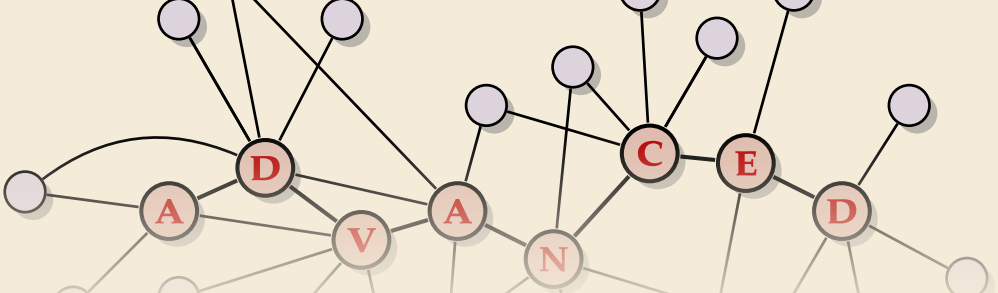


6 Advanced Parameterized Ideas

3 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version 2025-06-04 14:40 H

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 (LinearProgramming $\in$ 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

⇝ 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

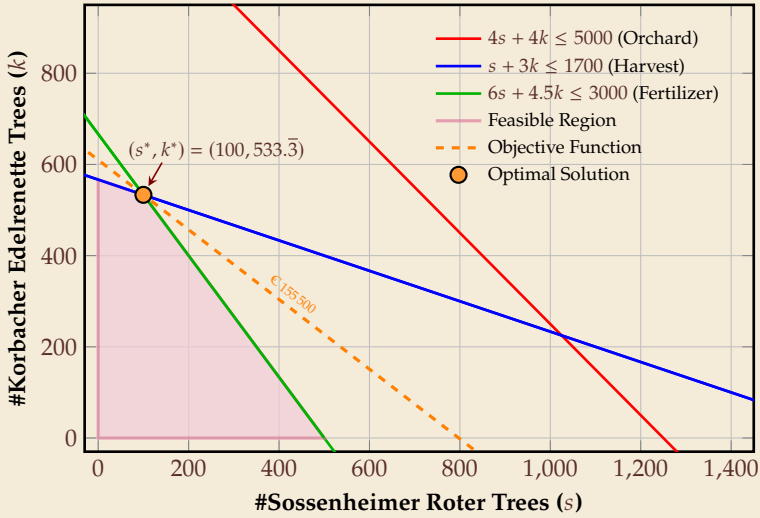
$$
\begin{aligned}
\textbf{Maximize:}\quad & 195s + 255k \\
\textbf{Subject to:}\quad & 4s + 4k \leq 5000 && \text{(Orchard constraint)} \\
& 6s + 4.5k \leq 3000 && \text{(Fertilizer constraint)} \\
& 1s + 3k \leq 1700 && \text{(Harvest constraint)} \\
& s \geq 0 && \text{(Non-negativity)} \\
& k \geq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

name of the LP

- **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



- ⇝ Hessy should plant
  - 100 *Sossenheimer Roter* trees and
  - $533+\frac{1}{3}$ *Korbacher Edelrenette* trees

  hmm . . .
  - Harvest **and** fertilizer *tight*
  - orchard space isn't
- ⇝ know what to change

LPs – The General Case

- General LP:

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

"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 "$-\infty$")

$\rightsquigarrow$ 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)

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

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!

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

⇝ We will briefly visit the conceptual ideas of the simplex algorithm

Recall: General Form of LPs

- General LP:

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

  - 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 ("cost"), coefficients c_j; objective value $z = c_1 x_1 + \cdots c_n x_n$

Enter Linear Algebra

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

- 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**: $\min \boldsymbol{c}^T \cdot \boldsymbol{x}$ (^{T}: transpose; $\cdot$: 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: $\boldsymbol{A}^{(\leq)}\boldsymbol{x} \overset{\text{elementwise } \leq}{\leqslant} \boldsymbol{b}^{(\leq)}$ and $\boldsymbol{A}^{(\geq)}\boldsymbol{x} \geq \boldsymbol{b}^{(\geq)}$

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

(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: $\max c^T x = -\min(-c)^T x$
- "$\geq$"-constraints: $A_{i,\bullet}\, x \geq b_i \iff (-A)_{i,\bullet}\, x \leq -b_i$
- *slack variables*: $A_{i,\bullet}\, x \leq b_i \iff A_{i,\bullet}\, x + x_{s_i} = b_i$ and $x_{s_i} \geq 0$
  (x_{s_i} is a new additional variable)
- *nonnegative*: variable $x_j \leqslant 0 \iff x_j = x_{j,+} - x_{j,-}$ and $x_{j,+}, x_{j,-} \geq 0$
  ($x_{j,+}$ and $x_{j,-}$ are new additional variables)

$\rightsquigarrow$ To solve LPs, can assume one of the following **normal forms**

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax \leq b \\ & x \geq 0 \end{array} \quad \text{or} \quad \begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b \\ & x \geq 0 \end{array} \quad \text{with } A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m, \text{ and } c \in \mathbb{R}^n$$

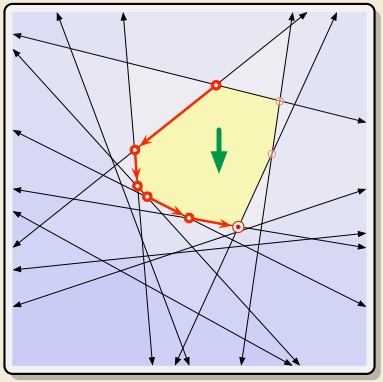
6.3 Linear Programs – The Simplex Algorithm

Simplex – Geometric Intuition

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

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

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



- 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^*)
- intersection of n hyperplanes $H_i^=$ is unique point (assuming nondegeneracy)
  - $\rightsquigarrow$ ***vertex*** $\{x_I\} = \bigcap_{i \in I} H_i^=$ (for $I \subset [m], |I| = n$)
- always have $c^T x^* = c^T \boldsymbol{x_{I^*}}$ 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 $|I \cap I'| = n - 1$

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

Simplex – Linear Algebra Realization

$$\begin{aligned} \min\ & c^T x \\ \text{s.t.}\ & Ax = b \\ & x \geq 0 \end{aligned}$$
+ *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: $\{A_{\bullet,j_1}, \ldots, A_{\bullet,j_m}\}$ (assuming nondegeneracy)

- **Notation:**
  - $x_J = (x_{j_1}, \ldots, x_{j_m})^T$ vector of *basis variables*
  - $x_{\bar{J}} = (x_{\bar{j}_1}, \ldots, x_{\bar{j}_{n-m}})^T$ 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 $A_{\bar{J}} = (A_{\bullet,\bar{j}_1}, \ldots, A_{\bullet,\bar{j}_{n-m}}) \in \mathbb{R}^{(n-m)\times m}$
  - c_J and $c_{\bar{J}}$ defined similarly

$\rightsquigarrow$ We have $Ax = b \iff A_J x_J + A_{\bar{J}} x_{\bar{J}} = b \iff \boxed{x_J = A_J^{-1} b - A_J^{-1} A_{\bar{J}} x_{\bar{J}}}$ (square & full rank)

x_J is uniquely determined by choosing $x_{\bar{J}}$

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

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

Simplex – Local Optimality Test

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

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

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

Convex function over a convex domain $\rightsquigarrow$ local opt $\implies$ global opt

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

- Otherwise: Bring $\bar{j}_k$ with $\tilde{c}_{\bar{j}_k} < 0$ into basis
  - This means we increase $x_{\bar{j}_k}$ as much as possible until some x_{j_k} becomes 0
  - $\rightsquigarrow$ 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)
  - 👎 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 $1, \dots, n$ with weights $w \in \mathbb{N}^n$ and values $v \in \mathbb{N}^n$
    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 $x_1 = 1$

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

- 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?

- Integrality! Optimal fractional Knapsack 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: $x \in \mathbb{Z}^n$

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

Example: Knapsack

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

intersection of halfspaces

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

⇝ Still get a lower bound on objective value

optimal objective value of LP $\leq$ 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.

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

$$\max \sum_{v \in V} x_v$$

$$\text{s.t. } x_v + x_w \leq 1 \quad (\forall vw \in E) \quad \text{(IS-ILP)}$$

$$x_v \in \{0, 1\} \quad (\forall v \in V)$$

$$\max \sum_{v \in V} x_v$$

$$\text{s.t. } x_v + x_w \leq 1 \quad (\forall vw \in E) \quad \text{(IS-LP)}$$

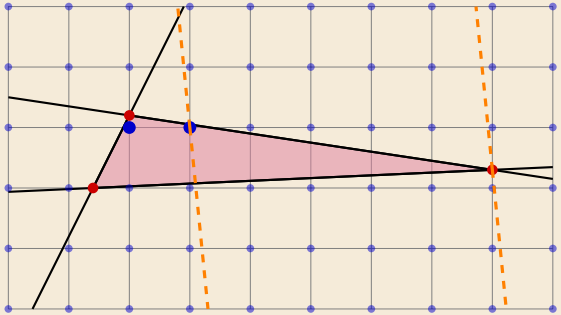
$$0 \leq x_v \leq 1 \quad (\forall v \in V)$$

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.

$\rightsquigarrow$ 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.

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 $|V_0| \leq 2k$.

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

Proof:

Kernel for VC [2]

Proof (cont.):

Kernel for VC [3]

Proof (cont.):

6.6 ETH-Based Lower Bounds

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. ◀

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* $\varepsilon > 0$.

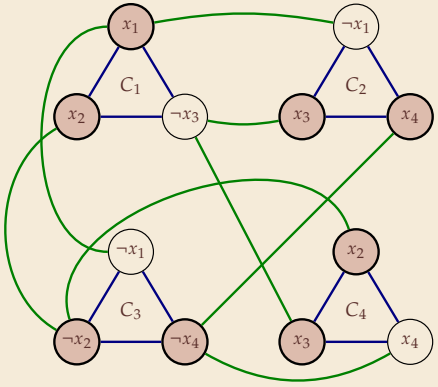
$\rightsquigarrow$ unless ETH false, no such $f(k, n)$-time algorithm for X exists.

- 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: $V = \{x_1 \mapsto 0, x_2 \mapsto 0, x_3 \mapsto 0, x_4 \mapsto 1\}$ ($0 \mathrel{\widehat{=}} \textit{false}, 1 \mathrel{\widehat{=}} \textit{true}$)
  - $\rightsquigarrow \varphi$ satisfiable

Sparsification Lemma

Lemma 6.3 (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]

Theorem 6.4 (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:

Lower Bounds – 3SAT [2]

Proof (cont.):

Lower Bounds – Vertex Cover

Theorem 6.5 (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:

Lower Bounds – Closest String

Theorem 6.6 (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*

$\rightsquigarrow$ 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 P $\neq$ NP), can show lower bounds for $f(k)$ part of any fpt algorithm