CSE 417

NP-completeness reductions

Spring 2006
Paul Beame

1

Polynomial time

- Define **P** (polynomial-time) to be
  - the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.
- **P** = $\cup_{k\geq 0}$TIME(n^k)

2

The complexity class NP

NP consists of all decision problems where

- You can **verify** the **YES** answers efficiently (in polynomial time) given a short (polynomial-size) **certificate**

And

- **No certificate** can fool your polynomial time verifier into saying **YES** for a **NO** instance

3

NP

- There are many natural, practical problems for which we don't know any polynomial-time algorithms
- e.g. decisionTSP:
  - Given a weighted graph **G** and an integer **k**, does there exist a tour that visits all vertices in **G** having total weight at most **k**?

4

More Precise Definition of NP

- A decision problem is in NP iff there is a polynomial time procedure verify(.,.), and an integer k such that
  - for every input **x** to the problem that is a **YES** instance there is a certificate **c** with $|\mathbf{c}| \leq |\mathbf{x}|^k$ such that **verify**(**x**,**c**) = **YES**

  and
  - for every input **x** to the problem that is a **NO** instance there does **not** exist a certificate **c** with $|\mathbf{c}| \leq |\mathbf{x}|^k$ such that **verify**(**x**,**c**) = **YES**

5

Keys to showing that a problem is in NP

- What's the output? (must be **YES**/**NO**)
- What must the input look like?
- Which inputs need a **YES** answer?
  - Call such inputs **YES** inputs/**YES** instances
- For every given **YES** input, is there a certificate that would help?
  - OK if some inputs need no certificate
- For any given **NO** input, is there a fake certificate that would trick you?

6

Example: CLIQUE is in NP

procedure **verify**(**x**,**c**)

if

c is a well-formed representation of a graph **G** = (**V**, **E**) and an integer **k**,

and

c is a well-formed representation of a vertex subset **U** of **V** of size **k**,

and

U is a clique in **G**,

then output "**YES**"

else output "**I'm unconvinced**"

Is it correct?

For every **x** = ⟨**G**,**k**⟩ such that **G** contains a **k**-clique, there is a certificate **c** that will cause **verify**(**x**,**c**) to say **YES**,

- **c** = a list of the vertices in such a **k**-clique

And no certificate can fool **verify**(**x**,·) into saying **YES** if either

- **x** isn't well-formed (the uninteresting case)
- **x** = ⟨**G**,**k**⟩ but **G** does not have any cliques of size **k** (the interesting case)

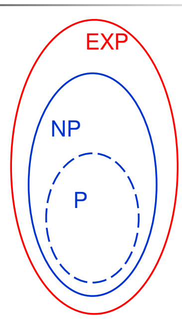
Solving NP problems without hints

- The only **obvious algorithm** for most of these problems is **brute force**:
  - try all possible certificates and check each one to see if it works.
  - *Exponential* time:
    - 2^n truth assignments for **n** variables
    - **n!** possible TSP tours of **n** vertices
    - $\binom{n}{k}$ possible **k** element subsets of **n** vertices
    - etc.

What We Know

- Nobody knows if all problems in **NP** can be done in polynomial time, i.e. does **P**=**NP**?
  - one of the most important open questions in all of science.
  - huge practical implications
- Every problem in **P** is in **NP**
  - one doesn't even need a certificate for problems in **P** so just ignore any hint you are given
- Every problem in **NP** is solvable in exponential time

P and NP



EXP = $\bigcup_{k\geq 0}$ **TIME**(2^{n^k})

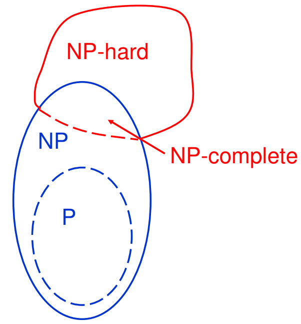
NP-hardness & NP-completeness

- Alternative approach to proving problems not in **P**
  - show that they are at least as hard as any problem in **NP**
- Rough definition:
  - A problem is NP-hard iff it is at least as hard as any problem in **NP**
  - A problem is NP-complete iff it is both
    - **NP**-hard
    - in **NP**

NP-hardness & NP-completeness

- Definition: A problem **B** is NP-hard iff every problem **A∈NP** satisfies $\mathbf{A} \le_P \mathbf{B}$
- Definition: A problem **B** is NP-complete iff **B** is NP-hard and **B ∈NP**
- Even though we seem to have lots of hard problems in **NP** it is not obvious that such super-hard problems even exist!

P and NP



Reductions by Simple Equivalence

- Show: Independent-Set $\le_P$ Clique
- Independent-Set:
  - Given a graph **G**=(**V**,**E**) and an integer **k**, is there a subset **U** of **V** with |**U**| ≥ **k** such that no two vertices in **U** are joined by an edge.
- Clique:
  - Given a graph **G**=(**V**,**E**) and an integer **k**, is there a subset **U** of **V** with |**U**| ≥ **k** such that every pair of vertices in **U** is joined by an edge.

Independent-Set $\le_P$ Clique

- Given ⟨**G**,**k**⟩ as input to Independent-Set where **G**=(**V**,**E**)
- Transform to ⟨**G'**,**k**⟩ where **G'**=(**V**,**E'**) has the same vertices as **G** but **E'** consists of **precisely** those edges that are not edges of **G**
- **U** is an independent set in **G**

⇔ **U** is a clique in **G'**

Satisfiability

- Boolean variables $\mathbf{x_1,\dots,x_n}$
  - taking values in {**0**,**1**}. **0**=false, **1**=true
- Literals
  - $\mathbf{x_i}$ or $\neg\mathbf{x_i}$ for **i**=**1**,...,**n**
- Clause
  - a logical OR of one or more literals
  - e.g. $(\mathbf{x_1} \vee \neg\mathbf{x_3} \vee \mathbf{x_7} \vee \mathbf{x_{12}})$
- CNF formula
  - a logical AND of a bunch of clauses

Satisfiability

- CNF formula example
  - $(\mathbf{x_1} \vee \neg\mathbf{x_3} \vee \mathbf{x_7} \vee \mathbf{x_{12}}) \wedge (\mathbf{x_2} \vee \neg\mathbf{x_4} \vee \mathbf{x_7} \vee \mathbf{x_5})$
- If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable
  - the one above is, the following isn't
  - $\mathbf{x_1} \wedge (\neg\mathbf{x_1} \vee \mathbf{x_2}) \wedge (\neg\mathbf{x_2} \vee \mathbf{x_3}) \wedge \neg\mathbf{x_3}$
- SAT: Given a formula F, is it satisfiable?

Cook-Levin Theorem

- Theorem (Cook-Levin 1971):
  SAT∈P ⇔ P=NP

Follows by showing that **SAT** is **NP-**complete

19

Implications of Cook's Theorem?

- There is at least one interesting super-hard problem in **NP**
- Is that such a big deal?
- YES!
  - There are lots of other problems that can be solved if we had a polynomial-time algorithm for **SAT**
  - Many of these problems are exactly as hard as **SAT**

20

Recall this useful property of polynomial-time reductions

- Theorem: If $A \leq_P B$ and $B \leq_P C$ then $A \leq_P C$

21

Cook-Levin Theorem & Implications

- Theorem: **SAT** is **NP**-complete
- Corollary: **C** is **NP**-hard ⇔ $SAT \leq_P C$
  - (or $B \leq_P C$ for any **NP**-complete problem **B**)
- Proof:
  - If **B** is **NP**-hard then every problem in **NP** polynomial-time reduces to **B**, in particular **SAT** does since it is in **NP**
  - For any problem **A** in **NP**, $A \leq_P SAT$ and so if $SAT \leq_P C$ we have $A \leq_P C$.
    - therefore **C** is **NP**-hard if $SAT \leq_P C$

22

Steps to Proving Problem B is NP-complete

- Show **B** is **NP**-hard:
  - State:`Reduction is from **NP**-hard Problem **A**'
  - 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 certificate is and why it works
  - Argue that it is polynomial-time to check.

23

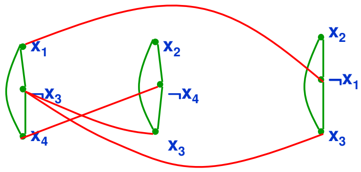
Another NP-complete problem: Satisfiability $\leq_P$ Independent-Set

- A Tricky Reduction:
  - mapping CNF formula **F** to a pair **<G,k>**
  - Let **m** be the number of clauses of **F**
  - Create a vertex in **G** for each literal in **F**
  - Join two vertices **u, v** in **G** by an edge iff
    - **u** and **v** correspond to literals in the same clause of **F**, (green edges) or
    - **u** and **v** correspond to literals **x** and **¬x** (or vice versa) for some variable **x**. (red edges).
  - Set **k=m**
  - Clearly polynomial-time

24

Satisfiability $\le^P$ Independent-Set

F: $(x_1 \lor \neg x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_3) \land (x_2 \lor \neg x_1 \lor x_3)$



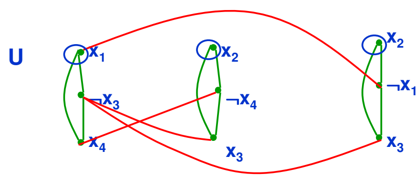
Satisfiability $\le^P$ Independent-Set

- Correctness:
  - If **F** is **satisfiable** then there is some assignment that satisfies at least one literal in each clause.
  - Consider the set **U** in **G** corresponding to the **first satisfied literal in each clause**.
    - $|U|=m$
    - Since **U** has only one vertex per clause, no two vertices in **U** are joined by green edges
    - Since a truth assignment never satisfies both **x** and **¬x**, **U** doesn't contain vertices labeled both **x** and **¬x** and so no vertices in **U** are joined by red edges
    - Therefore **G** has an independent set, **U**, of size at least **m**
  - Therefore (**G**,**m**) is a **YES** for independent set.

Satisfiability $\le^P$ Independent-Set

1 0 1 1 0 1 1 0 1

F: $(x_1 \lor \neg x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_3) \land (x_2 \lor \neg x_1 \lor x_3)$



Given assignment $x_1=x_2=x_3=x_4=1$,
U is as circled

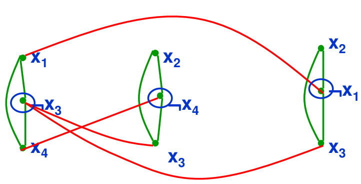
Satisfiability $\le^P$ Independent-Set

- Correctness continued:
  - If (**G**,**m**) is a **YES** for Independent-Set then there is a set **U** of **m** vertices in **G** containing no edge.
    - Therefore **U** has precisely one vertex per clause because of the green edges in **G**.
    - Because of the red edges in **G**, **U** does not contain vertices labeled both **x** and **¬x**
    - Build a truth assignment **A** that makes all literals labeling vertices in **U** true and for any variable not labeling a vertex in **U**, assigns its truth value arbitrarily.
    - By construction, **A** satisfies **F**
  - Therefore **F** is a **YES** for Satisfiability.

Satisfiability $\le^P$ Independent-Set

0 1 0 ? 1 0 ? 1 0

F: $(x_1 \lor \neg x_3 \lor x_4) \land (x_2 \lor \neg x_4 \lor x_3) \land (x_2 \lor \neg x_1 \lor x_3)$



Given **U**, satisfying assignment
is $x_1=x_3=x_4=0$, $x_2=0$ or 1

Independent-Set is NP-complete

- We just showed that **Independent-Set** is NP-hard and we already knew **Independent-Set** is in **NP**.
- Corollary: **Clique** is **NP**-complete
  - We showed already that **Independent-Set** $\le_P$ **Clique** and **Clique** is in **NP**.

Reductions from a Special Case to a General Case

- Show: Vertex-Cover $\le_P$ Set-Cover
- Vertex-Cover:
  - Given an undirected graph $G=(V,E)$ and an integer k is there a subset W of V of size at most k such that every edge of G has at least one endpoint in W? (i.e. W covers all edges of G).
- Set-Cover:
  - Given a set U of n elements, a collection $S_1,\ldots,S_m$ of subsets of U, and an integer k, does there exist a collection of at most k sets whose union is equal to U?

The Simple Reduction

- Transformation f maps $\langle G=(V,E),k\rangle$ to $\langle U,S_1,\ldots,S_m,k'\rangle$
  - $U \leftarrow E$
  - For each vertex $v \in V$ create a set S_v containing all edges that touch v
  - $k' \leftarrow k$
- Reduction f is clearly polynomial-time to compute
- We need to prove that the resulting algorithm gives the right answer.

Proof of Correctness

- Two directions:
  - If the answer to Vertex-Cover on (G,k) is YES then the answer for Set-Cover on $T(G,k)$ is YES
    - If a set W of k vertices covers all edges then the collection $\{S_v \mid v \in W\}$ of k sets covers all of U
  - If the answer to Set-Cover on $T(G,k)$ is YES then the answer for Vertex-Cover on (G,k) is YES
    - If a subcollection $S_{v_1},\ldots,S_{v_k}$ covers all of U then the set $\{v_1,\ldots,v_k\}$ is a vertex cover in G.

More Reductions

- Show: Independent Set $\le_P$ Vertex-Cover
- Vertex-Cover:
  - Given an undirected graph $G=(V,E)$ and an integer k is there a subset W of V of size at most k such that every edge of G has at least one endpoint in W? (i.e. W covers all edges of G).
- Independent-Set:
  - Given a graph $G=(V,E)$ and an integer k, is there a subset U of V with $|U| \ge k$ such that no two vertices in U are joined by an edge.

Reduction Idea

- Claim: In a graph $G=(V,E)$, S is an independent set iff $V\text{-}S$ is a vertex cover
- Proof:
  - $\Rightarrow$ Let S be an independent set in G
    - Then S contains at most one endpoint of each edge of G
    - At least one endpoint must be in $V\text{-}S$
    - $V\text{-}S$ is a vertex cover
  - $\Leftarrow$ Let $W=V\text{-}S$ be a vertex cover of G
    - Then S does not contain both endpoints of any edge (else W would miss that edge)
    - S is an independent set

Reduction

- Map $\langle G,k\rangle$ to $\langle G,n\text{-}k\rangle$
  - Previous lemma proves correctness
- Clearly polynomial time
- We also get that
  - Vertex-Cover $\le_P$ Independent Set

Problems we already know are NP-complete

- Satisfiability
- Independent-Set
- Clique
- Vertex-Cover

- There are 1000's of practical problems that are NP-complete, e.g. scheduling, optimal VLSI layout etc.

37

Is NP as bad as it gets?

- NO! **NP**-complete problems are frequently encountered, but there's worse:
  - Some problems provably require exponential time.
    - Ex: Does **P** halt on **x** in $2^{|x|}$ steps?
  - Some require 2^n, 2^{2^n}, $2^{2^{2^n}}$, ... steps
  - And of course, some are just plain uncomputable

38

A particularly useful problem for proving NP-completeness

- **3-SAT**: Given a CNF formula **F** having **precisely 3 variables per clause** (i.e., in 3-CNF), is **F** satisfiable?

- **Theorem: 3-SAT** is **NP**-complete
- **Alternate Proof based on CNFSAT:**
  - **3-SAT∈NP**
    - Certificate is a satisfying assignment
    - Just like SAT it is polynomial-time to check the certificate

39

CNFSAT $\le_P$ 3-SAT

- Reduction:
  - map CNF formula **F** to another CNF formula **G** that has precisely **3** variables per clause.
    - **G** has one or more clauses for each clause of **F**
    - **G** will have extra variables that don't appear in **F**
      - for each clause **C** of **F** there will be a different set of variables that are used only in the clauses of **G** that correspond to **C**

40

CNFSAT $\le_P$ 3-SAT

- Goal:
  - An assignment **a** to the original variables makes clause **C** true in **F** iff
    - there is an assignment to the extra variables that together with the assignment **a** will make all new clauses corresponding to **C** true.
- Define the reduction clause-by-clause
  - We'll use variable names z_j to denote the extra variables related to a single clause **C** to simplify notation
    - in reality, two different original clauses will not share z_j

41

CNFSAT $\le_P$ 3-SAT

- For each clause C in F:
  - If **C** has **3** variables:
    - Put **C** in **G** as is
  - If **C** has **2** variables, e.g. $C=(x_1 \lor \neg x_3)$
    - Use a new variable **z** and put two clauses in **G**
      $(x_1 \lor \neg x_3 \lor z) \land (x_1 \lor \neg x_3 \lor \neg z)$
    - If original **C** is true under assignment **a** then both new clauses will be true under **a**
    - If new clauses are both true under some assignment **b** then the value of **z** doesn't help in one of the two clauses so **C** must be true under **b**

42

CNFSAT $\le_p$ 3-SAT

- If **C** has **1** variable: e.g. $\mathbf{C=x_1}$
  - Use two new variables $\mathbf{z_1}$, $\mathbf{z_2}$ and put **4** new clauses in **G**
    $(x_1 \vee \neg z_1 \vee \neg z_2) \wedge (x_1 \vee \neg z_1 \vee z_2) \wedge$
    $(x_1 \vee z_1 \vee \neg z_2) \wedge (x_1 \vee z_1 \vee z_2)$
  - If original **C** is true under assignment **a** then all new clauses will be true under **a**
  - If new clauses are all true under some assignment **b** then the values of $\mathbf{z_1}$ and $\mathbf{z_2}$ don't help in one of the 4 clauses so **C** must be true under **b**

CNFSAT $\le_p$ 3-SAT

- If **C** has $\mathbf{k \ge 4}$ variables: e.g. $\mathbf{C}=(x_1 \vee \dots \vee x_k)$
  - Use **k-3** new variables $\mathbf{z_2,\dots,z_{k-2}}$ and put **k-2** new clauses in **G**
    $(x_1 \vee x_2 \vee z_2) \wedge (\neg z_2 \vee x_3 \vee z_3) \wedge (\neg z_3 \vee x_4 \vee z_4) \wedge \dots$
    $\wedge (\neg z_{k-3} \vee x_{k-2} \vee z_{k-2}) \wedge (\neg z_{k-2} \vee x_{k-1} \vee x_k)$
  - If original **C** is true under assignment **a** then some $\mathbf{x_i}$ is true for $\mathbf{i \le k}$. By setting $\mathbf{z_j}$ true for all $\mathbf{j<i}$ and false for all $\mathbf{j \ge i}$, we can extend **a** to make all new clauses true.
  - If new clauses are all true under some assignment **b** then some $\mathbf{x_i}$ must be true for $\mathbf{i \le k}$ because $z_2 \wedge (\neg z_2 \vee z_3) \wedge \dots \wedge (\neg z_{k-3} \vee z_{k-2}) \wedge \neg z_{k-2}$ is not satisfiable

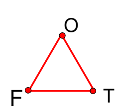
Graph Colorability

- 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:
  - Hint is an assignment of red,green,blue to the vertices of G
  - Easy to check that each edge is colored correctly

3-SAT $\le_p$ 3-Color

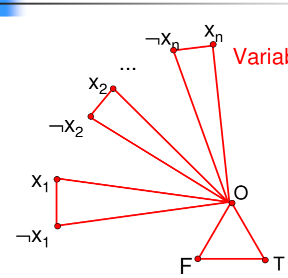
- Reduction:
  - We want to map a 3-CNF formula ⟨**F**⟩ to a graph ⟨**G**⟩ so that
    - **G** is 3-colorable iff **F** is satisfiable

3-SAT $\le_p$ 3-Color

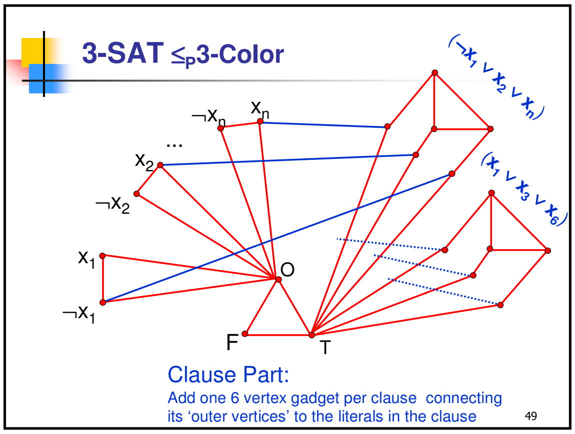


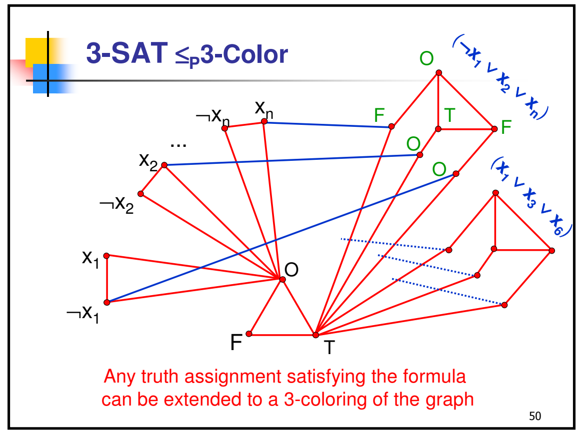
Base Triangle

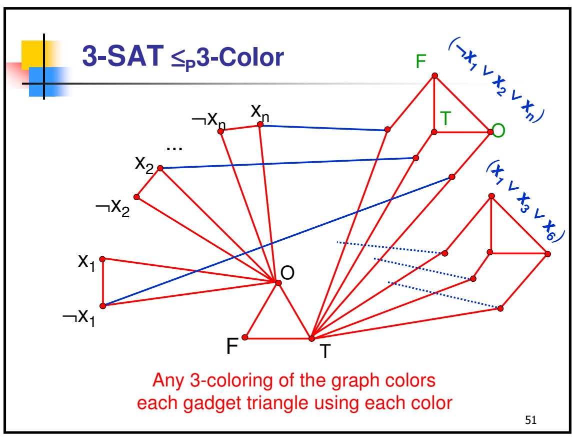
3-SAT $\le_p$ 3-Color

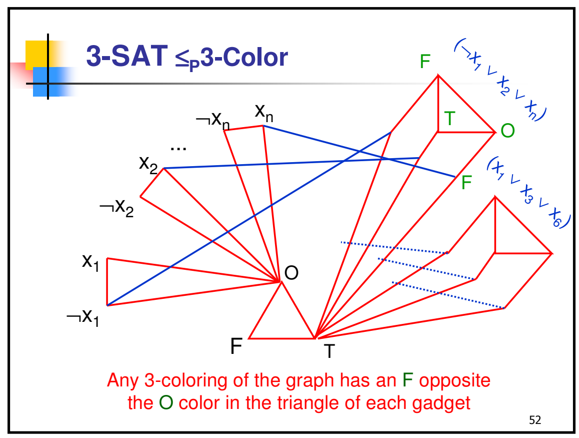


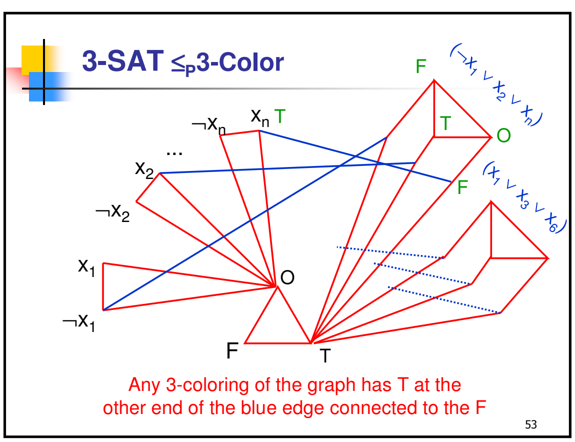
Variable Part:
in 3-coloring, variable colors correspond to some truth assignment (same color as T or F)

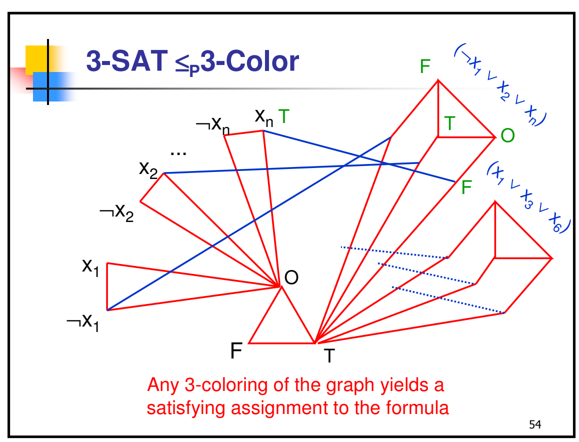

3-SAT $\leq_P$ 3-Color
$(\neg x_1 \vee x_2 \vee x_n)$
$(x_1 \vee x_3 \vee x_6)$
$\neg x_n$ x_n
...
x_2
$\neg x_2$
x_1
$\neg x_1$
O
F
T
Clause Part:
Add one 6 vertex gadget per clause connecting its 'outer vertices' to the literals in the clause
49


3-SAT $\leq_P$ 3-Color
$(\neg x_1 \vee x_2 \vee x_n)$
$(x_1 \vee x_3 \vee x_6)$
O T F F O O
$\neg x_n$ x_n
...
x_2
$\neg x_2$
x_1
$\neg x_1$
O
F
T
Any truth assignment satisfying the formula can be extended to a 3-coloring of the graph
50


3-SAT $\leq_P$ 3-Color
$(\neg x_1 \vee x_2 \vee x_n)$
$(x_1 \vee x_3 \vee x_6)$
F T O
$\neg x_n$ x_n
...
x_2
$\neg x_2$
x_1
$\neg x_1$
O
F
T
Any 3-coloring of the graph colors each gadget triangle using each color
51


3-SAT $\leq_P$ 3-Color
$(\neg x_1 \vee x_2 \vee x_n)$
$(x_1 \vee x_3 \vee x_6)$
F T O F
$\neg x_n$ x_n
...
x_2
$\neg x_2$
x_1
$\neg x_1$
O
F
T
Any 3-coloring of the graph has an F opposite the O color in the triangle of each gadget
52


3-SAT $\leq_P$ 3-Color
$(\neg x_1 \vee x_2 \vee x_n)$
$(x_1 \vee x_3 \vee x_6)$
F T O F
$\neg x_n$ x_n T
...
x_2
$\neg x_2$
x_1
$\neg x_1$
O
F
T
Any 3-coloring of the graph has T at the other end of the blue edge connected to the F
53


3-SAT $\leq_P$ 3-Color
$(\neg x_1 \vee x_2 \vee x_n)$
$(x_1 \vee x_3 \vee x_6)$
F T O F
$\neg x_n$ x_n T
...
x_2
$\neg x_2$
x_1
$\neg x_1$
O
F
T
Any 3-coloring of the graph yields a satisfying assignment to the formula
54

More NP-completeness

- Subset-Sum problem
  - 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

55

3-SAT $\leq_P$ Subset-Sum

- Given a 3-CNF formula with **m** clauses and **n** variables
- Will create **2m+2n** 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)

56

3-SAT $\leq_P$ Subset-Sum

$C_4 = (x_1 \vee \neg x_2 \vee x_5)$

				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

57