<u>NP-optimization problems</u>: An NP-optimization problem, Π, consists of:

① A set of <u>valid instances</u>, D_Π, recognizable in polynomial time. The size of an instance $I \in D_\Pi$, denoted by $|I|$, is defined as the number of bits needed to write I under the assumption that all numbers occuring in the instance are written in binary.

② Each instance $I \in D_\Pi$ has a set of <u>feasible solutions</u>, $S_\Pi(I)$. We require that $S_\Pi(I) \neq \phi$, and that every solution $s \in S_\Pi(I)$ is of length polynomially bounded in $|I|$. Furthermore, there is a polynomial time algorithm that, given a pair (I, s), decides whether $s \in S_\Pi(I)$.

③ There is a polynomial time computable <u>objective function</u>, obj_Π, that assigns a nonnegative rational number to each pair (I, s), where I is an instance and s is a feasible solution for I.

④ Π is specified to be either a <u>minimization problem</u> or a <u>maximization problem</u>.

The restriction of Π to unit cost instances is called the <u>cardinality version of Π</u>.

An <u>optimal solution</u> for an instance of a minimization (maximization) problem is a feasible solution that achieves the smallest (largest) objective function value. $OPT_\Pi(I)$ will denote the objective function value of an optimal solution to instance I.

With every NP-optimization problem, we can naturally associate a decision problem by giving a bound on the optimal solution. Thus, the decision version of NP-optimization problem Π consist of pairs (I, B), where I is an instance of Π and B is a rational number. If Π is a minimization (maximization) problem, then the answer to the decision version is "yes" if and only if there is a feasible solution to I of cost $\leq B$ ($\geq B$). If so, we will say that (I, B) is a "yes" instance; otherwise it is called a "no" instance.

A polynomial time algorithm for Π can help solve the decision version – by computing the cost of an optimal solution and comparing it with B. A polynomial-time algorithm for decision version of Π can be used to find the optimal solution – by performing binary search on B. Hardness for an NP-optimization problem is established by showing that its decision version is NP-Hard.

An <u>approximation algorithm</u> produces a feasible solution that is "close" to the optimal one, and is time efficient. Let Π be a minimization (maximization) problem, and let δ be a function, $\delta: Z^+ \rightarrow Q^+$, with $\delta \geq 1$ ($\delta \leq 1$). An algorithm A is said to be a factor δ <u>approximation algorithm</u> for Π if, on each instance I, A produces a feasible solution s for I such that $f_\Pi(I, s) \leq \delta(|I|).OPT(I)$ ($f_\Pi(I, s) \geq \delta(|I|).OPT(I)$), and the running time of A is bounded by a fixed polynomial in $|I|$.

<u>The Vertex Cover Problem</u>: Given an undirected graph $G = (V, E)$, and a cost function on vertices $c: V \to \mathbb{Q}^+$ find a minimum cost vertex cover, i.e., a set $V' \subseteq V$ such that every edge has at least one endpoint incident at V'. The special case, in which all vertices are of unit cost, will be called the cardinality vertex cover problem.

<u>A 2-approximation algorithm for the cardinality vertex cover problem</u>: Given a graph $H = (U, F)$, a subset of the edges $M \subseteq F$ is said to be a matching if no two edges of M share an endpoint. A matching of maximum cardinality in H is called a maximum matching, and a matching that is maximal under inclusion is called a maximal matching. A maximal matching can clearly be computed in polynomial time by simply greedily picking edges and removing endpoints of picked edges. The size of a maximal matching in G provides a lower bound. This is so because any vertex cover has to pick at least one endpoint of each matched edge.

<u>Algorithm</u>: Find a maximal matching in G and output the set of matched vertices.

No edge can be left uncovered by the set of vertices picked - otherwise such an edge could have been added to the matching, contradicting its maximality. Let M be the matching picked. $|M| \le OPT \Rightarrow |A| = 2|M| \le 2 \cdot OPT$
$\Rightarrow \frac{|A|}{OPT} \le 2$. <u>Tight examples</u>: ① <u>$K_{n,n}$</u>. $|A| = 2n$, $OPT = n$.

② <u>K_n</u> for odd n: $|A| = n-1$, $OPT = n-1$

A 2-approximation algorithm for weighted vertex cover:

ILP formulation of the problem:

$$\text{Min} \sum_{i \in V} w_i x_i$$

such that $\quad x_i + x_j \geq 1 \quad \forall (i,j) \in E$

$\qquad\qquad\quad x_i \in \{0,1\} \quad \forall i \in V$

LP relaxation of the above ILP:

$$\text{Min} \sum_{i \in V} w_i x_i$$

such that $\quad x_i + x_j \geq 1 \quad \forall (i,j) \in E$

$\qquad\qquad\quad 0 \leq x_i \leq 1 \quad \forall i \in V$

Let S^* denote a vertex cover of minimum weight. Then

$\underbrace{W_{LP}}_{\text{LP Solution}} \leq \underbrace{W(S^*)}_{\text{ILP Solution}}$.

ILP problem is NP-complete, therefore we cannot hope to solve the above ILP in polynomial time. LP problem is in P. We can solve it in polynomial time by using ellipsoid algorithm.

Given a fractional solution $\{x_i^*\}$, we define $S = \{i \in V : x_i^* \geq 1/2\}$.

The set S defined in this way is a vertex cover, and $W(S) \leq 2 \cdot W_{LP}$.

Consider an edge $e = (i,j) \Rightarrow x_i + x_j \geq 1 \Rightarrow$ either $x_i^* \geq 1/2$ or $x_j^* \geq 1/2 \Rightarrow$ at least one of i or j will be in S.

$$W_{LP} = \sum_i w_i x_i^* \geq \sum_{i \in S} w_i x_i^* \geq \frac{1}{2} \sum_{i \in S} w_i = \frac{1}{2} W(S)$$

$$\Rightarrow \underline{W(S) \leq 2 W_{LP} \leq 2 W(S^*)}$$

$$\Rightarrow \boxed{\frac{W(S)}{W(S^*)} \leq 2}$$