

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

1

POLYNOMIAL TRANSFORMATIONS

commonly used for NP-completeness and impossibility results

2

POLYNOMIAL TRANSFORMATIONS

For a decision problem Π , let $\mathcal{I}(\Pi)$ denote the set of all instances of Π . Let $\mathcal{I}_{yes}(\Pi)$ and $\mathcal{I}_{no}(\Pi)$ 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 $I \in \mathcal{I}_{yes}(\Pi_1)$, then $f(I) \in \mathcal{I}_{yes}(\Pi_2)$
- if $I \in \mathcal{I}_{no}(\Pi_1)$, then $f(I) \in \mathcal{I}_{no}(\Pi_2)$

[Mechanics] to give a polynomial transformation, you must:

- specify $f(I)$.
- show it runs in poly-time, and
- show I is a yes-instance of Π_1 IFF $f(I)$ is a yes-instance of Π_2 .

3

POLYNOMIAL TRANSFORMATIONS (CONT.)

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

Also known as Karp reductions and many-one reductions

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)

4

SUMMARIZING

THE MORE CONVENIENT DEFINITION

- Let Π_1 and Π_2 be decision problems
- $\Pi_1 \leq_P \Pi_2$ iff there exists $f : \mathcal{I}(\Pi_1) \rightarrow \mathcal{I}(\Pi_2)$ 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 $f(I) \in \mathcal{I}_{yes}(\Pi_2)$ then $I \in \mathcal{I}_{yes}(\Pi_1)$

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

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

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

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

5

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 $u, v \in W$, $u \neq v$.)



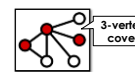
4-clique

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 $uv \in E$.)



3-vertex cover



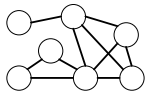
2-vertex cover

6

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 $1 \leq k \leq n$

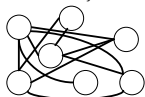
Want to solve $Clique(G, k)$



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

Construct instance $f(I) = (\bar{G}, n - k)$ 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 $VertexCover(\bar{G}, n - k)$



Consider the **complement graph** \bar{G} of G

Every edge of G is a non-edge of \bar{G} .
Every non-edge of G is an edge of \bar{G} .

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

7

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 $I \in \mathcal{J}_{yes}(CL)$ then $f(I) \in \mathcal{J}_{yes}(VC)$
 - If $f(I) \in \mathcal{J}_{yes}(VC)$ then $I \in \mathcal{J}_{yes}(CL)$

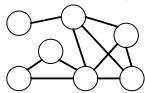
First let's show this

8

COMPLEXITY OF THE TRANSFORMATION

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 $Clique(G, k)$



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

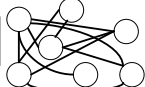
Time to compute $f(I)$?

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 $Size(I)$.

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 $VertexCover(\bar{G}, n - k)$



9

PROVING THIS IS A POLYNOMIAL TRANSFORMATION

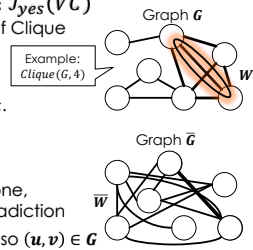
- We denote Clique by CL and Vertex-Cover by VC
- $CL \leq_p VC$ **iff** there exists $f : \mathcal{I}(CL) \rightarrow \mathcal{I}(VC)$ such that:
 - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(CL)$
 - **If $I \in \mathcal{J}_{yes}(CL)$ then $f(I) \in \mathcal{J}_{yes}(VC)$**
 - If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

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

10

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

- 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 $\bar{W} = V \setminus W$. Clearly $|\bar{W}| = n - k$.
- We **claim** \bar{W} is a vertex cover of \bar{G}
- Consider any edge $(u, v) \in \bar{G}$
- If either u or v is in \bar{W} , then we are done, so assume $u, v \notin \bar{W}$ to obtain a contradiction
- Then $u, v \in W$, and W is a clique in G , so $(u, v) \in G$
- But $(u, v) \in \bar{G}$ implies $(u, v) \notin G$. Contradiction!



11

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 $I \in \mathcal{J}_{yes}(CL)$ then $f(I) \in \mathcal{J}_{yes}(VC)$
 - **If $f(I) \in \mathcal{J}_{yes}(VC)$ then $I \in \mathcal{J}_{yes}(CL)$**

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

12

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

Suppose $f(I) = (\bar{G}, n - k)$ is a **yes**-instance of VC

Then there is a set of $n - k$ vertices \bar{W} that is a vertex cover of \bar{G}

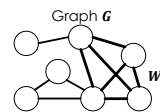
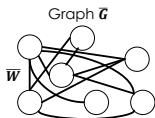
Define $W = V \setminus \bar{W}$. Clearly $|W| = k$.

We **claim** W is a clique in G

Since \bar{W} is a vertex cover of \bar{G} , **every edge** in \bar{G} has at least one endpoint in \bar{W}

Therefore, **no edge** in \bar{G} has two endpoints in W

So, in G , there are edges between all pairs of nodes in W . So, W is a clique in G .



So, we have demonstrated a polynomial transformation from CLIQUE to VERTEX-COVER 13

COMPLEXITY CLASS NP-COMPLETE

14

COMPLEXITY CLASS NP-COMPLETE (NPC)

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

$\Pi \in \text{NP}$

For all $\Pi' \in \text{NP}$, $\Pi' \leq_p \Pi$.

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

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

15

Satisfiability and the Cook-Levin Theorem

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

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

CNF-Satisfiability

Instance: A boolean formula F in n boolean variables x_1, \dots, 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.)

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

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

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

Theorem 7.14 (Cook-Levin Theorem)

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

SAT \in NPC.

16

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:

$\Pi_1 \in \text{NPC}$. Every $\Pi \in \text{NP}$ can be polynomially transformed to Π_1 .

$\Pi_1 \leq_p \Pi_2$, and And Π_1 can be polynomially transformed to Π_2 .

$\Pi_2 \in \text{NP}$. So, every $\Pi \in \text{NP}$ can be polynomially transformed to Π_2 .

Then $\Pi_2 \in \text{NPC}$. (Everything in NP can be poly-transformed to Π_2) + (Π_2 in NP) = definition of NPC

17

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?

18

IS SAT HARDER THAN 3-SAT? ONLY POLYNOMIALLY...

Proving SAT \leq_p 3SAT

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

- case 1** If $|C_j| = 1$, say $C_j = \{z\}$, construct four clauses $\{z, a, b\}, \{z, a, \bar{b}\}, \{z, \bar{a}, b\}, \{z, \bar{a}, \bar{b}\}$.
 Satisfying all 4 clauses is possible **IFF** $z = \text{true}$.
 Some trick, but padding the clause with **two** new variables.
- case 2** If $|C_j| = 2$, say $C_j = \{z_1, z_2\}$, construct two clauses $\{z_1, z_2, c\}, \{z_1, z_2, \bar{c}\}$.
 We add a **new variable c** as padding.
 To satisfy **both** of these clauses, one of z_1, z_2 must be true!
- case 3** If $|C_j| = 3$, then leave C_j unchanged.
 $(z_1 \vee z_2 \vee c) \wedge (z_1 \vee z_2 \vee \bar{c})$ is satisfiable **IFF** $(z_1 \vee z_2)$ is satisfiable!
- case 4** If $|C_j| \geq 4$, say $C_j = \{z_1, z_2, \dots, z_k\}$, then construct $k-2$ new clauses $\{z_1, z_2, d_1\}, \{d_1, z_3, d_2\}, \{d_2, z_4, d_3\}, \dots, \{d_{k-4}, z_{k-2}, d_{k-3}\}, \{d_{k-3}, z_{k-1}, z_k\}$.
 Here it gets trickier, because we aren't just "padding," but **"linking" many clauses**.
 So, the new clauses, together, are "equivalent" to the old one.

New variables $d_1 \dots d_{k-3}$ "link" the clauses.
 Key idea: satisfying all clauses is possible **IFF** some $z_i = \text{true}$.
 Let's prove this is a correct polynomial transformation.

CORRECTNESS

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

- Want to prove: **SAT \leq_p 3SAT**
- I.e., our transformation function f satisfies:
 - $f(I)$ is computable in poly-time, for all $I \in \mathcal{I}(\Pi_1)$
 - If $I \in \mathcal{J}_{\text{yes}}(\text{SAT})$ then $f(I) \in \mathcal{J}_{\text{yes}}(\text{3SAT})$
 Let's do this direction
 - If $f(I) \in \mathcal{J}_{\text{yes}}(\text{3SAT})$ then $I \in \mathcal{J}_{\text{yes}}(\text{SAT})$

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.
 $\{z, a, b\}, \{z, a, \bar{b}\}, \{z, \bar{a}, b\}, \{z, \bar{a}, \bar{b}\}$
- 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.
 $\{z_1, z_2, c\}, \{z_1, z_2, \bar{c}\}$
- 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 $d_i = \text{true}$ for $1 \leq i \leq t-2$ and define $d_i = \text{false}$ 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.
 $\{z_1, z_2, d_1\}, \{d_1, z_3, d_2\}, \{d_2, z_4, d_3\}, \dots, \{d_{k-4}, z_{k-2}, d_{k-3}\}, \{d_{k-3}, z_{k-1}, z_k\}$.

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 .

- Four clauses in $f(I)$ having the form $\{z, a, b\}, \{z, a, \bar{b}\}, \{z, \bar{a}, b\}, \{z, \bar{a}, \bar{b}\}$ are all satisfied if and only if $z = \text{true}$. Then the corresponding clause $\{z\}$ in I is satisfied.
- Two clauses in $f(I)$ having the form $\{z_1, z_2, c\}, \{z_1, z_2, \bar{c}\}$ are both satisfied if and only if at least one of $z_1, z_2 = \text{true}$. Then the corresponding clause $\{z_1, z_2\}$ in I is satisfied.
- If $C_j = \{z_1, z_2, z_3\}$ is a clause in $f(I)$, then C_j occurs unchanged in I .

CORRECTNESS

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

Correctness of the Transformation (

- Finally, consider the $k-2$ clauses in $f(I)$ arising from a clause $C_j = \{z_1, z_2, z_3, \dots, z_k\}$ in I , where $k > 3$. We show that at least one of $z_1, z_2, \dots, z_k = \text{true}$ if all $k-2$ of these clauses contain a true literal.
 Assume all of $z_1, z_2, \dots, z_k = \text{false}$. In order for the first clause to contain a true literal, $d_1 = \text{true}$. Then, in order for the second clause to contain a true literal, $d_2 = \text{true}$. This pattern continues, and in order for the second last clause to contain a true literal, $d_{k-3} = \text{true}$.
 But then the last clause contains no true literal, which is a contradiction.
 We have shown that at least one of $z_1, z_2, \dots, z_k = \text{true}$, which says that the clause $\{z_1, z_2, z_3, \dots, z_k\}$ 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-COMLETE.

Have we shown 3SAT is NP-COMLETE?
 Still need to show 3SAT \in NP!
 So every problem in NP can be transformed into 3SAT in polytime!
 24

PROVING 3SAT 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 $verify(I, C) = false$
 - **Contrapositive of case 2:** Suppose $verify(I, C) = true$; Prove I is a yes-instance

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

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

```

1 verify3SAT(I=(Clauses[1..m], n), C)
2   if C is not an array of n bits return false
3
4   numSat = 0
5   for each c in Clauses
6     for each (literal, (var, neg)) in c
7       if (C[var] && !neg) or !(C[var] && neg)
8         numSat++
9         break
10
11  return (numSat == m)
    
```

This takes $O((Clauses) \cdot n)$ time, which is polynomial in $Size(I)$

25

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 $verify(I, C) = false$
 - **Contrapositive 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 A . And, $verify(I, A)$ 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**.

26

RECAP

- To prove a problem Π is NP-COMPLETE
 - Show Π is **in NP, and**
 - 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, **or** proving correctness for a polynomial transformation,
 - Instead of proving statements about **no-instances**, it is usually easier to prove the **contrapositive**

27

IS 2-SAT ALSO HARD?

28

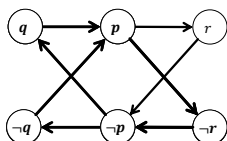
2-SAT EXAMPLES

- $(p \vee q) \wedge (\neg p \vee r) \wedge (\neg r \vee \neg p)$
 - 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 $\neg q \Rightarrow p$

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



Edges (implications of clauses)...

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

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

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

Therefore the formula **cannot** be satisfied!

29

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

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

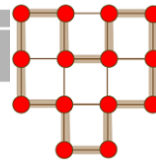
30

HOMWORK SLIDES

31

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!**

32

THE P=NP QUESTION

Theorem 7.12

If $P \cap NPC \neq \emptyset$, then $P = NP$.

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

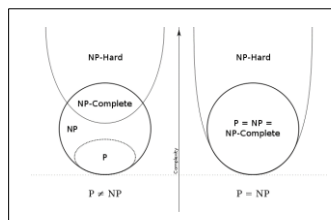
Proof.

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

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

33

TWO POSSIBLE REALITIES...



34

Theorem 7.10

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

Proof.

Suppose A is a poly-time algorithm for Π_2 , having complexity $O(m^k)$ 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:

- Given $I \in \mathcal{I}(\Pi_1)$, construct $f(I) \in \mathcal{I}(\Pi_2)$.
- 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 $n = \text{Size}(I)$. 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 $O(m^k)$. Since $m \in O(n^k)$, the time for step (2) is $O(n^{k^2})$, as is the total time to execute both steps.

PROPERTIES OF POLYNOMIAL TRANSFORMATIONS

35

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.)

PROPERTIES OF POLYNOMIAL TRANSFORMATIONS

36