CS 341: ALGORITHMS

Lecture 22: intractability IV – poly transformations, NP completeness

Readings: see website

Trevor Brown

https://student.cs.uwaterloo.ca/~cs341

trevor.brown@uwaterloo.ca

POLYNOMIAL **TRANSFORMATIONS**

commonly used for **NP-completeness** and **impossibility** results

POLYNOMIAL TRANSFORMATIONS

For a decision problem Π, let $\mathcal{I}(\Pi)$ denote the set of all instances of Π. Let $\mathcal{I}_{\mathbf{yes}}(\Pi)$ and $\mathcal{I}_{\mathbf{no}}(\Pi)$ denote the set of all yes-instances and no-instances (respectively) of Π.

Suppose that Π_1 and Π_2 are decision problems. We say that there is a **polynomial transformation** from Π_1 to Π_2 (denoted $\Pi_1 \leq_P \Pi_2$) if there exists a function $f : \mathcal{I}(\Pi_1) \rightarrow \mathcal{I}(\Pi_2)$ such that the following properties are satisfied:

$f(I)$ is computable in polynomial time (as a function of *size*(I), where $I \in \mathcal{I}(\Pi_1)$)

if $I \in \mathcal{I}_{\mathbf{yes}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\mathbf{yes}}(\Pi_2)$

if $I \in \mathcal{I}_{\mathbf{no}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\mathbf{no}}(\Pi_2)$

[Mechanics] to give a polynomial transformation, you must:
1. **specify** $f(I)$,
2. **show** it runs in poly-time, and
3. **show** I is a yes-instance of Π_1 **IFF** $f(I)$ is a yes-instance of Π_2.

POLYNOMIAL TRANSFORMATIONS (CONT.)

Also known as **Karp reductions** and **many-one reductions**

A polynomial transformation can be thought of as a (simple) special case of a polynomial-time Turing reduction, i.e., if $\Pi_1 \leq_P \Pi_2$, then $\Pi_1 \leq_P^T \Pi_2$.

Given a polynomial transformation f from Π_1 to Π_2, the corresponding Turing reduction is as follows:

Given $I \in \mathcal{I}(\Pi_1)$, construct $f(I) \in \mathcal{I}(\Pi_2)$.
Given an oracle for Π_2, say A, run $A(f(I))$.

We transform the instance, and then make a single call to the oracle.

Very important point: We do not know whether I is a yes-instance or a no-instance of Π_1 when we transform it to an instance $f(I)$ of Π_2.

To prove the implication "if $I \in \mathcal{I}_{\mathbf{no}}(\Pi_1)$, then $f(I) \in \mathcal{I}_{\mathbf{no}}(\Pi_2)$", we usually prove the contrapositive statement "if $f(I) \in \mathcal{I}_{\mathbf{yes}}(\Pi_2)$, then $I \in \mathcal{I}_{\mathbf{yes}}(\Pi_1)$.

We saw one instance where a contrapositive was easier to prove when we discussed Hamiltonian cycles

The contrapositive can help when it is hard to precisely characterize certificates for no-instances (or when such certificates don't prove much)

SUMMARIZING

THE MORE CONVENIENT DEFINITION

- Let Π_1 and Π_2 be decision problems
- **$\Pi_1 \leq_P \Pi_2$ iff** there exists $f : \mathcal{I}(\Pi_1) \to \mathcal{I}(\Pi_2)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(\Pi_1)$
  - If $I \in \mathcal{I}_{yes}(\Pi_1)$ then $f(I) \in \mathcal{I}_{yes}(\Pi_2)$
  - **If $\boldsymbol{f(I) \in \mathcal{I}_{yes}(\Pi_2)}$ then $\boldsymbol{I \in \mathcal{I}_{yes}(\Pi_1)}$**

Note: this is the same as saying
$\boldsymbol{(I \in \mathcal{I}_{yes}(\Pi_1)) \Leftrightarrow (f(I) \in \mathcal{I}_{yes}(\Pi_2))}$

This is the contrapositive. Was previously (2 slides ago):
If $I \in \mathcal{I}_{no}(\Pi_1)$ then $f(I) \in \mathcal{I}_{no}(\Pi_2)$

This property justifies correctness for the following generic **poly-time Karp reduction:**

```
P1toP2KarpReduction(I)
  fI = f(I)
  return OracleForP2(fI)
```

EXAMPLE POLYNOMIAL TRANSFORMATION

Problem 7.8

Clique

Instance: *An undirected graph* $G = (V, E)$ *and an integer* k, *where* $1 \leq k \leq |V|$.

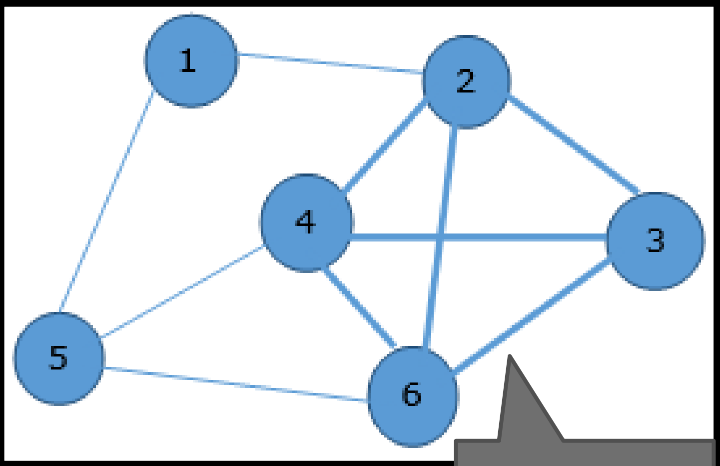
Question: *Does* G *contain a clique of size* $\geq k$? *(A* **clique** *is a subset of vertices* $W \subseteq V$ *such that* $uv \in E$ *for all* $u, v \in W$, $u \neq v$.*)*

Problem 7.9

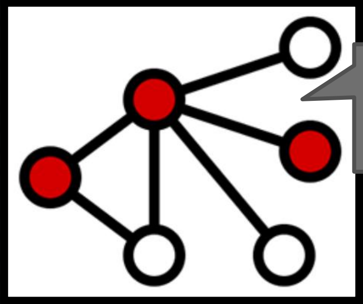
Vertex Cover

Instance: *An undirected graph* $G = (V, E)$ *and an integer* k, *where* $1 \leq k \leq |V|$.

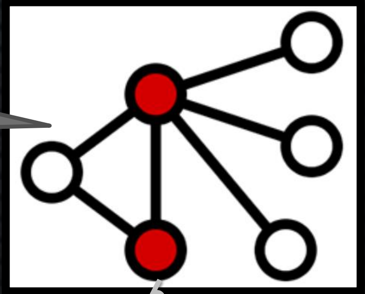
Question: *Does* G *contain a vertex cover of size* $\leq k$? *(A* **vertex cover** *is a subset of vertices* $W \subseteq V$ *such that* $\{u, v\} \cap W \neq \emptyset$ *for all edges* $uv \in E$.*)*



4-clique



3-vertex cover

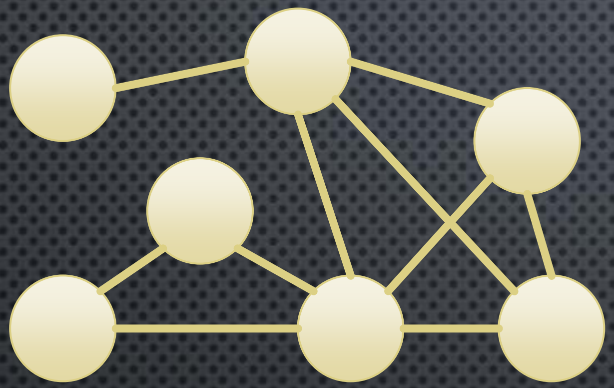


2-vertex cover

CLIQUE $\leq_P$ VERTEX-COVER

- Suppose $I = (G, k)$ is an instance of Clique where $G = (V, E), V = \{v_1, \dots, v_n\}$ and $1 \leq k \leq n$

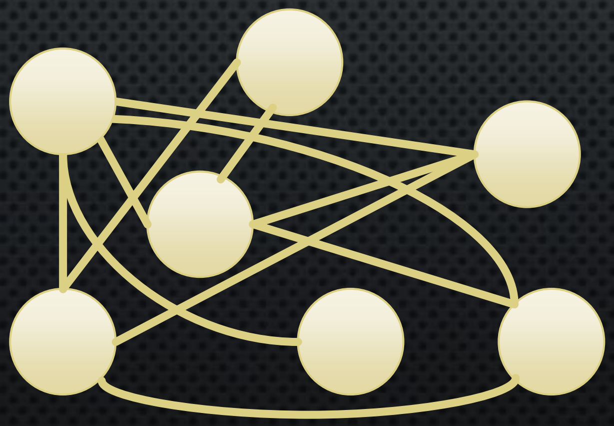
Want to solve $Clique(G, k)$



Claim: there is a k-clique in G ***iff*** there is an $(n - k)$ Vertex-Cover in $\overline{\boldsymbol{G}}$

- **Construct** instance $f(I) = (\overline{\boldsymbol{G}}, n - k)$ of Vertex-Cover, where $H = (V, \bar{E})$ and $v_i v_j \in \bar{E} \Leftrightarrow v_i v_j \notin E$

Idea: reduce to $VertexCover(\overline{\boldsymbol{G}}, n - k)$



Consider the **complement graph** $\overline{\boldsymbol{G}}$ of G

Every edge of G is a non-edge of $\overline{\boldsymbol{G}}$. Every non-edge of G is an edge of $\overline{\boldsymbol{G}}$.

Given an adjacency matrix for G, get $\overline{\boldsymbol{G}}$ by **flipping 0's and 1's.**

PROVING THIS IS A **POLYNOMIAL TRANSFORMATION**

- We denote Clique by CL and Vertex-Cover by VC
- $CL \leq_P VC$ **iff** there exists $f : \mathcal{I}(CL) \rightarrow \mathcal{I}(VC)$ such that:
  - **$f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(CL)$**
  - If $I \in \mathcal{I}_{yes}(CL)$ then $f(I) \in \mathcal{I}_{yes}(VC)$
  - If $f(I) \in \mathcal{I}_{yes}(VC)$ then $I \in \mathcal{I}_{yes}(CL)$

First let's show this

COMPLEXITY OF THE TRANSFORMATION

Assuming adjacency matrix, $Size(I) = \Theta(n^2 + \log_2 k)$

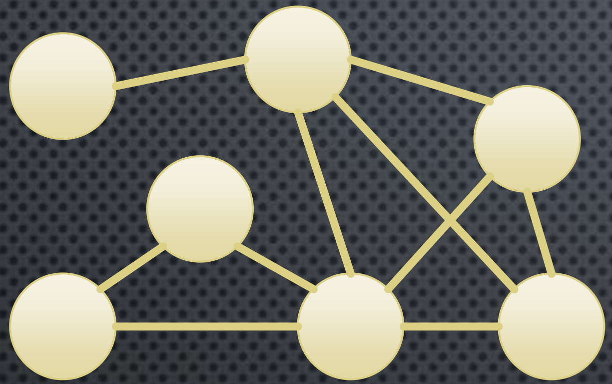
Time to compute $f(I)$?

Constructing $\bar{G}$ takes $O(n^2)$ time, and computing $n-k$ takes $O(\log n)$ time.

So computing $f(I)$ takes $O(n^2)$ time, which is polynomial in $Size(I)$.

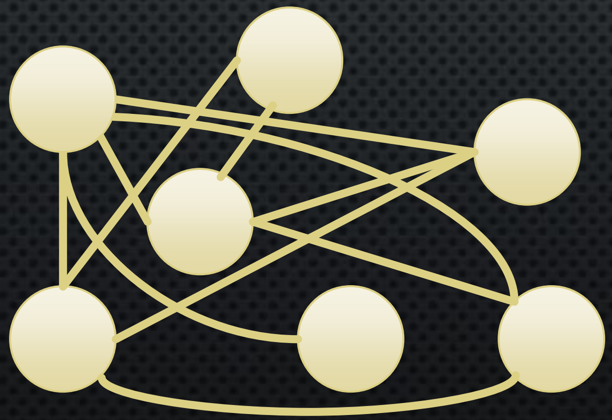
- Suppose $I = (G, k)$ is an instance of Clique where $G = (V, E), V = \{v_1, \dots, v_n\}$ and $1 \le k \le n$

Want to solve $Clique(G, k)$



- **Construct** instance $f(I) = (\bar{G}, n-k)$ of Vertex-Cover, where $\bar{G} = (V, \bar{E})$ and $v_i v_j \in \bar{E} \Leftrightarrow v_i v_j \notin E$

Idea: reduce to $VertexCover(\bar{G}, n-k)$



PROVING THIS IS A **POLYNOMIAL TRANSFORMATION**

- We denote Clique by CL and Vertex-Cover by VC
- $CL \leq_P VC$ **iff** there exists $f : \mathcal{I}(CL) \rightarrow \mathcal{I}(VC)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(CL)$
  - **If $\boldsymbol{I \in \mathcal{I}_{yes}(CL)}$ then $\boldsymbol{f(I) \in \mathcal{I}_{yes}(VC)}$**
  - If $f(I) \in \mathcal{I}_{yes}(VC)$ then $I \in \mathcal{I}_{yes}(CL)$

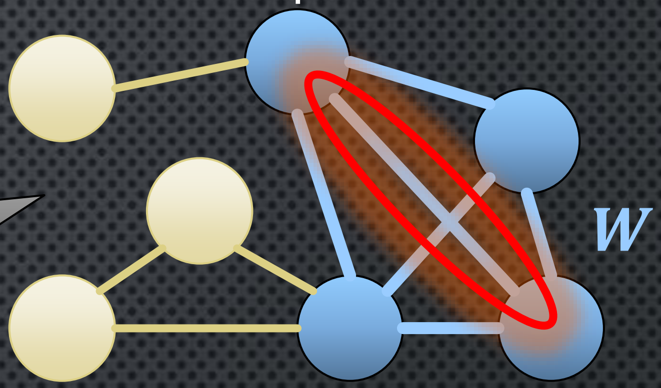
Now let's show this, i.e.,
if $\boldsymbol{G}$ contains a $\boldsymbol{k}$-clique then
$\boldsymbol{\overline{G}}$ contains an $\boldsymbol{(n-k)}$ vertex cover.

PROVING: $I \in \mathcal{I}_{yes}(CL) \Rightarrow f(I) \in \mathcal{I}_{yes}(VC)$

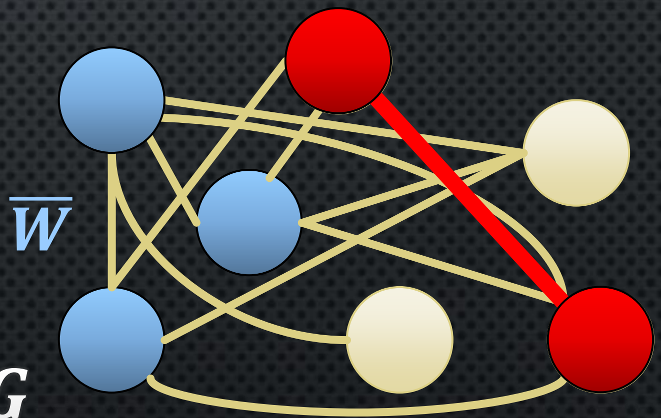
- Suppose $I = (G, k)$ is a **yes**-instance of Clique
- Then there is a set W of k vertices in a clique (with **all-to-all** edges)
- Define $\overline{\boldsymbol{W}} = V \setminus W$. Clearly $|\overline{\boldsymbol{W}}| = n - k$.
- We **claim** $\overline{\boldsymbol{W}}$ is a vertex cover of $\overline{\boldsymbol{G}}$
- Consider any edge $(u, v) \in \overline{\boldsymbol{G}}$
- If either u or v is in $\overline{\boldsymbol{W}}$, then we are done, so assume $u, v \notin \overline{\boldsymbol{W}}$ to obtain a contradiction
- Then $u, v \in W$, and W is a clique in G, so $(\boldsymbol{u}, \boldsymbol{v}) \in \boldsymbol{G}$
- But $(u, v) \in \overline{\boldsymbol{G}}$ implies $(u, v) \notin G$. Contradiction!

Example:
$Clique(G, 4)$

Graph $\boldsymbol{G}$



Graph $\overline{\boldsymbol{G}}$



PROVING THIS IS A **POLYNOMIAL TRANSFORMATION**

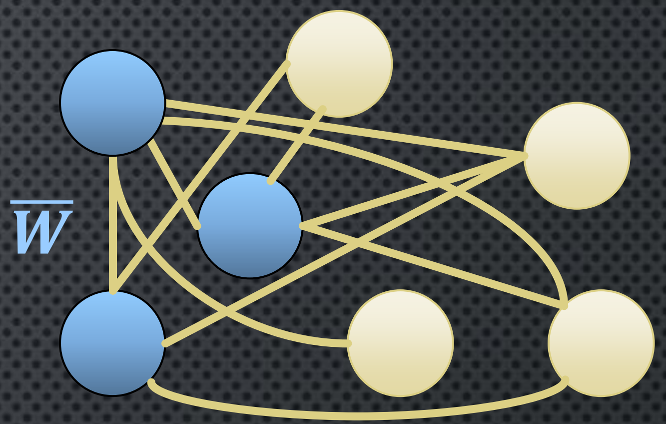
- We denote Clique by CL and Vertex-Cover by VC
- $CL \leq_P VC$ **iff** there exists $f : \mathcal{I}(CL) \rightarrow \mathcal{I}(VC)$ such that:
  - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(CL)$
  - If $I \in \mathcal{I}_{yes}(CL)$ then $f(I) \in \mathcal{I}_{yes}(VC)$
  - **If $\boldsymbol{f(I) \in \mathcal{I}_{yes}(VC)}$ then $\boldsymbol{I \in \mathcal{I}_{yes}(CL)}$**

Now let's show this, i.e.,
if $\overline{G}$ contains an $(n-k)$ vertex cover, then G contains a k-clique

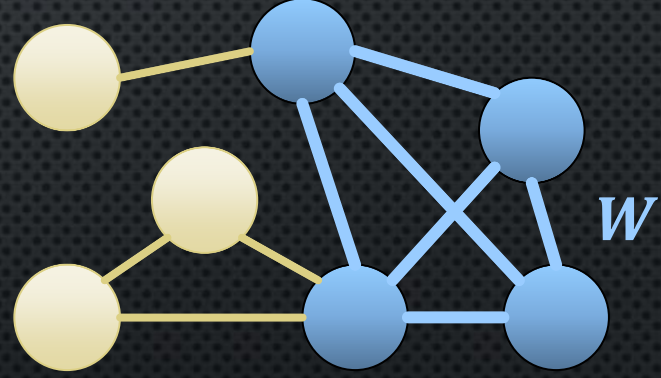
PROVING: $f(I) \in \mathcal{I}_{yes}(VC) \Rightarrow I \in \mathcal{I}_{yes}(CL)$

- Suppose $f(I) = (\overline{\boldsymbol{G}}, n-k)$ is a **yes**-instance of VC
- Then there is a set of $n-k$ vertices $\overline{\boldsymbol{W}}$ that is a vertex cover of $\overline{\boldsymbol{G}}$
- Define $W = V \setminus \overline{\boldsymbol{W}}$. Clearly $|W| = k$.
- We **claim** W is a clique in G
- Since $\overline{\boldsymbol{W}}$ is a vertex cover of $\overline{\boldsymbol{G}}$, **every edge** in $\overline{\boldsymbol{G}}$ has at least one endpoint in $\overline{\boldsymbol{W}}$
- Therefore, **no edge** in $\overline{\boldsymbol{G}}$ has two endpoints in W
- So, in $\boldsymbol{G}$, there are edges between all pairs of nodes in W. So, W is a clique in G.

Graph $\overline{\boldsymbol{G}}$



Graph $\boldsymbol{G}$



So, we have demonstrated a polynomial transformation from CLIQUE to VERTEX-COVER

COMPLEXITY CLASS **NP-COMPLETE**

COMPLEXITY CLASS **NP-COMPLETE** (NPC)

The complexity class **NPC** denotes the set of all decision problems Π that satisfy the following two properties:

$\Pi \in$ **NP**

For all $\Pi' \in$ **NP**, $\Pi' \leq_P \Pi$.

NPC is an abbreviation for **NP-complete**.

Note that the definition does not imply that NP-complete problems exist!

Satisfiability and the Cook-Levin Theorem

We will just call it the **SAT** problem

Challenging and powerful result!
How to prove **any NP problem** anyone will **ever** come up with is **solved** by a reduction to SAT?

Problem 7.13

CNF-Satisfiability

Example: $(p \lor q) \land (\neg p \lor r) \land (\neg r \lor \neg p \lor s \lor \neg s)$

Instance: *A boolean formula F in n boolean variables $x_1, \ldots, x_n$, such that F is the* **conjunction** *(logical "and") of m* **clauses**, *where each clause is the* **disjunction** *(logical "or") of literals. (A* **literal** *is a boolean variable or its negation.)*

Question: *Is there a truth assignment such that F evaluates to* **true**?

Variable: p, q
Literal: $p, \neg q$
Clause: $(p \lor q)$

Theorem 7.14 (Cook-Levin Theorem)

SAT $\in$ NPC.

Real-world problem people care about!
For example, used **extensively** to argue correctness for new processor designs.

Proving Problems NP-complete

Now that we **have one** NP-complete problem…

given any NP-complete problem, say Π_1, other problems in **NP** can be proven to be NP-complete via polynomial transformations from Π_1, as stated in the following theorem:

Theorem 7.15

Suppose that the following conditions are satisfied:

$\Pi_1 \in \mathbf{NPC}$,

$\Pi_1 \leq_P \Pi_2$, *and*

$\Pi_2 \in \mathbf{NP}$.

Then $\Pi_2 \in \mathbf{NPC}$.

Every $\Pi \in NP$ can be polynomially transformed to Π_1

And Π_1 can be polynomially transformed to Π_2

So, every $\Pi \in NP$ can be polynomially transformed to $\boldsymbol{\Pi_2}$

(Everything in NP can be poly-transformed to Π_2) + (Π_2 in NP) = definition of NPC

More Satisfiability Problems

Problem 7.16

Example: $(p \vee q \vee r) \wedge (\neg p \vee r \vee q) \wedge (\neg r \vee \neg p \vee s)$

3-SAT

Instance: *A boolean formula F in n boolean variables, such that F is the conjunction of m clauses, where each clause is the disjunction of exactly* **three** *literals.*

Question: *Is there a truth assignment such that F evaluates to* **true***?*

Problem 7.17

Example: $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg p)$

Satisfiable: $p = 0, q = 1, r \in \{0,1\}$

2-SAT

Instance: *A boolean formula F in n boolean variables, such that F is the conjunction of m clauses, where each clause is the disjunction of exactly* **two** *literals.*

Question: *Is there a truth assignment such that F evaluates to* **true***?*

IS **SAT HARDER** THAN 3-SAT? ONLY POLYNOMIALLY…

Proving SAT $\leq_P$ 3SAT

Suppose that $(X, \mathcal{C})$ is an instance of **SAT**, where $X = \{x_1, \ldots, x_n\}$ and $\mathcal{C} = \{C_1, \ldots, C_m\}$. For each C_j, do the following:

case 1 If $|C_j| = 1$, say $C_j = \{z\}$, construct four clauses

$$\{z, a, b\}, \{z, a, \overline{b}\}, \{z, \overline{a}, b\}, \{z, \overline{a}, \overline{b}\}.$$

case 2 If $|C_j| = 2$, say $C_j = \{z_1, z_2\}$, construct two clauses

$$\{z_1, z_2, c\}, \{z_1, z_2, \overline{c}\}.$$

case 3 If $|C_j| = 3$, then leave C_j unchanged.

case 4 If $|C_j| \geq 4$, say $C_j = \{z_1, z_2, \ldots, z_k\}$, then construct $k-2$ new clauses

$$\{z_1, z_2, d_1\}, \{\overline{d_1}, z_3, d_2\}, \{\overline{d_2}, z_4, d_3\}, \ldots,$$
$$\{\overline{d_{k-4}}, z_{k-2}, d_{k-3}\}, \{\overline{d_{k-3}}, z_{k-1}, z_k\}.$$

Satisfying all 4 clauses is possible **IFF $z = true$**

Same trick, but padding the clause with **two** new variables

We add a **<u>new</u> variable c** as padding

To satisfy **both** of these clauses, one of z_1, z_2 must be true!

$(z_1 \vee z_2 \vee c) \wedge (z_1 \vee z_2 \vee \neg c)$ is satisfiable **<u>IFF</u>** $(z_1 \vee z_2)$ is satisfiable!

So, the new clauses, together, are "equivalent" to the old one

New variables $d_1 \ldots d_{k-3}$ **"link"** the clauses

Here it gets trickier, because we aren't just "padding," but **"linking" many clauses**

Key idea: satisfying <u>all</u> clauses is possible **IFF some $z_i = true$**

Let's **prove** this is a **correct polynomial transformation**

CORRECTNESS

Sketch: let $\boldsymbol{L}$ be the number of **literals** in input $\boldsymbol{I}$. In our transformed input, we construct at most $\mathbf{4}\boldsymbol{L}$ **clauses**. Clearly this can be done in time $poly(4L)$, which is in $poly(L)$, which is in $poly(Size(I))$.

- Want to prove: $\mathbf{SAT} \leq_{\boldsymbol{P}} \mathbf{3SAT}$
- I.e., our transformation function $\boldsymbol{f}$ satisfies:
  - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(\Pi_1)$
  - If $I \in \mathcal{I}_{yes}(\mathbf{SAT})$ then $f(I) \in \mathcal{I}_{yes}(\mathbf{3SAT})$
  - **If $\boldsymbol{f}(\boldsymbol{I}) \in \boldsymbol{\mathcal{I}_{yes}}(\mathbf{3SAT})$ then $\boldsymbol{I} \in \boldsymbol{\mathcal{I}_{yes}}(\mathbf{SAT})$**

Let's do this direction

Correctness of the Transformation

Suppose I is a yes-instance of **SAT**. We show that $f(I)$ is a yes-instance of **3-SAT**. Fix a truth assignment for X in which every clause contains a true literal. We consider each clause C_j of the instance I.

If $C_j = \{z\}$, then z must be true. The corresponding four clauses in $f(I)$ each contain z, so they are all satisfied.

$\{z, a, b\}, \{z, a, \overline{b}\}, \{z, \overline{a}, b\}, \{z, \overline{a}, \overline{b}\}$

If $C_j = \{z_1, z_2\}$, then at least one of the z_1 or z_2 is true. The corresponding two clauses in $f(I)$ each contain z_1, z_2, so they are both satisfied.

$\{z_1, z_2, c\}, \{z_1, z_2, \overline{c}\}$

If $C_j = \{z_1, z_2, z_3\}$, then C_j occurs unchanged in $f(I)$.

Suppose $C_j = \{z_1, z_2, z_3, \dots, z_k\}$ where $k > 3$ and suppose $z_t \in C_j$ is a true literal. Define $d_i =$ **true** for $1 \leq i \leq t-2$ and define $d_i =$ **false** for $t-1 \leq i \leq k$. It is straightforward to verify that the $k-2$ corresponding clauses in $f(I)$ each contain a true literal.

$\{z_1, z_2, d_1\}, \{\overline{d_1}, z_3, d_2\}, \{\overline{d_2}, z_4, d_3\}, \dots,$
$\{\overline{d_{k-4}}, z_{k-2}, d_{k-3}\}, \{\overline{d_{k-3}}, z_{k-1}, z_k\}.$

CORRECTNESS

- Want to prove: $\mathbf{SAT} \leq_{\boldsymbol{P}} \mathbf{3SAT}$
- I.e., our transformation function $\boldsymbol{f}$ satisfies:
  - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(\Pi_1)$
  - If $I \in \mathcal{I}_{yes}(\mathbf{SAT})$ then $f(I) \in \mathcal{I}_{yes}(\mathbf{3SAT})$ — We just showed this
  - **If $\boldsymbol{f}(\boldsymbol{I}) \in \boldsymbol{\mathcal{I}}_{\boldsymbol{yes}}(\mathbf{3SAT})$ then $\boldsymbol{I} \in \boldsymbol{\mathcal{I}}_{\boldsymbol{yes}}(\mathbf{SAT})$** — Now let's show this

Conversely, suppose $f(I)$ is a yes-instance of **3-SAT**. We show that I is a yes-instance of **SAT**.

Consider each clause C in the SAT input I. We identify a corresponding set S of clauses in $f(I)$, and **we show C must be satisfied** because of the clauses in S.

(1) Four clauses in $f(I)$ having the form $\{z, a, b\}$, $\{z, a, \overline{b}\}$, $\{z, \overline{a}, b\}$ $\{z, \overline{a}, \overline{b}\}$ are all satisfied if and only if $z =$ **true**. Then the corresponding clause $\{z\}$ in I is satisfied.

(2) Two clauses in $f(I)$ having the form $\{z_1, z_2, c\}$, $\{z_1, z_2, \overline{c}\}$ are both satisfied if and only if at least one of $z_1, z_2 =$ **true**. Then the corresponding clause $\{z_1, z_2\}$ in I is satisfied.

(3) If $C_j = \{z_1, z_2, z_3\}$ is a clause in $f(I)$, then C_j occurs unchanged in I.

Correctness of the Transformation

$$\{z_1, z_2, d_1\}, \{\overline{d_1}, z_3, d_2\}, \{\overline{d_2}, z_4, d_3\}, \ldots,$$
$$\{\overline{d_{k-4}}, z_{k-2}, d_{k-3}\}, \{\overline{d_{k-3}}, z_{k-1}, z_k\}.$$

(4) Finally, consider the $k-2$ clauses in $f(I)$ arising from a clause $C_j = \{z_1, z_2, z_3, \ldots, z_k\}$ in I, where $k > 3$. We show that at least one of $z_1, z_2, \ldots, z_k =$ **true** if all $k-2$ of these clauses contain a true literal.

Assume all of $z_1, z_2, \ldots, z_k =$ **false**. In order for the first clause to contain a true literal, $d_1 =$ **true**. Then, in order for the second clause to contain a true literal, $d_2 =$ **true**. This pattern continues, and in order for the second last clause to contain a true literal, $d_{k-3} =$ **true**.

But then the last clause contains no true literal, which is a contradiction. We have shown that at least one of $z_1, z_2, \ldots, z_k =$ **true**, which says that the clause $\{z_1, z_2, z_3, \ldots, z_k\}$ contains a true literal, as required.

So, we have given a correct polynomial transformation from SAT to 3SAT.

So, if a problem Π can be transformed into SAT in polytime, it can also be transformed into 3SAT in polytime.

But wait… SAT is NP-COMPLETE.

Have we shown 3SAT is NP-COMPLETE?

Still need to show 3SAT ∈ NP!

So every problem in NP can be transformed into 3SAT in polytime!

PROVING 3SAT IS **IN NP**

3SAT input $I = (Clauses[1..m], n)$:
a list of m **clauses**, and the number n of variables.
Each clause contains literals. Each literal is a pair (var, neg): a variable $\in \{1..n\}$ & a negation bit

1. Define desired YES-certificate
2. Design a poly-time $verify(I, C)$ algorithm
3. Correctness proof
   - **Case 1:** Let I be any yes-instance; Find C such that $verify(I, C) = true$
   - **Case 2:** Let I be any no-instance, and C be any certificate; Prove $verify(I, C) = false$
   - **<u>Contrapositive</u> of case 2:** Suppose $verify(I, C) = true$; Prove I is a yes-instance

YES-certificate C = array with one bit per variable in $\{1..n\}$ representing a **satisfying assignment**

```
verify3SAT(I=(Clauses[1..m], n), C)
    if C is not an array of n bits return false

    numSat = 0
    for each c in Clauses
        for each literal (var, neg) in c
            if (C[var] && !neg) or (!C[var] && neg)
                numSat++
                break

    return (numSat == m)
```

This takes $O(|\text{Clauses}|)$ time, which is polynomial in $\text{Size}(I)$

MECHANICS OF SHOWING A PROBLEM IS **IN NP**

1. Define desired YES-certificate
2. Design a poly-time $verify(I, C)$ algorithm
3. Correctness proof
    - **Case 1:** Let I be any yes-instance; Find C such that $verify(I, C) = true$
    - **Case 2:** Let I be any no-instance, and C be any certificate; Prove $verify(I, C) = false$
    - **<u>Contrapositive</u> of case 2:** Suppose $verify(I, C) = true$; Prove I is a yes-instance

Let I be a yes-instance of 3SAT. Then it has a satisfying assignment $\boldsymbol{A_s}$. And, $verify(I, A_s)$ will see that each clause contains a literal satisfied by this assignment, so $verify$ will see $numSat = |Clauses|$ and return true.

Suppose $verify(I, C)$ returns true. Then $numSat = |Clauses|$, so $numSat$ was incremented in each iteration of the loop over clauses, so each clause contains a satisfied literal, so the 3SAT formula in I is satisfied by C, so I is a yes-instance.

It follows that **3SAT is in NP.**
Since we have already shown SAT $\leq_P$ 3SAT, we now know that **3SAT is NP-COMPLETE**.

RECAP

- To prove a problem Π is NP-COMPLETE
  - Show Π is **in NP**, **<u>and</u>**
  - Give a polynomial transformation **from** some NP-COMPLETE problem **to Π**
    - This involves an IFF correctness argument, **and** a polytime complexity argument
- When showing a problem is in NP, **<u>or</u>** proving correctness for a polynomial transformation,
  - Instead of proving statements about **no-instances**, it is usually easier to prove the **contrapositive**

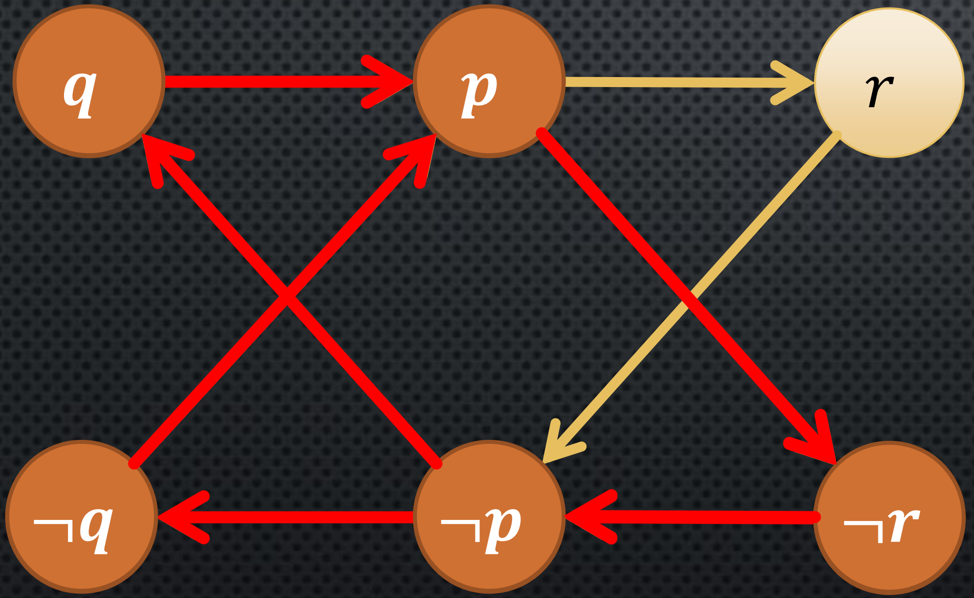
IS 2-SAT ALSO HARD?

2-SAT EXAMPLES

- $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg p)$
  - Satisfiable: $p = 0, q = 1, r \in \{0,1\}$
- $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg p) \wedge (p \vee \neg q)$

Logical refresher:
$p \Rightarrow q$ is **equivalent** to
$\neg p \vee q$.

Therefore, $p \vee q$ is **equivalent** to $\neg p \Rightarrow q$ **and equivalent** to $\neg q \Rightarrow p$



Edges (implications of clauses)…

$\neg p \Rightarrow q$	$p \Rightarrow r$	$r \Rightarrow \neg p$	$\neg p \Rightarrow \neg q$
$\neg q \Rightarrow p$	$\neg r \Rightarrow \neg p$	$p \Rightarrow \neg r$	$q \Rightarrow p$

$q \Rightarrow p \Rightarrow \neg r \Rightarrow \neg p \Rightarrow \neg q$ … so q cannot be $true$

$\neg q \Rightarrow p \Rightarrow \neg r \Rightarrow \neg p \Rightarrow q$ … so q cannot be $false$

Therefore the formula **cannot** be satisfied!

(variable names are integers in 1..|X|)

2-SAT can be solved in polynomial time. Suppose we are given an instance I of **2-SAT** on a set of boolean variables $\boldsymbol{X = \{1..|X|\}}$

(1) For every clause $x \vee y$ (where x and y are literals), construct two directed edges $\overline{x}y$ and $\overline{y}x$. We get a directed graph on vertex set $X \cup \overline{X}$.

(2) Determine the strongly connected components of this directed graph.

(3) I is a yes-instance if and only if there is no strongly connected component containing x and $\overline{x}$, for any $x \in X$.

Suppose no variable x is in the same SCC as $\bar{x}$, then to get a satisfying assignment do the following:
For each x, if $\exists$ path from x to $\bar{x}$, then set $x = false$ else set $x = true$.

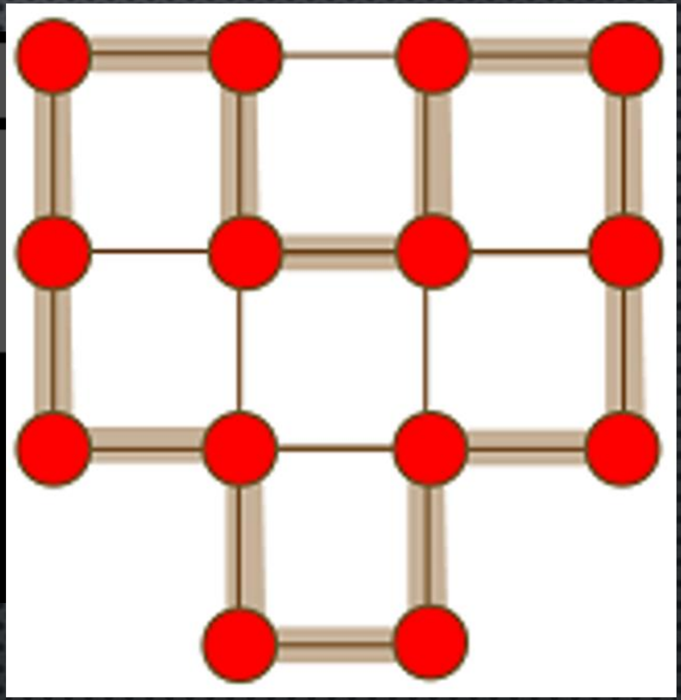
HOMEWORK SLIDES

RETURNING TO ANOTHER FAMILIAR PROBLEM

Problem 7.2

Hamiltonian Cycle
Instance: *An undirected graph $G = (V, E)$.*
Question: *Does G contain a hamiltonian cycle?*

A **hamiltonian cycle** is a cycle that passes through every vertex in V exactly once.



Turns out **Hamiltonian Cycle is NP complete** as well

Compare to **Euler tour/circuit**: a cycle that passes through each **edge** exactly once can be found in **polytime**!

THE P=NP QUESTION

Theorem 7.12

If $\mathbf{P} \cap \mathbf{NPC} \neq \emptyset$, *then* $\mathbf{P} = \mathbf{NP}$.

So, to win $1,000,000 just need to find one problem in NPC that can be reduced to a problem in P

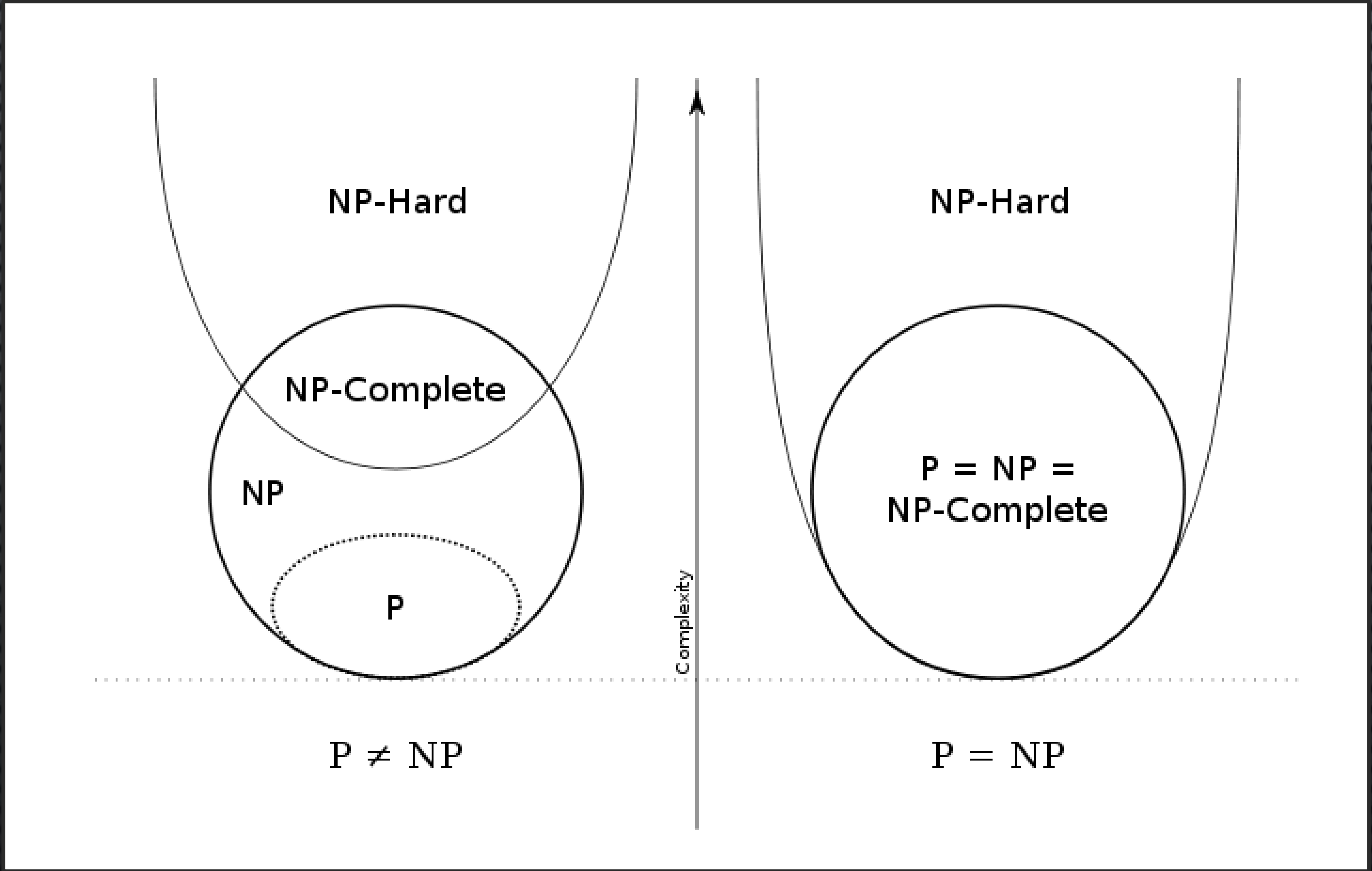
Proof.

We know that $\mathbf{P} \subseteq \mathbf{NP}$, so it suffices to show that $\mathbf{NP} \subseteq \mathbf{P}$. Suppose $\Pi \in \mathbf{P} \cap \mathbf{NPC}$ and let $\Pi' \in \mathbf{NP}$. We will show that $\Pi' \in \mathbf{P}$.

1. Since $\Pi' \in \mathbf{NP}$ and $\Pi \in \mathbf{NPC}$, it follows that $\Pi' \leq_P \Pi$ (definition of NP-completeness).
2. Since $\Pi' \leq_P \Pi$ and $\Pi \in \mathbf{P}$, it follows that $\Pi' \in \mathbf{P}$ (see last lecture)

$\square$

TWO POSSIBLE REALITIES…



PROPERTIES OF POLYNOMIAL TRANSFORMATIONS

Theorem 7.10

If Π_1 *and* Π_2 *are decision problems,* $\Pi_1 \leq_P \Pi_2$ *and* $\Pi_2 \in \mathbf{P}$*, then* $\Pi_1 \in \mathbf{P}$*.*

Proof.

Suppose A is a poly-time algorithm for Π_2, having complexity $O(m^\ell)$ on an instance of size m. Suppose f is a transformation from Π_1 to Π_2 having complexity $O(n^k)$ on an instance of size n. We solve Π_1 as follows:

1. Given $I \in \mathcal{I}(\Pi_1)$, construct $f(I) \in \mathcal{I}(\Pi_2)$.
2. Run $A(f(I))$.

It is clear that this yields the correct answer. We need to show that these two steps can be carried out in polynomial time as a function of $n = \mathit{Size}(I)$. Step (1) can be executed in time $O(n^k)$ and it yields an instance $f(I)$ having size $m \in O(n^k)$. Step (2) takes time $O(m^\ell)$. Since $m \in O(n^k)$, the time for step (2) is $O(n^{k\ell})$, as is the total time to execute both steps.

PROPERTIES OF POLYNOMIAL TRANSFORMATIONS

Theorem 7.11

Suppose that Π_1, Π_2 *and* Π_3 *are decision problems. If* $\Pi_1 \leq_P \Pi_2$ *and* $\Pi_2 \leq_P \Pi_3$, *then* $\Pi_1 \leq_P \Pi_3$.

Proof.

We have a polynomial transformation f from Π_1 to Π_2, and another polynomial transformation g from Π_2 to Π_3. We define $h = f \circ g$, i.e., $h(I) = g(f(I))$ for all instances I of Π_1. (Exercise: fill in the details.) □