

# Analysis of Algorithms, I

## CSOR W4231

Eleni Drinea  
*Computer Science Department*

Columbia University

Representative NP-complete problems: TSP, Set Cover

- 1 Review of last lecture
- 2 Representative  $\mathcal{NP}$ -complete problems
- 3 Integer Programming
- 4 Minimum-weight Set Cover
  - An integer programming formulation of Set Cover
  - The linear program relaxation
- 5 An approximation algorithm for Set Cover
  - Rounding the LP solution
  - An  $f$ -approximation algorithm for Set Cover

- 1 Review of last lecture
- 2 Representative  $\mathcal{NP}$ -complete problems
- 3 Integer Programming
- 4 Minimum-weight Set Cover
  - An integer programming formulation of Set Cover
  - The linear program relaxation
- 5 An approximation algorithm for Set Cover
  - Rounding the LP solution
  - An  $f$ -approximation algorithm for Set Cover

# Complexity classes $\mathcal{P}$ , $\mathcal{NP}$ and $\mathcal{NP}$ -complete

## Definition 1.

We define  $\mathcal{P}$  to be the set of problems that can be solved by polynomial-time algorithms.

## Definition 2.

We define  $\mathcal{NP}$  to be the set of decision problems that have an efficient certifier.

## Fact 3.

$$\mathcal{P} \subseteq \mathcal{NP}$$

## Definition 4.

A problem  $X(D)$  is  $\mathcal{NP}$ -complete if

1.  $X(D) \in \mathcal{NP}$  and
2. for all  $Y \in \mathcal{NP}$ ,  $Y \leq_P X$ .

# How do we show that a problem is $\mathcal{NP}$ -complete?

*Suppose we had an  $\mathcal{NP}$ -complete problem  $X$ .*

To show that another problem  $Y$  is  $\mathcal{NP}$ -complete, we use **transitivity of reductions**. So we “only” need show that

1.  $Y \in \mathcal{NP}$
2.  $X \leq_P Y$

*The first  $\mathcal{NP}$ -complete problem*

**Theorem 5 (Cook-Levin).**

*Circuit SAT is  $\mathcal{NP}$ -complete.*

# Satisfiability of boolean functions

**SAT:** Given a formula  $\phi$  in CNF with  $n$  variables and  $m$  clauses, is  $\phi$  satisfiable?

**3SAT:** Given a formula  $\phi$  in CNF with  $n$  variables and  $m$  clauses such that each clause has exactly 3 literals, is  $\phi$  satisfiable?

**Circuit-SAT:** Given a boolean combinatorial circuit  $C$ , is there an assignment of truth values to its inputs that causes the output to evaluate to 1?

## Lemma 6.

*Circuit-SAT*  $\leq_P$  *SAT*, *SAT*  $\leq_P$  *3SAT* and *3SAT*  $\leq_P$  *IS(D)*

## Common pitfalls when showing $\mathcal{NP}$ -completeness

1. Carry out the reduction in the wrong direction
2. Reduce from a problem not known to be  $\mathcal{NP}$ -complete
3. Exponential-time transformations
  - ▶ Subsets, permutations
4. Neglect to carefully prove both directions of equivalence of the original and the derived instances; that is,  $x$  is a **yes** instance of  $X$  *if and only if*  $y = R(x)$  is a **yes** instance of  $Y$
5. Neglect to show that the problem is in  $\mathcal{NP}$

### Suggestions

- ▶ You should think carefully which problem is most suitable to reduce from
- ▶ In absence of other ideas, reduce from 3SAT

# Today

- 1 Review of last lecture
- 2 Representative  $\mathcal{NP}$ -complete problems**
- 3 Integer Programming
- 4 Minimum-weight Set Cover
  - An integer programming formulation of Set Cover
  - The linear program relaxation
- 5 An approximation algorithm for Set Cover
  - Rounding the LP solution
  - An  $f$ -approximation algorithm for Set Cover



# The Traveling Salesman Problem (TSP)

**Tour:** a *simple* cycle that visits *every* vertex exactly once.

## Definition 7 (TSP(D)).

Given  $n$  cities  $\{1, \dots, n\}$ , a set of non-negative distances  $d_{ij}$  between every pair of cities and a budget  $B$ , is there a tour of length  $\leq B$ ?

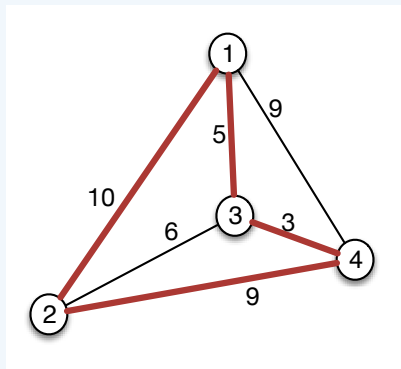
Equivalently, is there a **permutation**  $\pi$  such that

1.  $\pi(1) = \pi(n+1) = 1$ ; that is, we start and end at city 1
2. the total distance travelled satisfies

$$\sum_{i=1}^n d_{\pi(i)\pi(i+1)} \leq B$$

**Application:** Google street view car

## Example instance of TSP



Depending on the distances, TSP instances may be

- ▶ *Asymmetric*:  $d_{ij} \neq d_{ji}$
- ▶ *Symmetric*:  $d_{ij} = d_{ji}$
- ▶ *Metric*: satisfy the triangle inequality  $d_{ij} \leq d_{ik} + d_{kj}$
- ▶ *Euclidean*: e.g., cities are in  $\mathcal{R}^2$  hence city  $i$  corresponds to point  $(x_i, y_i)$ ; then  $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$

## A related problem and hardness of TSP(D)

**Hamiltonian Cycle:** Given a graph  $G = (V, E)$ , is there a simple cycle that visits every vertex exactly once?

### Claim 1.

Hamiltonian Cycle *is*  $\mathcal{NP}$ -complete.

**Proof:** Reduction from 3SAT (e.g., see your textbook).

### Claim 2.

Symmetric TSP(D) *is*  $\mathcal{NP}$ -complete.

**Proof:** reduction from undirected Hamiltonian Cycle.

## Proof of Claim 2 (Hamiltonian Cycle $\leq_P$ TSP(D))

1. Start from an arbitrary instance of undirected **Hamiltonian Cycle**, that is, an undirected graph  $G = (V, E)$ .
2. Construct the following instance  $(G' = (V', E', w), B)$  of TSP(D):  $G'$  is a *complete* weighted graph with  $V' = V$  such that for every edge  $e \in E'$ ,

$$w_e = \begin{cases} 1, & \text{if } e \in E \\ 2, & \text{otherwise} \end{cases}$$

3. Set the budget  $B = n$ .

This completes the reduction transformation.

Equivalence of the instances is straightforward:

- ▶ If  $G$  has a hamiltonian cycle, that cycle is a tour of length  $n$  in  $G'$ .
- ▶ If  $G'$  has a tour of length  $n$ , it must consist of edges of weight 1 (*why?*); thus all these edges appear in  $G$ .

## Concluding remarks on TSP

- ▶ Claim 1 also holds for directed Hamiltonian cycle. An exact analog of the proof of Claim 2 then shows that asymmetric TSP is  $\mathcal{NP}$ -complete.
- ▶ It is possible to reduce Hamiltonian cycle to Euclidean TSP, thus showing that even Euclidean TSP is  $\mathcal{NP}$ -complete.
- ▶ However, these problems are not similar in terms of how well they can be approximated: it is possible to provide very good approximate solutions to Euclidean TSP, which is not the case for Symmetric TSP.

## Packing and partitioning problems

- ▶ **Set Packing:** given a set  $U$  of  $a$  elements, a collection  $S_1, S_2, \dots, S_b$  of subsets of  $U$ , and a number  $k$ , is there a collection of at least  $k$  subsets such that no two of them intersect?
- ▶ **3D-Matching:** Given disjoint sets  $B, G, H$ , each of size  $n$ , and a set of triples  $T \subseteq B \times G \times H$ , is there a set of  $n$  triples in  $T$ , no two of which have an element in common?  
*Reduction from 3SAT.*

- ▶ **Subset sum:** Given natural numbers  $w_1, \dots, w_n$  and a (large) target weight  $W$ , is there a subset of  $w_1, \dots, w_n$  that adds up exactly to  $W$ ?

**Applications:** cryptography, scheduling

- ▶ **Minimum-weight solution to linear equations:** Given a system of linear equations in  $n$  variables with integer constants, and an integer  $B \leq n$ , does it have a rational solution with at most  $B$  non-zero entries?

**Applications:** coding theory, signal processing

# Similar problems with very different complexities

$\mathcal{NP}$ -complete	$\mathcal{P}$
max cut	min cut
longest path	shortest path
3D matching	matching
Hamiltonian cycle	Euler cycle
3-colorability	2-colorability
3-SAT	2-SAT
LCS of $n$ sequences	LCS of 2 sequences

More on  $\mathcal{NP}$ -completeness:

- ▶ *Computers and Intractability: A guide to the theory of  $\mathcal{NP}$ -completeness*, by Garey and Johnson
- ▶ *Computational Complexity*, by C. Papadimitriou



# Today

- 1 Review of last lecture
- 2 Representative  $\mathcal{NP}$ -complete problems
- 3 Integer Programming**
- 4 Minimum-weight Set Cover
  - An integer programming formulation of Set Cover
  - The linear program relaxation
- 5 An approximation algorithm for Set Cover
  - Rounding the LP solution
  - An  $f$ -approximation algorithm for Set Cover

# Integer Programming

**Integer programming (IP(D)):** Given a system of linear inequalities in  $n$  variables and  $m$  constraints with integer coefficients and an integer target value  $k$ , does it have an integer solution of value  $k$ ?

- ▶ Applications: production planning, scheduling trains, etc.

Example:

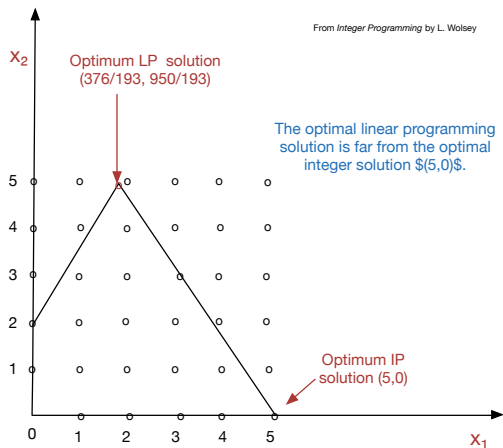
$$\begin{aligned} \max \quad & \mathbf{c}^T \mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \in \mathbf{Z}^n \end{aligned}$$

Here  $A$  is an  $m \times n$  matrix,  $\mathbf{b} \in \mathbf{R}^m$ ,  $\mathbf{c} \in \mathbf{R}^n$ ,  $\mathbf{x}$  is an integer vector with  $n$  components.

*What does the set of feasible solutions look like?*

# Rounding the LP is often insufficient

$$\begin{aligned} \max \quad & 1.00x_1 + 0.64x_2 \\ \text{subject to} \quad & x_1 \geq 0, x_2 \geq 0 \\ & 50x_1 + 31x_2 \leq 250 \\ & 3x_1 - 2x_2 \geq -4 \\ & x_1, x_2 \text{ integer} \end{aligned}$$



## *Is IP(D) hard?*

- ▶ IP(D) is in  $\mathcal{NP}$ .
- ▶ We can quickly solve LPs with several thousands of variables and constraints but there exist integer programs with 10 variables and 10 constraints that are very hard to solve.

## *Is IP(D) hard?*

- ▶ IP(D) is in  $\mathcal{NP}$ .
- ▶ We can quickly solve LPs with several thousands of variables and constraints but there exist integer programs with 10 variables and 10 constraints that are very hard to solve.
- ▶ This is not too surprising: integer programs restricted to solutions  $\mathbf{x} \in \{0, 1\}^n$  model **yes/no** decisions, which are generally hard.
- ▶ To formalize this intuition, we will reduce an  $\mathcal{NP}$ -complete problem to IP(D).

# Integer Programs for Vertex Cover and IS

First we formulate integer programs for two  $\mathcal{NP}$ -hard problems.

IP for Independent Set:

$$\begin{aligned} \max \quad & \sum_{i=0}^n x_i \\ \text{subject to} \quad & x_i + x_j \leq 1, \quad \text{for every } (i, j) \in E \\ & x_i \in \{0, 1\}, \quad \text{for every } i \in V \end{aligned}$$

IP for Vertex Cover:

$$\begin{aligned} \min \quad & \sum_{i=0}^n x_i \\ \text{subject to} \quad & x_i + x_j \geq 1, \quad \text{for every } (i, j) \in E \\ & x_i \in \{0, 1\}, \quad \text{for every } i \in V \end{aligned}$$

### Claim 3.

$$\text{VC(D)} \leq_P \text{IP(D)}$$

### Proof.

Reduction from arbitrary instance  $(G = (V, E), k)$  of VC(D) to the following integer program with target value  $k$ :

$$\begin{aligned} & \min && 0 \\ & \text{subject to} && x_i + x_j \geq 1, \quad \text{for every } (i, j) \in E \\ & && \sum_{i=1}^n x_i \leq k \\ & && x_i \in \{0, 1\}, \quad \text{for every } i \in V \end{aligned}$$

Equivalence of the instances is straightforward. □

# Similar problems with very different complexities (*new*)

$\mathcal{NP}$ -complete	$\mathcal{P}$
max cut	min cut
longest path	shortest path
3D matching	matching
Hamiltonian cycle	Euler cycle
3-colorability	2-colorability
3-SAT	2-SAT
LCS of $n$ sequences	LCS of 2 sequences
integer programming	linear programming

The theory of integer and linear programming and duality can guide the design of approximation algorithms, and exact solutions, for hard problems.



# Today

- 1 Review of last lecture
- 2 Representative  $\mathcal{NP}$ -complete problems
- 3 Integer Programming
- 4 Minimum-weight Set Cover**
  - An integer programming formulation of Set Cover
  - The linear program relaxation
- 5 An approximation algorithm for Set Cover
  - Rounding the LP solution
  - An  $f$ -approximation algorithm for Set Cover

# Minimum-weight Set Cover

## Input

- ▶ a set  $E = \{e_1, e_2, \dots, e_n\}$  of  $n$  elements
- ▶ a collection of subsets of these elements  $S_1, S_2, \dots, S_m$ , where each  $S_j \subseteq E$
- ▶ a non-negative weight  $w_j$  for every subset  $S_j$

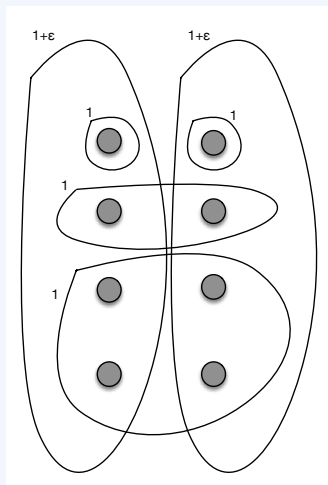
## Output

A minimum-weight collection of subsets that cover all of  $E$ .

In symbols: find an  $I \subseteq \{1, \dots, m\}$  such that  $\cup_{i \in I} S_i = E$  and  $\sum_{i \in I} w_i$  is minimum.

(Unweighted Set Cover:  $w_j = 1$  for all  $j$ )

# Example instance of Set Cover



$n = 8$  ground elements,  $m = 6$  subsets with weights  
 $w_1 = w_2 = w_3 = w_4 = 1$ ,  $w_5 = w_6 = 1 + \epsilon$ .

## Motivation: detect computer viruses

*Motivation (IBM AntiVirus): detect* features of boot sector viruses that do not occur in typical applications; then use them to *discover* more viruses

- ▶ **Ground elements:** known boot sector viruses ( $n \approx 150$ )
  - ▶ **Sets:** labelled by some three-byte sequence occurring in these viruses but not occurring in typical computer applications ( $m \approx 21000$ ); each set consisted of all the viruses that contained the three-byte sequence
  - ▶ **Output:** a small number of such sequences—much smaller than 150—that *cover* all known viruses
- ⇒ use the small set cover as features in a *neural classifier* to determine presence of a boot sector virus
- ⇒ *detect* new viruses (many boot sector viruses are written by modifying existing ones)

# Reduction via generalization

## Claim 4.

Set-Cover(D) is  $\mathcal{NP}$ -complete.

## Proof.

Reduction from VC(D). Input instance:  $(G = (V, E), k)$ .

- ▶ Set  $E = \{e_1, \dots, e_m\}$  to be the set of ground elements we want to *cover*.
- ▶ For every vertex  $j$ , set  $S_j$  to be the set of edges (ground elements) that are incident to—hence *covered* by—vertex  $j$ .
- ▶ Set  $w_j = 1$  for all  $1 \leq j \leq n$ .

Equivalence of instances: input graph has a vertex cover of size  $k$  if and only if  $E$  has a set cover of weight  $k$ . □

## Forming the integer program for Set Cover

**Variables:** we introduce one variable per set  $S_j$ ; intuitively,

$$x_j = \begin{cases} 1, & \text{if } S_j \text{ is included in the solution} \\ 0, & \text{otherwise} \end{cases}$$

**Constraints:** ensure that every element is *covered*:

for every element  $e_i$ , at least one of the sets  $S_j$   
containing  $e_i$  appears in the final solution

**Objective function:** minimize the sum of the weights of the sets included in the solution

# An integer programming formulation of Set Cover

Integer program for **Set Cover**:

$$\begin{aligned} \min \quad & \sum_{j=1}^m w_j x_j \\ \text{subject to} \quad & \sum_{j:e_i \in S_j} x_j \geq 1, \quad \text{for every } 1 \leq i \leq n \\ & x_j \in \{0, 1\}, \quad \text{for every } 1 \leq j \leq m \end{aligned}$$

# An integer programming formulation of Set Cover

Integer program for **Set Cover**:

$$\begin{aligned} \min \quad & \sum_{j=1}^m w_j x_j \\ \text{subject to} \quad & \sum_{j: e_i \in S_j} x_j \geq 1, \quad \text{for every } 1 \leq i \leq n \\ & x_j \in \{0, 1\}, \quad \text{for every } 1 \leq j \leq m \end{aligned}$$

Let  $Z_{IP}^*$  be the optimum value of this integer program;  
 $OPT$  be the value of the optimum solution to **Set Cover**.

$$Z_{IP}^* = OPT.$$

△ We cannot solve this integer program efficiently (*why?*).



# LP relaxation: a bound for the value of the IP

LP relaxation for **Set Cover**:

$$\begin{array}{ll} \min_{\mathbf{x} \geq \mathbf{0}} & \sum_{j=1}^m w_j x_j \\ \text{subject to} & \sum_{j: e_i \in S_j} x_j \geq 1, \quad \text{for every } 1 \leq i \leq n \end{array}$$

# LP relaxation: a bound for the value of the IP

LP relaxation for **Set Cover**:

$$\begin{aligned} \min_{\mathbf{x} \geq \mathbf{0}} \quad & \sum_{j=1}^m w_j x_j \\ \text{subject to} \quad & \sum_{j: e_i \in S_j} x_j \geq 1, \quad \text{for every } 1 \leq i \leq n \end{aligned}$$

- ▶ Every feasible solution to the original IP is a feasible solution to the LP relaxation.
- ▶ The value of any feasible solution to the original IP is the same in the LP (the objectives are the same).
- ▶ Let  $Z_{LP}^*$  be the optimum value of the LP relaxation.

$$Z_{LP}^* \leq Z_{IP}^* = OPT$$

# Today

- 1 Review of last lecture
- 2 Representative  $\mathcal{NP}$ -complete problems
- 3 Integer Programming
- 4 Minimum-weight Set Cover
  - An integer programming formulation of Set Cover
  - The linear program relaxation
- 5 An approximation algorithm for Set Cover
  - Rounding the LP solution
  - An  $f$ -approximation algorithm for Set Cover

# Rounding the solution to the LP

LP relaxation for **Set Cover**:

$$\begin{aligned} \min_{\mathbf{x} \geq \mathbf{0}} \quad & \sum_{j=1}^n w_j x_j \\ \text{subject to} \quad & \sum_{j: e_i \in S_j} x_j \geq 1, \quad \text{for every } 1 \leq i \leq n \end{aligned}$$

- ▶ Let  $x^*$  be an optimal solution to the LP relaxation.
- ▶ Let  $f_i = \#$  subsets  $S_j$  where element  $e_i$  appears.
- ▶ Let  $f = \max_{1 \leq i \leq n} f_i$ .
- ▶ Set

$$\hat{x}_j = \begin{cases} 1, & \text{if } x_j^* \geq 1/f \\ 0, & \text{if } x_j^* < 1/f \end{cases}$$

# Rounding yields a feasible solution to the original IP

The collection of sets  $S_j$  with  $\hat{x}_j = 1$  cover all the elements.

- ▶ Consider the optimal solution  $x^*$  for the LP relaxation.
- ▶ Fix any element  $e_i$ ; recall that  $e_i$  appears in  $f_i$  subsets.
- ▶ For simplicity, relabel these subsets as  $S_1, S_2, \dots, S_{f_i}$ . Then the optimal solution satisfies the constraint

$$x_1^* + x_2^* + \dots + x_{f_i}^* \geq 1$$

Let  $x_m^*$  be the maximum of  $x_1^*, x_2^*, \dots, x_{f_i}^*$ . Then

$$x_m^* \geq \frac{1}{f_i} \geq \frac{1}{f}$$

- ⇒ Our rounding procedure guarantees that, for every element  $e_i$ , at least one set  $S_j$  that *covers*  $e_i$  is chosen.

# An $f$ -approximation algorithm for Set Cover

How far is the solution obtained by the rounding procedure above from to the *optimal* solution to Set Cover?

- ▶ We do **not** know *OPT*!
- ▶ **But** we have a **bound** for it: the value  $Z_{LP}^*$  of the LP relaxation!

Recall that we set  $\hat{x}_j = 1$  if and only if  $x_j^* \geq 1/f$ . Then

$$\begin{aligned}\sum_j w_j \hat{x}_j &\leq \sum_j w_j (f x_j^*) = f \sum_j w_j x_j^* \\ &= f \cdot Z_{LP}^* \leq f \cdot OPT\end{aligned}$$

## Definition 8.

An  $\alpha$ -approximation algorithm for an optimization problem is a polynomial-time algorithm that, for all instances of the problem, produces a solution whose value is within a factor of  $\alpha$  of the value of the optimal solution.

## Remark 1.

- ▶  $\alpha$  is the approximation ratio or approximation factor
- ▶ For *minimization* problems,  $\alpha > 1$ .
- ▶ For *maximization* problems,  $\alpha < 1$ .

**Example 1:** the rounding procedure described on slide 53 gives an  $f$ -approximation algorithm for **Set Cover**:

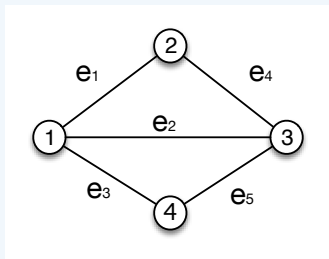
- ▶ it can be completed in polynomial-time
- ▶ it always returns a solution whose value is at most  $f$  times the value of the optimal solution.

**Remark:** if an element appears in too many sets (e.g.,  $f = \Omega(n)$ ), this algorithm does not provide a good approximation guarantee.

**Example 2:** a 2-approximation algorithm for VC is a polynomial-time algorithm that always returns a solution whose value is at most twice the value of the optimal solution.



## A 2-approximation algorithm for $VC$



- ▶ Let  $E = \{e_1, \dots, e_m\}$  be the set of edges in the graph.
- ▶ Let  $S_j$  be the set of edges (ground elements) that are covered by vertex  $j$ .
- ▶ For every edge  $e_i$ ,  $f_i = 2$ :  $e_i$  appears in exactly two subsets (*why?*).
- ▶ Hence  $f = \max_{1 \leq i \leq m} f_i = 2$ .