CS 577 - Intro to Algorithms

Computational Intractability

Dieter van Melkebeek

November 29, 2022

Outline

Motivation
- Recognizing infeasible approaches
- P vs NP problem

Topics
- Classes P and NP
- NP-hardness and NP-completeness

(Seemingly?) Intractable Problems

Independent Set
- Input: graph G
- Output: independent set S of G such that $|S|$ is maximized

Satisfiability
- Input: Boolean formula φ
- Output: satisfying assignment of φ, or report that none exists

More (Seemingly?) Intractable Problems

Graph coloring
- Input: graph $G = (V, E)$
- Output: $c : V \to [k]$ such that $(\forall (u,v) \in E)\, c(u) \neq c(v))$ and k is minimized

Traveling salesperson problem (TSP)
- Input: n cities and direct intercity travel costs for each pair
- Output: tour that visits every city once and has minimum total cost

Subset sum
- Input: $a_1, a_2, \ldots, a_n \in \mathbb{N}$; $t \in \mathbb{N}$
- Output: $I \subseteq [n]$ such that $\sum_{i \in I} a_i = t$, or report impossible

Common Pattern

On input x of length $n \doteq |x|$:

- Candidate solutions can be described by strings $y \in \{0,1\}^*$ with $|y| = n^c$ for some constant c.
- Whether a candidate solution $y \in \{0,1\}^{n^c}$ is valid for input x can be checked in time polynomial in n.

$$V(x,y) = \begin{cases} 1 & \text{if } y \text{ is valid for } x \\ 0 & \text{otherwise} \end{cases}$$

- [In case of optimization problem:]
  Objective $f(x,y)$ can be evaluated in time polynomial in n.

NP Decision/Search/Optimization

- Parameters:
  - $c \in \mathbb{N}$
  - $V : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ computable in polynomial time
  - $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$ computable in polynomial time
- Solution set for input $x \in \{0,1\}^n$:

$$S_x \doteq \{y \in \{0,1\}^{n^c} : V(x,y) = 1\}$$

- Goal:
  - Decision: Is $S_x \neq \emptyset$?
  - Search: Find $y \in S_x$ or report that no such y exists.
  - OptVal: Find $\min_{y \in S_x}(f(x,y))$ respectively $\max_{y \in S_x}(f(x,y))$.
  - OptSol: Find $y^* \in S_x$ such that $f(x,y^*) = \min_{y \in S_x}(f(x,y))$ respectively $f(x,y^*) = \max_{y \in S_x}(f(x,y))$

P vs NP

Definitions

- P: class of decision problems computable in polynomial time.
- NP: class decision problems for which yes-instances have certificates that can be verified in polynomial time, i.e., there exists $c \in \mathbb{N}$ and $V \in \mathrm{P}$ such that

$$x \text{ is yes-instance } \Leftrightarrow (\exists y \in \{0,1\}^{|x|^c})\, V(x,y) = 1.$$

Fact: P $\subseteq$ NP

Open: P = NP?

Conjecture: P $\neq$ NP

Note: Assuming P $\neq$ NP, the "hardest" problems in NP cannot be solved in polynomial time (but some problems in NP can).

NP-Hardness and NP-Completeness

Definition

B is NP-hard if $(\forall\, A \in \mathrm{NP})\, A \leq^p B$.

Definition

B is NP-complete if B is NP-hard and $B \in \mathrm{NP}$.

Proposition

Suppose B is NP-complete. Then $B \in \mathrm{P} \Leftrightarrow \mathrm{P} = \mathrm{NP}$.

Proof

$\Leftarrow$ $B \in \mathrm{NP}$ and $\mathrm{P} = \mathrm{NP}$ implies $B \in \mathrm{P}$.

$\Rightarrow$ Consider any $A \in \mathrm{NP}$.
$A \leq^p B$ and $B \in \mathrm{P}$ implies $A \in \mathrm{P}$.

Corollary

Assume P $\neq$ NP. If B is NP-hard then B cannot be solved in polynomial time.

Existence of NP-Complete Problems

Theorem (next lecture)

CNF-SAT is NP-hard.

Corollary

Independent Set is NP-hard.

Proof

Consider any $A \in \mathrm{NP}$.

- By the NP-hardness of CNF-SAT, $A \leq^p$ CNF-SAT.
- By previous lecture, CNF-SAT $\leq^p$ Independent Set.
- By transitivity, $A \leq^p$ Independent Set.

Proving NP-Hardness

Strategy

To show a new problem C is NP-hard:

- Find a known NP-hard problem B.
- Show that $B \leq^p C$.

Motivation

- Recognizing infeasible approaches.
- Convincing your boss
  See cartoon from: [Garey and Johnson, Computers and Intractability – A guide to the Theory of NP-Completeness]

Prevalence of NP-completeness

- Thousands of problems from all areas of science and engineering have been shown to be NP-complete.
- Considered strong evidence that P $\neq$ NP.
- In fact, almost all of the known problems in NP that are not known to be in P, have been shown to be NP-hard.
- Notorious exceptions include: graph isomorphism, factoring integers.

CS 577 - Intro to Algorithms

Computational Intractability

Dieter van Melkebeek

December 1, 2022

Outline

Motivation
- Recognizing infeasible approaches
- P vs NP problem

P, NP, and NPC
- P: decision problems that have polynomial-time algorithms
- NP: decision problems with yes-instances that have polynomial-time verifiable certificates
- Fact: $\mathrm{P} \subseteq \mathrm{NP}$
- Conjecture: $\mathrm{P} \neq \mathrm{NP}$
- NP-complete (NPC): hardest problems in NP
- Assume $\mathrm{P} \neq \mathrm{NP}$. If $B \in \mathrm{NPC}$ then $B \notin \mathrm{P}$.

Satisfiability: Circuit-SAT, 3-SAT, 2-SAT

NP Decision/Search/Optimization

- Parameters:
  - $c \in \mathbb{N}$
  - $V : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ computable in polynomial time
  - $f : \{0,1\}^* \times \{0,1\}^* \to \mathbb{R}$ computable in polynomial time
- Solution set for input $x \in \{0,1\}^n$:

$$S_x \doteq \{y \in \{0,1\}^{n^c} : V(x,y) = 1\}$$

- Goal:

  Decision: Is $S_x \neq \emptyset$?

  Search: Find $y \in S_x$ or report that no such y exists.

  OptVal: Find $\min_{y \in S_x}(f(x,y))$ respectively $\max_{y \in S_x}(f(x,y))$.

  OptSol: Find $y^* \in S_x$ such that $f(x,y^*) = \min_{y \in S_x}(f(x,y))$ respectively $f(x,y^*) = \max_{y \in S_x}(f(x,y))$

NP-Hardness and NP-Completeness

Definition
B is NP-hard if $(\forall A \in \mathrm{NP})\, A \leq^p B$.

Definition
B is NP-complete if B is NP-hard and $B \in \mathrm{NP}$.

Proposition
Suppose B is NP-complete. Then $B \in \mathrm{P} \Leftrightarrow \mathrm{P} = \mathrm{NP}$.

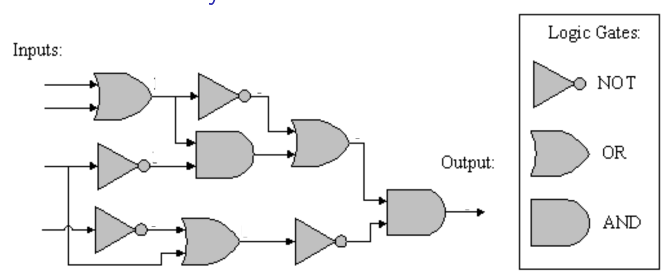
Proof

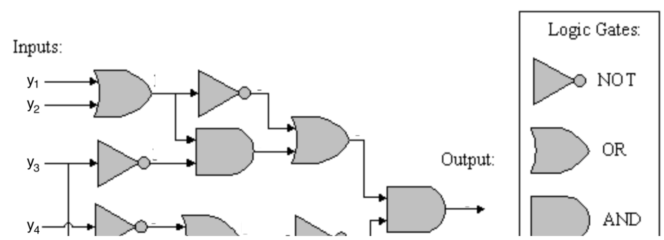
$\Leftarrow$ $B \in \mathrm{NP}$ and $\mathrm{P} = \mathrm{NP}$ implies $B \in \mathrm{P}$.

$\Rightarrow$ Consider any $A \in \mathrm{NP}$.
$A \leq^p B$ and $B \in \mathrm{P}$ implies $A \in \mathrm{P}$.

Corollary
Assume $\mathrm{P} \neq \mathrm{NP}$. If B is NP-hard then $B \notin \mathrm{P}$.

Circuit Satisfiability





Circuit-SAT is NP-Hard

Proof
- Need to show $A \leq^p$ Circuit-SAT for each A in NP.
- A is specified by $c \in \mathbb{N}$ and $V \in \mathrm{P}$ such that $x \in \{0,1\}^n$ is yes-instance $\Leftrightarrow S_x \doteq \{y \in \{0,1\}^{n^c} : V(x,y) = 1\} \neq \emptyset$.

Lemma
A Boolean circuit C_x on $\ell \doteq n^c$ inputs such that $C_x(y) = V(x,y)$ for all $y \in \{0,1\}^\ell$ can be constructed from x in time $n^{O(1)}$.

Mapping reduction

$$\begin{array}{ccl} A & & B = \text{Circuit-SAT} \\ x_A & \to & x_B = C_{x_A} \\ & & \downarrow \text{[blackbox]} \\ y_A = y_B & \leftarrow & y_B \text{ with } C_{x_A}(y_B) = 1 \end{array}$$

Satisfiability

Specification

Input: Boolean formula φ

Output: satisfying assignment of φ, or report that none exists

Restrictions
- CNF-SAT: φ is CNF, i.e., a conjunction of clauses.
  - Clause: disjunction of literals
  - Literal: variable or negated variable
- k-SAT for fixed $k \in \mathbb{N}$: φ is k-CNF, i.e., CNF in which each clause contains at most k literals.

$$\ell_1 \vee \ell_2 \vee \cdots \vee \ell_{k-1} \vee \ell_k \quad \equiv \quad \overline{\ell_1} \wedge \overline{\ell_2} \wedge \cdots \wedge \overline{\ell_{k-1}} \Rightarrow \ell_k$$

Complexity
- 3-SAT is NP-hard.
- 2-SAT can be solved in polynomial time.

Proving NP-Hardness

Strategy

To show a new problem C is NP-hard:

- Find a known NP-hard problem B.
- Show that $B \leq^p C$.

Justification

Consider any $A \in \text{NP}$.

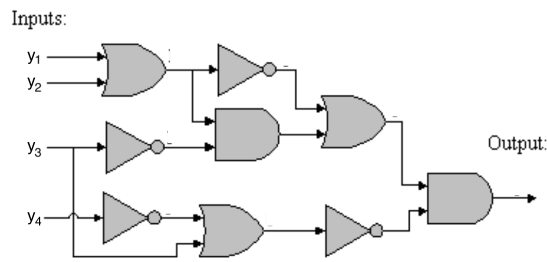
- By the NP-hardness of B, $A \leq^p B$.
- We show that $B \leq^p C$.
- Therefore $A \leq^p B \leq^p C$.
- By transitivity $A \leq^p C$.

3-SAT is NP-Hard

Strategy

Mapping reduction Circuit-SAT $\leq^p$ 3-SAT

Gadget reduction



Reduction Circuit-SAT $\leq^p$ 3-SAT

- Introduce a variable y_i for each input y_i of C, $i \in [\ell]$.
- Introduce a variable g for each gate g of C.
- For each gate g, include clauses with at most 3 literals each that force variable g to value of gate g on input $y_1 \ldots y_\ell$.
  - $g' = \text{ NOT } g \rightarrow \begin{cases} g \Rightarrow \overline{g'} \\ \overline{g} \Rightarrow g' \end{cases} \equiv \begin{cases} \overline{g} \vee \overline{g'} \\ g \vee g' \end{cases}$
  - $g' = g_1 \text{ AND } g_2 \rightarrow \begin{cases} \overline{g_1} \Rightarrow \overline{g'} \\ \overline{g_2} \Rightarrow \overline{g'} \\ g_1 \wedge g_2 \Rightarrow g' \end{cases} \equiv \begin{cases} g_1 \vee \overline{g'} \\ g_2 \vee \overline{g'} \\ \overline{g_1} \vee \overline{g_2} \vee g' \end{cases}$
- Add unit clause consisting of the variable for the output gate.

Correctness

- C has satisfying assignment $\Leftrightarrow$ φ has satisfying assignment.
- Each satisfying assignment for φ includes one for C.

Polynomial running time

2-SAT

$$(x_1 \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2})$$

Digraph representation G

- Introduce a vertex for each variable x_i that occurs in φ, and another one for its negation $\overline{x_i}$.
- Interpret each clause $\ell_1 \vee \ell_2$ as the implications $\overline{\ell_1} \Rightarrow \ell_2$ and $\overline{\ell_2} \Rightarrow \ell_1$.
- Include edges $(\overline{\ell_1}, \ell_2)$ and $(\overline{\ell_2}, \ell_1)$ in G.
- Handle unit clause ℓ as $\ell \vee \ell$.

Symmetry property

$$\ell_1 \rightsquigarrow \ell_2 \text{ in } G \Leftrightarrow \overline{\ell_2} \rightsquigarrow \overline{\ell_1} \text{ in } G$$

Polynomial-Time Algorithm for 2-SAT

Claim

φ has a satisfying assignment
$\Leftrightarrow$ for no variable x_i there are paths $x_i \rightsquigarrow \overline{x_i}$ and $\overline{x_i} \rightsquigarrow x_i$ in G.

Proof

$\Rightarrow$ By contraposition.

$\Leftarrow$ Algorithm for finding satisfying assignment:

while not all variables are assigned **do**
  pick unassigned literal ℓ such that $\ell \not\rightsquigarrow \overline{\ell}$
  for each ℓ' with $\ell \rightsquigarrow \ell'$ **do** $\ell' \leftarrow$ true

Correctness:

- Propagation ensures all clauses are satisfied.
- No conflicts because of hypothesis:
  If $\ell \rightsquigarrow x$ and $\ell \rightsquigarrow \overline{x}$ then $\ell \rightsquigarrow x$ and $x \rightsquigarrow \overline{\ell}$ so $\ell \rightsquigarrow \overline{\ell}$.

Running time: $O(n+m)$ for n variables and m clauses, using linear-time algorithm for finding strongly connected components.

CS 577 - Intro to Algorithms

Computational Intractability

Dieter van Melkebeek

December 6, 2022

Recap

- P: decision problems that have polynomial-time algorithms
- NP: decision problems with yes-instances that have polynomial-time verifiable certificates
- Fact: $\mathrm{P} \subseteq \mathrm{NP}$
- Conjecture: $\mathrm{P} \neq \mathrm{NP}$
- Definition: B is NP-hard if $(\forall A \in \mathrm{NP})\, A \leq^p B$.
- Assume $\mathrm{P} \neq \mathrm{NP}$. If B is NP-hard then $B \notin \mathrm{P}$.
- Theorem: Circuit-SAT is NP-hard.

Establishing NP-Hardness

Strategy

To show a new problem C is NP-hard:

- Find a known NP-hard problem B.
- Show that $B \leq^p C$.

Earlier instantiations

- Circuit-SAT $\leq^p$ 3-SAT
- 3-SAT $\leq^p$ Independent Set

Today's instantiations

- Independent Set $\leq^p$ Clique
- Independent Set $\leq^p$ Vertex Cover
- 3-SAT $\leq^p$ 3-Coloring
- 3-SAT $\leq^p$ Subset Sum

Independent Set vs Clique vs Vertex Cover

Definitions

Fix a graph $G = (V, E)$. A subset $S \subseteq V$ is:

- An independent set if $E \cap S \times S = \emptyset$.
- A clique if $S \times S \subseteq E$.
- A vertex cover if $E \subseteq S \times V$.

Relationships

- S is independent set in $G \Leftrightarrow S$ is clique in $\overline{G} \doteq (V, \overline{E})$.
- S is independent set in $G \Leftrightarrow \overline{S}$ is vertex cover in G.

Corollary

- Independent Set $\equiv^p$ Clique
- Independent Set $\equiv^p$ Vertex Cover

Satisfiability and Coloring

3-SAT

Input: 3-CNF formula φ

E.g.: $\varphi = (x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3)$

Output: whether φ has a satisfying assignment.

3-Coloring

Input: graph $G = (V, E)$

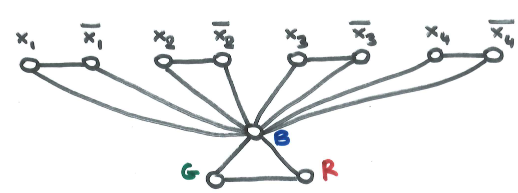
Output: whether G has a 3-coloring, i.e., a mapping $c : V \to [3]$ such that $(\forall (u, v) \in E)\, c(u) \neq c(v)$.

3-SAT $\leq^p$ 3-Coloring – variable gadgets

- Include a color palette: complete graph on vertices $\{\text{red}, \text{green}, \text{blue}\}$
- For each variable x_i, include two new vertices, one labeled x_i and the other $\overline{x_i}$.
- Include the edges $(x_i, \overline{x_i})$, (x_i, blue), and $(\overline{x_i}, \text{blue})$.
- Bijection between assignments to variables $x_1, \ldots, x_n$ and valid colorings with $\{\text{red}, \text{green}, \text{blue}\}$.

3-SAT $\leq^p$ 3-Coloring – variable gadgets

$$(x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3)$$



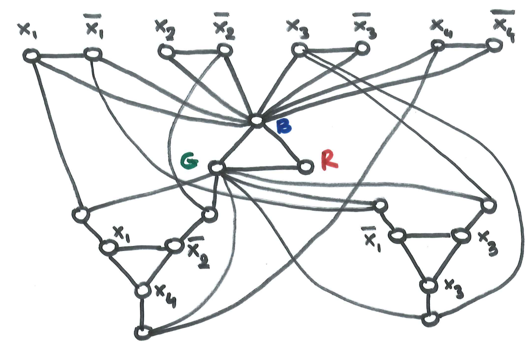
3-SAT $\leq^p$ 3-Coloring – clause gadgets & connections

- For each 3-clause C_j, include a complete graph on 3 new vertices, each labeled with a unique literal of C_j.
- Include for each new vertex v with label ℓ, another new vertex v'.
- Include the edges (v, v'), (v', green), and (v', u), where u denotes the vertex in the variable gadget labeled ℓ.
- A valid 3-coloring to the variable gadget can be extended to gadget for clause C_j iff underlying assignment satisfies C_j.
- Clauses with less than 3 literals can be handled by repeating a literal in the clause until there are three.

Conclusion: φ is satisfiable $\Leftrightarrow$ G is 3-colorable

3-SAT $\leq^p$ 3-Coloring – clause gadgets & connections

$(x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3)$



Satisfiability and Subset Sum

3-SAT

Input: 3-CNF formula φ

E.g.: $\varphi = (x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3)$

Output: whether φ has a satisfying assignment.

Subset Sum

Input: $a_1, a_2, \ldots, a_k \in \mathbb{N}$; $t \in \mathbb{N}$

Output: whether there exists $I \subseteq [k]$ such that $\sum_{i \in I} a_i = t$.

Note: Subset Sum $\leq^p$ Knapsack

3-SAT $\leq^p$ Subset Sum – variable gadgets

- For each variable x_i, include two numbers $a_{2i-1} = a_{2i} = 2^{i-1}$.
- Label a_{2i-1} with x_i, and a_{2i} with $\overline{x_i}$.
- Set $t = \sum_{i=1}^{n} 2^{i-1} = 2^n - 1$.
- Bijection between assignments to variables $x_1, \ldots, x_n$ and subsets $I \subseteq [2n]$ such that $\sum_{i \in I} a_i = t$.

3-SAT $\leq^p$ Subset Sum – variable gadgets

$(x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3)$

PHASE I

BIT POSITIONS

	x_4	x_3	x_2	x_1	
a_1	0	0	0	1	$\sim x_1$
a_2	0	0	0	1	$\sim \overline{x_1}$
a_3	0	0	1	0	$\sim x_2$
a_4	0	0	1	0	$\sim \overline{x_2}$
a_5	0	1	0	0	$\sim x_3$
a_6	0	1	0	0	$\sim \overline{x_3}$
a_7	1	0	0	0	$\sim x_4$
a_8	1	0	0	0	$\sim \overline{x_4}$
t	1	1	1	1	

3-SAT $\leq^p$ Subset Sum – clause gadgets & connections

For each clause C_j with k_j literals:

- Pick bit two new consecutive bit positions B_j.
- Set bits B_j to 01 in each number a_i labeled with literal in C_j.
- Set bits B_j in t equal to k_j (in binary).
- Include $k_j - 1$ new a_i with all bits zero except B_j set to 01.

Claim

- Consider subset $I \subseteq [2n]$ corresponding to assignment to variables $x_1, \ldots, x_n$.
- $\sum_{i \in I} a_i$ agrees with t in last n bit positions.
- I can be extended with subset of new indices to I' such that $\sum_{i \in I'} a_i$ agrees with t in positions B_j $\Leftrightarrow$ underlying assignment satisfies C_j.

Conclusion: φ is satisfiable $\Leftrightarrow$ $(\exists I') \sum_{i \in I'} a_i = t$.

3-SAT $\leq^p$ Subset Sum – clause gadgets & connections

$$(x_1 \vee \overline{x_2} \vee x_4) \wedge (\overline{x_1} \vee x_3)$$

PHASE 2

PHASE 1

BIT POSITIONS

	C_2	C_1	x_4	x_3	x_2	x_1	
a_1	00	01	0	0	0	1	$\sim x_1$
a_2	01	00	0	0	0	1	$\sim \overline{x_1}$
a_3	00	00	0	0	1	0	$\sim x_2$
a_4	00	01	0	0	1	0	$\sim \overline{x_2}$
a_5	01	00	0	1	0	0	$\sim x_3$
a_6	00	00	0	1	0	0	$\sim \overline{x_3}$
a_7	00	01	1	0	0	0	$\sim x_4$
a_8	00	00	1	0	0	0	$\sim \overline{x_4}$
a_9	00	01	0	0	0	0	
a_{10}	00	01	0	0	0	0	
a_{11}	01	00	0	0	0	0	
t	10	11	1	1	1	1	

Classical NP-Complete Problems

- Constraint satisfaction: Circuit-SAT, CNF-SAT, 3-SAT
- Packing: Independent Set, Clique
- Covering: Vertex Cover
- Partitioning: 3-Colorability
- Numerical: Subset Sum, Knapsack
- Sequencing: Traveling Salesperson

Hamiltonicity

Input: (di)graph $G = (V, E)$

Output: whether there exists a (directed) cycle/path that visits every vertex once

CS 577- Intro to Algorithms

Computational Intractability

Dieter van Melkebeek

December 8, 2022

Outline

How to handle NP-complete problems

- Instance structure
- Parameter bounds
- Approximations
- Heuristics

Instance Structure

Idea

Exploit structure of instances that occur in application setting.

Vertex Cover

Can be solved in polynomial time for:

- Trees
- Bipartite graphs
- Interval graphs
- ...

Parameter Bounds

Idea

Exploit bounds on parameters (other than input size) for instances that occur in application setting.

Vertex Cover

Using vertex cover size k as additional parameter:

- Polynomial-time solvable for each fixed k
  - Exhaustively try all $\binom{n}{k} = \Theta(n^k)$ possible subsets of size k.
- Fixed-parameter tractable
  - Running time $O(2^k \cdot (|V| + |E|))$
- Kernelization
  - Kernel consisting of at most k^2 edges

Fixed-Parameter Tractability

Definition
Instances of bit-length n with parameter k can be solved in time $f(k) \cdot n^c$ for some $f : \mathbb{N} \to \mathbb{N}$ and $c \in \mathbb{N}$.

Vertex Cover
- Principle of optimality applied to edge

  procedure $\text{VC-D}\textsc{ecision}(V, E, k)$
  if $E = \emptyset$ **then return** "yes"
  if $k = 0$ **then return** "no"
  pick $e = (u, v) \in E$
  return $\text{VC-D}\textsc{ecision}(V, E \setminus (\{u\} \times V), k-1)$ **or**
  $\text{VC-D}\textsc{ecision}(V, E \setminus (\{v\} \times V), k-1)$

- Running time: $O(2^k \cdot (|V| + |E|))$

Kernelization

Definition
Self-reduction where instances of bit-length n with parameter k are reduced in time n^c to instances of size at most $g(k)$ for some $g : \mathbb{N} \to \mathbb{N}$ and $c \in \mathbb{N}$.

Vertex Cover
- Vertices of degree more than k need to be included.
- A graph G' in which each vertex has degree at most d and has a vertex cover of size s, can have at most $s \cdot d$ edges.
- Kernelization:

  $S \leftarrow \{v \in V : \deg(v) > k\}$
  $E' \leftarrow E \setminus S \times V$
  if $|E'| > (k - |S|) \cdot k$ **then return** "no"
  return $\text{VC-D}\textsc{ecision}(V(E'), E', k - |S|)$

- Reduced instance $G' = (V(E'), E)$ has at most k^2 edges and $2k^2$ vertices.

Approximations

Idea
Instead of finding exact optimum, find valid solution whose objective value is close to that of exact optimum.

Definition
A ρ-approximation algorithm is a polynomial-time algorithm that guarantees closeness to within a multiplicative factor of ρ.

Vertex Cover
Has 2-approximation algorithms:

- Greedy
- Linear programming relaxation

Greedy 2-Approximation for Vertex Cover

- Consider maximal matching M in G, i.e., matching that cannot be extended.
- $\text{OPT} \geq |M|$
- Let S be set of all endpoints of edges in M.
  - S is a vertex cover.
  - $|S| \leq 2 \cdot |M|$
  - $|S| \leq 2 \cdot |M| \leq 2 \cdot \text{OPT}$

Linear Programming

- Optimizing a linear objective function over $\mathbb{R}^n$ under linear inequality constraints.
- Widely used algorithm: simplex
- Can be solved in polynomial time.
- No strongly polynomial-time algorithm known.

LP-Based 2-Approximation for Vertex Cover

Integral LP for Vertex Cover
- Variables: $x_v \in \mathbb{R}$ for each $v \in V$
- Objective: $\min f(x)$ where $f(x) \doteq \sum_{v \in V} x_v$
- Constraints:
  - $(\forall\, e = (u, v) \in E)\, x_u + x_v \geq 1$
  - $(\forall\, v \in V)\, 0 \leq x_v \leq 1$
  - All x_v are integral.

Relaxation
- Dropping integrality constraints yields genuine LP.
- Find solution of LP: x^*_v for $v \in V$.
- $f(x^*) \leq \text{OPT}$
- Let $S \doteq \{v \in V : x^*_v \geq 1/2\}$.
  - S is a vertex cover.
  - $|S| \leq 2 \cdot \sum_{v \in S} x^*_v \leq 2 \cdot \sum_{v \in V} x^*_v = 2 \cdot f(x^*) \leq 2 \cdot \text{OPT}$

Hardness of Approximation

- For every NP-hard approximation problem, achieving an approximation factor $\rho = 1$ is NP-hard.
- For every NP-hard approximation problem, there exists $\rho(n) > 1$ that is NP-hard to achieve.
- For some NP-hard approximation problems a tight threshold $\tau(n)$ for efficiently achievable $\rho(n)$ is known:
  - Every $\rho(n)$ worse than $\tau(n)$ can be achieved in polynomial time.
  - Achieving any $\rho(n)$ better than $\tau(n)$ is NP-hard.

Problem	$\tau(n)$
Knapsack	$1+\epsilon$
Vertex Cover	2
Set Cover	$\ln(n)$
Independent Set	$n^{1-\epsilon}$

Heuristics

- Algorithms that have returned good results in some settings, but no known guarantees.
- Often combine local search with restarts to get out of local optimum, using randomness.
- Often based on physical processes that minimize energy or entropy.
- Examples: Metropolis, simulated annealing, etc.