CSE 421

NP-Completeness

Yin Tat Lee

Latest News

arXiv:2203.00671v1 [cs.DS] 1 Mar 2022

Maximum Flow and Minimum-Cost Flow in Almost-Linear Time

(Preliminary Version)

Li Chen[*]
Georgia Tech
lichen@gatech.edu

Rasmus Kyng[†]
ETH Zurich
kyng@inf.ethz.ch

Yang P. Liu[‡]
Stanford University
yangpliu@stanford.edu

Richard Peng
University of Waterloo [§]
y5peng@uwaterloo.ca

Maximilian Probst Gutenberg[†]
ETH Zurich
maxprobst@ethz.ch

Sushant Sachdeva[¶]
University of Toronto
sachdeva@cs.toronto.edu

March 2, 2022

Abstract

We give an algorithm that computes exact maximum flows and minimum-cost flows on directed graphs with m edges and polynomially bounded integral demands, costs, and capacities in $m^{1+o(1)}$ time. Our algorithm builds the flow through a sequence of $m^{1+o(1)}$ approximate undirected minimum-ratio cycles, each of which is computed and processed in amortized $m^{o(1)}$ time using a dynamic data structure.

Our framework extends to an algorithm running in $m^{1+o(1)}$ time for computing flows that minimize general edge-separable convex functions to high accuracy. This gives an almost-linear time algorithm for several problems including entropy-regularized optimal transport, matrix scaling, p-norm flows, and isotonic regression.

Yesterday 5pm

Don’t quote me on this.
This paper is 1-day new.
I haven’t really read it.

Rough Idea

- The algorithm has m iterations.
- Each iterations send an approximate shortest path from s to t.
- They maintain $n^{o(1)}$ spanning tree-ish and use the shortest paths on these tree.

Don’t quote me on this.
This paper is 1-day new.
I haven’t really read it.

Rough Idea

- Use data structure to send the flow in $n^{o(1)}$ time.

- The length is defined to be how saturated is an edge. It is selected to ensures only the length of $n^{o(1)}$ edges changes sufficiently.

Don’t quote me on this.
This paper is 1-day new.
I haven’t really read it.

Rough Idea

I will “probably” have a 599 course on this next year.

- To find the data structure efficiently, the recursively reduce # of edges and # of vertices.

Decision Problems

A decision problem is a computational problem where the answer is just yes/no.

We can define a problem by a set $X \subset \{0,1\}^n$.
The answer for the input s is YES iff $s \in X$.

Certifier: Algorithm C(s, t) is a certifier for problem A if
$s \in X$ if and only if (There is a t such that $C(s, t) = YES$))

NP: Set of all decision problems for which there exists a poly-time certifier.

3CNF: $(x_1 \vee \overline{x_2} \vee x_9) \wedge (\overline{x_2} \vee x_3 \vee x_7) \wedge \cdots$

Cook-Levin Theorem

Theorem (Cook 71, Levin 73): 3-SAT is NP-complete, i.e., for all problems $A \in NP$, $A \leq_p$ 3-SAT.

Pf (Draft. Take CSE 431 for more.):

Since $A \in NP$, there is a polytime certifier C such that

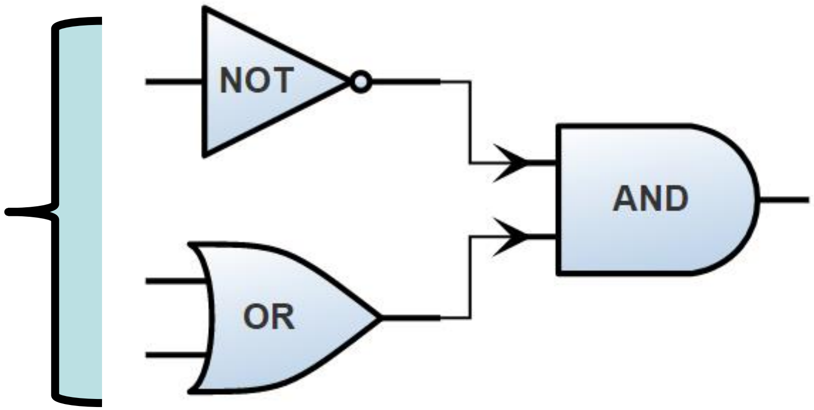
$$s \in A \text{ iff } C(s,t) = 1 \text{ for some } t$$

To solve the problem A, it suffices to find t.

Since C is polytime, we can convert C to a poly size circuit (of AND OR NOT).

- Some input are the given s.
- Some input are t.

Our goal is to find t to make the output is TRUE.



Cook-Levin Theorem

To find an input such that output is true,
we convert the circuit to 3CNF $(x_1 \vee \overline{x_2} \vee x_9) \wedge (\overline{x_2} \vee x_3 \vee x_7) \wedge \cdots$

- An OR gate with input a,b and output c can be represented by

$$(a \vee b \vee \bar{c}) \wedge (\bar{a} \vee c) \wedge (\bar{b} \vee c)$$

- A NOT gate with input a and output c can be represented by

$$(a \vee c) \wedge (\bar{a} \vee \bar{c})$$

- An AND gate can be represented by OR and NOT

X and Y = not ((not X) or (not Y))

Cook-Levin Theorem

To find an input such that output is true,
we convert the circuit to 3CNF $(x_1 \vee \overline{x_2} \vee x_9) \wedge (\overline{x_2} \vee x_3 \vee x_7) \wedge \cdots$

Suppose the circuit gate $C_1, C_2, \cdots, C_q$.

For each circuit C_i, we create a new variable c_i.

The relation between the inputs of C_i and its output c_i is a 3CNF.
We write that as $\overline{C}_i$

The whole formula is 3CNF is $\overline{C}_1 \wedge \overline{C}_2 \wedge \cdots \wedge \overline{C}_q \wedge c_q$.

Steps to Proving
Problem B is NP-complete

Show **B** is **NP**-hard:

- State which **NP-**hard Problem **A** you want to solve using **B.**
- Show what the map **f** is.
- Argue that **f** is polynomial time
- Argue correctness: two directions
  Yes for **A** implies Yes for **B** and vice versa.

Show **B** is in **NP**

- State what hint/certificate is and why it works
- Argue that it is polynomial-time to check.

Is NP-complete as bad as it gets?

- NO! **NP**-complete problems are frequently encountered, but there are worse:

- Some problems provably require exponential time.

- Ex: Does **M** halt on input **x** in $\mathbf{2^{|x|}}$ steps?

  Some require $2^n, 2^{2^n}, \cdots$ steps

  And some are just plain uncomputable.

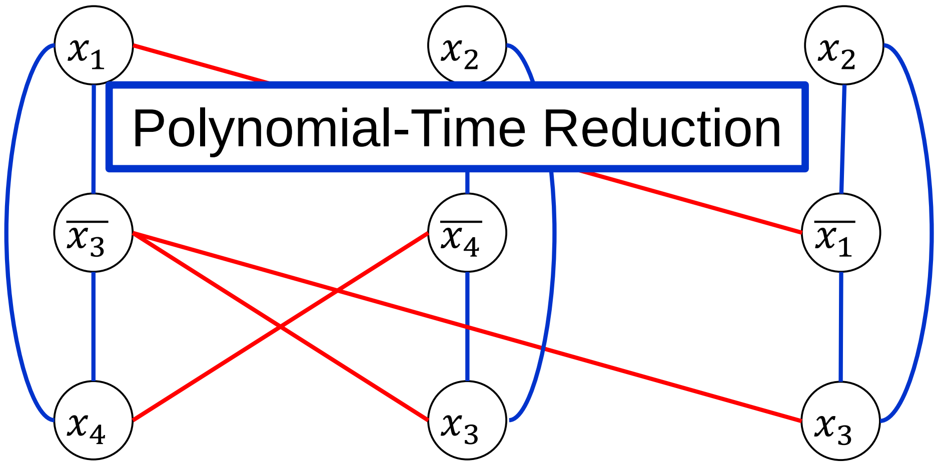
- I was wrong last lecture. There are natural problems that is not in P. Go is EXP-COMPLETE.

3-SAT $\leq_p$ Independent Set

Map a 3-CNF to (G,k). Say m is number of clauses

- Create a vertex for each literal
- Joint two literals if
  - They belong to the same clause (blue edges)
  - The literals are negations, e.g., $x_i, \overline{x_i}$ (red edges)
- Set k be the # of clauses.

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



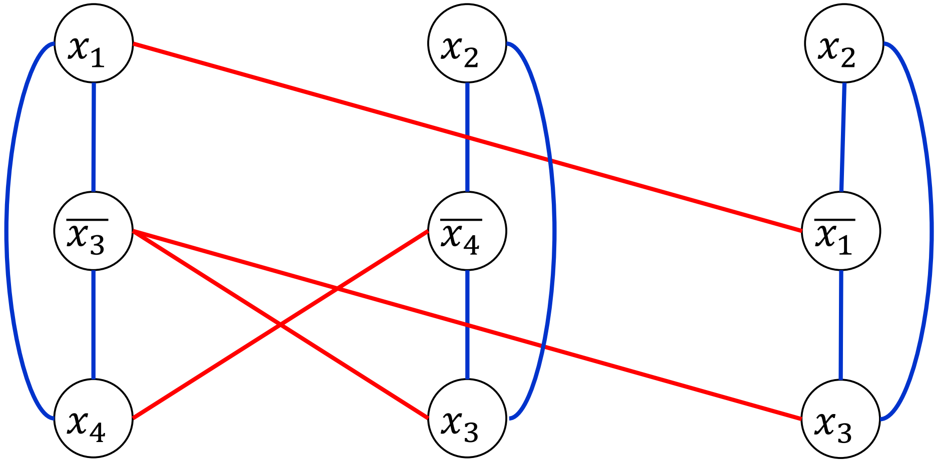
Correctness of 3-SAT $\leq_p$ Indep Set

F satisfiable => An independent of size k

Given a satisfying assignment, Choose one node from each clause where the literal is satisfied

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

Satisfying assignment: $x_1 = T, x_2 = F, x_3 = T, x_4 = F$



- S has exactly one node per clause => No blue edges between S
- S follows a truth-assignment => No red edges between S
- S has one node per clause => |S|=k

Correctness of 3-SAT $\leq_p$ Indep Set

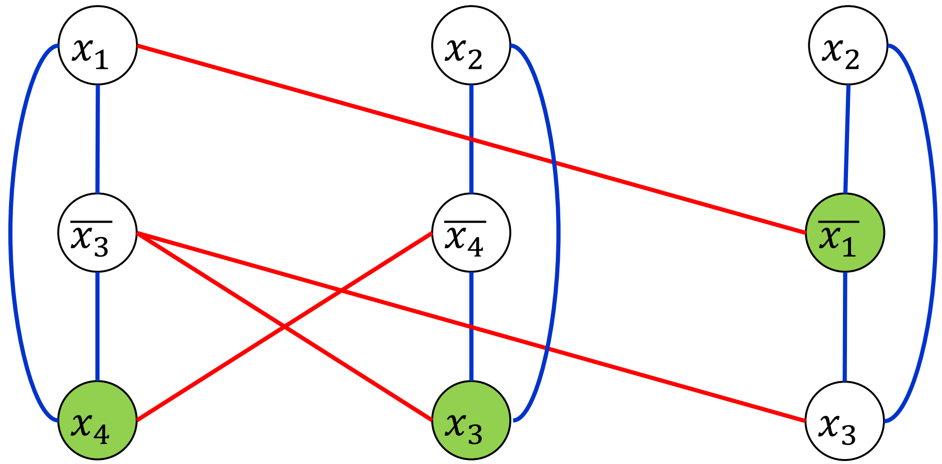
<u>An independent set of size k => A satisfying assignment</u>

Given an independent set S of size k.

S has exactly one vertex per clause (because of blue edges)

S does not have $x_i, \overline{x_i}$ (because of red edges)

So, S gives a satisfying assignment



Satisfying assignment: $x_1 = F, x_2 =?, x_3 = T, x_4 = T$

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

Yet another example of NP completeness

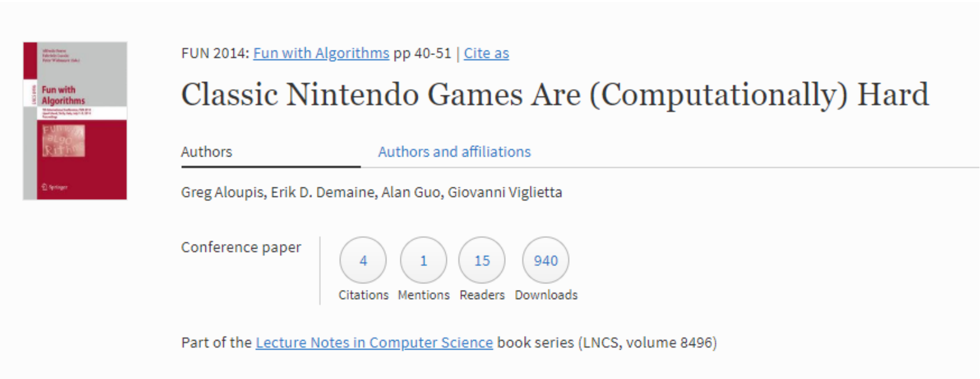
Prove that Super Mario Bros is NP-complete.

What do we need to show?

- The problem is in NP.
- Some NP complete problem is easier than Super Mario.

Approach:

- 3SAT $\leq_P$ Super Mario



FUN 2014: Fun with Algorithms pp 40-51 | Cite as

Classic Nintendo Games Are (Computationally) Hard

Authors | Authors and affiliations

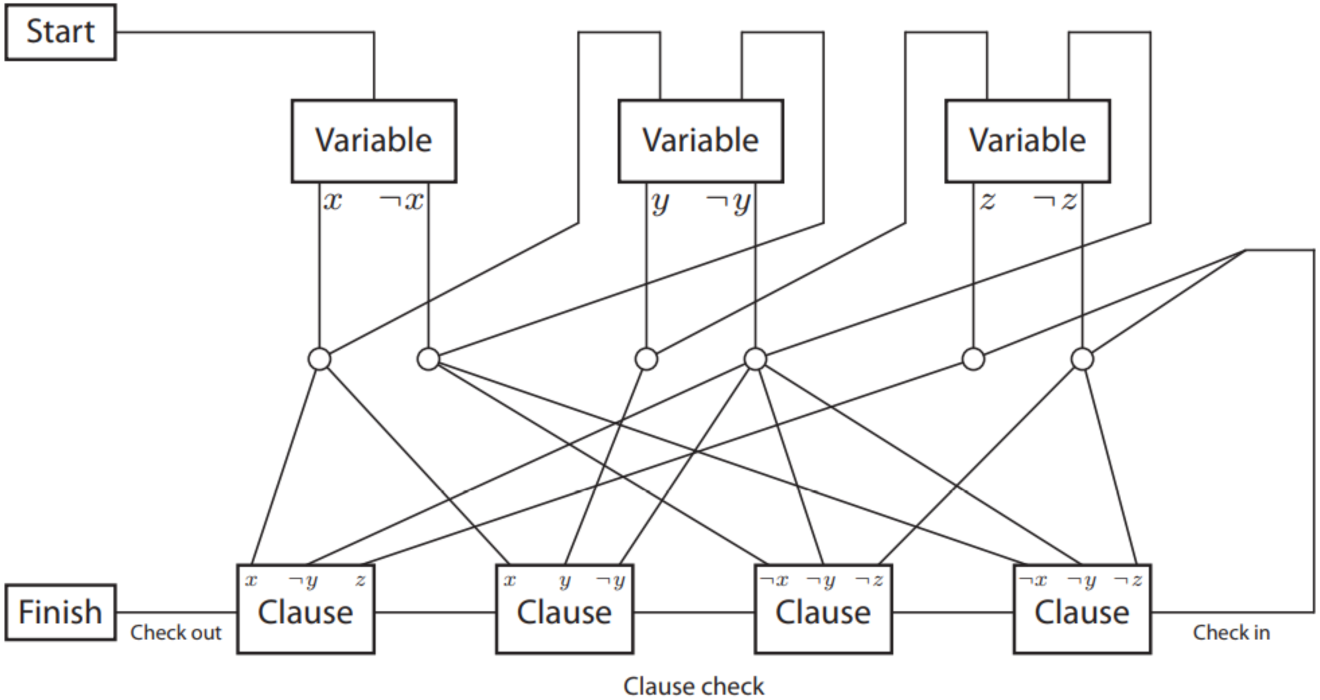
Greg Aloupis, Erik D. Demaine, Alan Guo, Giovanni Viglietta

Conference paper | 4 Citations | 1 Mentions | 15 Readers | 940 Downloads

Part of the Lecture Notes in Computer Science book series (LNCS, volume 8496)

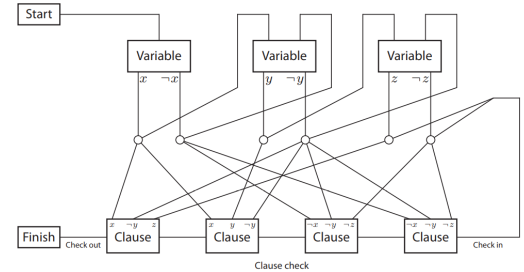
Yet another example of NP completeness

Given a 3SAT, we need to create a level.

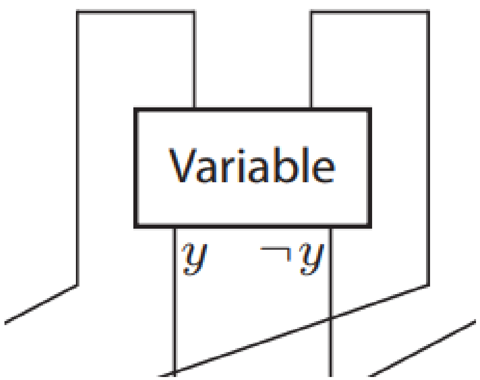


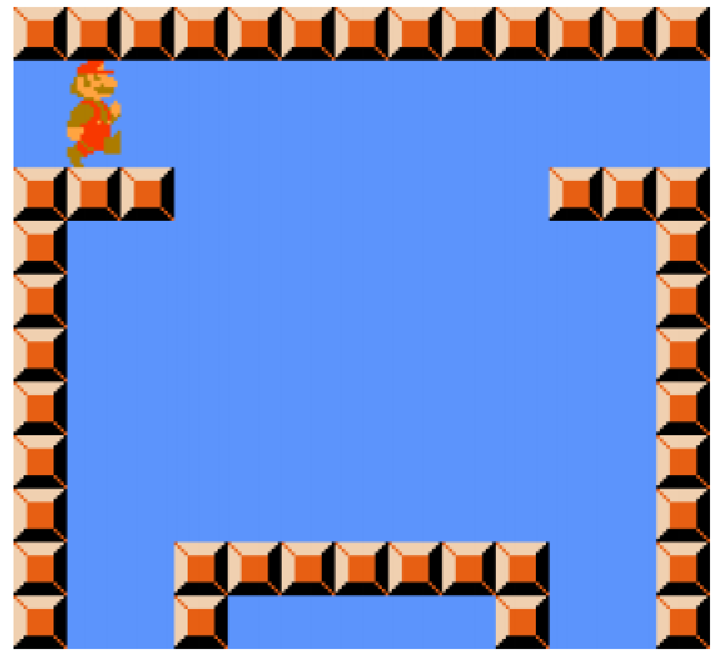
We ignore the following issues:

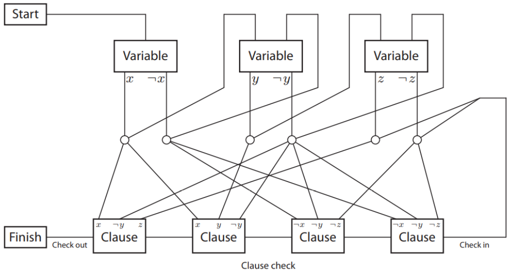
- Need to consider the “crossing” coz the level is 2-D.
- Assume Mario can go both left or right.



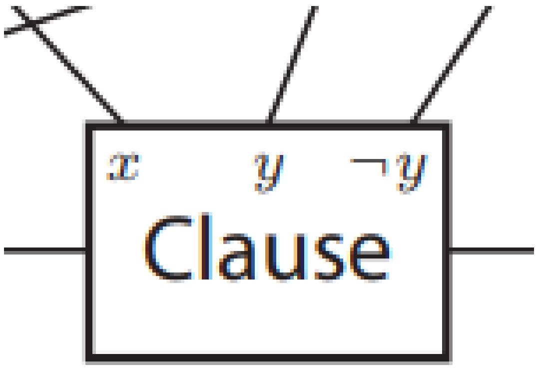
Question 1: How to create this part?







Question 2: How to create this part?



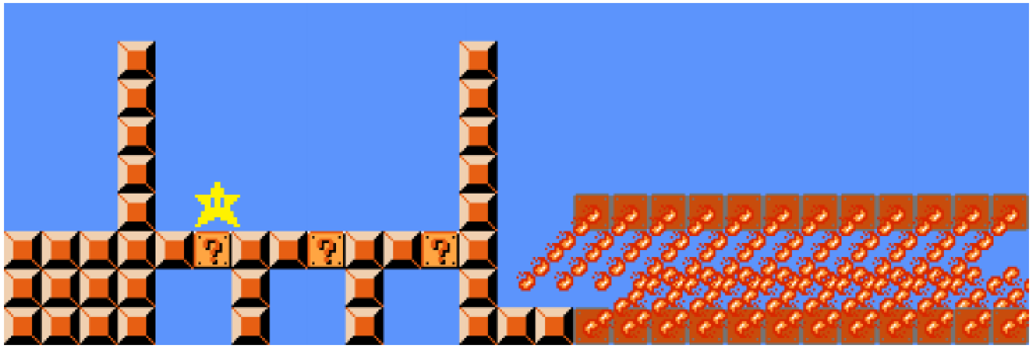


Figure 11: Clause gadget for Super Mario Bros.

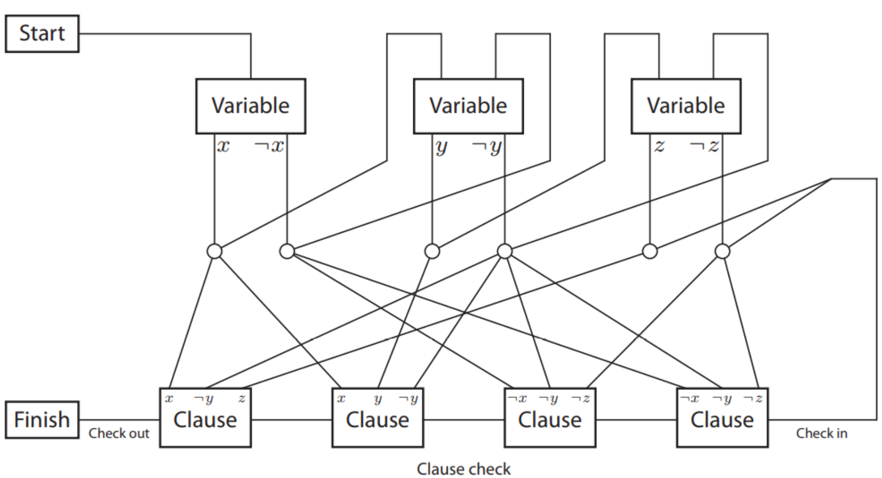
So, what you need to prove?

- If the 3SAT is satisfiable, then indeed the level is solvable.

Usually, this part is easy. This is basically due to the design of your reduction.

- If the level is solvable, then the 3SAT is satisfiable

This part usually requires more argument. Need to prove no tricky way to solve the problem without solving the 3SAT.



More NP-completeness

- Subset-Sum problem
  (Decision version of Knapsack)
  - Given **n** integers $w_1,\ldots,w_n$ and integer **W**
  - Is there a subset of the **n** input integers that adds up to exactly **W**?

- **O(nW)** solution from dynamic programming but if **W** and each w_i can be **n** bits long then this is exponential time

3-SAT $\leq_P$ Subset-Sum

- Given a 3-CNF formula with **m** clauses and **n** variables
- Will create 2**m**+2**n** numbers that are **m+n** digits long

  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

  - u_j and v_j (filler variables to handle number of false literals in clause C_j)

3-SAT $\leq_P$ Subset-Sum

$C_3 = (x_1 \lor \neg x_2 \lor x_5)$

				i						j		
	1	2	3	4	…	n	1	2	3	4	…	m
t_1	1	0	0	0	…	0	0	0	1	0	…	1
f_1	1	0	0	0	…	0	1	0	0	1	…	0
t_2	0	1	0	0	…	0	0	1	0	0	…	1
f_2	0	1	0	0	…	0	0	0	1	1	…	0
					…				….			
$u_1=v_1$	0	0	0	0	…	0	1	0	0	0	…	0
$u_2=v_2$	0	0	0	0	…	0	0	1	0	0	…	0
					…				….			
W	1	1	1	1	…	1	3	3	3	3	…	3

Graph Colorability

- Defn: Given a graph G=(V,E), and an integer k, a k-coloring of G is

  an assignment of up to k different colors to the vertices of G so that the endpoints of each edge have different colors.

- 3-Color: Given a graph G=(V,E), does G have a 3-coloring?
- Claim: 3-Color is NP-complete
- Proof: 3-Color is in NP:

  Certificate is an assignment of red,green,blue to the vertices of G

  Easy to check that each edge is colored correctly

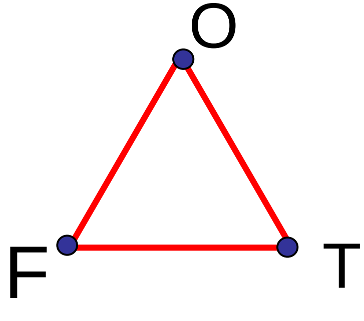
3-SAT $\leq_P$ 3-Color

- Reduction:

  We want to map a 3-CNF formula **F** to a graph **G** so that

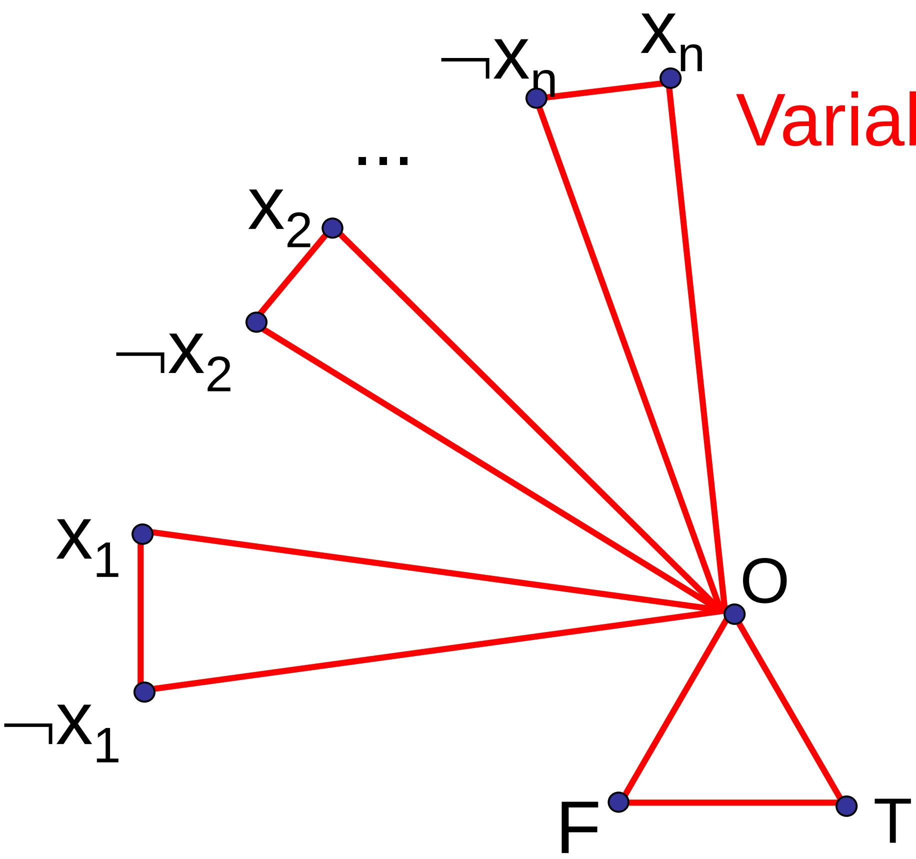
  - **G** is 3-colorable iff **F** is satisfiable

3-SAT $\leq_P$ 3-Color



Base Triangle

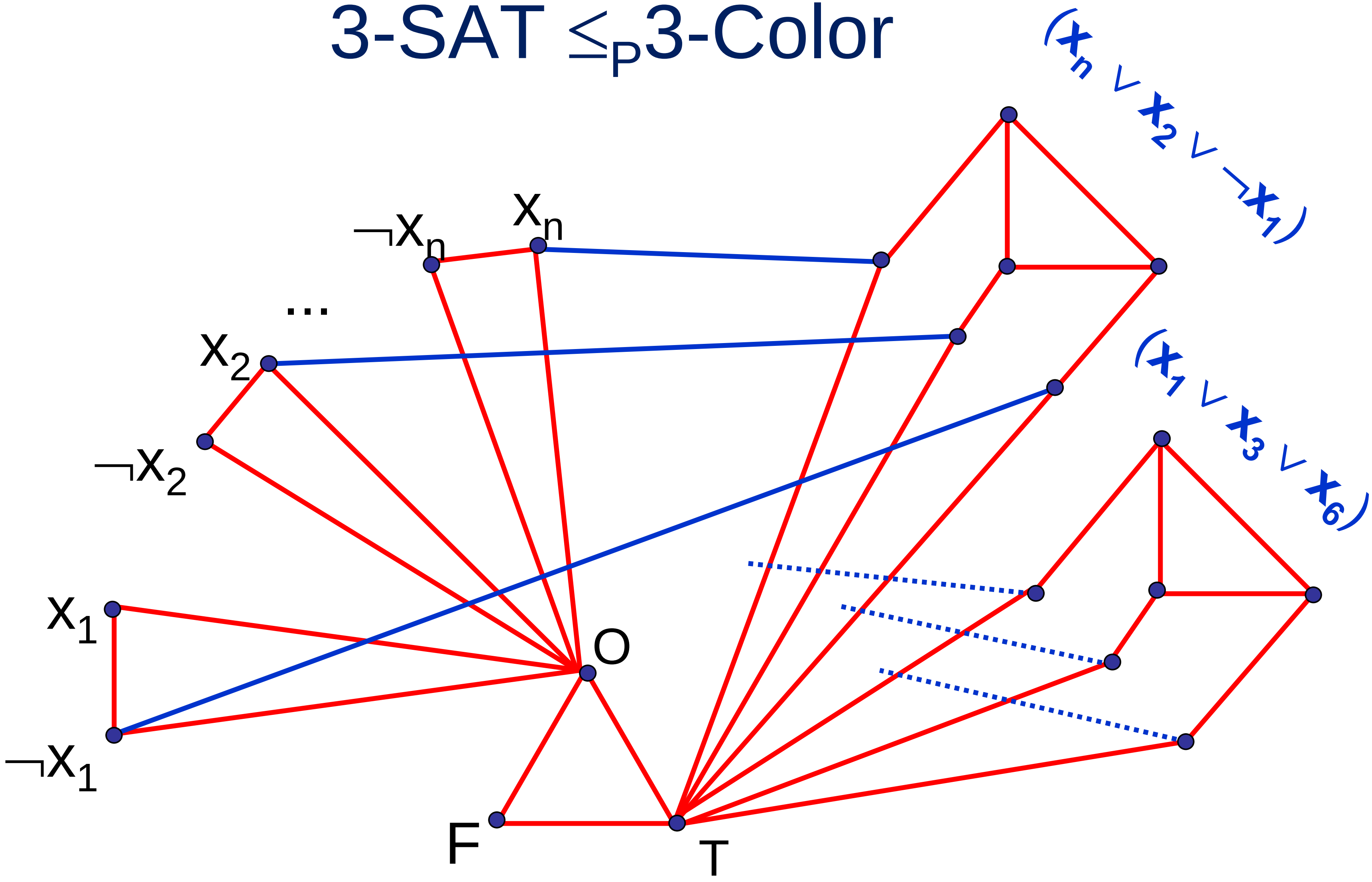
3-SAT $\leq_P$ 3-Color



Variable Part:

in 3-coloring, variable colors correspond to some truth assignment (same color as T or F)

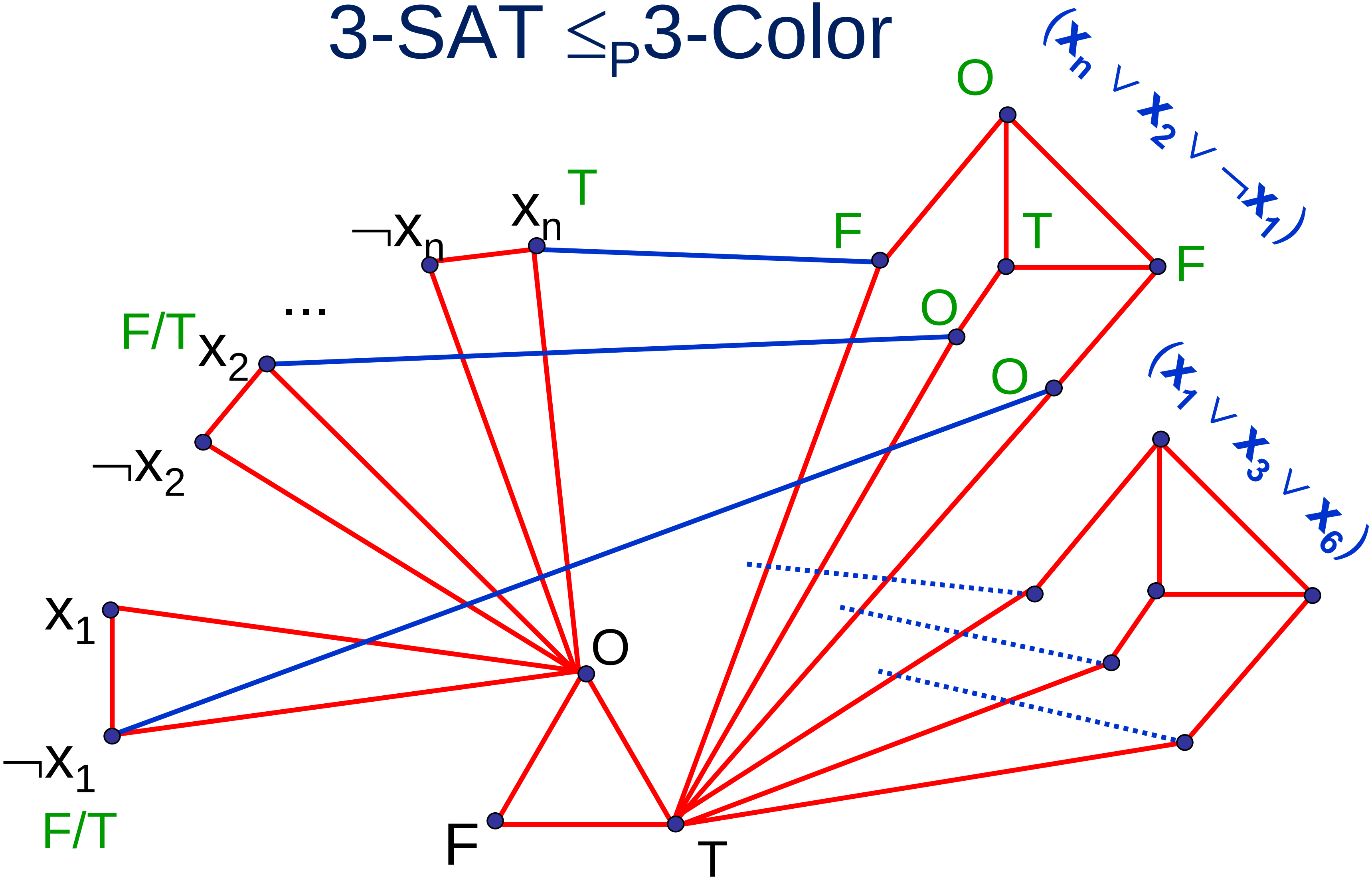
3-SAT $\leq_P$ 3-Color



Clause Part:

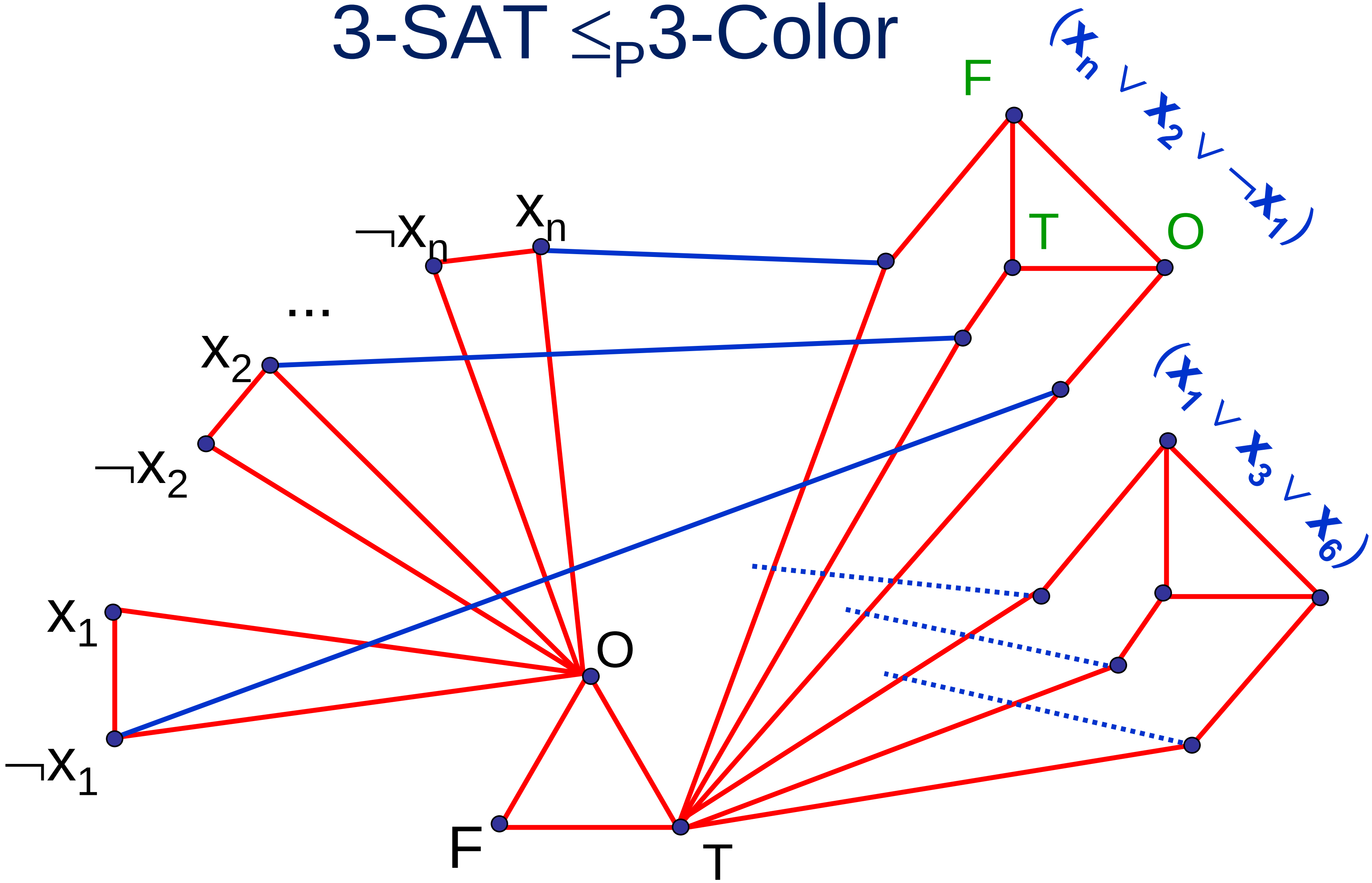
Add one 6 vertex gadget per clause connecting its ‘outer vertices’ to the literals in the clause

3-SAT $\leq_P$ 3-Color



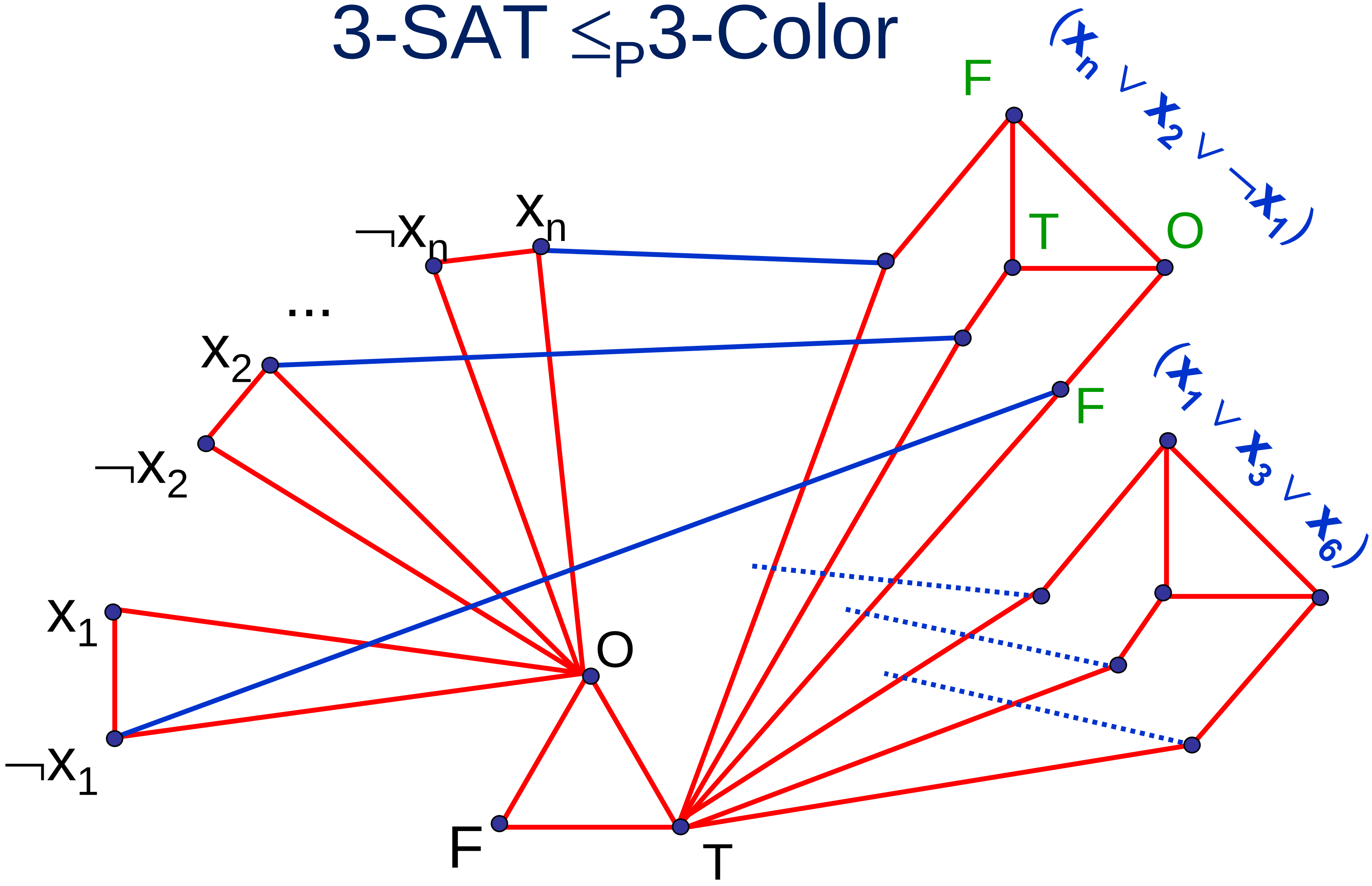
Any truth assignment satisfying the formula can be extended to a 3-coloring of the graph

3-SAT $\leq_P$ 3-Color



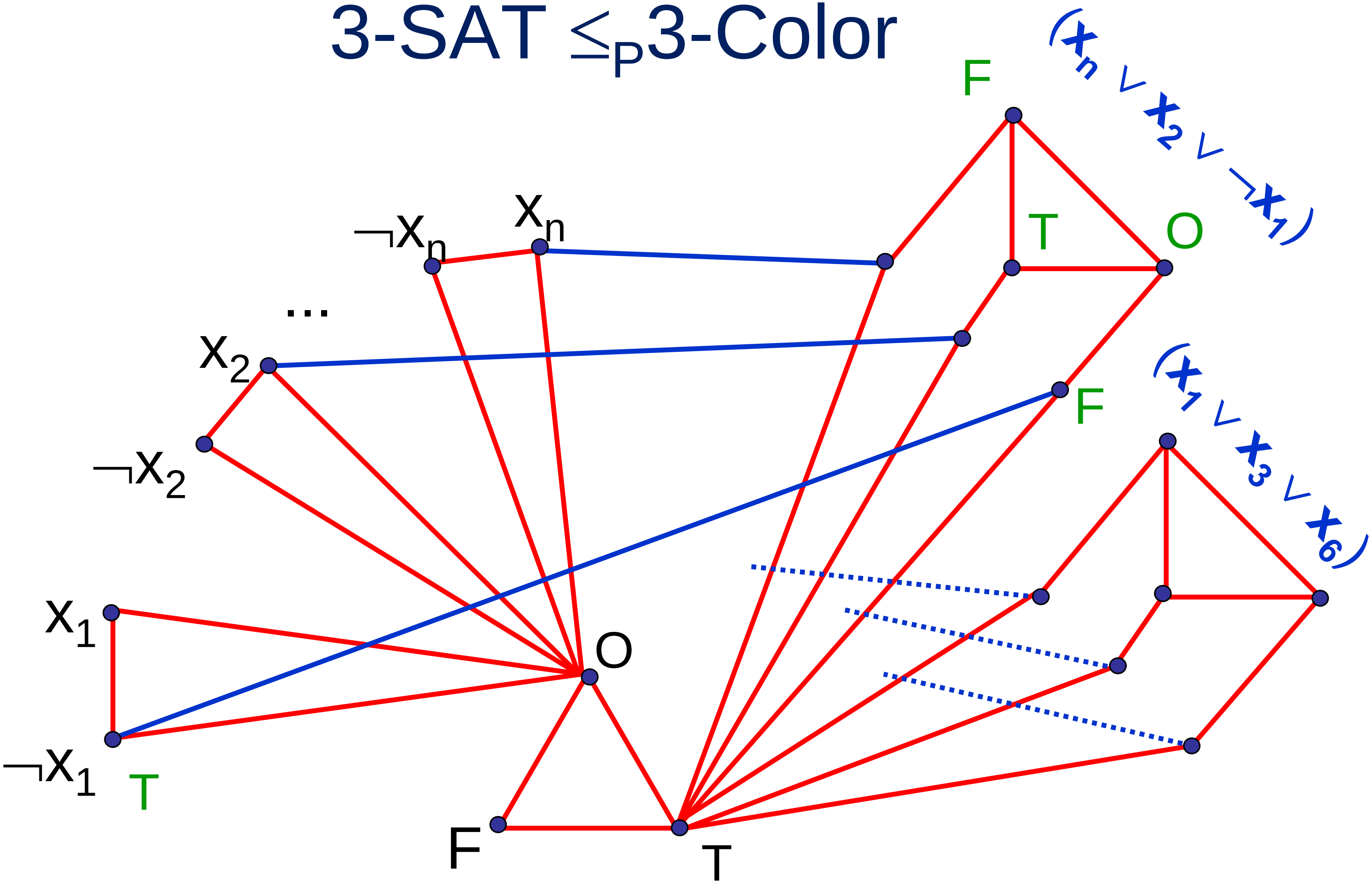
Any 3-coloring of the graph colors
each gadget triangle using each color

3-SAT $\leq_P$ 3-Color



Any 3-coloring of the graph has an F opposite the O color in the triangle of each gadget

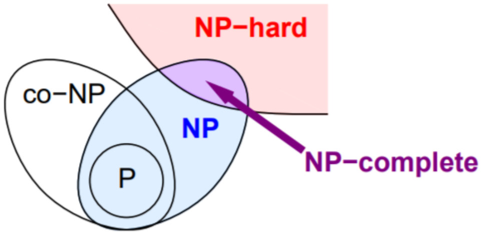
3-SAT $\leq_P$ 3-Color



Any 3-coloring of the graph has T at the other end of the blue edge connected to the F

Summary

- If a problem is NP-hard it does not mean that all instances are hared, e.g., Vertex-cover has a polynomial-time algorithm in trees
- We learned the crucial idea of polynomial-time reduction. This can be even used in algorithm design, e.g., we know how to solve max-flow so we reduce image segmentation to max-flow
- NP-Complete problems are the hardest problem in NP
- NP-hard problems may not necessarily belong to NP.
- Polynomial-time reductions are transitive relations



More of what we *think* the world looks like.