Coping with NP-hard Problems

- NP-Hardness
- Optimization Problems
- Approximation Algorithms

NP-Hardness

- A problem (not necessarily belong to NP) is NP-hard if it is proved to be "at least as hard" as the NP-complete problems.
  - All the NP-complete problems are NP-hard.
  - Any decision problem Π, whether a member of NP or not, to which we can transform an NP-complete problem will have the property that it cannot be solved in polynomial time unless P=NP. such a problem Π is "NP-hard", since it is, in a sense, at least as hard as the NP-complete problems.
- In general, The technique used for proving NP-hardness is polynomial time Turing reduction.
  - Recall: A Turing reduction from a problem Π to a problem Π', denoted by $\Pi \propto_T \Pi'$, is a algorithm A that solves Π by using a hypothetical subroutine S for solving Π' such that, if S were a polynomial time algorithm for Π', then A would be a polynomial time algorithm for Π.
  - $\Pi \propto_T \Pi' \Rightarrow \Pi'$ is at least as hard as Π.
  - If $\Pi \propto_T \Pi'$ and Π is NP-complete or NP-hard, then Π' is NP-hard.
  - Note: The strict definition needs OTM –Oracle Turing Machine. We omit it.

Kth Largest subset

Instance: A finite set A of positive integers, two nonnegative integers $K \le 2^{|A|}$ and $B \le \sum_{a\in A} a$.

Question: Are there at least K distinct subsets $A' \subseteq A$ that satisfies $\sum_{a\in A'} a \le B$?

- **Kth Largest Subset** does not appear to be in **NP.**
  - Since K can be as large as near $2^{|A|}$.
- **Kth Largest Subset is NP-hard.**
- **Proof:** Turing reduction from PARTITION.
  - Suppose $S[A, B, K]$ is a subroutine for solving the Kth Largest Subset problem, with parameters
    - $A=\{a_1, a_2, \ldots, a_n\}$,
    - $B \le \sum_{a\in A'} a$, and
    - $K \le 2^n$.
  - The corresponding algorithm for solving PARTITION with instance $A=\{a_1, a_2, \ldots, a_n\}$ is the next.

1. If $\sum_{a\in A} a$ is odd then return "no" else set $b= (\sum_{a\in A'} a)/2$.
2. Binarily search, using the assumed subroutine S, for the number of subsets $A' \subseteq A$ satisfying $\sum_{a\in A'} a \le b$.
   a. Set $L_{min}=0$, $L_{max}=2^n$.
   b. While $L_{min}< L_{max} - 1$ do
      - Set $L=(L_{min}+ L_{max})/2$ and call $S[A, b, L]$.
      - If the answer is "yes" then set $L_{min}=L$ else set $L_{max}=L$.
3. Call $S[A, b-1, L]$. If the answer is "yes", return "no", else return "yes".

- The algorithm would be a polynomial time for PARTITION if S were a polynomial time algorithm for Kth Largest Subset.
- Thus, Kth Largest Subset is NP-hard.

Optimization problems

- A combinatorial optimization problem Π is either a minimization problem or a maximization problem and consists of the following three parts:
  1. A set D_Π of instances;
  2. for each instance $I \in D_\Pi$, a finite set $S_\Pi(I)$ of candidate solutions for I; and
  3. A function m_Π that assigns to each instance $I \in D_\Pi$ and each candidate solution $\sigma \in S_\Pi(I)$ a positive rational number $m_\Pi(I,\sigma)$ called the solution value for σ.
- If Π is a minimization [maximization] problem, then an optimal solution for an instance $I \in D_\Pi$ is a candidate solution $\sigma^* \in S_\Pi(I)$ such that $m_\Pi(I,\sigma^*) \le m_\Pi(I,\sigma)$ [$m_\Pi(I,\sigma^*) \ge m_\Pi(I,\sigma)$] for all $\sigma \in S_\Pi(I)$.

Examples of NP-hard optimization problems.

- **Traveling Salesman (TS)**

  Instance: A graph $G=(V,E)$, a weight $W: E \rightarrow Z^+$.

  Objective: Find a Hamiltonian circuit C so that $\sum_{e\in C} W(e)$ is minimized.
- **Vertex Cover (VC)**

  Instance: A graph $G=(V,E)$.

  Objective: Find a vertex cover C in G so that |C| is minimized.

Approximation Algorithms

- Definitions
- Absolute approximations
- Relative approximations
- Hardness of approximations

Approximation Algorithms

- An approximation algorithm A for optimization problem Π guarantees:
  - Feasibility: For each instance $I \in D_\Pi$ it finds a solution $\sigma \in S_\Pi(I)$.
  - Performance: Denote by $A(I)$ the value of the solution it produces for $I \in D_\Pi$, by OPT(I) the value of the optimal solution for $I \in D_\Pi$, then
    - A is an absolute approximation algorithm for Π if there is a constant k, such that $|A(I)\text{-}OPT(I)| \le k$ for every $I \in D_\Pi$.
    - A is a relative approximation algorithm for Π if there is a constant k, such that $A(I)/OPT(I) \le k$ if Π is to minimize, and $OPT(I)/A(I) \le k$ if Π is to maximize, for every $I \in D_\Pi$.
- For relative approximation algorithm A, the constant k is called the performance ratio of A.
- Obviously:
  - performance ratio should be no less than 1, and
  - The smaller the performance ratio is, the better the algorithm is.

Absolute approximations (I)

Maximum Programs Stored

Instance: n programs of sizes respective $L_1, L_1, \ldots, L_n$, two disks of the same size L.

Objective: Store maximum number of programs in the two disks with the constraint that a program must be wholly put in one disk.

Comment: The problem is NP-hard since an algorithm for it can be used to solve the PARTITION problem, by looking at the solution for the case of $L=(\sum L_i)/2$.

- An absolute approximation algorithm A:

  Repeatedly put the smallest waiting program in the first disk, until the present program cannot be accomodated, then do the same on the second disk.
- The algorithm is clearly of polynomial time.
- For any instance I of Maximum Programs Stored, we have |OPT(I)-A(I)|≤1.
- Proof: suppose $L_1 \le L_1 \le \ldots \le L_n$. and $A(I)=K$.
  - Let p be the maximum integer between 0 and n such that $\sum_{I \le i \le p} L_i \le 2L$, then $OPT(I) \le p$.
  - Let j be the maximum integer between 0 and n such that $\sum_{I \le i \le j} L_i \le L$, then
    a. $j \le p$,
    b. algorithm A stores the first j programs in disk 1, and
    c. $\sum_{j<i<p} L_i \le L$.
  - The c in the above implies that the algorithm A puts at least the next p-j-1 programs, from the (j+1)-*th* to the (p-1)-*th*, into disk 2.
  - Thus A(I)≥p-1, giving us that |OPT(I)-A(I)|≤1.