COMPSCI 530: Design and Analysis of Algorithms	October 22, 2013
Lecture #15	
Lecturer: Debmalya Panigrahi	*Scribe: Ang Li*

1 Overview

In this lecture, we cover basic definitions involving NP, NP-Hard, and NP-Complete problems, give examples of each, and demonstrate polynomial time reductions and approximate algorithms through classical examples such as Vertex Cover and Set Cover.

2 Decision Problems in NP and CO-NP

Definition 1. *A decision problem is in NP if for* **any yes instance***, there is a succinctly verifiable proof or certificate.*

Example 1. Hamiltonian Cycle (HAM) is in NP. Given $G = (V, E)$, is there a cycle in G such that every vertex is visited exactly once?

Proof. Given a Hamiltonian Cycle, we can simply check whether each vertex in the graph is visited exactly once to verify. □

Example 2. Vertex Cover (VC) is in NP. Given $G = (V, E)$, are there k vertices in the graph such that every edge is incident on at least one of the k vertices?

Proof. Given a vertex cover of size k, we can simply check whether each edge in the graph is incident on at least one of the k vertices. □

Definition 2. *A decision problem is in CO-NP if for* **any no instance***, there is a succinctly verifiable proof or certificate.*

Example 3. The complement of the Hamiltonian Cycle ($\overline{HAM}$) is in CO-NP. Given $G = (V, E)$, is there no cycle in G such that every vertex is visited exactly once?

Proof. When there is no Hamiltonian Cycle existing, we cannot prove that there is no Hamiltonian Cycle in the given graph in polynomial time. However, given a Hamiltonian Cycle from the graph, we can easily prove that the statement is wrong by verifying that particular Hamiltonian Cycle in polynomial time. □

Theorem 1. $P \subseteq NP \cap CO - NP$

Proof. Any problem in NP has a polynomial time certificate for any Yes instance. Any problem in CO-NP has a polynomial time certificate for any No instance. Any problem in P is polynomial-time, efficiently solvable. Any instance of P is in both NP and CO-NP because an empty certificate is needed to verify either a Yes or a No instance. The efficient, polynomial-time algorithm for the problem can simply be run to determine the exact answer to the problem. □

3 NP-Hard, NP-Complete, and Polynomial-Time Reductions

Definition 3. *Polynomial-time reductions can be performed for a problem B to a problem P in the following way. Given an instance of B, polynomial time is spent on converting it to an instance of P. The oracle for solving P can then be used to solve that particular instance of P. Finally, the outcome of P is transformed to the outcome of B in polynomial time.*

Definition 4. *A problem P is in NP-Hard if any problem B in NP can be reduced in polynomial-time to P, that is $B \leq P$.*

Definition 5. *A problem P is in NP-Complete if it is both NP-Hard and in NP.*

Theorem 2. *Circuit Satisfiability (SAT) is NP-Hard*

Example 1. Reduction from SAT to 3SAT.

Given an instance of SAT: $(l_1 \vee l_2 \vee ... \vee l_k) \wedge ....$ The following 3SAT instance can be shown to be equivalent to the original SAT problem: $(l_1 \vee l_2 \vee x_1) \wedge (\neg x_1 \vee l_3 \vee x_2) \wedge ... \wedge (\neg x_{k-3} \vee l_{k-1} \vee l_k) \wedge ....$

Proof. If the original SAT clause is satisfiable, one of the variables must be set to true. In the reduced 3SAT clauses corresponding to the SAT clause, when the same variable is set to true, all the x variables can be set in a unique way such that the clauses evaluate to true. If the original SAT clause is not satisfiable, none of the variables can be true. In the reduced 3SAT clauses, if none of the l variables is true, there's no way for the clauses to evaluate to true regardless of the x variables. Hence, the reduced instance of 3SAT is satisfiable if and only if the original SAT problem is satisfiable. □

Example 2. Reduction from 3SAT to Independent Set.

Given an instance of 3SAT: $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2)$. There is an equivalent instance of independent set with $G = (V, E)$ and k, such that there are a set of k vertices with no two vertices sharing an edge if and only if the 3SAT problem is satisfiable.

Proof. For each clause in the 3SAT problem, construct a clique of at most 3 vertices, and finally add edges between opposite literals in the combined graph to obtain G, and k is set to be the number of clauses. If the original 3 SAT problem is satisfiable, at least one literal from each clause will be set to true, and one of the corresponding vertices in G can be picked without sharing any edge, for every clause. Since opposite literals cannot be true at the same time in 3SAT, we can be sure that the vertices picked will form an independent set of size k in G. On the other hand, if the original 3SAT problem is not satisfiable, it is impossible to have an independent set of size k because at most one vertex can be chosen from each clique, and we have zero vertex chosen from at least one of the clique (the unsatisfiable clause). □

Example 3. Reduction from Independent Set to Vertex Cover.

Given an instance of Independent Set: $G_1 = (V_1, E_1)$ and k_1. There's an equivalent instance of Vertex Cover, $G_2 = (V_2, E_2)$ and k_2. And the two decision problems are either both Yes or both No.

Proof. Set $G_2 = G_1 = G$, $k_2 = |V| - k_1$. We can observe that G has an independent set of size k if and only if G has a vertex cover of size $|V| - k$, because $VC = \{v \notin IS\}$, $IS = \{v \notin VC\}$ according to definitions. □

Example 4. Reduction from 3SAT to Integer Linear Programming.

Given an instance of 3SAT: $(x_1 \vee \neg x_2 \vee x_3) \wedge ...$, there is an equivalent instance of integer feasibility LP that is feasible if and only if the 3SAT instance is satisfiable.

Proof. Every clause in 3SAT produces a linear constraint: $x_1 + (1 - x_2) + x_3 \geq 1, x_i \in \{0,1\}$. If a 3SAT clause is satisfiable, the corresponding true variable can be set to 1 in the LP and the constraint will be satisfied. If a 3SAT clause is not satisfiable, the corresponding linear constraint will be violated and the LP will be infeasible as a result. □

Example 5. Reduction from Independent Set to Clique Number.

Given an instance of Independent Set: $G_1 = (V_1, E_1)$ and k_1. There's an equivalent instance of Clique Number(size of largest clique), $G_2 = (V_2, E_2)$ and k_2. And the two decision problems are either both YES or both No.

Proof. Set $G_2 = \bar{G_1} = G_1(V, \bar{E})$, whereby $\bar{E} = \{e \notin E\}$, and $k_2 = k_1 = k$. It can be observed that G_2 has a clique of size k if and only if G_1 has an independent set of size k, because any two of the k vertices sharing no common edge in G_1 has an edge between them in G_2. □

4 Approximation Algorithms

Definition 6. *Approximation Ratio for an Approximation Algorithm (ALGO) is defined as:*

$$Approximation\ Ratio\ (Maximization) = max\ \frac{value of OPT}{value of ALGO}. \tag{1}$$

$$Approximation\ Ratio\ (Minimization) = max\ \frac{value of ALGO}{value of OPT}. \tag{2}$$

Example 1. A 2-approximation algorithm for Vertex Cover.

Repeat until graph is empty:
—pick an edge and add the vertices at both ends to the Vertex Cover
—remove all edges incident on either of the vertices

Lemma 3. *In any iteration, if the algorithm picked edge e = (u, v) and added both vertices to the vertex cover, then either u or v is in the optimal solution, and this algorithm produces a 2-approximation.*

Proof. Consider the edge e picked in any iteration. For the optimal vertex cover solution cover this edge, either u or v has to be in the optimal solution OPT for edge e to be covered with this vertex cover. The approximation algorithm included both vertices, thus incurring an approximation ratio of 2. □

Example 2. A log(N)-approximation algorithm for Set Cover.

Problem statement: Given a Universe U of elements, sets $S_i \subseteq U$, find a collection of subsets of minimum size such that all elements are covered.

Algorithm: Greedy Algorithm.

Repeat until all elements are covered:
— picked the set that covers the maximum number of uncovered elements.

Theorem 4. *This Greedy Algorithm produces O(log n)-approximation.*

Proof. Suppose that during some iteration of the algorithm there are k elements left to be covered. The remaining elements can be covered using at most OPT sets in the optimal solution. Hence, the following is true.

$$Number\ of\ elements\ covered\ by\ each\ set \geq \frac{k}{OPT}\ on\ average \tag{3}$$

As a result, there is a subset remaining that has cost-benefit ratio of at most $\frac{OPT}{k}$. Since the algorithm always chooses the set with minimum cost-benefit ratio, it is guaranteed that

$$\frac{cost}{benefit} \leq \frac{OPT}{k} \tag{4}$$

Finally, the total cost of the approximation algorithm is calculated using the harmonic sum formula,

$$Total\ Cost \leq \sum_{k=n}^{1} \frac{OPT}{k} = OPT \cdot H_n = OPT \cdot O(logn) \tag{5}$$

□

5 Summary

In this lecture, we covered some basic examples of NP, CO-NP problems, illustrated polynomial-time reductions between well-known NP-Hard problems, and developed techniques of applying an approximation algorithm and proving its approximation ratio for some NP-Hard problems using Vertex Cover and Set Cover as examples.