UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

# CS2102 Analysis and Design of Algorithms

## NP-Completeness and Approximation Algorithms

+ Reductions

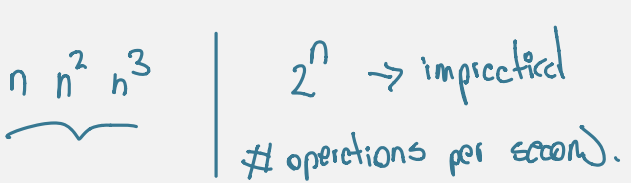
M.Sc. Bryan Gonzales Vega
bgonzales.vega@gmail.com

University of Engineering and Technology

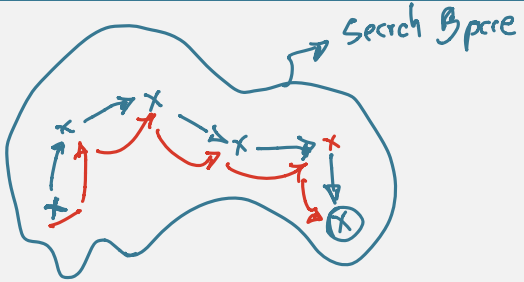
# Lecture Content

1. Easy and hard problems ✓
2. Reductions ✓
3. Approximation Algorithms ✓

# Search Problems

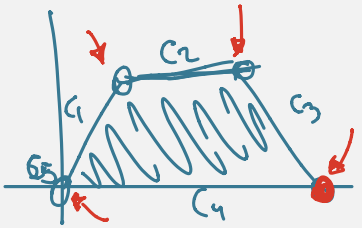


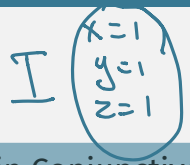

- Polynomial vs exponential
- Usually, an efficient algorithm searches for a solution among an exponential number of candidates
- We don't have an efficient algorithm yet ☹. An efficient algorithm for one such problem automatically gives efficient algorithms for all these problems ☺.
- $1M Millennium Prize

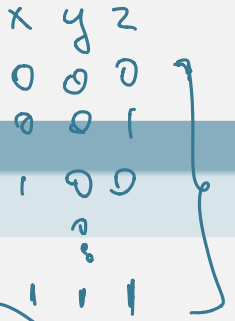



**Search problem**

A search problem is defined by an algorithm $\mathcal{C}$ that takes an instance $\mathcal{I}$ and a candidate solution $\mathcal{S}$, and runs in time polynomial in the length of $\mathcal{I}$. We say that $\mathcal{S}$ is a solution to $\mathcal{I}$ iff $\mathcal{C}(\mathcal{S}, \mathcal{I}) = \textit{true}$.






**Formula in Conjunctive Normal Form (CNF)**

C: $(x \vee y \vee z)(x \vee \bar{y})(y \vee \bar{z})(z \vee \bar{x})(\bar{x} \vee \bar{y} \vee \bar{z})$

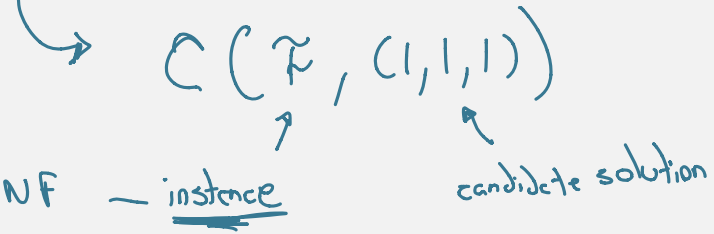
- $x, y, z$ are boolean variables
- Literals are variables $(x, y, z)$ and their negations $(\bar{x}, \bar{y}, \bar{z})$
- Clauses are disjunctions (logical or) of literals
- Many hard problems are stated in terms of SAT naturally

3-SAT

**Satisfiability (SAT)**

**Input:** Formula $\mathcal{F}$ in Conjunctive Normal ✓ C

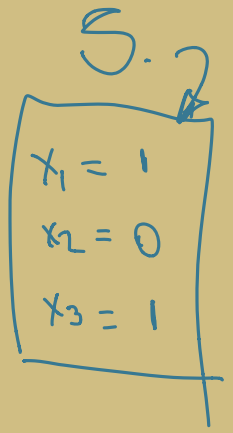
**Output:** An assignment of Boolean values to the variables of F satisfying all clauses, if exists



# Exercises

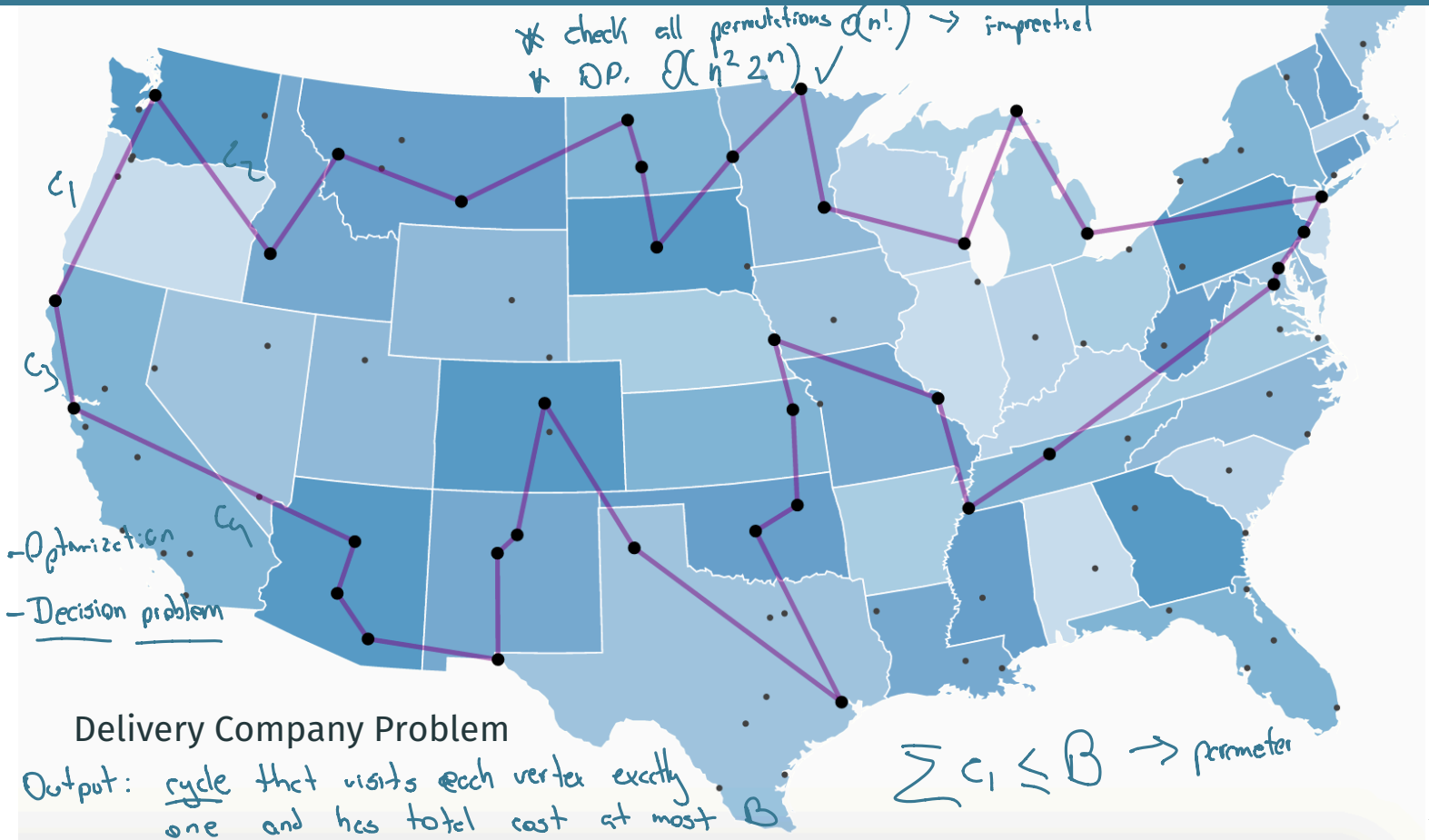
Is the following formula satisfiable?

$$F: \quad (x_1 \vee \bar{x_2} \vee \bar{x_3})(x_3)(x_1 \vee x_2)(\bar{x_1} \vee \bar{x_2})$$

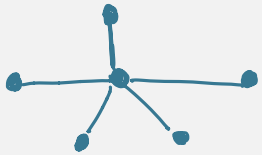


# Easy and hard problems

# Traveling Salesman Problem (TSP)



# Traveling Salesman Problem (TSP)



## Minimum Spanning Tree (MST)

- Decision version: given $n$ cities, connect them by $(n-1)$ roads of minimal total length
- Can be solved efficiently

## Traveling Salesman Problem (TSP)

- Decision version: given $n$ cities, connect them in a path by $(n-1)$ roads of minimal total length
- No polynomial algorithm known!

- TSP is a search problem: given a sequence of vertices, it is easy to check whether it is a cycle visiting all the vertices of total length at most $b$.
- TSP is usually stated as an optimization problem; we stated its decision version to guarantee that a candidate solution can be efficiently checked for correctness.

# Hamiltonian Cycle Problem

## Eulerian Cycle

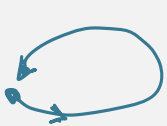
Find a cycle visiting each edge exactly once

**Input:** A graph

**Output:** A cycle that visits each edge of the graph exactly once.

## Theorem

A graph has an Eulerian cycle if and only if it is connected and the degree of each vertex is even.



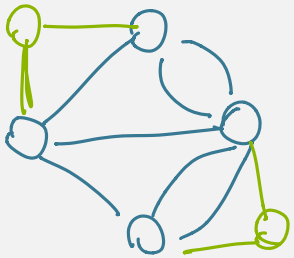
## Hamiltonian Cycle

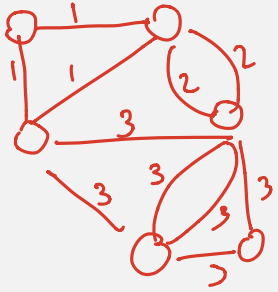
Find a cycle visiting each vertex exactly once

**Input:** A graph

**Output:** A cycle that visits each vertex of the graph exactly once.

No polynomial algorithm known!



+C

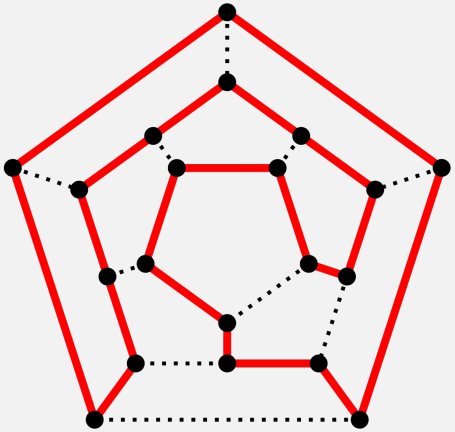


Figure 1: Hamiltonian Cycle

**Hamiltonian Cycle**

Find a cycle visiting each vertex exactly once

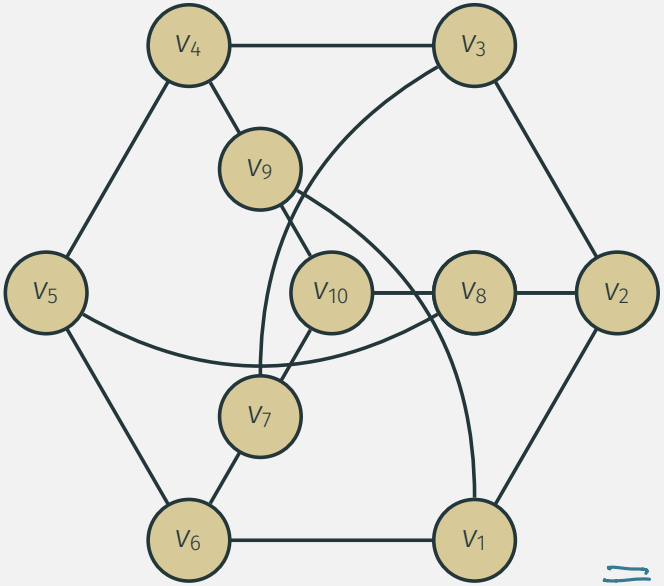
**Input:** A graph

**Output:** A cycle that visits each vertex of the graph exactly once.

No polynomial algorithm known!

Given a cycle → check its correctness

# Longest Path Problem




- Dijkstra
- Bellman-Ford
- BFS

**Shortest Path Problem**

Find a simple path from $s$ to $t$ of total length at most $b$

Can be solved efficiently

**Longest Path Problem**

Find a simple path from $s$ to $t$ of total length at least $b$

No polynomial algorithm known!

$=$ Hamiltonian Path

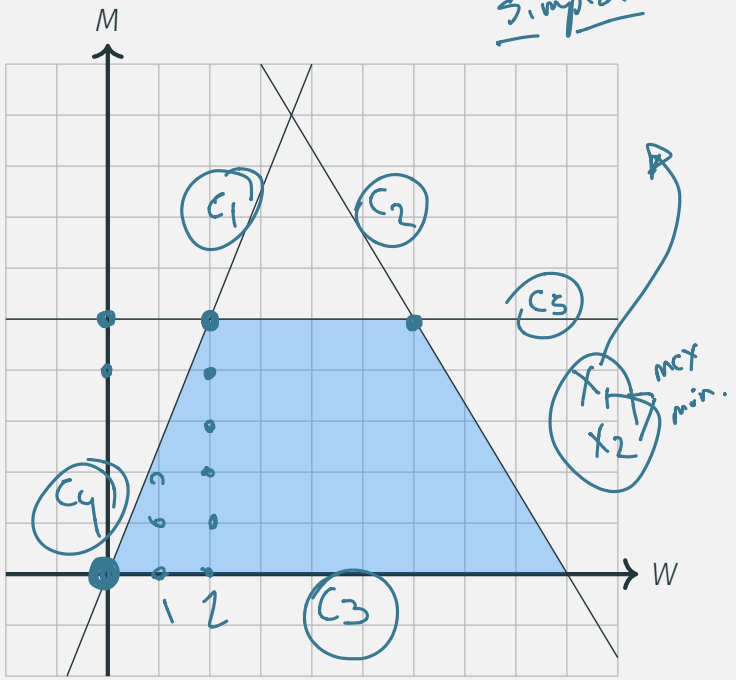
# Integer Linear Programming Problem

**Integer Linear Programming**

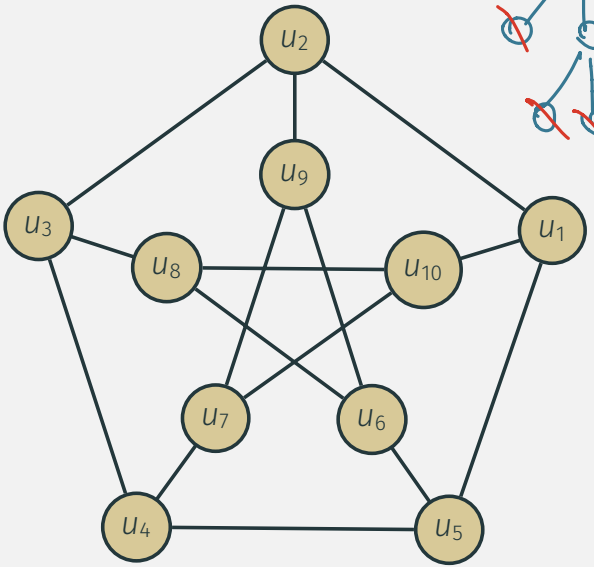
Find an integer solution of a system of linear inequalities
No polynomial algorithm known!

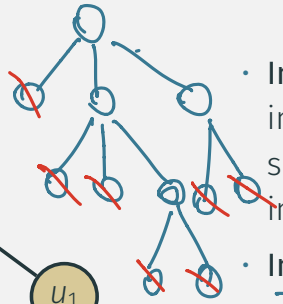
**Linear Programming**

Find a real solution of a system of linear inequalities
Can be solved efficiently



# Independent Set Problem



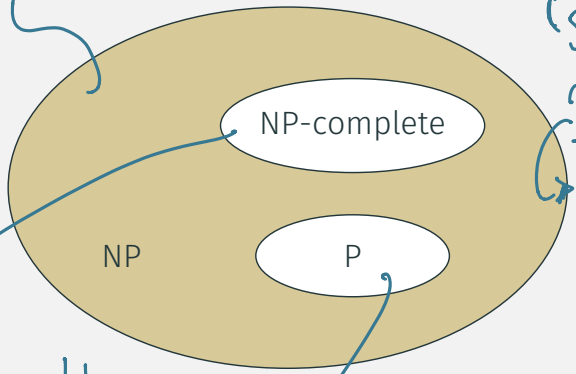
- **Independent set in a tree**. A maximal independent set in a tree can be found by a simple greedy algorithm: it is safe to take into a solution all the leaves.
- **Independent set in a graph**:

**Independent Set**

**Input:** A graph and a constraint $b$.

**Output:** A subset of vertices of size at least b such that no two of them are adjacent.

# P and NP Problems




Problems whose solution can be verified in polynomial time.

- T.S.P.
- Integer Linear Programming
- Longest Path
- Independent Set (graph)

Millenial Prize Problem

P vs NP

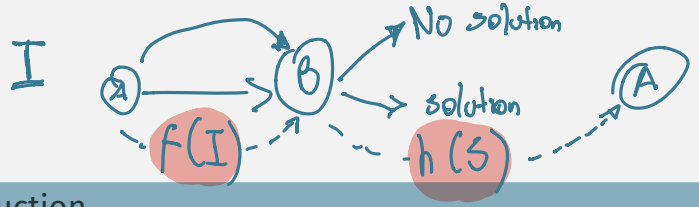
- NP stands for non-deterministic polynomial time: can guess a solution and then verify its correctness in polynomial time.
- NP contains all problems whose solutions can be efficiently verified
- If $P = NP$, then all search problems can be solved in polynomial time.
- If $P \neq NP$, then there exist search problems that cannot be solved in polynomial time.

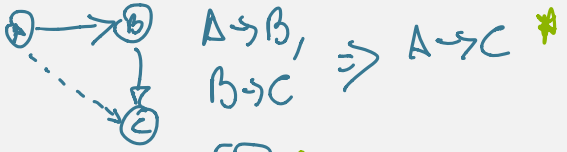
NP-complete: if all search problems are reduced to it

All search problems that can be solved in polynomial time.

- M.S.T.
- Shortest Path
- Indep. Set (trees)
- Linear Programming

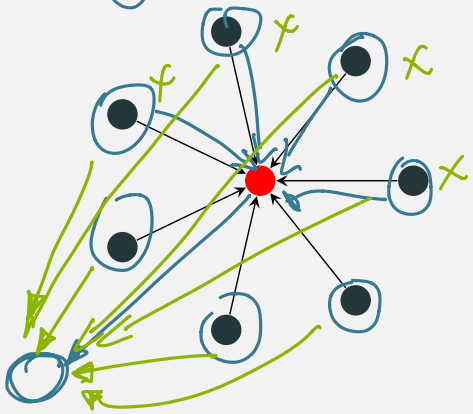
# Reductions

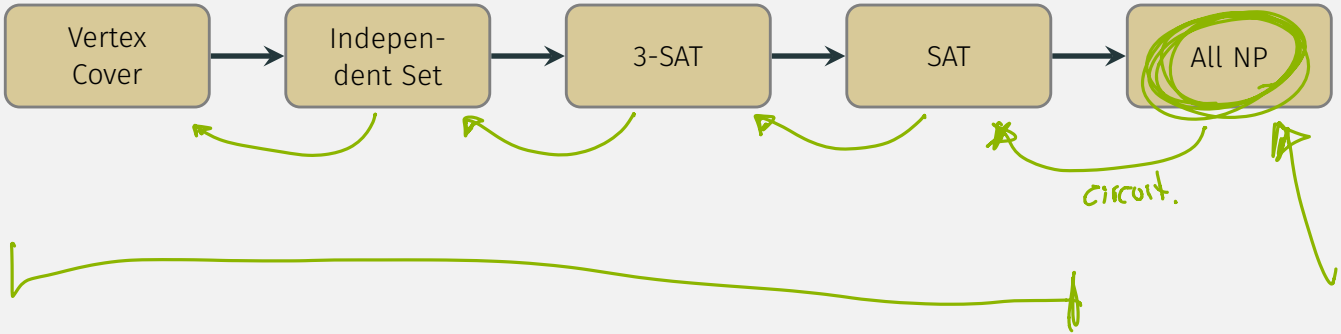



**Reduction**

We say that a search problem $\mathcal{A}$ is reduced to a search problem $\mathcal{B}$ and write $\mathcal{A} \to \mathcal{B}$, if there exists a polynomial time algorithm $f$ that converts any instance $\mathcal{I}$ of $\mathcal{A}$ into an instance $f(\mathcal{I})$ of $\mathcal{B}$, together with a polynomial time algorithm $h$ that converts any solution $\mathcal{S}$ to $f(\mathcal{I})$ back to a solution $h(\mathcal{S})$ to $\mathcal{A}$.





# Reductions

- if $\mathcal{B}$ is easy (can be solved in polynomial time), then so is $\mathcal{A}$
- if $\mathcal{A}$ is hard (cannot be solved in polynomial time), then so is $\mathcal{B}$

**Lemma**

If $\mathcal{A} \to \mathcal{B}$ and $\mathcal{B} \to \mathcal{C}$, then $\mathcal{A} \to \mathcal{C}$.

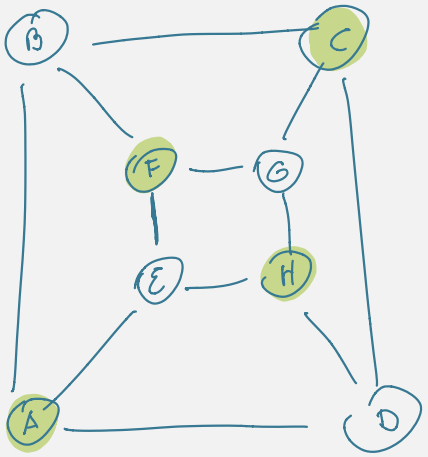
**Corolary**

If $\mathcal{A} \to \mathcal{B}$ and $\mathcal{A}$ is NP-complete, then so is $\mathcal{B}$

Independent Set → Vertex Cover (Reduction)

(I.S.) (V.C)

↳ constraint : B




I.S.

→ }E, C{

- }A, C, F, H{

C.V.

} A, B, D F G H {

- }B, D, E, G{

Given a graph G(V, E):

}I{ = IS(G) ⟺ }V - I{ = VC(G)

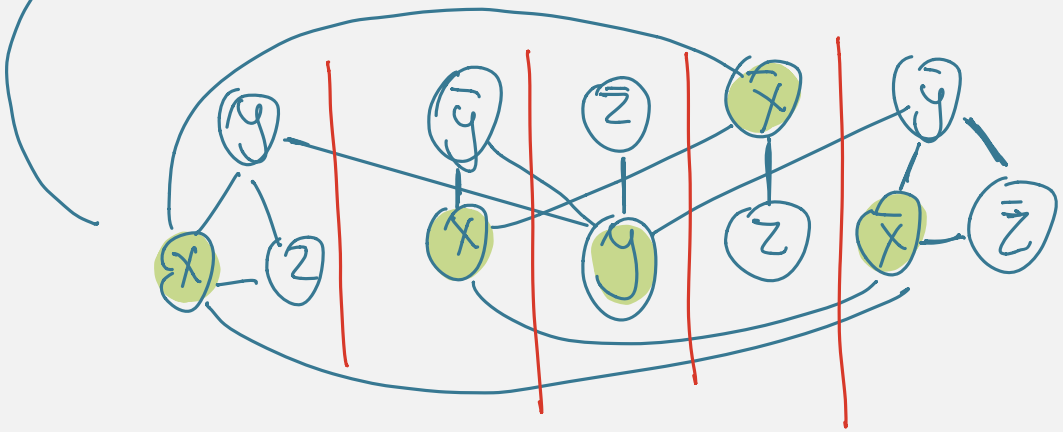
SAT → Independent Set

CNF

x, y, z

$$(x \vee y \vee z)(x \vee \bar{y})(y \vee \bar{z})(z \vee \bar{x})(\bar{x} \vee \bar{y} \vee \bar{z})$$

CNF




This graph contains an independent set of size = # clauses iff CNF is satisfiable

SAT → 3-SAT

F → F'

F is satisfiable iff F' is satisfiable ✓

$F = (l_1 \vee l_2 \vee A) \circ\circ\circ$ (A: ≥ 2)

$F' = (l_1 \vee l_2 \vee y)(\bar{y} \vee A) \circ\circ\circ\circ$

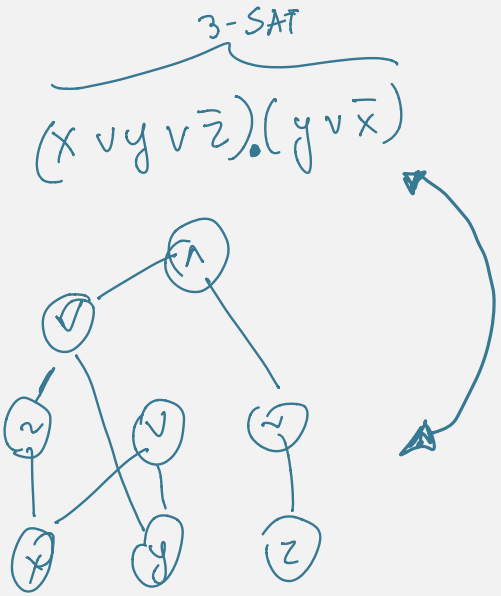
≤ 3-SAT | 3-SAT

if A satisfies F: $y = 1$

if $l_1$ or $l_2$ satisfies F: $y = 0$

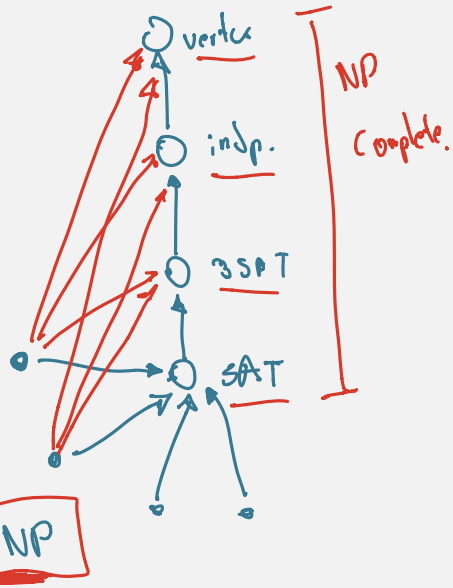
3-SAT ⟶ All NP

↳ Boolean circuit ↗



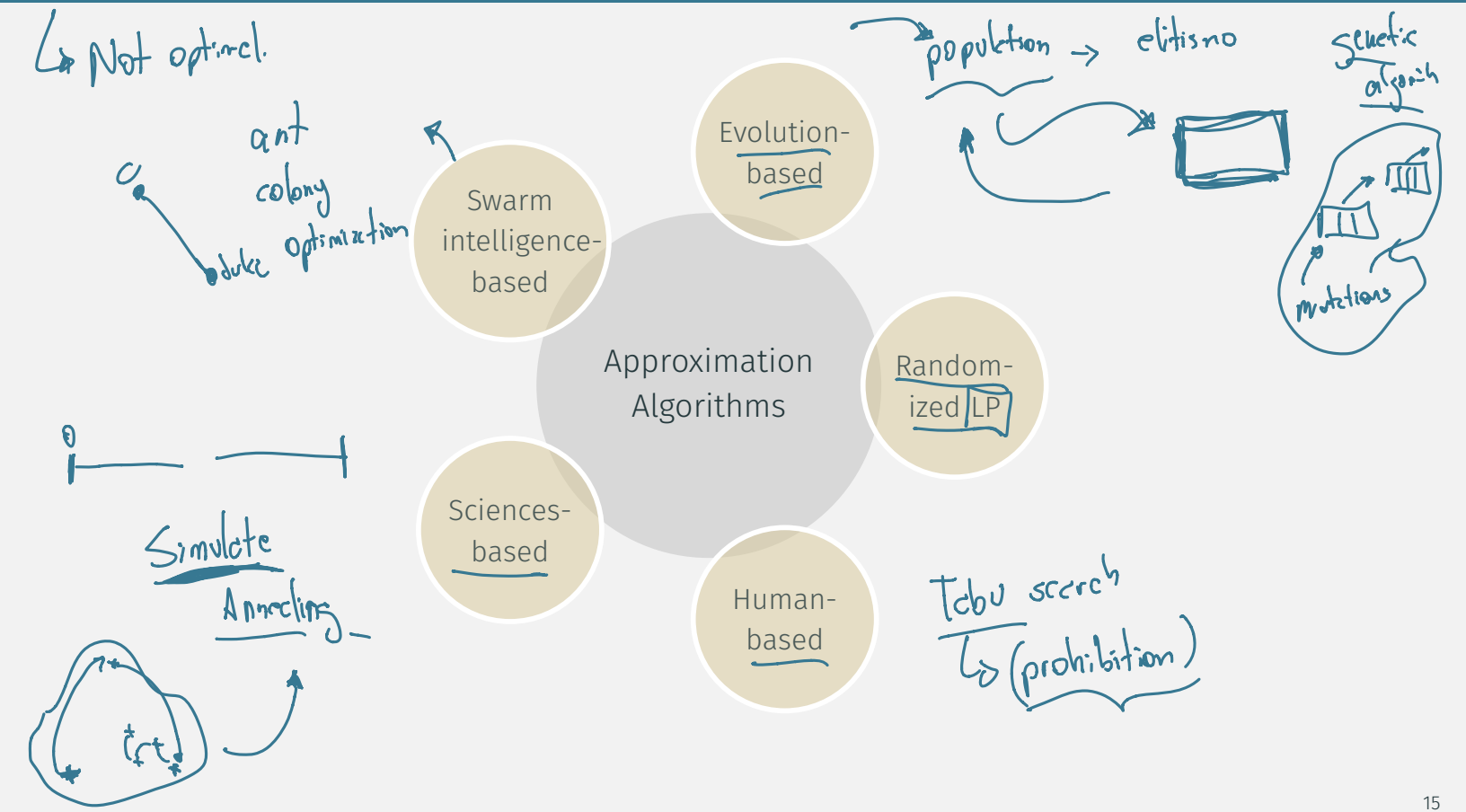

Algorithm into circuit

↳ Computer is a circuit

# Approximation Algorithms

# Approximation Algorithms



# Cont’d