CSE 6140/ CX 4140:

Computational Science and Engineering ALGORITHMS

Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina, Jennifer Welch, George Bebis, and Kevin Wayne

Class policies

1. http://t-square.gatech.edu/.
3. GT Academic Honor Code: www.honor.gatech.edu.
4. No plagiarizing: "to steal and pass off (the ideas or words of another) as one's own: use (another's production) without crediting the source."
5. Late homeworks will not be accepted.
6. Homeworks: you may work with other students in the class. However, each student must write homework solutions in their own words independently, and upload their own homework solutions to T-Square with the collaborators names annotated on every copy of the submission.
7. No collaboration is permitted on exams. The midterm and final exams will be in-class, closed- book exams. You will be allowed to take a "cheat sheet" (double-sided 8.5 x 11 sheet of paper) into each exam.
8. Unauthorized use of **any previous semester course materials**, such as tests, quizzes, homework, projects, and any other coursework, **is prohibited in this course**. Using these materials will be considered a direct violation of academic policy and will be dealt with according to the GT Academic Honor Code.

Further comments

- For proofs, you need to find the correct level of detail, following the recipes seen in class. Please avoid incomplete or very wordy proofs, where you try to disguise answers you aren't sure of.

- Avoid combining multiple styles of proofs (exchange argument + greedy choice/optimal substructure + induction)

- Define clearly the notations that you use: if you return a complexity in O(n), you must have defined n!

- Blindly copy pasting code with 0 explanation (not even pseudocode) is not acceptable.

Reduction By Simple Equivalence

Basic reduction strategies.

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

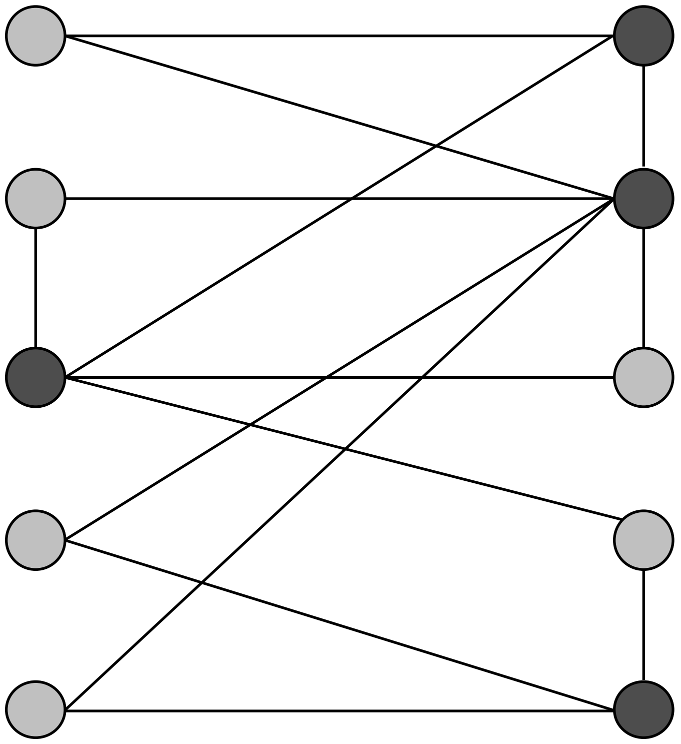
Vertex Cover

MINIMUM VERTEX COVER: Given a graph $G = (V, E)$, **find the smallest** subset of vertices $S \subseteq V$ such that for each edge at least one of its endpoints is in S?

VERTEX COVER: Given a graph $G = (V, E)$ and an integer k, **is there a subset** of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in S?

Ex. Is there a vertex cover of size ≤ 4? Yes.

Ex. Is there a vertex cover of size ≤ 3? No.



● vertex cover

Set Cover

SET COVER: Given a set U of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of U, and an integer k, does there exist a collection of $\leq k$ of these sets whose union is equal to U?

Sample application.

- m available pieces of software.
- Set U of n capabilities that we would like our system to have.
- The ith piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all n capabilities using fewest pieces of software.

Ex:

$U = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$k = 2$

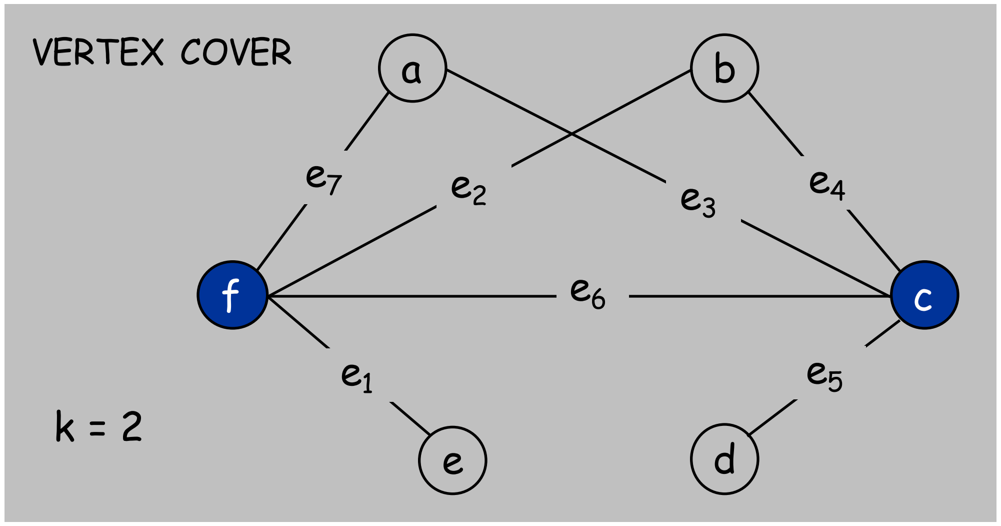
$S_1 = \{3, 7\}$	$S_4 = \{2, 4\}$
$S_2 = \{3, 4, 5, 6\}$	$S_5 = \{5\}$
$S_3 = \{1\}$	$S_6 = \{1, 2, 6, 7\}$

Vertex Cover Reduces to Set Cover (KT 8.1)

Claim. VERTEX-COVER $\leq_P$ SET-COVER.
Pf. Given a VERTEX-COVER instance {G = (V, E), k}, we construct a SET-COVER instance {U, {S}, k'} whose size equals the size of the vertex cover instance.

Construction. (Proof of correctness on paper)

- Create SET-COVER instance:
  - $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$, $k'=k$
- Prove that vertex cover of size $\leq$ k iff set-cover of size $\leq$ k.



SET COVER

$U = \{1, 2, 3, 4, 5, 6, 7\}$
$k = 2$

$S_a = \{3, 7\}$	$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$	$S_d = \{5\}$
$S_e = \{1\}$	$S_f = \{1, 2, 6, 7\}$

Summary

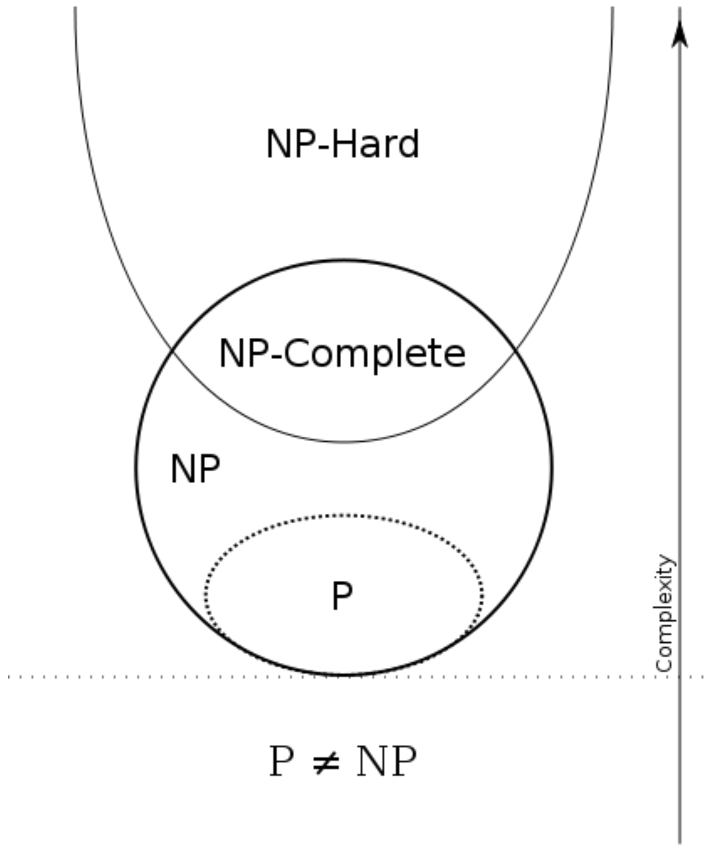
- Problems
  - Decision problems (yes/no)
  - Optimization problems (solution with best score)
- P
  - Decision problems (decision problems) that can be solved in polynomial time
  - Can be solved “efficiently”
- NP
  - Decision problems whose “YES” answer can be verified in polynomial time, if we already have the proof (or witness)
- Polynomial reductions
  - $A \leq_p B$: exhibit function f that transforms input of A **to** input of B in poly time, and prove that A(i) = YES $\Leftrightarrow$ B(f(i)) = YES

NP-Completeness (formally)

- A problem Y is **NP-hard** if $X \leq_p Y$ for all $X \in$ **NP**
  - A problem is NP-hard iff a polynomial-time algorithm for it implies a polynomial-time algorithm for every problem in NP
  - NP-hard problems are at least as hard as any NP problem
- A problem Y is **NP-complete** if:

  (1) $Y \in$ **NP**

  (2) Y is **NP-hard**



Establishing NP-Completeness

- Remark. Once we establish first "natural" NP-complete problem, others fall like dominoes.

- Recipe to establish NP-completeness of problem Y.
  - Step 1. Show that Y is in NP.
  - Step 2. Choose an NP-complete problem X.
  - Step 3. Prove that $X \le_P Y$ (poly-time reduction).

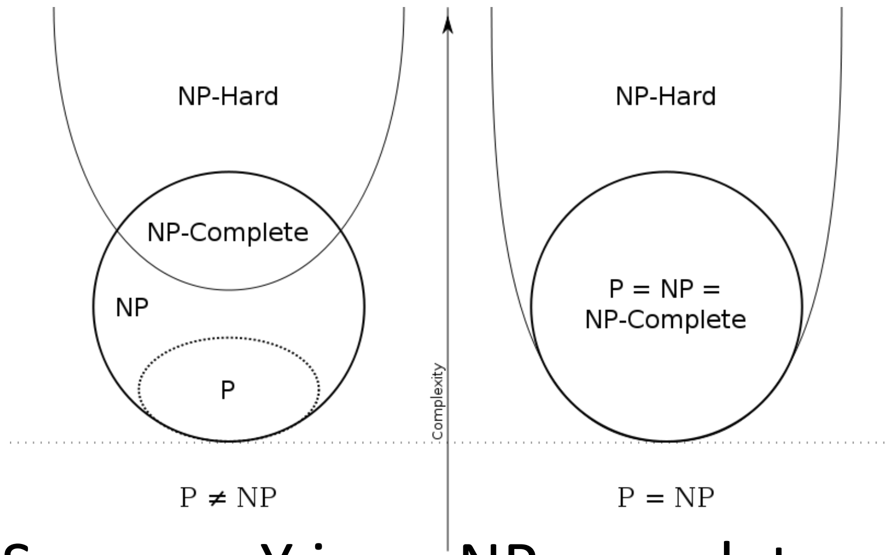
- Justification for Recipe: If X is an NP-complete problem, and Y is a problem in NP with the property that $X \le_P Y$ then Y is NP-complete.

- Pf. Let W be any problem in NP. Then $W \underbrace{\le_P}_{\text{by definition of NP-complete}} X \underbrace{\le_P}_{\text{by assumption}} Y$.
  - By transitivity, $W \le_P Y$.
  - Hence Y is NP-complete.

Establishing NP-Completeness

- Recipe to establish NP-completeness of problem Y.
  - Step 1. Show that Y is in NP.
    - Describe how a potential **solution**/witness will be represented
    - Describe a **procedure to check** whether the potential witness is a correct solution to the problem instance, and argue that this procedure takes **polynomial time**
  - Step 2. Choose an NP-complete problem X.
  - Step 3. Prove that $X \leq_P Y$ (X is **poly-time reducible** to Y).
    - Describe a **procedure f that converts** the inputs i of X to inputs of Y in **polynomial time**
    - Show that the reduction is correct by showing that $X(i) = \text{YES} \Leftrightarrow Y(f(i)) = \text{YES}$ (**if and only if**, proof in both directions)

Revisit "Is P = NP?"



- Theorem. Suppose Y is an NP-complete problem. Y is solvable in poly-time if and only if P = NP.
- Pf. $\Leftarrow$ If P = NP then Y is in P. Hence Y can be solved in poly-time.
- Pf. $\Rightarrow$ Suppose Y can be solved in poly-time.
  - Let X be any problem in NP. Then, we know that X ≤ p Y by definition of NP-complete and Y being NP-complete problem. Then we can solve X in poly-time by solving Y in poly-time. This implies any problem X in NP is also in P, i.e. NP $\subseteq$ P.
  - We already know P $\subseteq$ NP. Thus P = NP.

P & NP-Complete Problems

- **Shortest simple path**
  - Given a graph G = (V, E) find a **shortest** path from a source to all other vertices
  - <u>Polynomial solution:</u> O(VE)
- **Longest simple path**
  - Given a graph G = (V, E) find a **longest** path from a source to all other vertices
  - <u>NP-complete</u>

P & NP-Complete Problems

- **Euler tour**
  - G = (V, E) a connected, directed graph find a cycle that traverses **each edge** of G exactly once (may visit a vertex multiple times)
  - <u>Polynomial solution O(E)</u>
- **Hamiltonian cycle**
  - G = (V, E) a connected, directed graph find a cycle that visits **each vertex** of G exactly once
  - <u>NP-complete</u>

Extent and Impact of NP-Completeness

- Extent of NP-completeness. [Papadimitriou 1995]
  - Prime intellectual export of CS to other disciplines.
  - 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."

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.

Satisfiability Problem (SAT)

- **Satisfiability problem:** given a logical expression Φ, find an assignment of values (F, T) to variables x_i that causes Φ to evaluate to T:

$$\Phi = x_1 \vee \neg x_2 \wedge x_3 \vee \neg x_4$$

- Boolean variables: take on values T or F
  - Ex: x, y
- Literal: variable or negation of a variable
  - Ex: x, $\neg$ x (also denoted by $\bar{x}$)

- SAT is in NP: given a value assignment, check the Boolean logic of Φ evaluates to True (linear time)
- SAT was the first problem shown to be NP-complete! **(Cook–Levin theorem)**

CNF Satisfiability

- CNF is a special case of SAT
- Φ is in "Conjunctive Normal Form" (CNF)
  - "AND" of expressions (i.e., clauses)
  - Each clause contains only "OR"s of the variables and their negations

E.g.: $\Phi = (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2) \wedge (\neg x_1 \vee \neg x_2)$

clauses

- **Satisfiability (SAT) problem:** given a CNF formula Φ, find an assignment of values (False/0, True/1) to variables x_i that causes Φ to evaluate to True (**NP-complete**)

Definition of 3SAT / 3CNF

- A subcase of CNF problem:
  - Contains three literals per clause
- E.g.:
  - $\Phi = (x1 \lor \neg x1 \lor \neg x2) \land (x3 \lor x2 \lor x4) \land (\neg x1 \lor \neg x3 \lor \neg x4)$
- Is 3SAT in NP?
  - Yes, because SAT is in NP. Also easy to prove it directly.
- Is 3SAT NP-complete?
  - Not obvious. It has a more regular structure, which can perhaps be exploited to get an efficient algorithm
  - In fact, 2SAT does have a polynomial time algorithm

Showing 3SAT is NP-Complete by Reduction

- (1) To show 3SAT is in NP:
  - A certificate is a truth (0/1) assignment to the variables
  - Certifier: check that each clause has at least one literal set to true according to the certificate
- (2) Choose SAT as known NP-complete problem
- (3) Describe a reduction from SAT inputs to 3SAT inputs
  - Computable in poly time
  - SAT input is satisfiable iff constructed 3SAT input is satisfiable

  - (3a) Transform I_1 (instance of SAT) into I_2 (instance of 3SAT) in polynomial time
  - (3b,3c) Prove that I_1 has a solution $\Leftrightarrow$ I_2 has a solution

(3a) Reduction from SAT to 3SAT: I_1 -> I_2

- We are given an arbitrary CNF formula C = c1∧ c2 ∧ … ∧ cm over set of variables U, this is instance I_1
  - each ci is a clause (disjunction of literals)

- We will replace each clause ci with a set of clauses Ci', and may use some extra variables Ui' just for this clause
- Each clause in Ci' will have exactly 3 literals
- Transformed input will be conjunction of all the clauses in all the Ci’, this is an instance I_2 of 3SAT
- New clauses are carefully chosen…

(3a) Reduction from SAT to 3SAT: I_1 -> I_2

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$ (the z's are literals)

- Case 1: $k = 1$.
  - E.g. $c_i = z_1$
  - Use extra variables y_i^1 and y_i^2.
  - Replace clause c_i with 4 clauses:

    $(z_1 \vee \overline{y_i^1} \vee y_i^2)$

    $(z_1 \vee y_i^1 \vee \overline{y_i^2})$

    $(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

    $(z_1 \vee y_i^1 \vee y_i^2)$
  - Note that no matter what values we give the y variables, in one of the 4 clauses we will be forced to use z_1 to satisfy it

(3a) Reduction from SAT to 3SAT: $I_1 \to I_2$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 2: $k = 2$.
  - E.g. $c_i = z_1 \vee z_2$
  - Use extra variable y_i^1.
  - Replace c_i with 2 clauses:

  $(z_1 \vee z_2 \vee \overline{y_i^1})$

  $(z_1 \vee z_2 \vee y_i^1)$

(3a) Reduction from SAT to 3SAT: I_1 -> I_2

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 3: $k = 3$.
  - No extra variables are needed.
  - Keep c_i:

    $(z_1 \vee z_2 \vee z_3)$

(3a) Reduction from SAT to 3SAT: $I_1 \to I_2$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

(3a) Why is Reduction Poly Time?

- The running time of the reduction (the algorithm to compute the 3SAT formula C', given the SAT formula C) is proportional to the size of C'
- Rules for constructing C' are simple to calculate

(3a) Size of New Formula

- **Original clause with 1 literal** becomes 4 clauses with 3 literals each (1 to 12 literals conversion)
- **Original clause with 2 literals** becomes 2 clauses with 3 literals each (2 to 6 literals conversion)
- **Original clause with 3 literals** becomes 1 clause with 3 literals
- **Original clause with k > 3 literals** becomes k-2 clauses with 3 literals each (k to 3(k-2) literals conversion)

- So new formula C' is only a constant factor larger than the original formula
  - total *L* literals in formula C to *cL* literals in C', where c is a constant

(3bc) Correctness of Reduction

- Show that CNF formula C is satisfiable iff the 3-CNF formula C' constructed is satisfiable, i.e., sol(I_1) ⇔ sol(I_2)

- Step 3b (=>) Suppose original CNF formula C is satisfiable, i.e., I_1 has a solution. That means it has a truth assignment A to the variables that make the formula C evaluate to true.
- Come up with a satisfying truth assignment for the reduced 3SAT formula C', i.e., a solution to instance I_2.
  - For variables in U, use same truth assignments as for C.
  - How to assign T/F to the new variables in C'?

(3b) Truth Assignment for New Var.: sol(I_1) => sol(I_2)

Let $c_i = z_1$

- Case 1: k = 1.
- Use extra variables y_i^1 and y_i^2.
  - Replace c_i with 4 clauses:

$(z_1 \vee \overline{y_i^1} \vee y_i^2)$

$(z_1 \vee y_i^1 \vee \overline{y_i^2})$

$(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

$(z_1 \vee y_i^1 \vee y_i^2)$

Since z_1 is true, it does not matter how we assign y_i^1 and y_i^2

(3b) Truth Assignment for New Var.: sol(I_1) => sol(I_2)

Let $c_i = (z_1 \vee z_2)$

- Case 2: k = 2.
  - Use extra variable y_i^1.
  - Replace c_i with 2 clauses:

    $(z_1 \vee z_2 \vee y_i^1)$

    $(z_1 \vee z_2 \vee \overline{y_i^1})$

Since either z_1 or z_2 is true, it does not matter how we assign y_i^1

(3b) Truth Assignment for New Var.: sol(I_1) => sol(I_2)

Let $c_i = (z_1 \lor z_2 \lor z_3)$

- Case 3: k = 3.
  - No extra variables are needed.
  - Keep c_i:

    $(z_1 \lor z_2 \lor z_3)$

No new variables.

(3b) Truth Assignment for New Var.: sol(I_1) => sol(I_2)

Let $c_i = z_1 \lor z_2 \lor \ldots \lor z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with $k-2$ clauses:

$(z_1 \lor z_2 \lor y_i^1)$

$(\overline{y_i^1} \lor z_3 \lor y_i^2)$

$(\overline{y_i^2} \lor z_4 \lor y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \lor z_{k-3} \lor y_i^{k-4})$

$(\overline{y_i^{k-4}} \lor z_{k-2} \lor y_i^{k-3})$

$(\overline{y_i^{k-3}} \lor z_{k-1} \lor z_k)$

If first true literal is z_1 or z_2, set all y_i's to false: then all later clauses have a true literal

(3b) Truth Assignment for New Var.: sol(I_1) => sol(I_2)

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with $k-2$ clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

If first true literal is z_{k-1} or z_k, set all y_i's to true: then all earlier clauses have a true literal

(3b) Truth Assignment for New Var.: sol(I_1) => sol(I_2)

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with $k-2$ clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

If first true literal is in between, set all earlier y_i's to true and all later y_i's to false

(3c) Correctness of Reduction: sol(I_2) => sol(I_1)

- (<=) Suppose the newly constructed 3SAT formula C' is satisfiable, i.e., I_2 has a solution. We must show that the original SAT formula C is also satisfiable, i.e., I_1 has a solution.
- Use the same satisfying truth assignment for C as for C' (ignoring new variables).
- Show each original clause has at least one true literal in it.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \dots \vee z_k$

- Case 1: $k = 1$.
- Use extra variables y_i^1 and y_i^2.
  - Replace c_i with 4 clauses:

$(z_1 \vee \overline{y_i^1} \vee y_i^2)$

$(z_1 \vee y_i^1 \vee \overline{y_i^2})$

$(z_1 \vee \overline{y_i^1} \vee \overline{y_i^2})$

$(z_1 \vee y_i^1 \vee y_i^2)$

For every assignment of y_i^1 and y_i^2, in order for all 4 clauses to have a true literal, z_1 must be true.

(3c) $sol(I_2)$ => $sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 2: k = 2.
  - Use extra variable y_i^1.
  - Replace c_i with 2 clauses:

    $(z_1 \vee z_2 \vee y_i^1)$

    $(z_1 \vee z_2 \vee \overline{y_i^1})$

For either assignment of y_i^1, in order for both clauses to have a true literal, z_1 or z_2 must be true.

(3c) sol(I_2) => sol(I_1)

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 3: $k = 3$.
  - No extra variables are needed.
  - Keep c_i:

    $(z_1 \vee z_2 \vee z_3)$

No new variables.

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \dots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \dots, y_i^{k-3}$.
  - Replace c_i with $k-2$ clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

$\dots$

$\dots$

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false.

(3c) sol(I_2) => sol(I_1)

Let $c_i = z_1 \vee z_2 \vee \dots \vee z_k$

- Case 4: k > 3.
  - Use extra variables $y_i^1, \dots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(\underline{z_1} \vee z_2 \vee y_i^1)$

$(y_i^1 \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true,

(3c) sol(I_2) => sol(I_1)

Let $c_i = z_1 \lor z_2 \lor \dots \lor z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \dots, y_i^{k-3}$.
  - Replace c_i with $k-2$ clauses:

$(z_1 \lor z_2 \lor y_i^1)$

$(\overline{y_i^1} \lor z_3 \lor y_i^2)$

$(\overline{y_i^2} \lor z_4 \lor y_i^3)$

$\dots$

$\dots$

$(\overline{y_i^{k-5}} \lor z_{k-3} \lor y_i^{k-4})$

$(\overline{y_i^{k-4}} \lor z_{k-2} \lor y_i^{k-3})$

$(\overline{y_i^{k-3}} \lor z_{k-1} \lor z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, so y_i^2 must be true...

(3c) sol(I_2) => sol(I_1)

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with $k-2$ clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, so y_i^2 must be true…

(3c) $sol(I_2) \Rightarrow sol(I_1)$

Let $c_i = z_1 \vee z_2 \vee \dots \vee z_k$

- Case 4: $k > 3$.
  - Use extra variables $y_i^1, \dots, y_i^{k-3}$.
  - Replace c_i with $k-2$ clauses:

$(z_1 \vee z_2 \vee y_i^1)$

$(\overline{y_i^1} \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, so y_i^2 must be true…

(3c) sol(I_2) => sol(I_1)

Let $c_i = z_1 \vee z_2 \vee \ldots \vee z_k$

- Case 4: k > 3.
  - Use extra variables $y_i^1, \ldots, y_i^{k-3}$.
  - Replace c_i with k-2 clauses:

$(\underline{z_1} \vee z_2 \vee y_i^1)$

$(y_i^1 \vee z_3 \vee y_i^2)$

$(\overline{y_i^2} \vee z_4 \vee y_i^3)$

. . .

. . .

$(\overline{y_i^{k-5}} \vee z_{k-3} \vee y_i^{k-4})$

$(\overline{y_i^{k-4}} \vee z_{k-2} \vee y_i^{k-3})$

$(\overline{y_i^{k-3}} \vee z_{k-1} \vee z_k)$

Suppose in contradiction all z_i's are false. Then y_i^1 must be true, ..., y_i^{k-3} must be true, so last clause is false.

Conclusion

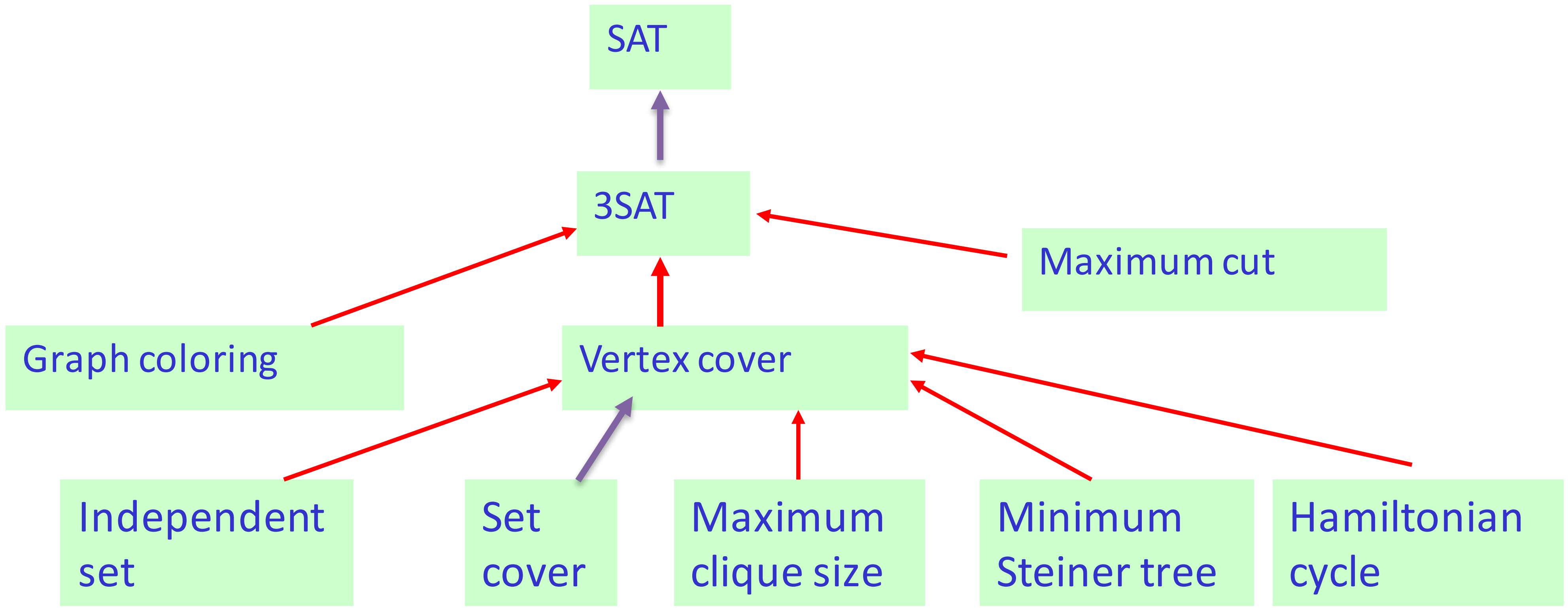
- (1) 3SAT is in NP
- (2) We know that SAT is NPC, we want to prove that 3SAT is more difficult than SAT, hence SAT $\leq_P$ 3SAT
- (3a) Take an instance I_1 of SAT, transform it in polynomial time into an instance I_2 of 3SAT
- (3b) Show that if I_1 has a solution, then I_2 has a solution
- (3c) Show that if I_2 has a solution, then I_1 has a solution

- 3SAT is NP-complete! This is your very first NP-completeness proof. Now you can do reductions from 3SAT.

- (All pbs in NP) $\leq_P$ SAT $\leq_P$ 3SAT

Examples of NP-complete problems

Summary of some NPC problems



find more NP-complete problems in

- http://en.wikipedia.org/wiki/List_of_NP-complete_problems
- **Garey-Johnson: Computers and Intractability: A Guide to the Theory of NP-Completeness**