CSE 421
Introduction to Algorithms

Lecture 24: More NP-completeness

NP-hardness & NP-completeness

Notion of hardness we **can** prove that is useful unless $\mathbf{P} = \mathbf{NP}$:

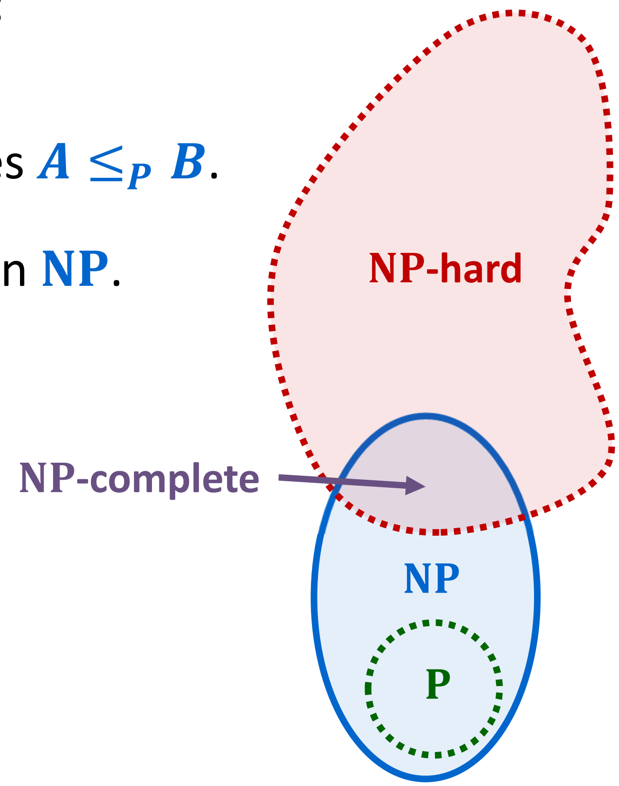
Defn: Problem $\boldsymbol{B}$ is $\mathbf{NP}$-hard iff **every** problem $\boldsymbol{A} \in \mathbf{NP}$ satisfies $\boldsymbol{A} \le_{\boldsymbol{P}} \boldsymbol{B}$.

This means that $\boldsymbol{B}$ is at least as hard as every problem in $\mathbf{NP}$.

Defn: Problem $\boldsymbol{B}$ is $\mathbf{NP}$-complete iff

- $\boldsymbol{B} \in \mathbf{NP}$ and
- $\boldsymbol{B}$ is $\mathbf{NP}$-hard.

This means that $\boldsymbol{B}$ is a hardest problem in $\mathbf{NP}$.



Extent and Impact of NP-Completeness

Extent of NP-completeness. [Papadimitriou 1995]

- 6,000 citations per year (title, abstract, keywords).
  - more than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors, and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."

NP-completeness can guide scientific inquiry.

- 1926: Ising introduces simple model for phase transitions.
- 1944: Onsager solves 2D case in tour de force.
- 19xx: Feynman and other top minds seek 3D solution.
- 2000: Istrail proves 3D problem NP-complete.

Cook-Levin Theorem and implications

Theorem [Cook 1971, Levin 1973]: **3SAT** is **NP**-complete

Proof: See CSE 431.

Corollary: If **3SAT** $\leq_P$ **B** then **B** is **NP**-hard.

By the same kind of reasoning we have

Theorem: If **A** $\leq_P$ **B** for some **NP**-hard **A** then **B** is **NP**-hard.

NP-complete problems so far

So far:

3SAT → Independent-Set → Clique

$$\begin{array}{l} \textbf{3SAT} \rightarrow \textbf{Independent-Set} \rightarrow \textbf{Clique} \\ \qquad\qquad\quad \downarrow \\ \qquad\qquad \textbf{Vertex-Cover} \rightarrow \textbf{01-Programming} \rightarrow \textbf{Integer-Programming} \end{array}$$

Steps to Proving Problem $\boldsymbol{B}$ is $\mathbf{NP}$-complete

- Show $\boldsymbol{B}$ is in $\mathbf{NP}$
  - State what the hint/certificate is.
  - Argue that it is polynomial-time to check and you won't get fooled.
- Show $\boldsymbol{B}$ is $\mathbf{NP}$-hard:
  - State: "Reduction is from $\mathbf{NP}$-hard Problem $\boldsymbol{A}$"
  - Show what the reduction function $\boldsymbol{f}$ is.
  - Argue that $\boldsymbol{f}$ is polynomial time.
  - Argue correctness in two directions:
    - $\boldsymbol{x}$ a **YES** for $\boldsymbol{A}$ implies $\boldsymbol{f}(\boldsymbol{x})$ is a **YES** for $\boldsymbol{B}$
      - Do this by showing how to convert a certificate for $\boldsymbol{x}$ being **YES** for $\boldsymbol{A}$ to a certificate for $\boldsymbol{f}(\boldsymbol{x})$ being a **YES** for $\boldsymbol{B}$.
    - $\boldsymbol{f}(\boldsymbol{x})$ a **YES** for $\boldsymbol{B}$ implies $\boldsymbol{x}$ is a **YES** for $\boldsymbol{A}$
      - … by converting certificates for $\boldsymbol{f}(\boldsymbol{x})$ to certificates for $\boldsymbol{x}$

Reduction from a Special Case to a General Case

Set-Cover:

Given a set $\boldsymbol{U}$ (universe) of $\boldsymbol{m}$ elements, a collection $\boldsymbol{S_1, \dots, S_n}$ of subsets of $\boldsymbol{U}$, and an integer $\boldsymbol{k}$

Is there a sub-collection (the cover) of $\leq \boldsymbol{k}$ sets whose union is equal to $\boldsymbol{U}$?

Theorem: Set-Cover is $\mathbf{NP}$-complete

Proof:

1. **Set-Cover** is in $\mathbf{NP}$:
   a) Certificate is a set $\boldsymbol{T} \subseteq \{\mathbf{1}, \dots, \boldsymbol{n}\}$ defining a supposed cover.
   b) Verifier outputs **YES** if $|\boldsymbol{T}| \leq \boldsymbol{k}$ and $\bigcup_{i \in T} \boldsymbol{S_i} = \boldsymbol{U}$; otherwise, answers **NO**.
   This computation is clearly polynomial-time

Set-Cover is NP-complete

Proof (continued):

2. **Set-Cover** is **NP**-hard

   Claim: Vertex-Cover $\leq_P$ Set-Cover

   a) Reduction function f takes and input a graph $G = (V, E)$ and integer k and produces a universe U, sets $S_1, \ldots, S_n \subseteq U$ and integer k' as follows:
      - $U = E$ (good idea since the objects being covered in **Vertex-Cover** are edges.)
      - Write $V = \{v_1, \ldots, v_n\}$.
        For each $i = 1, \ldots, n$ define S_i to be the set of edges in E that v_i touches.
      - $k' = k$.

   b) Clearly function f is polynomial time to compute.

   c) Correctness (⇒): Suppose that graph G has a vertex cover W of size $\leq k$.
      Define the set $T = \{i \mid v_i \in W\}$. Then $|T| = |W| \leq k$.
      Also since W is a vertex cover, $\bigcup_{i \in T} S_i = \{\, e \in E \mid \text{some } v_i \in W \text{ touches } e \,\} = E = U$.
      Therefore U has set cover T from $S_1, \ldots, S_n$ of size $\leq k$.

Set-Cover is NP-complete

Proof (continued):

2. Set-Cover is **NP**-hard

   Claim: Vertex-Cover $\leq_P$ Set-Cover

   a) Reduction function f takes and input a graph $G = (V, E)$ and integer k and produces a universe U, sets $S_1, \dots, S_n \subseteq U$ and integer k' as follows:
      - $U = E$ (good idea since the objects being covered in Vertex-Cover are edges.)
      - Write $V = \{v_1, \dots, v_n\}$.
        For each $i = 1, \dots, n$ define S_i to be the set of edges in E that v_i touches.
      - $k' = k$.

   b) c) …

   d) Correctness ($\Leftarrow$): Suppose that U has a set cover T from $S_1, \dots, S_n$ of size $\leq k$.
      Define the set $W = \{v_i \mid i \in T\}$. Then $|W| = |T| \leq k$.
      Also since T is a set cover, $U = E = \bigcup_{i\in T} S_i = \bigcup_{i\in T}\{e \in E \mid v_i \text{ touches } e\}$. But this is the same as $E = \bigcup_{v\in W}\{e \in E \mid v \text{ touches } e\}$, so graph G has vertex cover W of size $\leq k$. ■

Recall: Graph Colorability

Defn: A undirected graph $G = (V, E)$ is k-colorable iff
we can assign one of k colors to each vertex of V s.t.
for every edge (u, v) has different colored endpoints, $\chi(u) \neq \chi(v)$.
"edges are not monochromatic"

Theorem: 3Color is $\mathbf{NP}$-complete

Proof:

1. **3Color** is in $\mathbf{NP}$:
   - We already showed this; the certificate was the coloring.
2. **3Color** is $\mathbf{NP}$-hard:

   Claim: 3SAT$\leq_P$3Color

   We need to find a function f that maps a 3CNF formula F to a graph G s.t.
   F is satisfiable $\Leftrightarrow$ G is 3-colorable.

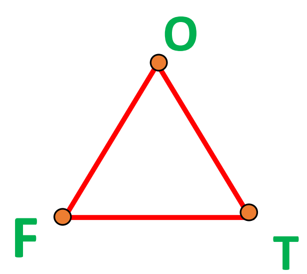
3SAT $\leq_P$ 3Color

Start with a base triangle with vertices **T**, **F**, and **O**.

We can assume that **T**, **F**, and **O** are the three colors used.

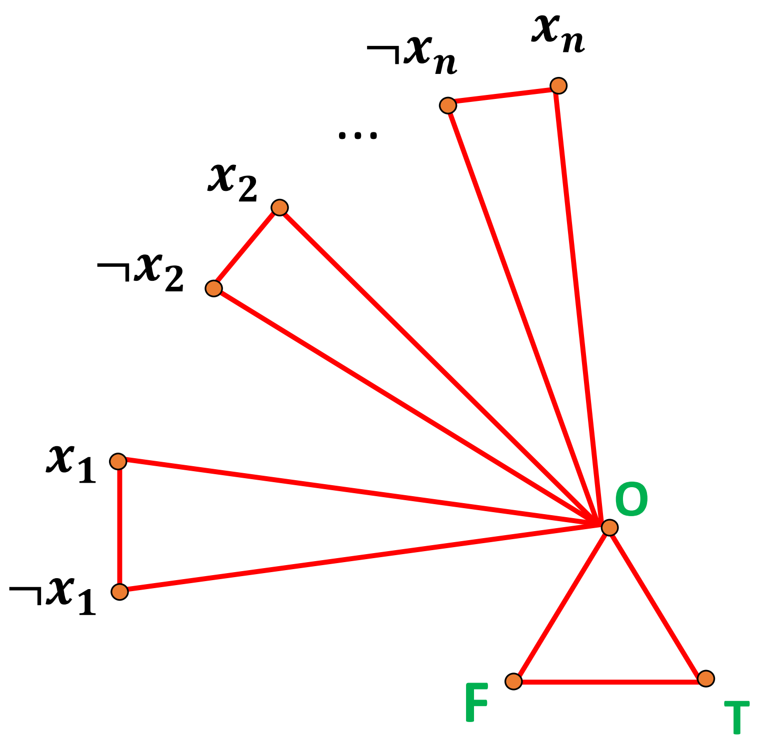
- Intuition: **T** and **F** will stand for *true* and *false*; **O** will stand for *other*.

To represent the properties of the 3CNF formula $\boldsymbol{F}$ we will need both a Boolean variable part and a clause part.



Base Triangle

$3SAT \leq_P 3Color$

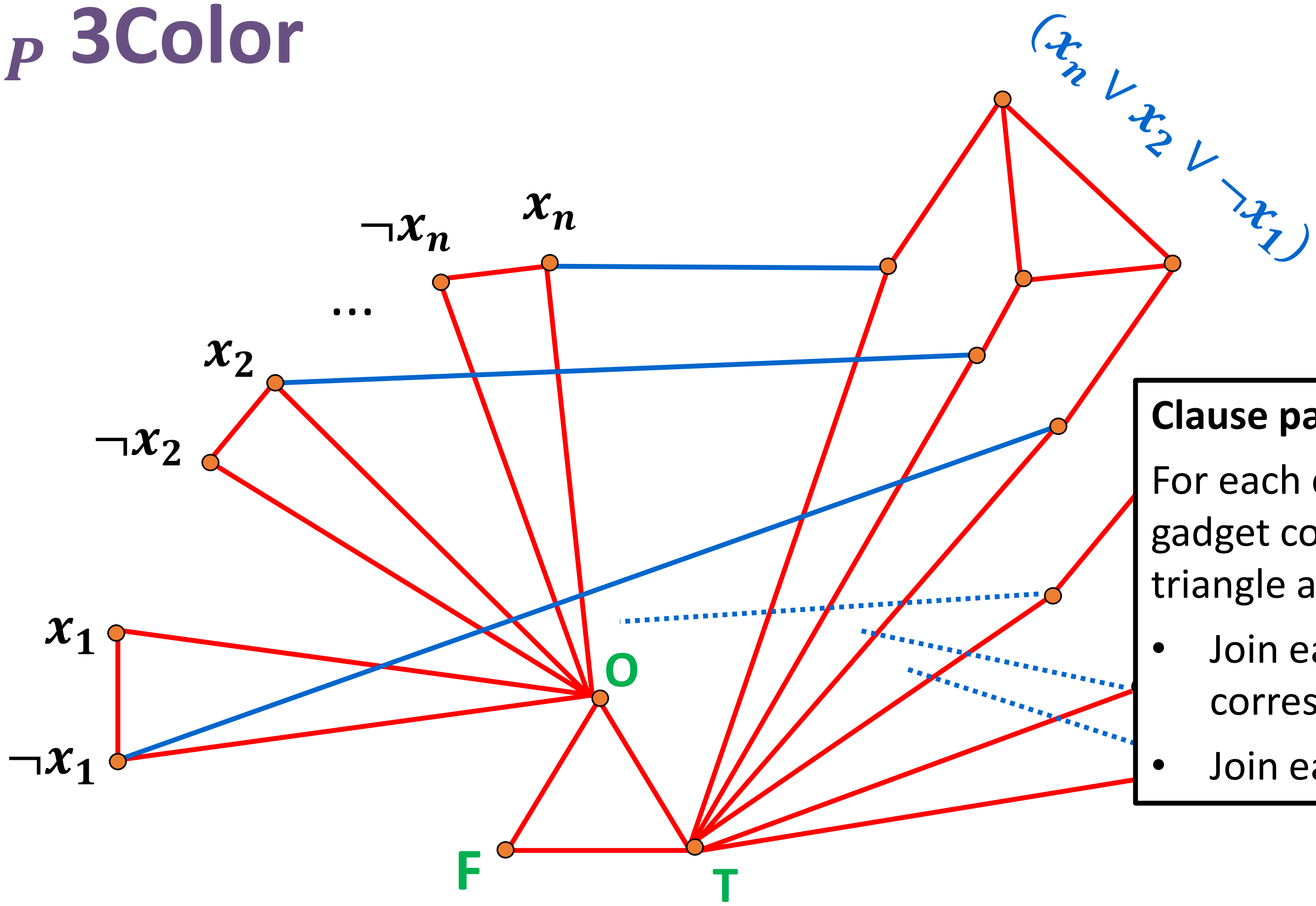


Base Triangle

Boolean variable part:

- For each Boolean variable add a triangle with two nodes labelled by literals as shown.
- Since both nodes are joined to node **O** and to each other, they must have opposite colors **T** and **F** in any 3-coloring.
- So, any 3-coloring corresponds to a unique truth assignment.

3SAT $\leq_P$ 3Color



Clause Part

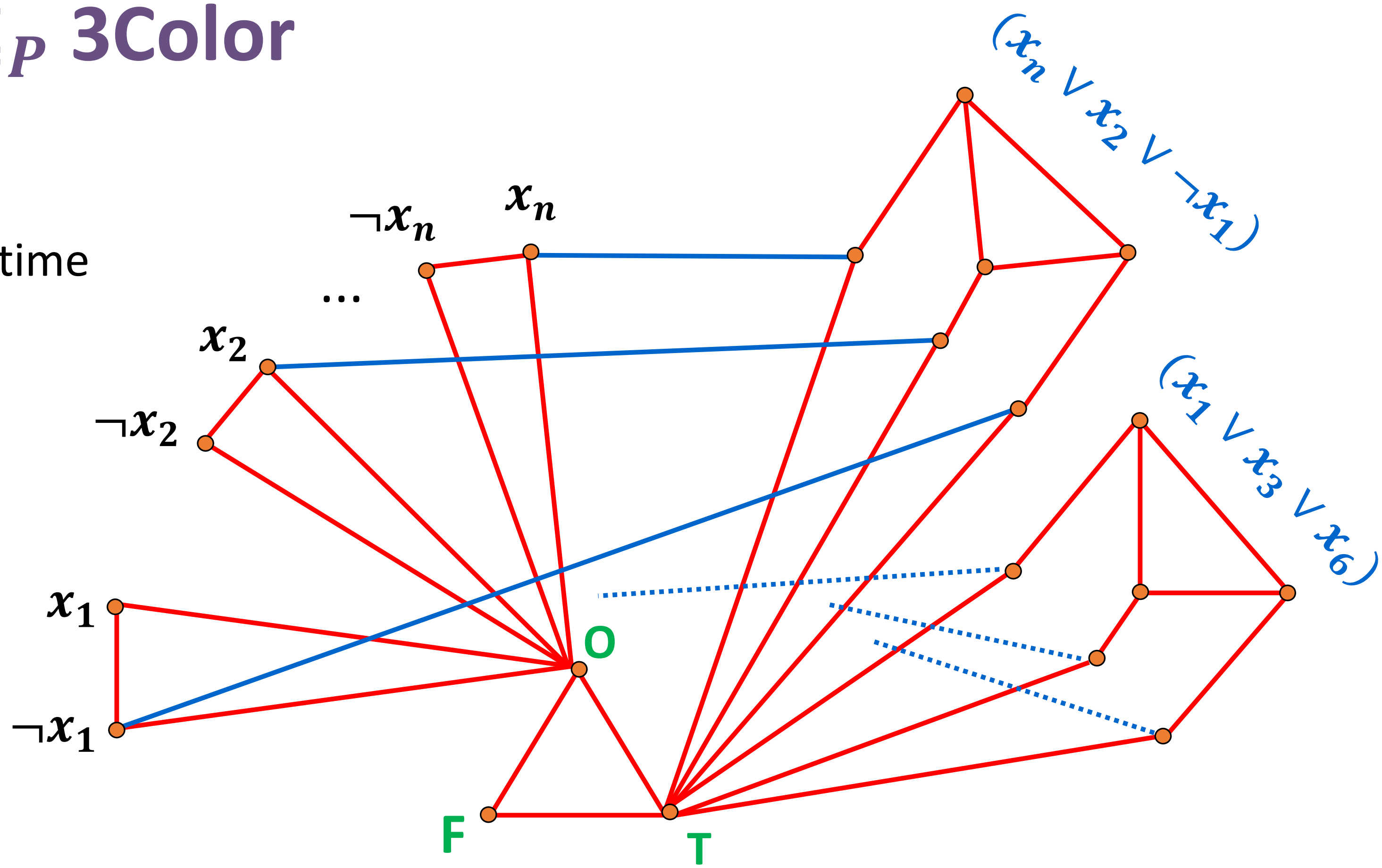
Clause part:

For each clause of F add a gadget consisting of a triangle and 3 "outer" nodes.

- Join each outer node to a corresponding literal node
- Join each outer node to **T**

3SAT $\leq_P$ 3Color

Clearly only polynomial-time to produce.



Clause Part

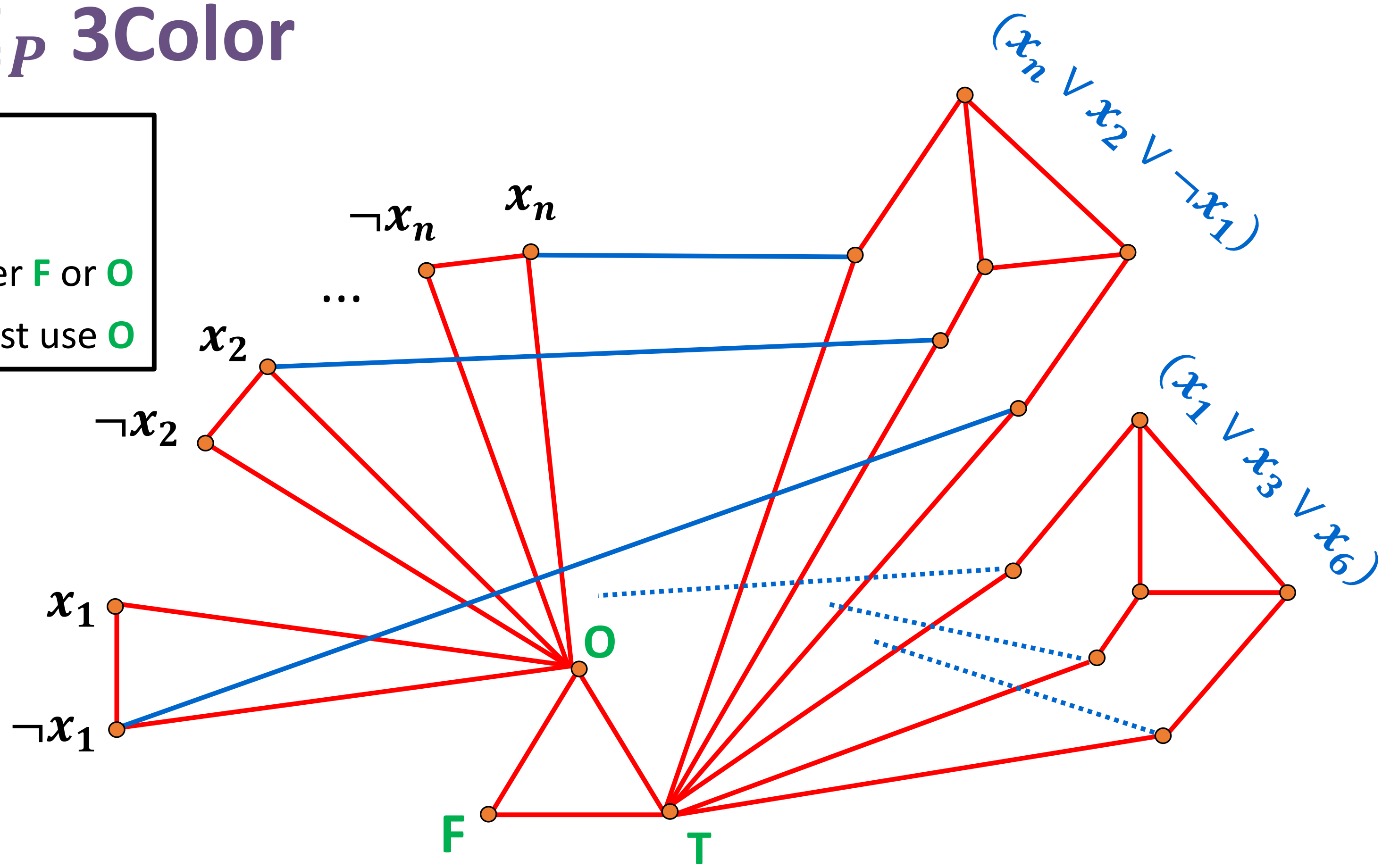
3SAT $\leq_P$ 3Color

Key property:

In any 3-coloring:

outer nodes either F or O

inner triangle must use O



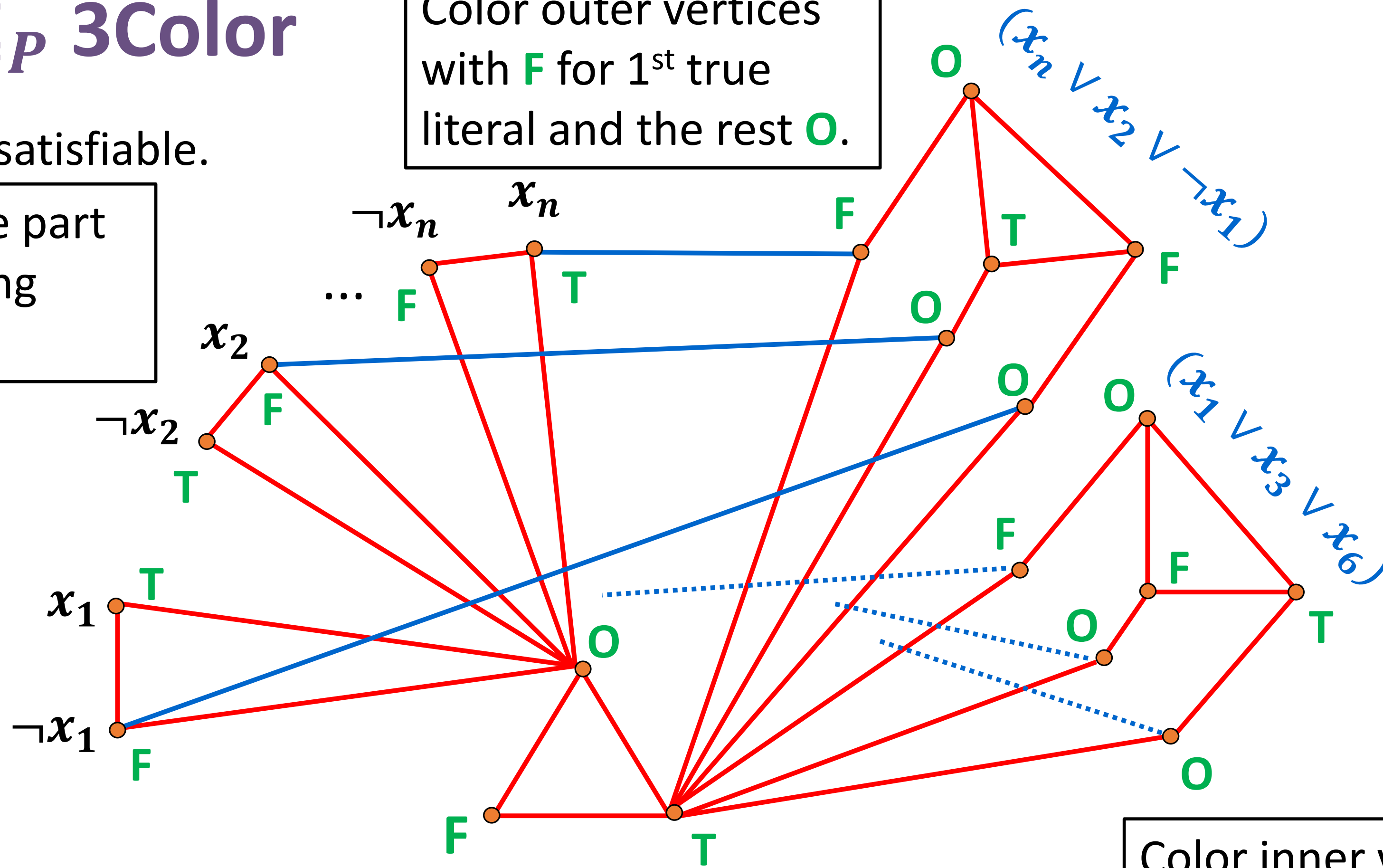
Clause Part

3SAT $\leq_P$ 3Color

Suppose F is satisfiable.

Color variable part using satisfying assignment.

Color outer vertices with F for 1st true literal and the rest O.



Color inner vertices with O opposite F.

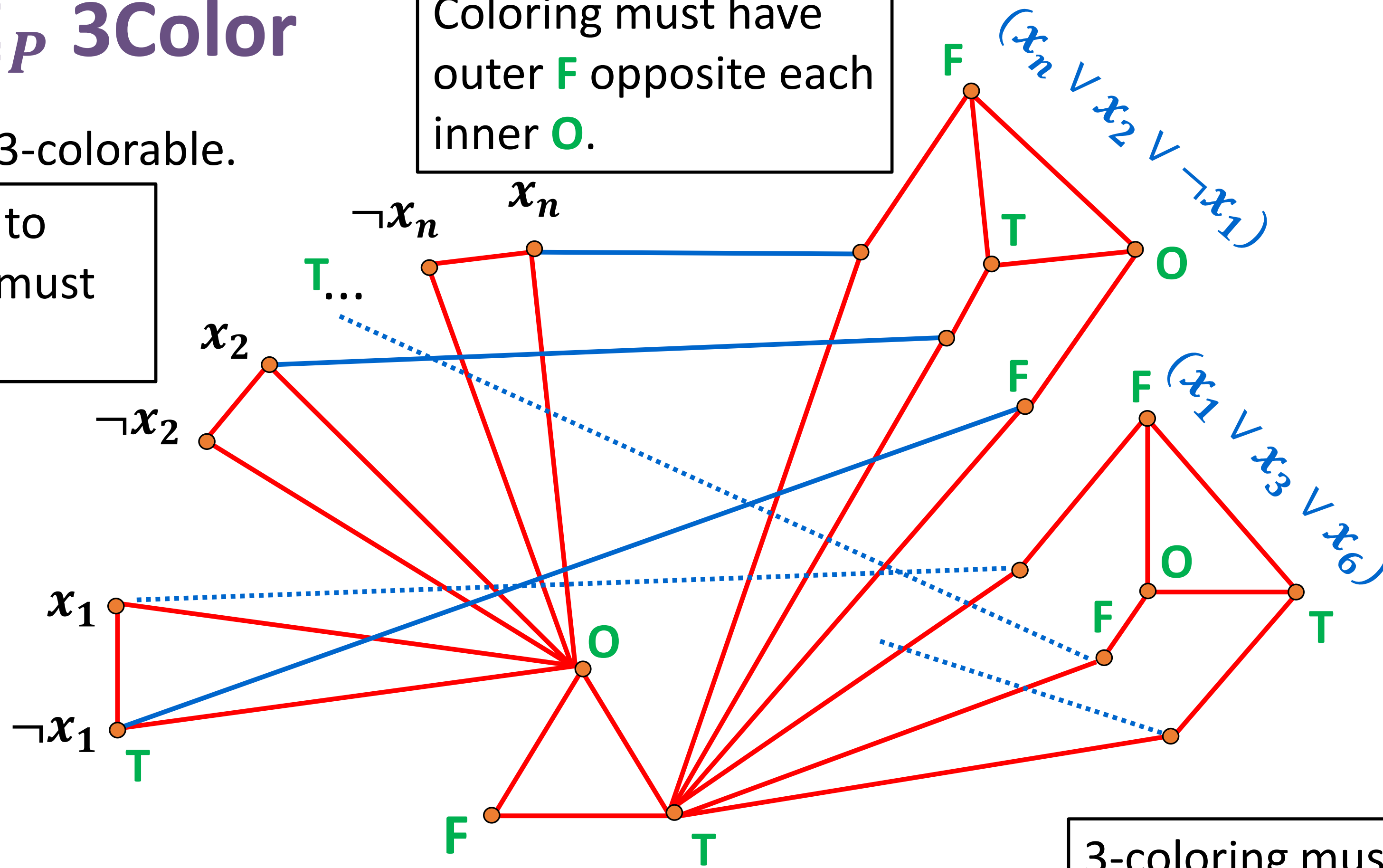
Therefore G is 3-colorable

3SAT $\leq_P$ 3Color

Suppose G is 3-colorable.

Literal joined to each outer F must be colored T.

Coloring must have outer F opposite each inner O.



3-coloring must use O on each inner triangle

Each clause has a literal that is T satisfying F ∎

More NP-completeness

Subset-Sum: (Decision version of **Knapsack**)

Given: $\boldsymbol{n}$ integers $\boldsymbol{w_1, \ldots, w_n}$ and integer $\boldsymbol{W}$

Is there a subset of the $\boldsymbol{n}$ input integers that adds up to exactly $\boldsymbol{W}$?

$\boldsymbol{O(nW)}$ solution from dynamic programming but if $\boldsymbol{W}$ and each $\boldsymbol{w_i}$ can be $\boldsymbol{n}$ bits long then this is exponential time.

Theorem: Subset-Sum is $\mathbf{NP}$-complete

Proof:

1. **Subset-Sum** is in $\mathbf{NP}$:
   a) Certificate is $\boldsymbol{n}$ bits representing a subset $\boldsymbol{S}$ of $\{\mathbf{1}, \ldots, \boldsymbol{n}\}$.
   b) Check that $\sum_{i \in S} w_i = W$.
2. **Subset-Sum** is $\mathbf{NP}$-hard

   Claim: 3SAT$\leq_P$Subset-Sum

3SAT$\leq_P$Subset-Sum

Given a 3-CNF formula F with m clauses and n variables

- We will create an input for **Subset-Sum** with $2m + 2n$ numbers that are $m + n$ digits long.
- We will ensure that no matter how we sum them there won't be any carries so each digit in the target W will force a separate constraint.
- Instead of calling them $w_1, \ldots, w_{2n+2m}$ we will use mnemonic names:
  - Two numbers for each variable x_i
    - t_i and f_i (corresponding to x_i being true or x_i being false)
  - Two extra numbers for each clause C_j
    - a_j and b_j (two identical filler numbers to handle number of false literals in clause C_j)
- We define them by giving their decimal representation...

3SAT$\leq_P$Subset-Sum

We include two $n+m$ digit numbers for each Boolean variable x_i

	1	2	3	i	…	n	1	2	3	j	…	m	
$t_i =$	0	0	0	1	…	0	1	0	0	0	…	1	Clauses C_1 and C_m contain x_i
$f_i =$	0	0	0	1	…	0	0	1	0	1	…	0	Clauses C_2 and C_j contain $\neg x_i$

Boolean part in the first n positions:

- Digit i of both t_i and f_i are 1; the rest are 0

Clause part in the next m positions:

- Digit j of t_i is 1 if clause C_j contains literal x_i; the rest are 0
- Digit j of f_i is 1 if clause C_j contains literal $\neg x_i$; the rest are 0

$3SAT \le_p$ Subset-Sum

We also include two extra identical $n + m$ digit numbers for each clause C_j

$$
\begin{array}{rcccccccccccc}
 & 1 & 2 & 3 & i & \dots & n & 1 & 2 & \dots & j & \dots & m \\
a_j = & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 & \dots & 0 \\
b_j = & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 1 & \dots & 0
\end{array}
$$

These are:

- All 0 in the Boolean columns
- Digit j of both a_j and b_j are 1 in the clause columns; the rest are 0

$3SAT \le_P$ Subset-Sum

Boolean variable part: First n digit positions ensure that exactly one of t_i or f_i is included in any subset summing to W.

				i						j		
	1	2	3	4	…	n	1	2	3	4	…	m
$t_1 =$	1	0	0	0	…	0	1	0	0	0	…	1
$f_1 =$	1	0	0	0	…	0	0	1	0	1	…	0
$t_2 =$	0	1	0	0	…	0	0	1	0	0	…	0
$f_2 =$	0	1	0	0	…	0	1	0	0	0	…	0
$t_3 =$	0	0	1	0	…	0	1	0	0	0	…	0
$f_3 =$	0	0	1	0	…	0	0	0	1	1	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$a_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$b_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$a_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
$b_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$W =$	1	1	1	1	…	1	3	3	3	3	…	3

$C_1 = (x_1 \vee \neg x_2 \vee x_3)$

$C_2 = (\neg x_1 \vee x_2 \vee x_5)$

$C_3 = (\neg x_3 \vee x_4 \vee x_7)$

$C_4 = (\neg x_1 \vee \neg x_3 \vee x_9)$

…

$C_m = (x_1 \vee \neg x_8 \vee x_{22})$

Clause part:
1's in each digit position j correspond to the 3 literals that would make clause C_j true.

Every column in the clause part of the block of t's and f's has exactly 3 1's.

The a's and b's add exactly 2 more possible 1's per column

3SAT$\leq_P$Subset-Sum

Boolean variable part:
First n digit positions ensure that exactly one of t_i or f_i is included in any subset summing to W.

				i						j		
	1	2	3	4	…	n	1	2	3	4	…	m
$t_1 =$	1	0	0	0	…	0	1	0	0	0	…	1
$f_1 =$	1	0	0	0	…	0	0	1	0	1	…	0
$t_2 =$	0	1	0	0	…	0	0	1	0	0	…	0
$f_2 =$	0	1	0	0	…	0	1	0	0	0	…	0
$t_3 =$	0	0	1	0	…	0	1	0	0	0	…	0
$f_3 =$	0	0	1	0	…	0	0	0	1	1	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$a_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$b_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$a_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
$b_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$W =$	1	1	1	1	…	1	3	3	3	3	…	3

$C_1 = (x_1 \lor \neg x_2 \lor x_3)$

$C_2 = (\neg x_1 \lor x_2 \lor x_5)$

$C_3 = (\neg x_3 \lor x_4 \lor x_7)$

$\boxed{C_4 = (\neg x_1 \lor \neg x_3 \lor x_9)}$

…

$C_m = (x_1 \lor \neg x_8 \lor x_{22})$

Key idea of clause columns:
Column j can sum to the target column sum of $\mathbf{3}$
$\Leftrightarrow$ at least one of the t_i or f_i rows included in the subset contains a $\mathbf{1}$ in column j

The a's and b's add exactly 2 more possible $\mathbf{1}$'s per column

$3SAT \le_P$ Subset-Sum

If F satisfiable choose one of t_i or f_i depending on the satisfying assignment. Their sum will have exactly one 1 in each of the first n digits and at least one 1 in every clause digit position. Also include 0, 1, or 2 of each a_j, b_j pair to add to W.

				i						j		
	1	2	3	4	…	n	1	2	3	4	…	m
$t_1 =$	1	0	0	0	…	0	1	0	0	0	…	1
$f_1 =$	1	0	0	0	…	0	0	1	0	1	…	0
$t_2 =$	0	1	0	0	…	0	0	1	0	0	…	0
$f_2 =$	0	1	0	0	…	0	1	0	0	0	…	0
$t_3 =$	0	0	1	0	…	0	1	0	0	0	…	0
$f_3 =$	0	0	1	0	…	0	0	0	1	1	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$a_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$b_1 =$	0	0	0	0	…	0	1	0	0	0	…	0
$a_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
$b_2 =$	0	0	0	0	…	0	0	1	0	0	…	0
…	…	…	…	…	…	…	…	…	…	…	…	…
$W =$	1	1	1	1	…	1	3	3	3	3	…	3

If some subset sums to W must have exactly one of t_i or f_i for each i.
Set variable x_i to true if t_i used and false if f_i used.
Must have three 1's in each clause digit column j since things sum to W.
At most two of these can come from a_j, b_j to one of these 1's must come from the choices of the truth assignment $\Rightarrow$ every clause C_j is satisfied so F is satisfiable.

■

Some other NP-complete examples you should know

Hamiltonian-Cycle: **Given** a directed graph $G = (V, E)$. Is there a cycle in G that visits each vertex in V exactly once?

Hamiltonian-Path: **Given** a directed graph $G = (V, E)$. Is there a path p in G of length $n - 1$ that visits each vertex in V exactly once?

Same problems are also $\mathbf{NP}$-complete for undirected graphs

Note: If we asked about visiting each *edge* exactly once instead of each vertex, the corresponding problems are called **Eulerian-Cycle**, **Eulerian-Path** and are polynomial-time solvable.

Travelling-Salesperson Problem (TSP)

Travelling-Salesperson Problem (TSP):

Given: a set of n cities $v_1, \dots, v_n$ and distance function d that gives distance $d(v_i, v_j)$ between each pair of cities

Find the shortest tour that visits all n cities.

DecisionTSP:

Given: a set of n cities $v_1, \dots, v_n$ and distance function d that gives distance $d(v_i, v_j)$ between each pair of cities *and* a distance D

Is there a tour of total length at most D that visits all n cities?

Hamiltonian-Cycle $\leq_P$ DecisionTSP

Define the reduction given $G = (V, E)$:

- Vertices $V = \{v_1, \dots, v_n\}$ become cities
- Define $d(v_i, v_j) = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 2 & \text{if not} \end{cases}$
- Distance $D = |V|$.

Claim: There is a Hamiltonian cycle in G $\Leftrightarrow$ there is a tour of length $|V|$

NP-complete problems we've covered

3SAT → Independent-Set → Clique

↓

Vertex-Cover → 01-Programming → Integer-Programming

↓

Set-Cover

→ 3Color

→ Subset-Sum

→ Hamiltonian-Cycle → DecisionTSP

→ Hamiltonian-Path

More Hard Computational Problems

- Aerospace engineering: optimal mesh partitioning for finite elements.
- Biology: protein folding.
- Chemical engineering: heat exchanger network synthesis.
- Civil engineering: equilibrium of urban traffic flow.
- Economics: computation of arbitrage in financial markets with friction.
- Electrical engineering: VLSI layout.
- Environmental engineering: optimal placement of contaminant sensors.
- Financial engineering: find minimum risk portfolio of given return.
- Game theory: find Nash equilibrium that maximizes social welfare.
- Genomics: phylogeny reconstruction.
- Mechanical engineering: structure of turbulence in sheared flows.
- Medicine: reconstructing 3-D shape from biplane angiocardiogram.
- Operations research: optimal resource allocation.
- Physics: partition function of 3-D Ising model in statistical mechanics.
- Politics: Shapley-Shubik voting power.
- Pop culture: Minesweeper consistency.
- Statistics: optimal experimental design.