**Lecture 17**

**NP-Complete Problems**

**Approximate Algorithms**

---

**Polynomial Reductions** — a formal way to say "as hard as".

- Reduction is a transformation from one problem to another.
  - Formally, let $T$ be a function from input set for a decision problem $U$ into the input set for a decision problem $V$, such that
    - For every string $x$, if $x$ is a yes input for $U$, then $T(x)$ is a yes input for $V$.
    - For every string $x$, if $x$ is a no input for $U$, then $T(x)$ is a no input for $V$. (Or equivalently, $T(x)$ is a yes input for $V$, then $x$ is a yes input for $U$).
  - $T$ is a polynomial reduction when it can be computed in polynomial bounded time.
  - Problem $U$ is polynomially reducible to $V$, denoted as $U \leq_p V$, if there exists a polynomial reduction from $U$ to $V$.

---

**Example:** A dominating set of a graph $G$ is a subset of vertices $W$ such that every vertex of $G$ is either in $W$ or is adjacent to a vertex in $W$.

**Decision Problem:**
- Input: A graph $G$ and integer $k$.
- Question: Does $G$ have a dominating set of size at most $k$?

**Optimization Problem:**
- Input: A graph $G$.
- Output: A smallest dominating set of $G$.

The Claim: The optimization problem for dominating set is no harder than the decision problem.

**Proof:** Use Polynomial reduction.

- **Step 1:** For input from $1$ to $G$, call the algorithm for the decision problem as a subroutine on input $(G,k)$. This will give us $M$, the size of the smallest dominating set of $G$.
- **Step 2:** Initialize all vertices in $G$ as unmarked. Now choose an unmarked vertex $v$ in $G$.
  - Add a new vertex $X$ attached by an edge to $v$. Ask the decision algorithm if the modified graph has a dominating set of size $M$. If yes, then mark $v$ as a member of the dominating set and remove the vertex $X$. Repeat until all vertices are marked as members of the dominating set.

* *Module a polynomial reduction.

---

**Brief Recap**

- A problem is said to be very hard, but its solution can be verified relatively easily, i.e., in polynomial time. Is this problem in NP?
- Given two problems $A$ and $B$, how do we check the claim that $A$ is as hard as or even harder than $B$, in another word, $A$ is "B-hard"?
- What is NP-Complete?
- What is Cook’s Theorem? Why is it important?
- What are the three requirements for Polynomial Reductions?

---

**What to do in case of NP-Complete problems?**

- Use a heuristic
- Find an approximate algorithm
- Use exponential time algorithm anyway
Let $S = \{s_1, \ldots, s_n\}$ be an input, in non-increasing order, for the bin packing problem and let $\text{opt}(S)$ be the optimal number of bins for $S$. All of the objects placed by FFD in extra bins (i.e., bins with index larger than $\text{opt}(S)$) have size at most $1/3$.

**Proof**

Let $i$ be the index of the first object placed by FFD in bin $\text{opt}(S)+1$. $S_i$ must be no larger than $1/3$.

Suppose $S_i > 1/3$. Then $S_i, \ldots, S_{\lfloor \text{opt}(S) \rfloor}$, at most two objects each. For some $k > 0$, the first $k$ bins hold one object each, and bins $B_{\lfloor \text{opt}(S) \rfloor}, \ldots, B_{\text{opt}(S)}$ hold two objects each. $S_i > 1/3$ can not fit even by an optimal solution. But an optimal solution must fit object in one of the first $\text{opt}(S)$ bins. Therefore, the assumption that $S_i > 1/3$ must be false.

**Lemma 13.10**

For an input $S = \{s_1, \ldots, s_n\}$ the number of objects placed by FFD in extra bins is at most $\text{opt}(S) - 1$.

**Proof**

Assume FFD puts $\text{opt}(S)$ objects with sizes $t_1, \ldots, t_{\text{opt}(S)}$ in extra bins. Let $b_i$ be the used space for each of the first $\text{opt}(S)$ bins. Then

$$\sum_{i=1}^{\text{opt}(S)} s_i \geq \sum_{i=1}^{\text{opt}(S)} t_i = \sum_{i=1}^{\text{opt}(S)} (b_i + t_i) > \text{opt}(S)$$

However, since all the objects fit in $\text{opt}(S)$ bins, we must have

$$\sum_{i=1}^{\text{opt}(S)} s_i < \text{opt}(S)$$

Therefore, the assumption of $\text{opt}(S)$ objects being put into extra bins is not valid.

**Travelling Salesman Problem (TSP)**

Optimization problem: Given a complete, weighted graph, find a minimum-weight Hamiltonian cycle.

Decision Problem: Given a complete, weighted graph and an integer $k$, is there a Hamiltonian cycle with total weight at most $k$?

**Nearest-Neighbor Strategy**

Select an arbitrary vertex $v$ to start the cycle $C$. $V = S$

While there are vertices not yet in $C$, select an edge $vw$ of minimum weight where $w$ is not in $C$. Add edge $vw$ to $C$. $V = \emptyset$. Add the edge $vw$ to $C$.
Shortest-Link Strategy
shortestLink(TSP(V,E,W))
R = E; // R is remaining edges
C = empty; // C is cycle edges
while R is not empty
  Remove the lightest edge, vw, from R
  If we do not make a cycle with edges in C
  and we would not be the third edge in C incident on v or w
  Add vw to C
  Add the edge connecting the endpoints of the path in C
  return C.

Performance evaluation
Theorem 13.22 Let A be any approximation algorithm for the TSP. If there is any constant c such that A(G) ≤ c∗(G) for all instances, then P = NP.

Why DNA computing?
- The information density of DNA is greater than that of silicon: 1 bit can be stored in about one cubic nanometer.
- Operations on DNA are massive parallel

Adleman’s DNA algorithm for Hamiltonian path problem
1. Generate DNA strands to represent paths in G
2. Use biochemical processes to extract strands satisfying the following
   a. Extract strands that start at vstart and end at vend (discard the rest)
   b. Extract strands that include n vertices (Discard the rest)
   c. Extract strands that contain every vertex
   d. Any strand that remains represents a Hamiltonian path from v start to vend. If no strand remains, G has no such path