{R}^n$ (*normal vector* for hyperplane $\{x \in \mathbb{R}^n : c^T x = 0\}$)
  - *"Roll a ball downhill inside feasible region"*
  - $\rightsquigarrow$ Optimal point x^* must lie on boundary!

    (assuming finite optimal objective value z^*)

assuming nondegeneracy

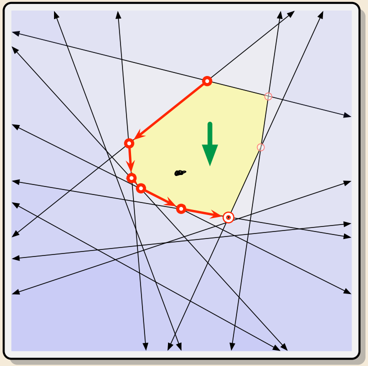
- intersection of n hyperplanes $H_i^=$ is unique point
  - $\rightsquigarrow$ ***vertex*** $\{x_I\} = \bigcap_{i \in I} H_i^=$ (for $I \subset [m]$, $|I| = n$)

Simplex – Geometric Intuition

$$\begin{array}{lrl} \min & \boldsymbol{c}^T\boldsymbol{x} & \\ \text{s.t.} & A\boldsymbol{x} & \leq \boldsymbol{b} \\ & \boldsymbol{x} & \geq \boldsymbol{0} \end{array}$$
+ *nondegeneracy*

- constraint $A_{i,\bullet}\boldsymbol{x} \leq b_i$ defines a *hyperplane*/*halfspace* $\boxed{n = 2, m = 12}$

$\rightsquigarrow$ $H_i^= = \{\boldsymbol{x} \in \mathbb{R}^n : A_{i,\bullet}\boldsymbol{x} = b_i\}$
$H_i = \{\boldsymbol{x} \in \mathbb{R}^n : A_{i,\bullet}\boldsymbol{x} \leq b_i\}$



- $\boldsymbol{c}$ = **direction** of improvement in $\mathbb{R}^n$ (*normal vector* for hyperplane $\{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{c}^T\boldsymbol{x} = 0\}$)
  - *"Roll a ball downhill inside feasible region"*

  $\rightsquigarrow$ Optimal point $\boldsymbol{x}^*$ must lie on boundary!

  (assuming finite optimal objective value z^*)

assuming nondegeneracy

- intersection of n hyperplanes $H_i^=$ is unique point

  $\rightsquigarrow$ ***vertex*** $\{\boldsymbol{x}_I\} = \bigcap_{i \in I} H_i^=$ (for $I \subset [m]$, $|I| = n$)

- always have $\boldsymbol{c}^T\boldsymbol{x}^* = \boldsymbol{c}^T\boldsymbol{x}_{I^*}$ for a **vertex** $\boldsymbol{x}_{I^*}$

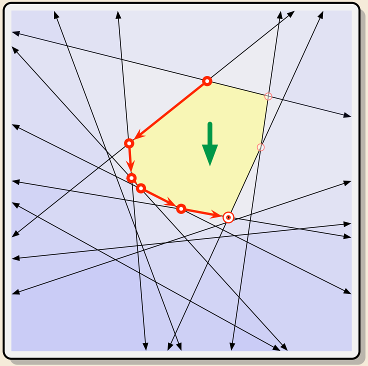
Simplex – Geometric Intuition

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & Ax \le b \\ & x \ge 0 \end{aligned}$$
+ *nondegeneracy*

- constraint $A_{i,\bullet}x \le b_i$ defines a *hyperplane*/*halfspace*

$\boxed{n = 2, m = 12}$

⇝ $H_i^{=} = \{x \in \mathbb{R}^n : A_{i,\bullet}x = b_i\}$
   $H_i = \{x \in \mathbb{R}^n : A_{i,\bullet}x \le b_i\}$



- c = **direction** of improvement in $\mathbb{R}^n$
  (*normal vector* for hyperplane $\{x \in \mathbb{R}^n : c^T x = 0\}$)
  - *"Roll a ball downhill inside feasible region"*

  ⇝ Optimal point x^* must lie on boundary!
  (assuming finite optimal objective value z^*)

assuming nondegeneracy

- intersection of n hyperplanes $H_i^{=}$ is unique point

  ⇝ ***vertex*** $\{x_I\} = \bigcap_{i \in I} H_i^{=}$ (for $I \subset [m]$, $|I| = n$)

- always have $c^T x^* = c^T x_{I^*}$ for a **vertex** x_{I^*}
  - "only" $\binom{m}{n}$ vertices x_I (all n-subsets of $[m]$)

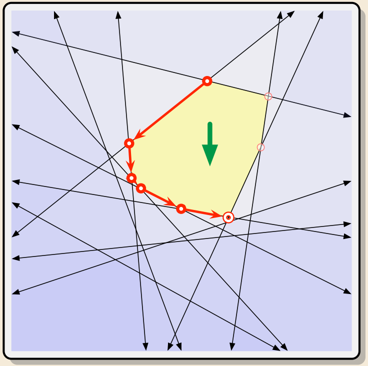
(n const ⇝ polytime brute force)

Simplex – Geometric Intuition

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \le b \\ & x \ge 0 \end{aligned}$$
+ *nondegeneracy*

- constraint $A_{i,\bullet} x \le b_i$ defines a *hyperplane*/*halfspace* $\boxed{n = 2, m = 12}$

⇝ $H_i^= = \{x \in \mathbb{R}^n : A_{i,\bullet} x = b_i\}$
$H_i = \{x \in \mathbb{R}^n : A_{i,\bullet} x \le b_i\}$



- c = **direction** of improvement in $\mathbb{R}^n$ (*normal vector* for hyperplane $\{x \in \mathbb{R}^n : c^T x = 0\}$)
  - *"Roll a ball downhill inside feasible region"*

  ⇝ Optimal point x^* must lie on boundary!

  (assuming finite optimal objective value z^*)

assuming nondegeneracy

- intersection of n hyperplanes $H_i^=$ is unique point

  ⇝ ***vertex*** $\{x_I\} = \bigcap_{i \in I} H_i^=$ (for $I \subset [m]$, $|I| = n$)

- always have $c^T x^* = c^T x_{I^*}$ for a **vertex** x_{I^*}
  - "only" $\binom{m}{n}$ vertices x_I (all n-subsets of $[m]$)

  ⇝ *Simplex algorithm*:
  Move to better neighbor until optimal.
  - x_I and $x_{I'}$ neighbors if $|I \cap I'| = n - 1$

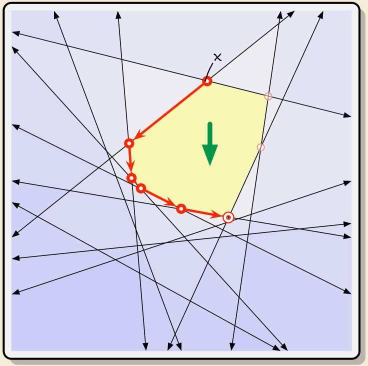
Simplex – Geometric Intuition

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \le b \\ & x \ge 0 \end{array}$$
+ *nondegeneracy*

- constraint $A_{i,\bullet}x \le b_i$ defines a *hyperplane*/*halfspace*

$\boxed{n = 2, m = 12}$

$\rightsquigarrow$ $H_i^= = \{x \in \mathbb{R}^n : A_{i,\bullet}x = b_i\}$
$H_i = \{x \in \mathbb{R}^n : A_{i,\bullet}x \le b_i\}$



- c = **direction** of improvement in $\mathbb{R}^n$ (*normal vector* for hyperplane $\{x \in \mathbb{R}^n : c^T x = 0\}$)
  - *"Roll a ball downhill inside feasible region"*
  - $\rightsquigarrow$ Optimal point x^* must lie on boundary!
    (assuming finite optimal objective value z^*)

assuming nondegeneracy

- intersection of n hyperplanes $H_i^=$ is unique point
  - $\rightsquigarrow$ ***vertex*** $\{x_I\} = \bigcap_{i \in I} H_i^=$ (for $I \subset [m]$, $|I| = n$)
- always have $c^T x^* = c^T x_{I^*}$ for a **vertex** x_{I^*}
  - "only" $\binom{m}{n}$ vertices x_I (all n-subsets of $[m]$)
  - $\rightsquigarrow$ *Simplex algorithm*: Move to better neighbor until optimal.
  - x_I and $x_{I'}$ neighbors if $|I \cap I'| = n - 1$

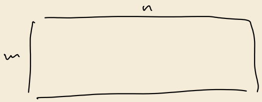
```
procedure simplexIteration(H = {H_1, ..., H_m}):
    if ⋂ H == ∅ return INFEASIBLE
    x := any feasible vertex
    while x is not locally optimal // c "against wall"
        // pivot towards better objective function
        if ∀ feasible neighbor vertex x' : c^T x' > c^T x
            return UNBOUNDED
        else
            x := some feasible lower neighbor of x
    return x
```

Simplex – Linear Algebra Realization

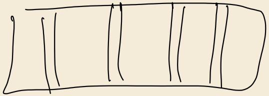
$$\boxed{\begin{aligned} \min\ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.}\ \ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ &+ \textit{nondegeneracy} \end{aligned}}$$

- Here use equality constraints $\rightsquigarrow$ $\underline{m \leq n}$
- Assume $\operatorname{rank}(A) = m$ (nondegeneracy)
- every $J = \{j_1, \ldots, j_m\} \subseteq [n]$ corresponds to *basis* of A: $\{A_{\bullet,j_1}, \ldots, A_{\bullet,j_m}\}$

assuming nondegeneracy



Simplex – Linear Algebra Realization



$$\begin{aligned} \min\ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$
+ *nondegeneracy*

- Here use equality constraints $\rightsquigarrow$ $m \leq n$
- Assume $\operatorname{rank}(A) = m$ (nondegeneracy)
- every $J = \{j_1, \ldots, j_m\} \subseteq [n]$ corresponds to *basis* of A: $\{\boldsymbol{A}_{\bullet,j_1}, \ldots, \boldsymbol{A}_{\bullet,j_m}\}$

  assuming nondegeneracy

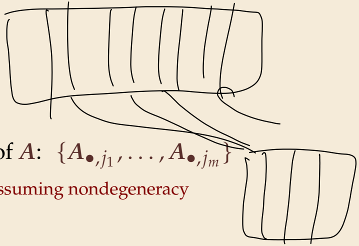
- **Notation:**
  - $\boldsymbol{x}_J = (x_{j_1}, \ldots, x_{j_m})^T$ vector of *basis variables*
  - $\boldsymbol{x}_{\bar{J}} = (x_{\bar{j}_1}, \ldots, x_{\bar{j}_{n-m}})^T$ vector of *non-basis variables* for $\bar{J} = [n] \setminus J = \{\bar{j}_1, \ldots, \bar{j}_{n-m}\}$

Simplex – Linear Algebra Realization

$$\begin{array}{rrcl} \min & \boldsymbol{c}^T\boldsymbol{x} & & \\ \text{s. t.} & A\boldsymbol{x} & = & \boldsymbol{b} \\ & \boldsymbol{x} & \geq & \boldsymbol{0} \end{array}$$
+ nondegeneracy

- Here use equality constraints $\rightsquigarrow$ $m \leq n$
- Assume $\operatorname{rank}(A) = m$ (nondegeneracy)
- every $J = \{j_1, \ldots, j_m\} \subseteq [n]$ corresponds to *basis* of A: $\{\boldsymbol{A}_{\bullet,j_1}, \ldots, \boldsymbol{A}_{\bullet,j_m}\}$

assuming nondegeneracy



- **Notation:**
  - $\boldsymbol{x}_J = (x_{j_1}, \ldots, x_{j_m})^T$ vector of *basis variables*
  - $\boldsymbol{x}_{\bar{J}} = (x_{\bar{j}_1}, \ldots, x_{\bar{j}_{n-m}})^T$ vector of *non-basis variables* for $\bar{J} = [n] \setminus J = \{\bar{j}_1, \ldots, \bar{j}_{n-m}\}$
  - $\boldsymbol{A}_J = (\boldsymbol{A}_{\bullet,j_1}, \ldots, \boldsymbol{A}_{\bullet,j_m}) \in \mathbb{R}^{m\times m}$; similarly $\boldsymbol{A}_{\bar{J}} = (\boldsymbol{A}_{\bullet,\bar{j}_1}, \ldots, \boldsymbol{A}_{\bullet,\bar{j}_{n-m}}) \in \mathbb{R}^{(n-m)\times m}$
  - $\boldsymbol{c}_J$ and $\boldsymbol{c}_{\bar{J}}$ defined similarly

Simplex – Linear Algebra Realization

$$\begin{array}{rrcl} \min & \boldsymbol{c}^T\boldsymbol{x} & & \\ \text{s.t.} & A\boldsymbol{x} & = & \boldsymbol{b} \\ & \boldsymbol{x} & \geq & \boldsymbol{0} \end{array}$$

+ nondegeneracy

- Here use equality constraints $\rightsquigarrow$ $m \leq n$
- Assume $\operatorname{rank}(A) = m$ (nondegeneracy)
- every $J = \{j_1, \ldots, j_m\} \subseteq [n]$ corresponds to *basis* of A: $\{A_{\bullet,j_1}, \ldots, A_{\bullet,j_m}\}$
  assuming nondegeneracy

- **Notation:**
  - $\boldsymbol{x}_J = (x_{j_1}, \ldots, x_{j_m})^T$ vector of *basis variables*
  - $\boldsymbol{x}_{\bar{J}} = (x_{\bar{j}_1}, \ldots, x_{\bar{j}_{n-m}})^T$ vector of *non-basis variables* for $\bar{J} = [n] \setminus J = \{\bar{j}_1, \ldots, \bar{j}_{n-m}\}$
  - $A_J = (A_{\bullet,j_1}, \ldots, A_{\bullet,j_m}) \in \mathbb{R}^{m \times m}$; similarly $A_{\bar{J}} = (A_{\bullet,\bar{j}_1}, \ldots, A_{\bullet,\bar{j}_{n-m}}) \in \mathbb{R}^{(n-m)\times m}$
  - $\boldsymbol{c}_J$ and $\boldsymbol{c}_{\bar{J}}$ defined similarly

square & full rank

$\rightsquigarrow$ We have $A\boldsymbol{x} = \boldsymbol{b} \iff A_J\boldsymbol{x}_J + A_{\bar{J}}\boldsymbol{x}_{\bar{J}} = \boldsymbol{b} \iff \boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$

$\boldsymbol{x}_J$ is uniquely determined by choosing $\boldsymbol{x}_{\bar{J}}$

Simplex – Linear Algebra Realization

$$\begin{aligned} \min\ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s. t.}\ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

+ *nondegeneracy*

- Here use equality constraints $\rightsquigarrow$ $m \leq n$
- Assume $\operatorname{rank}(A) = m$ (nondegeneracy)
- every $J = \{j_1, \ldots, j_m\} \subseteq [n]$ corresponds to *basis* of A: $\{\boldsymbol{A}_{\bullet,j_1}, \ldots, \boldsymbol{A}_{\bullet,j_m}\}$

  assuming nondegeneracy

- **Notation:**
  - $\boldsymbol{x}_J = (x_{j_1}, \ldots, x_{j_m})^T$ vector of *basis variables*
  - $\boldsymbol{x}_{\bar{J}} = (x_{\bar{j}_1}, \ldots, x_{\bar{j}_{n-m}})^T$ vector of *non-basis variables* for $\bar{J} = [n] \setminus J = \{\bar{j}_1, \ldots, \bar{j}_{n-m}\}$
  - $A_J = (\boldsymbol{A}_{\bullet,j_1}, \ldots, \boldsymbol{A}_{\bullet,j_m}) \in \mathbb{R}^{m \times m}$; similarly $A_{\bar{J}} = (\boldsymbol{A}_{\bullet,\bar{j}_1}, \ldots, \boldsymbol{A}_{\bullet,\bar{j}_{n-m}}) \in \mathbb{R}^{(n-m)\times m}$
  - $\boldsymbol{c}_J$ and $\boldsymbol{c}_{\bar{J}}$ defined similarly

square & full rank

$\rightsquigarrow$ We have $A\boldsymbol{x} = \boldsymbol{b} \iff A_J\boldsymbol{x}_J + A_{\bar{J}}\boldsymbol{x}_{\bar{J}} = \boldsymbol{b} \iff \boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$

$\boldsymbol{x}_J$ is uniquely determined by choosing $\boldsymbol{x}_{\bar{J}}$

- *basic solution* setting $\boldsymbol{x}_{\bar{J}} = \boldsymbol{0}$ gives $\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b}$ $\rightsquigarrow$ correspond to *vertices* from before
  - may or may not be a *feasible basic solution*: $\boldsymbol{x}_J \geq \boldsymbol{0}$?

$\rightsquigarrow$ given J, can easily compute basic solution and check feasibility

Simplex – Local Optimality Test

- basic solution: $\boxed{x_J = A_J^{-1}b - A_J^{-1}A_{\bar{J}}x_{\bar{J}}}$ and $x_{\bar{J}} = \mathbf{0}$

$$\begin{array}{rrcl} \min & c^T x & & \\ \text{s. t.} & Ax & = & b \\ & x & \geq & \mathbf{0} \end{array}$$

+ *nondegeneracy*

Simplex – Local Optimality Test

$$\begin{array}{|lrcl|} \hline \min & \boldsymbol{c}^T \boldsymbol{x} & & \\ \text{s.t.} & A\boldsymbol{x} & = & \boldsymbol{b} \\ & \boldsymbol{x} & \ge & \boldsymbol{0} \\ \multicolumn{4}{|l|}{+ \textit{ nondegeneracy}} \\ \hline \end{array}$$

- basic solution: $\boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$ and $\boldsymbol{x}_{\bar{J}} = \boldsymbol{0}$
- How to locally modify basic solution without violating constraints?
  - can't change x_{j_k} for $j_k \in J$ (equality constraint);
  - can't *decrease* $x_{\bar{j}_k}$ for $\bar{j}_k \in \bar{J}$ (nonnegativity);

  $\rightsquigarrow$ can only increase $x_{\bar{j}_k}$ by small $\delta > 0$

Simplex – Local Optimality Test

- basic solution: $\boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$ and $\boldsymbol{x}_{\bar{J}} = \boldsymbol{0}$

$$\min \ \boldsymbol{c}^T\boldsymbol{x} \quad \text{s.t.} \ A\boldsymbol{x} = \boldsymbol{b}, \ \boldsymbol{x} \ge \boldsymbol{0}$$
+ *nondegeneracy*

- How to locally modify basic solution without violating constraints?
  - can't change x_{j_k} for $j_k \in J$ (equality constraint);
  - can't *decrease* $x_{\bar{j}_k}$ for $\bar{j}_k \in \bar{J}$ (nonnegativity);
  - $\rightsquigarrow$ can only increase $x_{\bar{j}_k}$ by small $\delta > 0$

- rewrite cost: $\boldsymbol{c}^T\boldsymbol{x} = \boldsymbol{c}_J\boldsymbol{x}_J + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}}$

Simplex – Local Optimality Test

$$\begin{aligned} \min\ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \end{aligned}$$
+ *nondegeneracy*

- basic solution: $\boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$ and $\boldsymbol{x}_{\bar{J}} = \boldsymbol{0}$
- How to locally modify basic solution without violating constraints?
  - can't change x_{j_k} for $j_k \in J$ (equality constraint);
  - can't *decrease* $x_{\bar{j}_k}$ for $\bar{j}_k \in \bar{J}$ (nonnegativity);
  - ⇝ can only increase $x_{\bar{j}_k}$ by small $\delta > 0$
- rewrite cost:

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} &= \boldsymbol{c}_J\boldsymbol{x}_J + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \\ &= \boldsymbol{c}_J\left(A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}\right) + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \end{aligned}$$

Simplex – Local Optimality Test

$$\begin{aligned} \min \ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s. t.} \ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$
+ *nondegeneracy*

- basic solution: $\boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$ and $\boldsymbol{x}_{\bar{J}} = \boldsymbol{0}$
- How to locally modify basic solution without violating constraints?
  - can't change x_{j_k} for $j_k \in J$ (equality constraint);
  - can't *decrease* $x_{\bar{j}_k}$ for $\bar{j}_k \in \bar{J}$ (nonnegativity);
  - ⇝ can only increase $x_{\bar{j}_k}$ by small $\delta > 0$
- rewrite cost:

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} &= \boldsymbol{c}_J\boldsymbol{x}_J + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \\ &= \boldsymbol{c}_J\left(A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}\right) + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \\ &= \boldsymbol{c}_J A_J^{-1}\boldsymbol{b} + \underbrace{\left(\boldsymbol{c}_{\bar{J}}^T - \boldsymbol{c}_J A_J^{-1}A_{\bar{J}}\right)}_{\tilde{\boldsymbol{c}}_{\bar{J}}^T}\boldsymbol{x}_{\bar{J}} \end{aligned}$$

Simplex – Local Optimality Test

$$\begin{aligned} \min\ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \end{aligned}$$
+ *nondegeneracy*

- basic solution: $\boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$ and $\boldsymbol{x}_{\bar{J}} = \boldsymbol{0}$
- How to locally modify basic solution without violating constraints?
  - can't change x_{j_k} for $j_k \in J$ (equality constraint);
  - can't *decrease* $x_{\bar{j}_k}$ for $\bar{j}_k \in \bar{J}$ (nonnegativity);
  - ⇝ can only increase $x_{\bar{j}_k}$ by small $\delta > 0$
- rewrite cost:

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} &= \boldsymbol{c}_J\boldsymbol{x}_J + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \\ &= \boldsymbol{c}_J\left(A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}\right) + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \\ &= \boldsymbol{c}_JA_J^{-1}\boldsymbol{b} + \underbrace{\left(\boldsymbol{c}_{\bar{J}}^T - \boldsymbol{c}_JA_J^{-1}A_{\bar{J}}\right)}_{\tilde{\boldsymbol{c}}_{\bar{J}}^T}\boldsymbol{x}_{\bar{J}} \end{aligned}$$

Convex function over a convex domain
⇝ local opt $\Longrightarrow$ global opt

⇝ **No** (local) improvement possible $\iff$ $\tilde{\boldsymbol{c}}_{\bar{J}} \ge 0$ $\iff$ current basic solution **optimal**

Simplex – Local Optimality Test

$$\begin{aligned} \min \ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \end{aligned}$$
+ *nondegeneracy*

- basic solution: $\boxed{\boldsymbol{x}_J = A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}}$ and $\boldsymbol{x}_{\bar{J}} = \boldsymbol{0}$
- How to locally modify basic solution without violating constraints?
  - can't change x_{j_k} for $j_k \in J$ (equality constraint);
  - can't *decrease* $x_{\bar{j}_k}$ for $\bar{j}_k \in \bar{J}$ (nonnegativity);
  - ⇝ can only increase $x_{\bar{j}_k}$ by small $\delta > 0$
- rewrite cost:

$$\begin{aligned} \boldsymbol{c}^T\boldsymbol{x} &= \boldsymbol{c}_J\boldsymbol{x}_J + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \\ &= \boldsymbol{c}_J\left(A_J^{-1}\boldsymbol{b} - A_J^{-1}A_{\bar{J}}\boldsymbol{x}_{\bar{J}}\right) + \boldsymbol{c}_{\bar{J}}^T\boldsymbol{x}_{\bar{J}} \\ &= \boldsymbol{c}_J A_J^{-1}\boldsymbol{b} + \underbrace{\left(\boldsymbol{c}_{\bar{J}}^T - \boldsymbol{c}_J A_J^{-1}A_{\bar{J}}\right)}_{\tilde{\boldsymbol{c}}_{\bar{J}}^T}\boldsymbol{x}_{\bar{J}} \end{aligned}$$

Convex function over a convex domain
⇝ local opt $\Longrightarrow$ global opt

⇝ **No** (local) improvement possible $\iff$ $\tilde{\boldsymbol{c}}_{\bar{J}} \ge 0$ $\iff$ current basic solution **optimal**

- Otherwise: Bring $\bar{j}_k$ with $\tilde{\boldsymbol{c}}_{\bar{j}_k} < 0$ into basis
  - This means we increase $x_{\bar{j}_k}$ as much as possible until some x_{j_k} becomes 0
  - ⇝ corresponds to moving to neighbor vertex

Summary LP Algorithms

- **Simplex Algorithm**
  - 👍 simple and mostly combinatorial algorithm
  - 👍 easy to implement
  - 👍 usually fast in practice (in most open source solvers)

Summary LP Algorithms

- **Simplex Algorithm**
  - 👍 simple and mostly combinatorial algorithm
  - 👍 easy to implement
  - 👍 usually fast in practice (in most open source solvers)
  - 👎 worst case running time actually **exponential**
    details depend on how better neighboring vertex is chosen (*pivoting rule*)
    but no rule known that guarantees polytime
    - 👍 but *smoothed analysis* proves: random perturbations of input yield expected polytime on any input

Summary LP Algorithms

- **Simplex Algorithm**
  - 👍 simple and mostly combinatorial algorithm
  - 👍 easy to implement
  - 👍 usually fast in practice (in most open source solvers)
  - 👎 worst case running time actually **exponential**
    details depend on how better neighboring vertex is chosen (*pivoting rule*)
    but no rule known that guarantees polytime
    - 👍 but *smoothed analysis* proves: random perturbations of input yield expected polytime on any input

- **Alternative methods**
  - **ellipsoid method** (separation-oracle based)
  - **interior-point methods** (numeric algorithms)
  - 👍 worst case polytime
  - 👍 interior-point method fastest in practice
  - 👎 more complicated, harder to implement well

6.4 Integer Linear Programs

When LPs Are Too Smooth

- Many natural optimization problems have linear objective and constraints
  - Example: **The Knapsack Problem**

Given: items $1, \ldots, n$ with weights $\boldsymbol{w} \in \mathbb{N}^n$ and values $\boldsymbol{v} \in \mathbb{N}^n$
knapsack weight capacity $b \in \mathbb{N}$

Goal: Select subset of items of maximal total value, subject to fitting in the knapsack

$\rightsquigarrow$ Introduce variable x_i, such that "item included" iff $x_1 = 1$

$$\begin{aligned} \max \quad & \boldsymbol{v}^T \boldsymbol{x} \\ \text{s. t.} \quad & \boldsymbol{w}^T \boldsymbol{x} \le b \\ & \boldsymbol{x} \le \mathbf{1} \\ & \boldsymbol{x} \ge \mathbf{0} \end{aligned} \qquad \text{(Knapsack)}$$

When LPs Are Too Smooth

- Many natural optimization problems have linear objective and constraints
  - Example: **The Knapsack Problem**

    Given: items $1, \ldots, n$ with weights $\boldsymbol{w} \in \mathbb{N}^n$ and values $\boldsymbol{v} \in \mathbb{N}^n$
    knapsack weight capacity $b \in \mathbb{N}$

    Goal: Select subset of items of maximal total value, subject to fitting in the knapsack

  $\rightsquigarrow$ Introduce variable x_i, such that "item included" iff $x_1 = 1$

$$\begin{aligned} \max \quad & \boldsymbol{v}^T \boldsymbol{x} \\ \text{s. t.} \quad & \boldsymbol{w}^T \boldsymbol{x} \le b \\ & \boldsymbol{x} \le \mathbf{1} \\ & \boldsymbol{x} \ge \mathbf{0} \end{aligned} \qquad \text{(Knapsack)}$$

- via LP solvers, we obtain exact worst-case polytime algorithms

When LPs Are Too Smooth

- Many natural optimization problems have linear objective and constraints
  - Example: **The Knapsack Problem**

    Given: items $1, \ldots, n$ with weights $\boldsymbol{w} \in \mathbb{N}^n$ and values $\boldsymbol{v} \in \mathbb{N}^n$
    knapsack weight capacity $b \in \mathbb{N}$

    Goal: Select subset of items of maximal total value, subject to fitting in the knapsack

  ⇝ Introduce variable x_i, such that "item included" iff $x_1 = 1$

$$
\begin{aligned}
\max \quad & \boldsymbol{v}^T \boldsymbol{x} \\
\text{s. t.} \quad & \boldsymbol{w}^T \boldsymbol{x} \le b \\
& \boldsymbol{x} \le \mathbf{1} \\
& \boldsymbol{x} \ge \mathbf{0}
\end{aligned}
\qquad \text{(Knapsack)}
$$

- via LP solvers, we obtain exact worst-case polytime algorithms

- Hold on; where's the catch?
  These problems are NP-hard; so there must be something wrong?

When LPs Are Too Smooth

- Many natural optimization problems have linear objective and constraints
  - Example: **The Knapsack Problem**

    Given: items $1, \ldots, n$ with weights $\boldsymbol{w} \in \mathbb{N}^n$ and values $\boldsymbol{v} \in \mathbb{N}^n$
    knapsack weight capacity $b \in \mathbb{N}$

    Goal: Select subset of items of maximal total value, subject to fitting in the knapsack

  ⇝ Introduce variable x_i, such that "item included" iff $x_1 = 1$

$$\begin{aligned} \max \quad & \boldsymbol{v}^T \boldsymbol{x} \\ \text{s. t.} \quad & \boldsymbol{w}^T \boldsymbol{x} \le b \\ & \boldsymbol{x} \le \mathbf{1} \\ & \boldsymbol{x} \ge \mathbf{0} \end{aligned} \qquad \text{(Knapsack)}$$

- via LP solvers, we obtain exact worst-case polytime algorithms

- Hold on; where's the catch?
  These problems are NP-hard; so there must be something wrong?

ϟ Integrality! Optimal fractional Knapsack $\boldsymbol{x}^*$ can be nonsensical:
Could have $x_i = \frac{1}{2}$ for a single high-value item of weight $2b$, etc.

Integer Linear Programs

- A *(mixed)* ***integer*** *linear program* (ILP/IP resp. MILP) is a linear program, where (some) variables are constrained to integers, $x_i \in \mathbb{Z}$.
  - focus here on the case that all variables are integral: $\boldsymbol{x} \in \mathbb{Z}^n$

$$\boxed{\begin{array}{lrcll} \min & \boldsymbol{c}^T\boldsymbol{x} & & & \\ \text{s.t.} & A\boldsymbol{x} & \le & \boldsymbol{b} & \qquad \text{(ILP)} \\ & \boldsymbol{x} & \ge & \boldsymbol{0} & \\ & \boldsymbol{x} & \in & \mathbb{Z}^n & \end{array}}$$

Integer Linear Programs

- A *(mixed)* ***integer** linear program* (ILP/IP resp. MILP) is a linear program, where (some) variables are constrained to integers, $x_i \in \mathbb{Z}$.
  - focus here on the case that all variables are integral: $\boldsymbol{x} \in \mathbb{Z}^n$

$$\begin{aligned} \min\ & \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} \le \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{aligned} \qquad \text{(ILP)}$$

Example: Knapsack

$$\begin{aligned} \max\ & \boldsymbol{v}^T \boldsymbol{x} \\ \text{s.t.}\ & \boldsymbol{w}^T \boldsymbol{x} \le b \\ & \boldsymbol{x} \le \boldsymbol{1} \\ & \boldsymbol{x} \ge \boldsymbol{0} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{aligned} \qquad \text{(Knapsack-ILP)}$$

Integer Linear Programs

- A *(mixed)* ***integer** linear program* (ILP/IP resp. MILP) is a linear program, where (some) variables are constrained to integers, $x_i \in \mathbb{Z}$.
  - focus here on the case that all variables are integral: $\boldsymbol{x} \in \mathbb{Z}^n$

$$\begin{aligned} \min\ \ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.}\ \ & A\boldsymbol{x} \le \boldsymbol{b} \\ & \boldsymbol{x} \ge \boldsymbol{0} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{aligned} \qquad \text{(ILP)}$$

Example: Knapsack

$$\begin{aligned} \max\ \ & \boldsymbol{v}^T\boldsymbol{x} \\ \text{s.t.}\ \ & \boldsymbol{w}^T\boldsymbol{x} \le b \\ & \boldsymbol{x} \le \boldsymbol{1} \\ & \boldsymbol{x} \ge \boldsymbol{0} \\ & \boldsymbol{x} \in \mathbb{Z}^n \end{aligned} \qquad \text{(Knapsack-ILP)}$$

intersection of halfspaces

$\rightsquigarrow$ feasibility region of an LP is a *polyhedron* $P = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \le \boldsymbol{b}, \boldsymbol{x} \ge \boldsymbol{0}\}$
feasibility region of an ILP is the intersection of P with the integer lattice:
$P_{\mathbb{Z}} = P \cap \mathbb{Z}^n \subset P$

Solving ILP is NP-hard. (0/1 - INTEGER PROGRAMMING)

Integer Linear Programs

- A *(mixed) **integer** linear program* (ILP/IP resp. MILP) is a linear program, where (some) variables are constrained to integers, $x_i \in \mathbb{Z}$.
  - focus here on the case that all variables are integral: $\boldsymbol{x} \in \mathbb{Z}^n$

$$\begin{array}{lrcl} \min & \boldsymbol{c}^T\boldsymbol{x} & & \\ \text{s.t.} & A\boldsymbol{x} & \leq & \boldsymbol{b} \\ & \boldsymbol{x} & \geq & \boldsymbol{0} \\ & \boldsymbol{x} & \in & \mathbb{Z}^n \end{array} \qquad \text{(ILP)}$$

Example: Knapsack

$$\begin{array}{lrcl} \max & \boldsymbol{v}^T\boldsymbol{x} & & \\ \text{s.t.} & \boldsymbol{w}^T\boldsymbol{x} & \leq & b \\ & \boldsymbol{x} & \leq & \boldsymbol{1} \\ & \boldsymbol{x} & \geq & \boldsymbol{0} \\ & \boldsymbol{x} & \in & \mathbb{Z}^n \end{array} \qquad \text{(Knapsack-ILP)}$$

intersection of halfspaces

⇝ feasibility region of an LP is a *polyhedron* $P = \{\boldsymbol{x} \in \mathbb{R}^n : A\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}\}$
feasibility region of an ILP is the intersection of P with the integer lattice:
$\underline{P_{\mathbb{Z}} = P \cap \mathbb{Z}^n \subset P}$

⇝ Still get a lower bound on objective value

$\boxed{\text{optimal objective value of LP} \ \leq \ \text{optimal objective value of ILP}}$

LP Relaxations

- Given a combinatorial optimization problem as ILP,
  its *LP relaxation* is the LP obtained by dropping all integrality constraints.

Idea: cheap way to get lower bounds for optimal value

LP Relaxations

- Given a combinatorial optimization problem as ILP,
  its *LP relaxation* is the LP obtained by dropping all integrality constraints.

- **Example:** Independent Set
  - Given: $G = (V, E)$
    Goal: Maximum-cardinality independent set
  - Introduce variable $x_v \in \{0,1\}$ for $v \in V$

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad x_v + x_w &\le 1 \qquad (\forall vw \in E) \qquad \text{(IS-ILP)} \\ x_v &\in \{0,1\} \quad (\forall v \in V) \end{aligned}$$

$$\begin{aligned} \max \quad & \sum_{v \in V} x_v \\ \text{s.t.} \quad x_v + x_w &\le 1 \qquad (\forall vw \in E) \qquad \text{(IS-LP)} \\ 0 \le x_v &\le 1 \qquad (\forall v \in V) \end{aligned}$$

Integrality Gap

- The ratio $\frac{z^*_{\text{ILP}}}{z^*_{\text{LP}}}$ is called the *integrality gap* of an LP relaxation.

(can also refer to integrality gap of a problem

Integrality Gap

- The ratio $\dfrac{z^*_{\mathrm{ILP}}}{z^*_{\mathrm{LP}}}$ is called the *integrality gap* of an LP relaxation.
  - Hessy James's apple trees: use 533 instead of $533.33\ldots$ trees

  $\rightsquigarrow$ actual profit € 155 415 instead of € 155 500 $\rightsquigarrow$ minuscule difference

Integrality Gap

- The ratio $\dfrac{z^*_{\mathrm{ILP}}}{z^*_{\mathrm{LP}}}$ is called the *integrality gap* of an LP relaxation.
  - Hessy James's apple trees: use 533 instead of $533.33\ldots$ trees

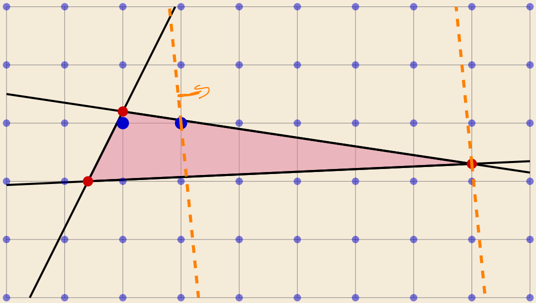
    $\rightsquigarrow$ actual profit €155 415 instead of €155 500 $\rightsquigarrow$ minuscule difference
  - If integrality gap is small, can potentially use LP for approximate solutions $\rightsquigarrow$ Unit 12

Integrality Gap

- The ratio $\dfrac{z^*_{\mathrm{ILP}}}{z^*_{\mathrm{LP}}}$ is called the *integrality gap* of an LP relaxation.
  - Hessy James's apple trees: use 533 instead of $533.33\ldots$ trees

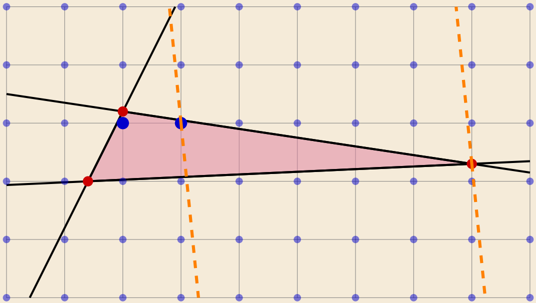
  $\rightsquigarrow$ actual profit € 155 415 instead of € 155 500 $\rightsquigarrow$ minuscule difference
  - If integrality gap is small, can potentially use LP for approximate solutions $\rightsquigarrow$ Unit 12

- in the worst case, integrality gap can be bad



Integrality Gap

- The ratio $\frac{z^*_{\text{ILP}}}{z^*_{\text{LP}}}$ is called the *integrality gap* of an LP relaxation.
  - Hessy James's apple trees: use 533 instead of 533.33 . . . trees
  - ⇝ actual profit € 155 415 instead of € 155 500 ⇝ minuscule difference
  - If integrality gap is small, can potentially use LP for approximate solutions ⇝ Unit 12
- in the worst case, integrality gap can be bad



- actual example: Independent Set
  - Consider complete graph $G = K_n$
  - Largest independent set is single vertex ⇝ $z^*_{\text{ILP}} = 1$
  - Fractional solution possible with $z^*_{\text{LP}} = n/2$ by setting all $x_v = \frac{1}{2}$
  - ⇝ unbounded integrality gap

6.5 LP-Based Kernelization

Vertex Cover as (Integer) Linear Program

Consider optimization version of VERTEXCOVER:
Given: Graph $G = (V, E)$
Goal: Vertex cover of G with minimal cardinality.

solvable in $O(1.3^k n^c)$

$O(k^2)$ kernel

Vertex Cover as (Integer) Linear Program

Consider optimization version of VertexCover:
Given: Graph $G = (V, E)$
Goal: Vertex cover of G with minimal cardinality.

$\rightsquigarrow$ equivalent to the following integer linear program

$$\begin{aligned} &\min \textstyle\sum_{v \in V} x_v \\ &\text{s. t. } x_u + x_v \ge 1 \quad \text{for all } \{u, v\} \in E \\ &\qquad x_v \in \{0, 1\} \quad \text{for all } v \in V \end{aligned}$$

(indp. set
max $\sum x_v$
s.t. $x_u + x_v \le 1$)

Vertex Cover as (Integer) Linear Program

Consider optimization version of VertexCover:
Given: Graph $G = (V, E)$
Goal: Vertex cover of G with minimal cardinality.

$\rightsquigarrow$ equivalent to the following integer linear program

$$\begin{aligned} &\min \textstyle\sum_{v \in V} x_v \\ &\text{s. t. } x_u + x_v \geq 1 \quad \text{for all } \{u, v\} \in E \\ &\qquad x_v \in \{0, 1\} \quad \text{for all } v \in V \end{aligned}$$

Consider *relaxation* to $x_v \in \mathbb{R}$, $x_v \geq 0$.
$\rightsquigarrow$ LP that can by solved in polytime.

"trick" for LPs:
grow the feasibility region
in a way that never changes
z^* or $\{x^* : c^T x^* = z^*\}$

Vertex Cover as (Integer) Linear Program

Consider optimization version of VertexCover:
Given: Graph $G = (V, E)$
Goal: Vertex cover of G with minimal cardinality.

$\rightsquigarrow$ equivalent to the following integer linear program

$$\begin{aligned} &\min \textstyle\sum_{v\in V} x_v \\ &\text{s. t. } x_u + x_v \geq 1 \quad \text{for all } \{u, v\} \in E \\ &\qquad x_v \in \{0, 1\} \quad \text{for all } v \in V \end{aligned}$$

Consider *relaxation* to $x_v \in \mathbb{R}$, $x_v \geq 0$.
$\rightsquigarrow$ LP that can by solved in polytime. any x^* will satisfy $0 \leq x^* \leq 1$

For an *optimal* solution $\vec{x}$ of the *relaxation*, we define

$$\begin{aligned} I_0 &= \{v \in V : x_v < \tfrac{1}{2}\} \\ V_0 &= \{v \in V : x_v = \tfrac{1}{2}\} \\ C_0 &= \{v \in V : x_v > \tfrac{1}{2}\} \end{aligned}$$

Kernel for VC

Theorem 6.1 (Kernel for Vertex Cover)

Let $(G = (V, E), k)$ an instance of p-VERTEX-COVER.

1. There exists a minimal vertex cover S with $C_0 \subseteq S$ and $S \cap I_0 = \emptyset$.
2. V_0 implies a problem kernel $\big(G[V_0], k - |C_0|\big)$ with $|V_0| \le 2k$.

Here $G[V_0]$ is the induced subgraph of V_0 in G.

Proof:

ad (1) Let S^* be optimal VC for G

Claim: $S := (S^* \setminus I_0) \cup C_0$ is also optimal VC

$= (S^* \setminus S_I) \dot\cup \bar{S}_C$ $\qquad S_I = S^* \cap I_0, \quad \bar{S}_C = C_0 \setminus S^*$

"S VC" only edges with endpoints in I_0 could remain uncovered

$e = vw \qquad v \in I_0 \quad \Rightarrow \quad x^*_v < \frac{1}{2} \quad \Rightarrow \quad x^*_w > \frac{1}{2} \quad \Rightarrow \quad w \in C_0$ ✓

(all other edges unchanged

Kernel for VC [2]

Proof (cont.):

"$|S| = |S^*|$" $S_I \subseteq S^*$, $\overline{S_C} \cap S^* = \emptyset$

$$\Rightarrow |S| = |S^*| - |S_I| + |\overline{S_C}|$$

suffices to show that $|\overline{S_C}| \le |S_I|$

$$\varepsilon := \min\{x_v - \tfrac{1}{2} : v \in C_0\} > 0$$

$x' = x^*$ except for

- $u \in S_I$ $x'_u := x^*_u + \varepsilon$
- $u \in \overline{S_C}$ $x'_u := x^*_u - \varepsilon \ge \frac{1}{2}$ (*)

Claim: x' fulfills constraints of LP

$x'_v + x'_w \overset{!}{\ge} 1$ for $vw \in E$ could only be violated for vw with $v \in \overline{S_C}$ ($= C_0 \setminus S^*$)

Kernel for VC [3]

(1) $w \in I_0 \setminus S^*$ $\quad v \notin S^*$ $\quad w \notin S^*$ $\quad \lightning$ S^* VC

Proof (cont.):

(2) $w \in S_I$ $\rightsquigarrow$ $x'_w = x^*_w + \varepsilon$

x^* feasible

$$x'_w + x'_v = x^*_w + x^*_v \geq 1$$

(3) $w \notin I_0$ $\Rightarrow$ $x'_w \geq \frac{1}{2}$, $\quad \underset{(*)}{x'_v \geq \frac{1}{2}}$

$$\Rightarrow x'_w + x'_v \geq 1$$

$\square$ claim

x optimal

$$\sum_v x'_v \geq \sum_v x_v$$

$$\sum_v x_v + \underset{>0}{\varepsilon} \underbrace{\left(|S_I| - |S_C|\right)}_{\geq 0} \Rightarrow |S_C| \leq |S_I|$$

$\square_{(1)}$

ad (2) LP objective $\sum_v x_v^*$ $\leq$ ILP objective for any input

$$\Rightarrow |S^*| \geq \sum_v x_v^*$$

apply reduction rule until only V_0 vertices left:

solve LP relaxation of VC $\leadsto x^* \leadsto I_0, V_0, C_0$

delete vertices in I_0, include in solution from C_0 and delete

$$\Rightarrow x_v^* = \frac{1}{2} \quad \forall v \in V_0$$

$$\Rightarrow |S^*| \geq \sum_v x_v^* = \frac{1}{2} |V_0|$$

If $|V_0| > 2k$ $\Rightarrow$ no VC for size $\leq k$ (output No instance)

otherwise $|V_0| \leq 2k$ $\leadsto$ output as kernel

$\square$.

6.6 Lower Bounds by ETH

so far: W[1]-hardness to show "probably $\notin$ FPT"

how about problems in FPT

can we get lower bounds for $f(k)$ in fpt runtime?

Example: VC allows fpt algorithm

with time $O(\underline{1.4^k}\, n^2)$

can we do better?

unlikely $f(k) = O(k^c)$ for c const

but could be subexponential

$$2^{\sqrt{n}} \qquad 2^{\log^5(n)} = n^{\log^4 n}$$

The Exponential Time Hypothesis

Definition 6.2 (Exponential-Time Hypothesis)

The *Exponential-Time Hypothesis (ETH)* asserts that there is a constant $\varepsilon > 0$ so that every algorithm for p-3SAT requires $\Omega(2^{\varepsilon k})$ time, where k is the number of variables. ◀

variables

$(1+\delta)^k \quad \delta > 0$

The Exponential Time Hypothesis

Definition 6.3 (Exponential-Time Hypothesis)

The *Exponential-Time Hypothesis (ETH)* asserts that there is a constant $\varepsilon > 0$ so that every algorithm for p-3SAT requires $\Omega(2^{\varepsilon k})$ time, where k is the number of variables. ◀

Equivalent formulations:

- There is a $\delta > 0$ so that every 3-SAT algorithm needs $\Omega((1+\delta)^k)$ time.
- There is no $O(2^{o(k)} n^c)$-time algorithm for 3-SAT.
- There is no subexponential-time algorithm for 3-SAT.

Lower Bounds Conditional on ETH

- **Idea:** Show that solving X in time $f(k, n)$
  implies a $O(2^{\varepsilon k} n^c)$ algorithm for 3SAT *for all* $\varepsilon > 0$.

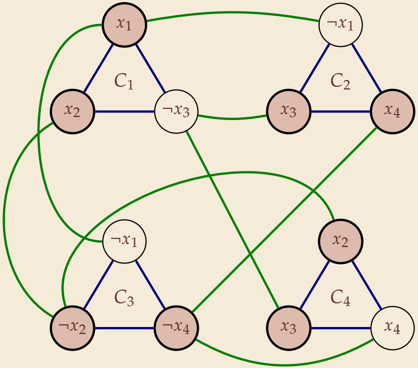
$\rightsquigarrow$ unless ETH false, no such $f(k, n)$-time algorithm for X exists.

- That needs a 3SAT-reduction that preserves parameter k tightly.

Recall: Classical Reduction from 3SAT to Vertex Cover

(ii) 3SAT $\leq_p$ VertexCover – Example

$$\varphi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee x_4)$$



Set S (a VC of size $2n$)

- **Idea:** Vertices ***not in*** vertex cover S define a variable assignment.
  - Cannot be contradictory, otherwise "negation"-edge not covered.
  - Must take ≥ 2 vertices per clause into S (otherwise triangle not covered)
  - $\rightsquigarrow$ $|S| \geq 2n$ for every vertex cover.
- In the example:
  - Fat vertices form a vertex cover for G
  - corresponding assignment: $V = \{x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 0, x_4 \mapsto 1\}$ ($0 \mathrel{\widehat{=}} \mathit{false}, 1 \mathrel{\widehat{=}} \mathit{true}$)
  - $\rightsquigarrow$ φ satisfiable

Sparsification Lemma

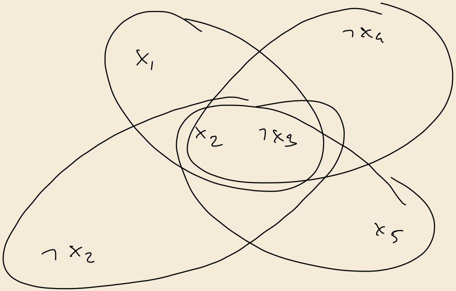
Lemma 6.4 (Sparsification Lemma)

For all $\varepsilon > 0$, there is a constant K so that we can compute for every formula φ in 3-CNF with n clauses over k variables an equivalent formula $\bigvee_{i=1}^{t} \psi_i$ where each ψ_i is in 3-CNF and over the same k variables and has $\leq K \cdot k$ clauses. Moreover, $t \leq 2^{\varepsilon k}$ and the computation takes $O(2^{\varepsilon k} n^c)$ time. ◀

Rough Idea:

Iteratively remove *sunflowers* by retaining only the *heart* or only the *petals*.

Proof in Impagliazzo, Paturi, Zane (2001): *Which Problems Have Strongly Exponential Complexity?*



Lower Bounds – 3SAT [1]

Lemma 6.4 (Sparsification Lemma)

For all $\varepsilon > 0$, there is a constant K so that we can compute for every formula φ in 3-CNF with n clauses over k variables an equivalent formula $\bigvee_{i=1}^{t} \psi_i$ where each ψ_i is in 3-CNF and over the same k variables and has $\leq K \cdot k$ clauses. Moreover, $t \leq 2^{\varepsilon k}$ and the computation takes $O(2^{\varepsilon k} n^c)$ time.

Theorem 6.5 (Lower Bound by Size)

Unless ETH fails, there is a constant $c > 0$ so that every algorithm for p-3SAT needs time $\Omega(2^{c(n+k)})$ where n is the number of clauses and k is the number of variables.

Proof: Assume $\forall c > 0$ A_c is an algorithm that solves p-3SAT in $O(2^{c(n+k)} n^b)$.

Let $\delta > 0$ given. To show: $\exists$ algorithm B_δ that solves p-3SAT in $O(2^{\delta k} \cdot n^{b'})$ $\rightsquigarrow$ $\lightning$ ETH.

Set $\varepsilon = \frac{\delta}{2}$ and let $K = K(\varepsilon)$ the constant from sparsification lemma.

(1) Construct from input φ equiv. formula $\bigvee_{i=1}^{t} \psi_i$ as in " with $\varepsilon = \frac{\delta}{2}$

(2) Call A_c for each ψ_i with $c = \frac{\delta}{2(K+1)}$

(3) Return true iff any ψ_i satisfiable

Lower Bounds – 3SAT [2]

Proof (cont.):

Running time of B_δ $\qquad$ $n_i = |\psi_i| \leq K \cdot k$

(1) $O(2^{\varepsilon k} n^{c'}) = O(2^{\frac{\delta}{2} k} n^{c'})$

(2) $O(\underbrace{t}_{\leq 2^{\varepsilon k}} \cdot 2^{c(n_i+k)} \cdot n^b) = O\Big(2^{\varepsilon k} \cdot 2^{\frac{\delta}{2(K+1)} (\overbrace{n_i}^{\leq K\cdot k} + k)} \cdot n^b\Big)$

$= O(2^{\frac{\delta}{2} k} \cdot 2^{\frac{\delta}{2} k} \cdot n^b) = O(2^{\delta k} \cdot n^b)$

$\Rightarrow$ total time $O(2^{\delta \cdot k} n^{b+c'})$

Lower Bounds – Vertex Cover

the same k variables and has $\le K \cdot k$ clauses. Moreover, ... and the computation takes $O(2^{\varepsilon k} n^c)$ time.

Theorem 6.5 (Lower Bound by Size)
Unless ETH fails, there is a constant $c > 0$ so that every algorithm for p-3SAT needs time $\Omega(2^{c(n+k)})$ where n is the number of clauses and k is the number of variables.

Theorem 6.6 (No Subexponential Algorithm Vertex Cover)

Unless ETH fails, there is a constant $c > 0$ so that every algorithm for p-VERTEX-COVER needs time $\Omega(2^{ck})$.

⇝ Apart from constant basis, exponential dependence on k likely best possible.

Proof: Same template

Assume $\forall c > 0$ A_c is an algorithm that solves p-VC in $O(2^{ck} \cdot n^b)$

$\delta > 0$ given ⇝ construct B_δ solving 3-SAT in time $O(2^{\delta \cdot n} n^{b'})$

(1) Given φ, construct (G,k) using "standard" reduction

(2) Call A_c on (G,k) $\quad c = \frac{\delta}{2}$ ...... $k = 2n$ in

$\Rightarrow B_\delta$ solves 3SAT in $O(2^{ck} \cdot n^b) = O(2^{\frac{\delta}{2} \cdot 2n} n^b) = O(2^{\delta n} n^b)$

Lower Bounds – Closest String

Theorem 6.7 (Lower Bound Closest String)

Unless ETH fails, there is a constant $c > 0$ so that every algorithm for p-Closest-String needs time $\Omega(2^{c(k \lg k)}) = \Omega(k^{ck})$. ◀

Proof omitted.

see Cygan et al. (2015): *Parameterized Algorithms*

⇝ Again, apart from constant in basis, k^k growth in k likely best possible.

Summary

- LPs as a versatile tool
- in particular, give linear-size kernel for p-VERTEXCOVER
- assuming the Exponential Time Hypothesis (instead of only $\mathsf{P} \neq \mathsf{NP}$), can show lower bounds for $f(k)$ part of ant fpt algorithm