CSE 421: Introduction to Algorithms

NP-completeness

Winter 2003
Paul Beame

1

Computational Complexity

- Classify problems according to the amount of computational resources used by the best algorithms that solve them
- Recall:
  - worst-case running time of an algorithm
    - **max** # steps algorithm takes on any input of size **n**
- Define:
  - **TIME**(**f**(**n**)) to be the set of all decision problems solved by algorithms having worst-case running time **O**(**f**(**n**))

2

Decision problems

- Computational complexity usually analyzed using decision problems
  - answer is just 1 or 0 (yes or no).
- Why?
  - much simpler to deal with
  - *deciding* whether **G** has a path from s to t, is certainly no harder than *finding* a path from s to t in G, so a *lower* bound on deciding is also a lower bound on finding
  - Less important, but if you have a good decider, you can often use it to get a good finder.

3

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

4

Beyond P?

- 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**?

5

Relative Complexity of Problems

- Want a notion that allows us to compare the complexity of problems
  - Want to be able to make statements of the form

    "If we could solve problem **R** in polynomial time then we can solve problem **L** in polynomial time"

    "Problem **R** is at least as hard as problem **L**"

6

Polynomial Time Reduction

- **L** $£_P$ **R** if there is an algorithm for **L** using a 'black box' (subroutine) that solves **R** that
  - Uses only a polynomial number of steps
  - Makes only a polynomial number of calls to a subroutine for **R**
- Thus, poly time algorithm for **R** implies poly time algorithm for **L**
  - Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!
- If you can prove there is no fast algorithm for **L**, then that proves there is no fast algorithm for **R**

A Special kind of Polynomial-Time Reduction

- We will always use a restricted form of polynomial-time reduction often called Karp or many-one reduction
- $\mathbf{L} \leq_P^1 \mathbf{R}$ if and only if there is an algorithm for **L** given a black box solving **R** that on input **x**
  - Runs for polynomial time computing an input **T(x)**
  - Makes one call to the black box for **R**
  - Returns the answer that the black box gave

We say that the function **T** is the reduction

Why the name reduction?

- Weird: it maps an easier problem into a harder one
- Same sense as saying Maxwell reduced the problem of analyzing electricity & magnetism to solving partial differential equations
  - solving partial differential equations in general is a much harder problem than solving E&M problems

A geek joke

- An engineer
  - is placed in a kitchen with an empty kettle on the table and told to boil water; she fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - she is next confronted with a kettle full of water sitting on the counter and told to boil water; she puts it on the stove, turns on the gas and boils water.
- A mathematician
  - is placed in a kitchen with an empty kettle on the table and told to boil water; he fills the kettle with water, puts it on the stove, turns on the gas and boils water.
  - he is next confronted with a kettle full of water sitting on the counter and told to boil water: he empties the kettle in the sink, places the empty kettle on the table and says, "I've reduced this to an already solved problem".

Reductions from a Special Case to a General Case

- Show: Vertex-Cover $£_P$ 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 $\mathbf{S_1,\ldots,S_m}$ 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 **T** maps $(\mathbf{G}=(\mathbf{V},\mathbf{E}),\mathbf{k})$ to $(\mathbf{U},\mathbf{S_1},\ldots,\mathbf{S_m},\mathbf{k'})$
  - **U** ¬ **E**
  - For each vertex **v**∈**V** create a set $\mathbf{S_v}$ containing all edges that touch **v**
  - **k'** ¬ **k**
- Reduction **T** 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 {$\mathbf{S_v}$ | **v** Î **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 $\mathbf{S_{v_1},\ldots,S_{v_k}}$ covers all of **U** then the set $\mathbf{\{v_1,\ldots,v_k\}}$ is a vertex cover in **G**.

13

Reductions by Simple Equivalence

- Show: Independent-Set $£_P$ 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.

14

Independent-Set $\leq_P$ 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'**

15

More Reductions

- Show: Independent Set $£_P$ 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**| ≥ **k** such that no two vertices in **U** are joined by an edge.

16

Reduction Idea

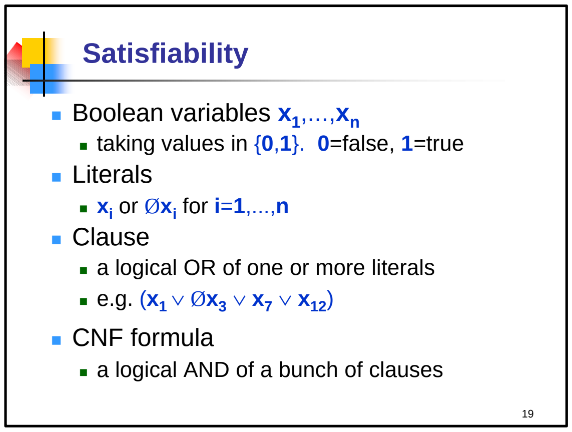
- Claim: In a graph **G**=(**V**,**E**), **S** is an independent set iff **V**-**S** is a vertex cover
- Proof:
  - ⇒ Let **S** be an independent set in **G**
    - Then **S** contains at most one endpoint of each edge of **G**
    - At least one endpoint must be in **V-S**
    - **V-S** is a vertex cover
  - ⇐Let **W=V-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

17

Reduction

- Map (**G**,**k**) to (**G**,**n-k**)
  - Previous lemma proves correctness
- Clearly polynomial time
- We also get that
  - Vertex-Cover $£_P$ Independent Set

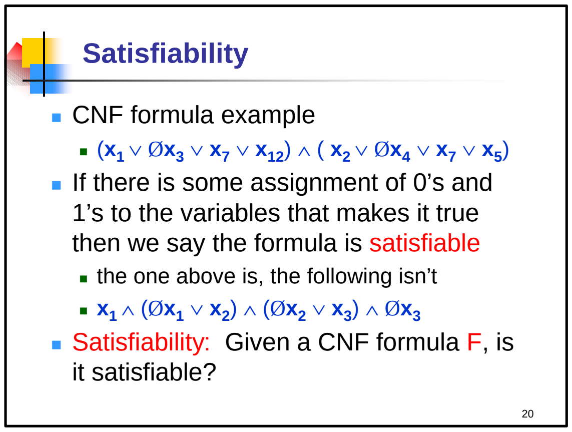
18



Satisfiability

- Boolean variables $x_1,\ldots,x_n$
  - taking values in $\{0,1\}$. 0=false, 1=true
- Literals
  - x_i or $Øx_i$ for $i=1,\ldots,n$
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \lor Øx_3 \lor x_7 \lor x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses

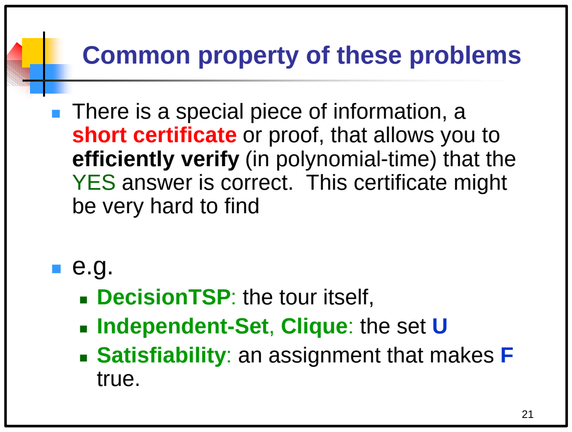
19



Satisfiability

- CNF formula example
  - $(x_1 \lor Øx_3 \lor x_7 \lor x_{12}) \land (x_2 \lor Øx_4 \lor x_7 \lor x_5)$
- 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
  - $x_1 \land (Øx_1 \lor x_2) \land (Øx_2 \lor x_3) \land Øx_3$
- Satisfiability: Given a CNF formula **F**, is it satisfiable?

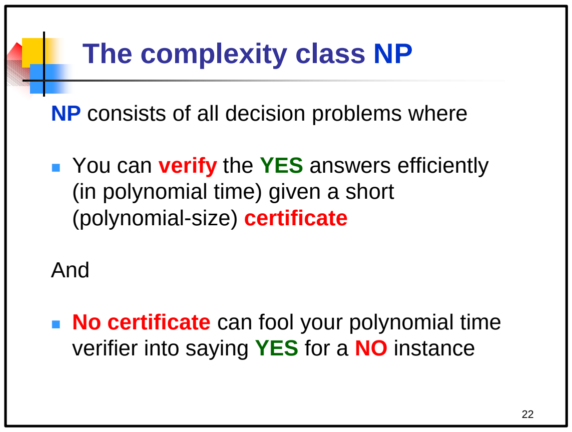
20



Common property of these problems

- There is a special piece of information, a **short certificate** or proof, that allows you to **efficiently verify** (in polynomial-time) that the YES answer is correct. This certificate might be very hard to find
- e.g.
  - **DecisionTSP**: the tour itself,
  - **Independent-Set**, **Clique**: the set **U**
  - **Satisfiability**: an assignment that makes **F** true.

21



The complexity class NP

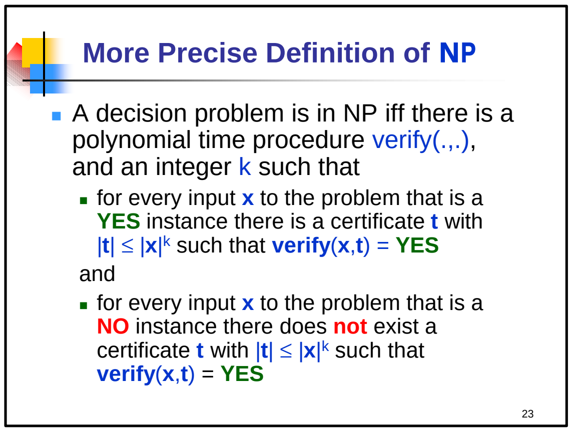
NP consists of all decision problems where

- 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

22

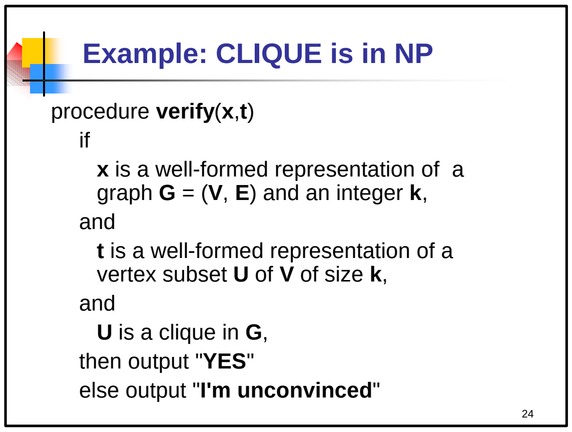


More Precise Definition of NP

- 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 **t** with $|t| \le |x|^k$ such that **verify(x,t)** = **YES**

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

23



Example: CLIQUE is in NP

```
procedure verify(x,t)
    if
        x is a well-formed representation of a
        graph G = (V, E) and an integer k,
    and
        t 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"
```

24

Is it correct?

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

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

And no certificate can fool **verify(x,χ)** 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)

25

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?

26

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.

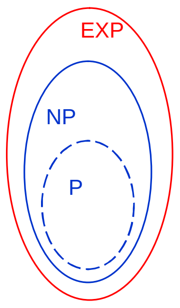
27

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 in exponential time

28

P and NP



$\mathrm{EXP} = \mathrm{U}_{k^3 0}\mathrm{TIME}(2^{n^k})$

29

NP-hardness & NP-completeness

- Some problems in **NP** seem hard
  - people have looked for efficient algorithms for them for hundreds of years without success
- However
  - nobody knows how to **prove** that they are really hard to solve, i.e. $\mathbf{P}^1$ **NP**

30

Problems in NP that seem hard

- Some Examples in **NP**
  - Satisfiability
  - Independent-Set
  - Clique
  - Vertex Cover
- All hard to solve; certificates seem to help on all
- Fast solution to *any* gives fast solution to *all!*

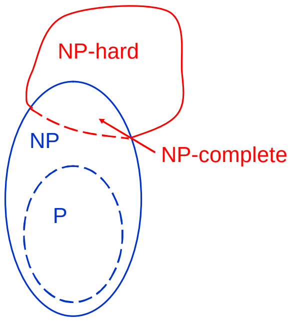
31

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**

32

P and NP



33

NP-hardness & NP-completeness

- Definition: A problem **R** is NP-hard iff every problem **L Î NP** satisfies **L £$_P$R**
- Definition: A problem **R** is NP-complete iff **R** is NP-hard and **R Î NP**
- Even though we seem to have lots of hard problems in **NP** it is not obvious that such super-hard problems even exist!

34

Cook's Theorem

- Theorem (Cook 1971): **Satisfiability** is **NP**-complete
- Recall
  - CNF formula
    - e.g. ($x_1 \vee$ Ø$x_3 \vee x_7 \vee x_{12}$) $\wedge$ ($x_2 \vee$ Ø$x_4 \vee x_7 \vee x_5$)
  - 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
  - **Satisfiability:** Given a CNF formula F, is it satisfiable?

35

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 **Satisfiability**
  - Many of these problems are exactly as hard as **Satisfiability**

36

A useful property of polynomial-time reductions

- Theorem: If L £$_P$R and R £$_P$S then L £$_P$S
- Proof idea: (Using $\leq_P^1$)
  - Compose the reduction T from L to R with the reduction T' from R to S to get a new reduction T''(x)=T'(T(x)) from L to S.
  - The general case is similar and uses the fact that the composition of two polynomials is also a polynomial

37

Cook's Theorem & Implications

- Theorem (Cook 1971): Satisfiability is NP-complete
  For proof see CSE 431
- Corollary: R is NP-hard ⇔ Satisfiability £$_P$R
  - (or Q £$_P$R for any NP-complete problem Q)
- Proof:
  - If R is NP-hard then every problem in NP polynomial-time reduces to R, in particular Satisfiability does since it is in NP
  - For any problem L in NP, L £$_P$Satisfiability and so if Satisfiability £$_P$R we have L £$_P$ R.
    - therefore R is NP-hard if Satisfiability £$_P$R

38

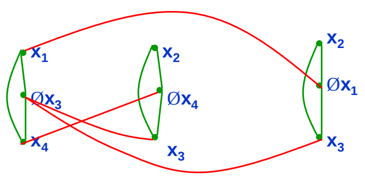
Another NP-complete problem: Satisfiability £$_P$Independent-Set

- 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 Øx (or vice versa) for some variable x. (red edges).
  - Set k=m
  - Clearly polynomial-time

39

Satisfiability £^{P}Independent-Set

F: (x_1 Ú Øx_3 Ú x_4) Ù (x_2 Ú Øx_4 Ú x_3) Ù (x_2 Ú Øx_1 Ú x_3)



40

Satisfiability £^{P}Independent-Set

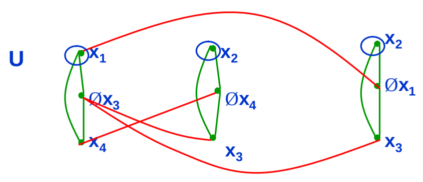
- Correctness:
  - If F is satisfiable then there is some assignment that satisfies at least one literal in each clause.
  - Consider the set U in G corresponding to the **first satisfied literal in each clause**.
    - |U|=m
    - Since U has only one vertex per clause, no two vertices in U are joined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

41

Satisfiability £^{P}Independent-Set

$\overset{1}{}$ 1 0 1 1 0 1 1 0 1

F: (x_1 Ú Øx_3 Ú x_4) Ù (x_2 Ú Øx_4 Ú x_3) Ù (x_2 Ú Øx_1 Ú x_3)



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

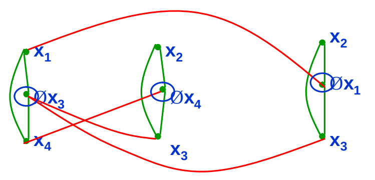
42

Satisfiability £^{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 **Øx**
    - 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 £^{p}Independent-Set

```
     0      1    0        ?      1     0        ?      1     0
F:  (x1 Ú Øx3 Ú x4) Ù ( x2 Ú Øx4 Ú x3) Ù ( x2 Ú Øx1 Ú x3)
```



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** £$_P$ **Clique** and **Clique** is in **NP**.

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.

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

Steps to Proving Problem R is NP-complete

- Show **R** is **NP**-hard:
  - State:`Reduction is from **NP**-hard Problem **L**'
  - Show what the map **T** is
  - Argue that **T** is polynomial time
  - Argue correctness: two directions Yes for **L** implies Yes for **R** and vice versa.
- Show **R** is in **NP**
  - State what hint is and why it works
  - Argue that it is polynomial-time to check.

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?
- Claim: 3-SAT is NP-complete
- Proof:
  - 3-SAT Î NP
    - Certificate is a satisfying assignment
    - Just like Satisfiability it is polynomial-time to check the certificate

49

Satisfiability £$_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

50

Satisfiability £$_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$_j$

51

Satisfiability £$_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$_1$ Ú Øx$_3$)
    - Use a new variable z and put two clauses in G (x$_1$ Ú Øx$_3$ Ú z) ∧ (x$_1$ Ú Øx$_3$ Ú Øz)
    - 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

52

Satisfiability £$_P$3-SAT

- If C has 1 variable: e.g. C=x$_1$
  - Use two new variables z$_1$, z$_2$ and put 4 new clauses in G (x$_1$ Ú Øz$_1$ Ú Øz$_2$) ∧ (x$_1$ Ú Øz$_1$ Ú z$_2$) ∧ (x$_1$ Ú z$_1$ Ú Øz$_2$) ∧ (x$_1$ Ú z$_1$ Ú 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 z$_1$ and z$_2$ don't help in one of the 4 clauses so C must be true under b

53

Satisfiability £$_P$3-SAT

- If C has k ³ 4 variables: e.g. C=(x$_1$ Ú ... Ú x$_k$)
  - Use k-3 new variables z$_2$,...,z$_{k-2}$ and put k-2 new clauses in G
    (x$_1$ Ú x$_2$ Ú z$_2$) ∧ (Øz$_2$ Ú x$_3$ Ú z$_3$) ∧ (Øz$_3$ Ú x$_4$ Ú z$_4$) ∧ ... ∧ (Øz$_{k-3}$ Ú x$_{k-2}$ Ú z$_{k-2}$) ∧ (Øz$_{k-2}$ Ú x$_{k-1}$ Ú x$_k$)
  - If original C is true under assignment a then some x$_i$ is true for i £ k. By setting z$_j$ true for all j<i and false for all j ≥ i, we can extend a to make all new clauses true.
  - If new clauses are all true under some assignment b then some x$_i$ must be true for i ≤ k because z$_2$ ∧ (Øz$_2$ Ú z$_3$) ∧ ... ∧ (Øz$_{k-3}$ Ú z$_{k-2}$) ∧ Øz$_{k-2}$ is not satisfiable

54

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

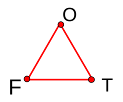
55

3-SAT £$_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

56

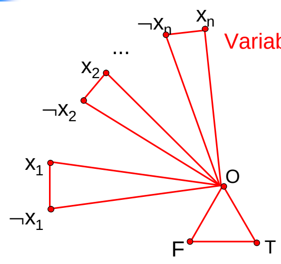
3-SAT £$_P$3-Color



Base Triangle

57

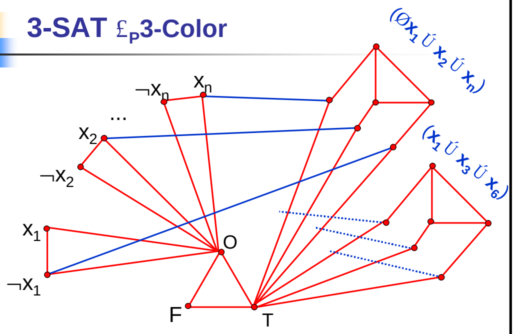
3-SAT £$_P$3-Color



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

58

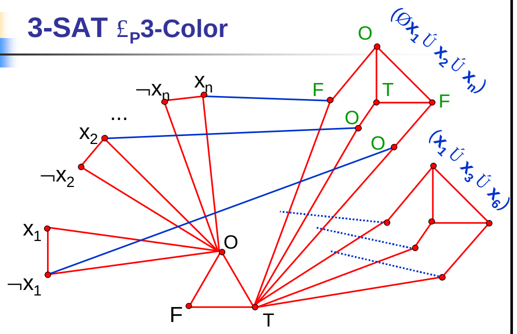
3-SAT £$_P$3-Color



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

59

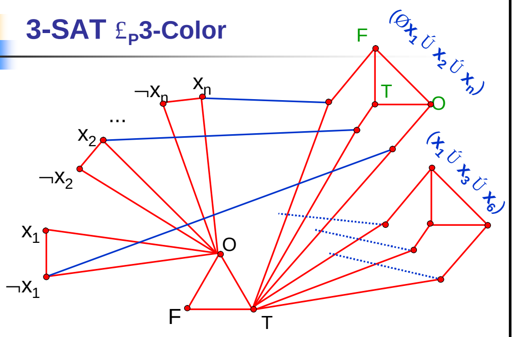
3-SAT £$_P$3-Color



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

60

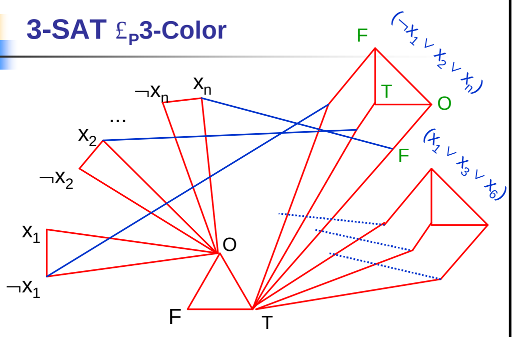
3-SAT £$_P$3-Color



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

61

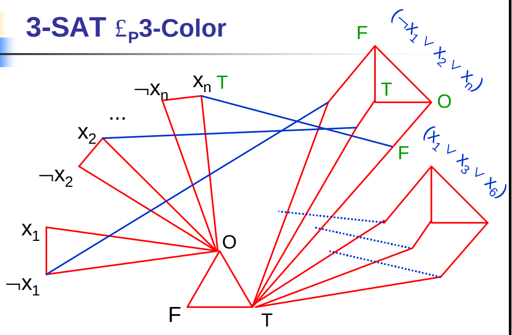
3-SAT £$_P$3-Color



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

62

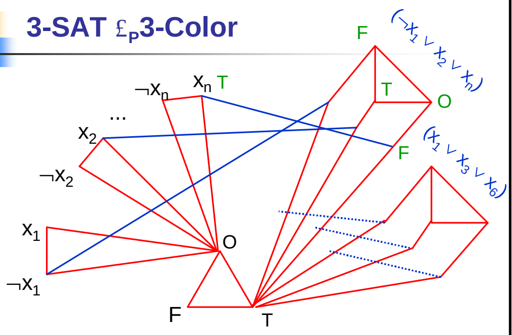
3-SAT £$_P$3-Color



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

63

3-SAT £$_P$3-Color



Any 3-coloring of the graph yields a
satisfying assignment to the formula

64

More NP-completeness

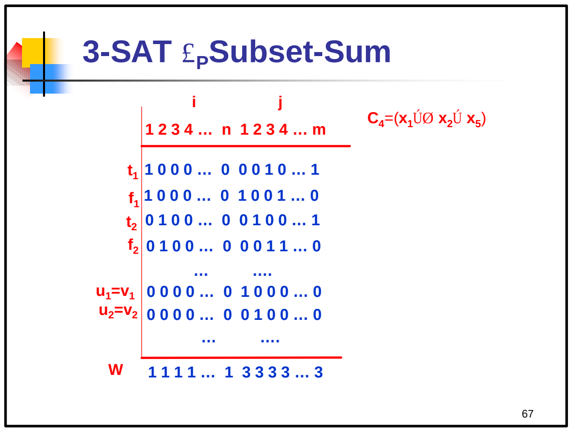
- Subset-Sum problem
  - Given **n** integers $\mathbf{w_1,\dots,w_n}$ and integer **W**
  - Is there a subset of the **n** input integers that adds up to exactly **W**?
- **O(nW)** solution from dynamic programming but if **W** and each $\mathbf{w_i}$ can be **n** bits long then this is exponential time

65

3-SAT £$_P$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 $\mathbf{x_i}$
    - $\mathbf{t_i}$ and $\mathbf{f_i}$ (corresponding to $\mathbf{x_i}$ being true or $\mathbf{x_i}$ being false)
  - Two extra numbers for each clause
    - $\mathbf{u_j}$ and $\mathbf{v_j}$ (filler variables to handle number of false literals in clause $\mathbf{C_j}$)

66

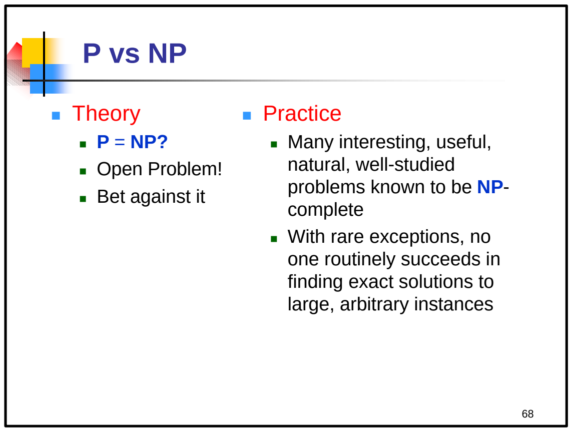


3-SAT £$_P$Subset-Sum

C_4=(x_1ÚØ x_2Ú 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

67



P vs NP

- Theory
  - **P = NP?**
  - Open Problem!
  - Bet against it
- Practice
  - Many interesting, useful, natural, well-studied problems known to be **NP**-complete
  - With rare exceptions, no one routinely succeeds in finding exact solutions to large, arbitrary instances

68