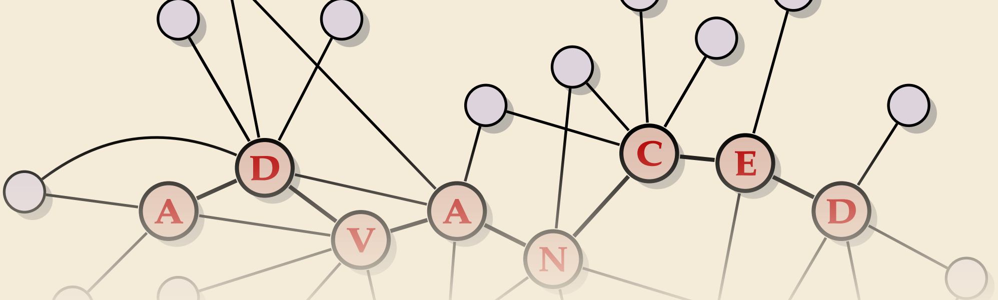


6 Advanced Parameterized Ideas

3 June 2025

Prof. Dr. Sebastian Wild

CS627 (Summer 2025)
Philipps-Universität Marburg
version `2025-06-03 00:28` 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

$\rightsquigarrow$ 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:

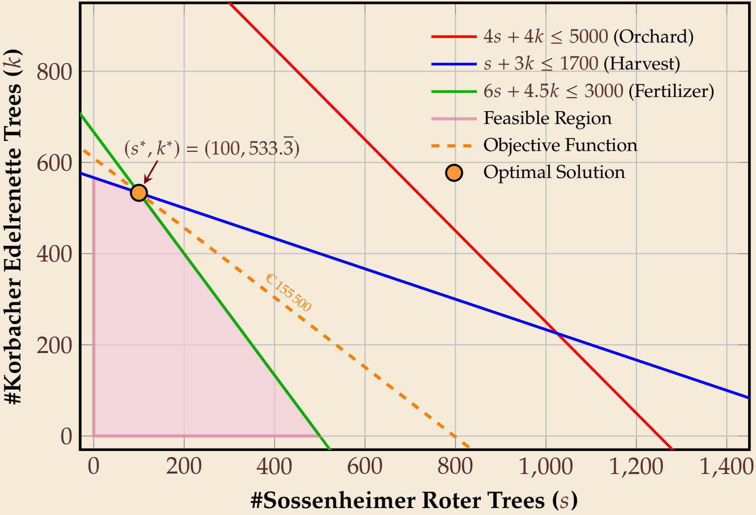
optimization goal — objective function — constraint

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

- **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



⇝ 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:

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

"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)

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

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*

$\rightsquigarrow$ 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 ("=", "≤", or "≥")
  - 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

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

- 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
  - “=”-**constraints**

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

    elementwise $\le$
  - similarly for “$\le$” and “$\ge$” constraints: $A^{(\le)}\boldsymbol{x} \le \boldsymbol{b}^{(\le)}$ and $A^{(\ge)}\boldsymbol{x} \ge \boldsymbol{b}^{(\ge)}$

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

  (generally write $A_{i,\bullet}$ for the ith row of A and $A_{\bullet,j}$ 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: $\boldsymbol{A}_{i,\bullet}\boldsymbol{x} \;\geq\; b_i \quad\iff\quad (-\boldsymbol{A})_{i,\bullet}\boldsymbol{x} \;\leq\; -b_i$
- *slack variables*: $\boldsymbol{A}_{i,\bullet}\boldsymbol{x} \;\leq\; b_i \quad\iff\quad \boldsymbol{A}_{i,\bullet}\boldsymbol{x} + \boldsymbol{x_{s_i}} \;=\; b_i$ and $x_{s_i} \;\geq\; 0$

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

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

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

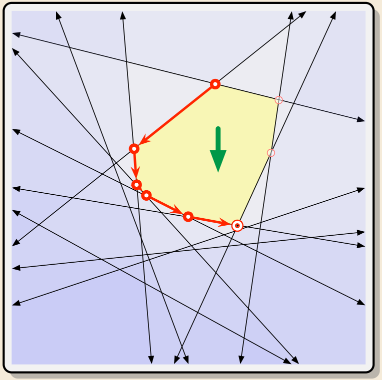
6.3 Linear Programs – The Simplex Algorithm

Simplex – Geometric Intuition

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

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

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



- $\boldsymbol{c}$ = **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 halfspaces 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** $\boldsymbol{x}_{I^*}$
  - "only" $\binom{m}{n}$ vertices $\boldsymbol{x}_I$ (all n-subsets of $[m]$)
  - $\rightsquigarrow$ *Simplex algorithm*: Move to better neighbor until optimal.
  - $\boldsymbol{x}_I$ and $\boldsymbol{x}_{I'}$ 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

$$\begin{aligned} \min\ & \boldsymbol{c}^T\boldsymbol{x} \\ \text{s.t.}\ & A\boldsymbol{x} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$
+ *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 $\bar{J} = [n] \setminus J = \{\bar{j}_1, \ldots, \bar{j}_{n-m}\}$
  - $\boldsymbol{A}_J = (\boldsymbol{A}_{\bullet,j_1}, \ldots, \boldsymbol{A}_{\bullet,j_m}) \in \mathbb{R}^{m\times m}$; similarly $\boldsymbol{A}_{\bar{J}} = (\boldsymbol{A}_{\bullet,\bar{j}_1}, \ldots, \boldsymbol{A}_{\bullet,\bar{j}_{n-m}}) \in \mathbb{R}^{(m-n)\times m}$
  - $\boldsymbol{c}_J$ and $\boldsymbol{c}_{\bar{J}}$ defined similarly

$\rightsquigarrow$ 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

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

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

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 $\tilde{c}_{\bar{j}_k} < 0$ 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)
  - 👎 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**
- 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?

Integer Linear Programs

- x

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 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.

6.6 Lower Bounds by ETH

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. ◀

Alternative formulations:

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

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 fails, no such $f(k, n)$-time algorithm for X exists.

Problem: Need a reduction that preserves parameter k.

Recap: Reduction from 3SAT to Vertex Cover

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*.

Lower Bounds

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. ◀

Lower Bounds [2]

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 $\Omega(2^{ck})$. ◀

Lower Bounds [3]

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 $\Omega(2^{c(k \operatorname{ld} k)}) = \Omega(k^{ck})$. ◀