Part 12 SAT Problem

(**Definition 1**) A **literal** is a Boolean variable or its negation, e.g., x_2 and $\overline{x}_3$. The operations between each literal include **disjunction** $\vee$ (i.e., 'OR') and **conjunction** $\wedge$ (i.e., 'AND'). A **clause** is the **disjunction** of **literals** or a single **literal**, e.g., $x_1 \vee \overline{x}_2 \vee x_3$, $\overline{x}_2 \vee \overline{x}_3$, x_2, and $\overline{x}_4$.

Conjunction Normal Form (**CNF**) is a formula that is expressed as the **conjunction** of **clauses**. Any Boolean formula involving 'AND', 'OR', and 'NOT' can be transformed into the **CNF**.

Notes: For example, $(x_1 \vee \overline{x}_2 \vee x_3) \wedge (x_2 \vee \overline{x}_3) \wedge (\overline{x}_4 \vee x_1)$ is a Boolean formula written in **CNF**.

(**Definition 2**) (**Satisfiability** (**SAT**) **Problem**) Given a Boolean formula in **CNF**, determine whether there is an **assignment** (of true or false values) to the variables such that the formula is **satisfied**, i.e., the whole formula is true.

Notes: SAT is the first **NP-Complete** problem being found.

(**Definition 3**) (**MAX-SAT Problem**) Given n Boolean **variables** $\{x_1, x_2, \cdots, x_n\}$ (each of which can be set to true or false) and m **clauses** $\{C_1, C_2, \cdots, C_m\}$ (each of which is a disjunction of literals), find an **assignment** (with *true* or *false* values) to the given variables that **maximizes** the number of clauses satisfied.

Notes: The **SAT Problem** and **MAX-SAT Problem** equal in **difficulty**. Namely, if we can solve the **MAX-SAT Problem** in **polynomial** time, then we can solve the **SAT Problem** in **polynomial** time. However, SAT Problem is **NP-Complete**.

(**Algorithm 1**) (**Randomized Algorithm**) Set each variable x_i independently to be *true* or *false* each with the **probability** 1/2.

(**Theorem 1**) The **approximation ratio** of **Algorithm 1** is 1/2.

Proof of **Theorem 1**. Let $C_j = x_1 \vee x_2 \vee \cdots \vee x_k$ denote an arbitrary **clause** with length of k. For the **probability** that the clause C_j is satisfied, we have

$$\begin{aligned} P(C_j \text{ is satisfied}) &= 1 - P(\text{all the } k \text{ literals are } \textit{false}) \\ &= 1 - (\frac{1}{2})^k \\ &\geq \frac{1}{2} \end{aligned}.$$

Let S be the number of clauses that is satisfied. For the **expected** number of clauses that is satisfied, we have

$$\mathrm{E}[S] = \sum_{j=1}^{m} 1 \cdot P(C_j \text{ is satisfied}) \geq \frac{m}{2}.$$

Let $OPT_{\mathrm{SAT}}(I)$ denote the maximum number of clauses satisfied. We have $OPT_{\mathrm{SAT}}(I)=m$. Hence, the approximation ratio of **Algorithm 1** is

$$\frac{\mathrm{E}[S]}{OPT_{\mathrm{SAT}}(I)} \le \frac{m/2}{m} = \frac{1}{2}\cdot$$

(**Example 1**) (**MAX-3SAT Problem**) In the **MAX-3SAT Problem**, where each clause has exactly 3 literals, the **expected** number of clauses satisfied is

$$E[S] = m(1-(\frac{1}{2})^3) = \frac{7}{8}m \ge \frac{7}{8}OPT_{\mathrm{SAT}}(I)\cdot$$

Hence, the **approximation ratio** of **Algorithm 1** w.r.t. **MAX-3SAT Problem** is 7/8. This is a lower bound show that it's impossible to achieve an approximation factor better than 7/8 for **MAX-3SAT Problem**, unless *P=NP*. The simple algorithm (i.e., **Algorithm 1**) is probably the best possible.

(**Example 2**) (**ILP** for **MAX-SAT Problem**) Consider the **MAX-SAT Problem** with 3 clauses:

$$x_1 \vee \overline{x}_2 \vee x_3,\ \ \overline{x}_1 \vee x_2 \vee \overline{x}_3,\ \ x_2 \vee x_3\,.$$

For each **clause** C_j, introduce a variable $z_j \in \{0,1\}$ with the definition that

$$z_j = \begin{cases} 1,\ C_j \text{ is } \textit{true} \\ 0, C_j \text{ if } \textit{false} \end{cases}.$$

For each **Boolean variable** x_i, introduce a variable $y_i \in \{0,1\}$ with the definition that

$$y_i = \begin{cases} 1,\ x_i \text{ is } \textit{true} \\ 0, x_i \text{ is } \textit{false} \end{cases}.$$

We can reformulate the aforementioned **MAX-SAT Problem** as the following **ILP**:

$$\begin{aligned} &\max z_1 + z_2 + z_3 \\ &\text{s.t. } y_1 + (1-y_2) + y_3 \ge z_1 \\ &\quad (1-y_1) + y_2 + (1-y_3) \ge z_2 \\ &\quad y_2 + y_3 \ge z_3 \\ &\quad z_j \in \{0,1\}, y_i \in \{0,1\} \end{aligned}.$$

The corresponding **LP-Relaxation** can be represented as follow:

$$\begin{aligned} &\max z_1 + z_2 + z_3 \\ &\text{s.t. } y_1 + (1-y_2) + y_3 \ge z_1 \\ &\quad (1-y_1) + y_2 + (1-y_3) \ge z_2 \\ &\quad y_2 + y_3 \ge z_3 \\ &\quad 0 \le z_j \le 1, 0 \le y_i \le 1 \end{aligned}.$$

(**Algorithm 2**) (**Randomized Rounding Algorithm**)

(1) Obtain the **optimal solution** to the **LP-Relaxation** (as introduced in **Example 2**) denoting as $\{y_1^*,\cdots,y_k^*\}$ and $\{z_1^*,\cdots,z_m^*\}$.

(2) Set each variable x_i to be true with the probability y_i^*.

(**Theorem 2**) The **approximation ratio** of **Algorithm 2** is $(1-1/e)$.

Proof of **Theorem 2**. Without loss of generality, consider an arbitrary clause $C_j = x_1 \vee \cdots \vee x_k$.

For the probability that C_j satisfied, we have

$$\begin{aligned} P(C_j \text{ is satisfied}) &= 1 - P(\text{all the literals are } \textit{false}) \\ &= 1 - \prod_{i=1}^{k} (1 - y_i^*) \end{aligned}.$$

Consider the worst case, in which all the variables $\{y_1^*, \cdots, y_k^*\}$ are with the same value, i.e., $y_1^* = \cdots = y_k^*$, $\prod_{i=1}^{k} (1 - y_i^*)$ will have the largest value. Since we also have the **constraint** that $\sum_{i=1}^{k} y_i^* \geq z_j^*$, we have

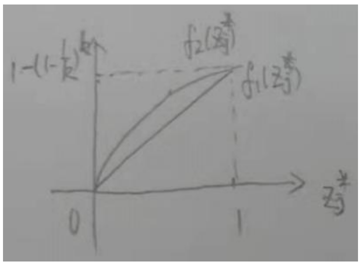
$$y_1^* = \cdots = y_k^* = \frac{z_j^*}{k}.$$

For the probability that C_j satisfied, we further have

$$\begin{aligned} P(C_j \text{ is satisfied}) &= 1 - \prod_{i=1}^{k} (1 - y_i^*) \\ &\geq 1 - (1 - \frac{z_j^*}{k})^k \end{aligned}.$$

Treat k as a fixe value. Consider the functions w.r.t. z_j^* that $y = f_1(z_j^*) = [1 - (1 - \frac{1}{k})^k] \cdot z_j^*$ and $y = f_2(z_j^*) = 1 - (1 - \frac{z_j^*}{k})^k$. Obviously, $y = f_1(z_j^*)$ and $y = f_2(z_j^*)$ are both **monotone increasing**.

Note that we have the constraint $0 \leq z_j^* \leq 1$. Consider the range of [0, 1]. Since the 2nd derivative of $y = f_2(z_j^*)$ is negative, we have the following visualization result:



In the visualization result, we have

$$1 - (1 - \frac{z_j^*}{k})^k \geq [1 - (1 - \frac{1}{k})^k] \cdot z_j^*,$$

for $0 \leq z_j^* \leq 1$. Thus, for the probability that C_j satisfied, we further have

$$\begin{aligned} P(C_j \text{ is satisfied}) &\geq 1 - (1 - \frac{z_j^*}{k})^k \\ &\geq [1 - (1 - \frac{1}{k})^k] \cdot z_j^* \\ &\geq (1 - \frac{1}{e}) z_j^* \end{aligned},$$

where we use the fact that $\lim_{k\to\infty}(1-1/k)^k = 1/e$. Let S be the number of clauses satisfied given by the **Algorithm 2**. Let $OPT_{\text{SAT}}(I)$ be the **maximum** number of clauses satisfied. Then, we have $OPT_{\text{SAT}}(I) \le \sum_{j=1}^{m} z_j^*$. For the **expected** number of clauses satisfied, we have

$$\begin{aligned}
\mathrm{E}[S] &= \sum_{j=1}^{m}[1\cdot P(C_j \text{ is satisfied})] \\
&\ge \sum_{j=1}^{m}(1-\frac{1}{e})z_j^* \\
&= (1-\frac{1}{e})\sum_{j=1}^{m} z_j^* \\
&\ge (1-\frac{1}{e})OPT_{\text{SAT}}(I)
\end{aligned}.$$

In summary, the **approximation ratio** of **Algorithm 2** is

$$\frac{\mathrm{E}[S]}{OPT_{\text{SAT}}(I)} \ge \frac{(1-1/e)OPT_{\text{SAT}}(I)}{OPT_{\text{SAT}}(I)} = 1-\frac{1}{e}.$$

Notes: **Algorithm 1** does better for **longer clauses**, while **Algorithm 2** does better for **shorter clauses**. One can design a better approximation algorithm by combing both of the algorithms.

(**Algorithm 3**) (**Combined Algorithm**)

(1) Run both **Algorithm 1** and **Algorithm 2**.

(2) Choose the better solution, i.e., with the larger number of clauses satisfied.

(**Theorem 3**) **Algorithm 3** is a randomized 3/4-approximation algorithm for **MAX-SAT**.

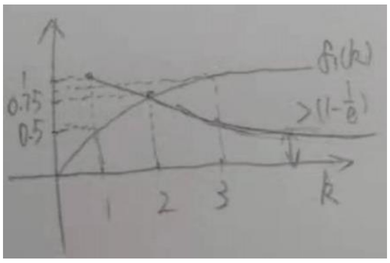
Proof of **Theorem 3**. Let X_1 and X_2 be the random variables denoting the value of solutions (i.e., number of clauses satisfied) returned by **Algorithm 1** and **Algorithm 2**, respectively. Let l_j be the length of a clause C_j.

For the expected number of clauses satisfied of **Algorithm 3**, we have

$$\begin{aligned}
\mathrm{E}[\max\{X_1, X_2\}] &\ge \mathrm{E}[(X_1+X_2)/2] \\
&= \frac{1}{2}\mathrm{E}[X_1]+\frac{1}{2}\mathrm{E}[X_2] \\
&\ge \frac{1}{2}\sum_{j=1}^{m}(1-(\frac{1}{2})^{l_j}) + \frac{1}{2}\sum_{j=1}^{m}[1-(1-\frac{1}{l_j})^{l_j}]\cdot z_j^* \\
&\ge \frac{1}{2}\sum_{j=1}^{m}(1-(\frac{1}{2})^{l_j})\cdot z_j^* + \frac{1}{2}\sum_{j=1}^{m}[1-(1-\frac{1}{l_j})^{l_j}]\cdot z_j^* \\
&= \sum_{j=1}^{m} z_j^* \cdot \frac{1}{2}[(1-\frac{1}{2^{l_j}}) + (1-(1-\frac{1}{l_j})^{l_j})]
\end{aligned}.$$

Assume all the clauses have the same length k. Consider the following 2 functions w.r.t. k:

$$y = f_1(k) = 1-(\frac{1}{2})^k \quad \text{and} \quad y = f_2(k) = 1-(1-\frac{1}{k})^k:$$



For k=1,

$$\frac{1}{2}[f_1(k)+f_2(k)]=\frac{1}{2}(\frac{1}{2}+1)=\frac{3}{4}\cdot$$

For k=2,

$$\frac{1}{2}[f_1(k)+f_2(k)]=\frac{1}{2}(\frac{3}{4}+\frac{3}{4})=\frac{3}{4}\cdot$$

For $k\geq 3$,

$$\frac{1}{2}[f_1(k)+f_2(k)]\geq\frac{1}{2}(\frac{7}{8}+1-\frac{1}{e})\geq\frac{3}{4}\cdot$$

Hence, we have $(f_1(k)+f_2(k))/2\geq 3/4$, for any $k\geq 1$. For the expected number of clauses satisfied, we further have

$$\begin{aligned}\mathrm{E}[\max\{X_1,X_2\}] &\geq \sum_{j=1}^{m} z_j^*\cdot\frac{1}{2}[f_1(l_j)+f_2(l_j)]\\ &\geq \sum_{j=1}^{m}\frac{3}{4}z_j^*\\ &\geq \frac{3}{4}OPT_{\mathrm{SAT}}(I)\end{aligned}.$$

In summary, the **approximation ratio** of **Algorithm 3** is

$$\frac{\mathrm{E}[\max\{X_1,X_2\}]}{OPT_{\mathrm{SAT}}(I)}\geq\frac{3OPT_{\mathrm{SAT}}(I)/4}{OPT_{\mathrm{SAT}}(I)}=\frac{3}{4}\cdot$$

(**Example 3**) (**MAX-2SAT Problem**) Consider the MAX-2SAT Problem, where each clause must have 1 or 2 literals and we need to find the assignment to satisfy the maximum number of clauses.

For **Algorithm 1**, suppose there're exactly 2 literals in a clause C_j, where we have

$$\begin{aligned}P(C_j\text{ is satisfied}) &= 1-P(\text{both the 2 literals are }\textit{false})\\ &= 1-(\frac{1}{2})^2=\frac{3}{4}\end{aligned}.$$

Suppose there is exactly 1 literal in the clause C_j. We further have

$$\begin{aligned}P(C_j\text{ is satisfied}) &= 1-P(\text{the single literal is }\textit{false})\\ &= 1-\frac{1}{2}=\frac{1}{2}\end{aligned}.$$

In summary, for the **MAX-2SAT** Problem, <u>the probability that a clause C_j is satisfied for **Algorithm 1**</u> should have the following property:

$$P(C_j\text{ is satisfied})\geq\frac{1}{2}\cdot$$

Hence, <u>the **approximation ratio** of **Algorithm 1** for **MAX-2SAT** is 1/2</u>.

Similarly, for **Algorithm 2**, the probability that a clause C_j is satisfied should have the following property:

$$\begin{aligned} P(C_j \text{ is satisfied}) &\geq [1-(1-\frac{1}{k})^k]\cdot z_j^* \\ &\geq \frac{3}{4}\cdot z_j^* \end{aligned}.$$

Hence, the **approximation ratio** of **Algorithm 2** for **MAX-2SAT** is 3/4.

(**Definition 4**) (**Vector Programming** of **MAX-SAT**) For each Boolean variable x_i in a Boolean formula, introduce a variable $y_i \in \{-1,+1\}$. Also, introduce an extra variable $y_0 \in \{-1,+1\}$. Let $y_i=y_0$ when variable x_i is *true* and $y_i \neq y_0$ when x_i is *false*. Then, we have $y_iy_0=1$ when x_i is *ture* and $y_iy_0=-1$ when x_i is *false*.

Consider a **clause** with only one **literal** x_i. Define the **value** of x_i as

$$v(x_i) = (1+y_iy_0)/2,$$

where $v(x_i)=1$ if the clause is satisfied (i.e., x_i is *true*) and $v(x_i)$ if the clause is not satisfied (i.e., x_i is *false*). Similarly, consider a **clause** with only one **literal** $\bar{x}_i$. Define the value of $\bar{x}_i$ as

$$v(\bar{x}_i) = (1-y_iy_0)/2.$$

Furthermore, consider a **clause** with exactly two **literals** $x_i \vee x_j$. The value of $x_i \vee x_j$ is

$$\begin{aligned} v(x_i \vee x_j) &= 1-v(\bar{x}_i)\cdot v(\bar{x}_j) \\ &= 1-(1-y_iy_0)(1-y_jy_0)/4 \\ &= 1-(1-y_iy_0-y_jy_0+y_iy_jy_0^2)/4 \\ &= (3+y_iy_0+y_jy_0-y_iy_j)/4 \\ &= [(1+y_iy_0)+(1+y_jy_0)+(1-y_iy_j)]/4 \end{aligned}.$$

Similarly, we also have the values of $\bar{x}_i \vee x_j$, $x_i \vee \bar{x}_j$, and $\bar{x}_i \vee \bar{x}_j$:

$$v(\bar{x}_i \vee x_j) = 1-v(x_i)v(\bar{x}_j) = [(1-y_iy_0)+(1+y_jy_0)+(1+y_iy_j)]/4,$$

$$v(x_i \vee \bar{x}_j) = 1-v(\bar{x}_i)v(x_j) = [(1+y_iy_0)+(1-y_jy_0)+(1+y_iy_j)]/4,$$

$$v(x_i \vee x_j) = 1-v(\bar{x}_i)v(\bar{x}_j) = [(1-y_iy_0)+(1-y_jy_0)+(1-y_iy_j)]/4.$$

In general, the objective of the **MAX-SAT Problem** for an arbitrary Boolean formula can be reformulated as follow:

$$\begin{aligned} &\max \sum_{0\leq i<j\leq n} 2[a_{ij}\cdot\frac{(1+y_iy_j)}{2}+b_{ij}\cdot\frac{(1-y_iy_j)}{2}], \\ &\text{s.t. } y_i^2=1, \text{ for } 0\leq i\leq n \end{aligned}$$

where a_{ij} and b_{ij} are constants. We can relax the aforementioned ILP as the following **vector**

programming by replace each variable yi with an (n+1)-dimensional vector $\mathbf{v}_i$:

$$\max \sum_{0\le i<j\le n} 2[a_{ij}\frac{(1+\mathbf{v}_i\cdot\mathbf{v}_j)}{2}+b_{ij}\frac{(1-\mathbf{v}_i\cdot\mathbf{v}_j)}{2}].$$
$$\text{s.t. } \|\mathbf{v}_i\|=1, \text{ for } 0\le i\le n$$

(**Algorithm 4**) (**Randomized Rounding Algorithm** via **Vector Programming**)

(1) Solve the **vector program** of **MAX-SAT** (near) optimally.

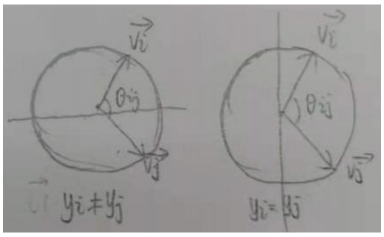
(2) Use the same approach as in **MAX-Cut** for the **randomized rounding**, i.e., randomly choose a hyper plane H, in which we let all variables y_i with vectors **above** H have the same value and all variables with vectors **below** H have the same value. Finally, we let each Boolean variable x_i be *true* if y_iy_0=1 and let x_i be *false*, otherwise.

(**Theorem 4**) The **approximation ratio** of **Algorithm 4** is 0.878.

Proof of **Theorem 4**. Let OPT_{SAT} be the **optimal value** of the **MAX-SAT** problem. Let $\{\mathbf{v}_1,\mathbf{v}_2,\cdots,\mathbf{v}_n\}$ be the optimal solution to the **vector programming**. Since vector programming is the relaxation of the **ILP** of **MAX-SAT**, we have

$$OPT_{\mathrm{SAT}}\le 2\sum_{0\le i<j\le n}[a_{ij}\frac{1+\mathbf{v}_i\cdot\mathbf{v}_j}{2}+b_{ij}\frac{1-\mathbf{v}_i\cdot\mathbf{v}_j}{2}]=OPT_{\mathrm{VP}},$$

where OPT_{VP} denotes the optimal value of the **vector programming**.



Without the loss of generality, consider two vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ (w.r.t. variables y_i and y_j). Let θ_{ij} be the angle between $\mathbf{v}_i$ and $\mathbf{v}_j$. The **probability** that $y_i\neq y_j$ (i.e., $\mathbf{v}_i$ and $\mathbf{v}_j$ are on two sides of the selected hyper-plane) is

$$P(y_i\neq y_j)=\frac{\theta_{ij}}{\pi}\ge 0.878\cdot\frac{1-\cos\theta_{ij}}{2}=0.878\cdot\frac{1-\mathbf{v}_i\cdot\mathbf{v}_j}{2},$$

where we used the property $\frac{\tau/\pi}{(1-\cos\tau)/2}\ge 0.878$ (for $0\le\theta_{ij}\le\pi$) proved in the analysis of the algorithm for **MAX-CUT**.

Further, for the probability that $y_i=y_j$ (i.e., $\mathbf{v}_i$ and $\mathbf{v}_j$ are on one side of the hyper-plane) is

$$P(y_i=y_j)=\frac{\pi-\theta_{ij}}{\pi}.$$

Let $\delta_{ij}=\pi-\theta_{ij}$. We also have $0\le\delta_{ij}\le\pi$, so we have

$$\frac{\delta_{ij}/\pi}{(1-\cos\delta_{ij})/2}=\frac{(\pi-\theta_{ij})/\pi}{(1-\cos(\pi-\theta_{ij}))/2}=\frac{(\pi-\theta_{ij})/\pi}{(1+\cos\theta_{ij})/2}\ge 0.878\cdot$$

Hence, we have

$$P(y_i=y_j)=\frac{\pi-\theta_{ij}}{\pi}\ge 0.878\cdot\frac{1+\cos\theta_{ij}}{2}=0.878\cdot\frac{1+\mathbf{v}_i\cdot\mathbf{v}_j}{2}.$$

Let S be the number of satisfied clauses in the given Boolean formula. The expected number of satisfied clauses is

$$\begin{aligned}
\mathrm{E}[S] &= 2\sum_{0\le i<j\le n}[a_{ij}P(y_i = y_j)/2 + b_{ij}P(y_i \ne y_j)/2] \\
&\ge 2\cdot 0.878\sum_{0\le i<j\le n}[a_{ij}\frac{1+\mathbf{v}_i\cdot\mathbf{v}_j}{2} + b_{ij}\frac{1-\mathbf{v}_i\cdot\mathbf{v}_j}{2}] \\
&= 0.878\cdot OPT_{\mathrm{VP}}
\end{aligned}.$$

In summary, the **approximation ratio** of **Algorithm 4** is

$$\frac{\mathrm{E}[S]}{OPT_{SAT}} \ge \frac{0.878\cdot OPT_{\mathrm{VP}}}{OPT_{\mathrm{VP}}} = 0.878\cdot$$

(**Definition 5**) (**Approximability Hierarchy**) Assume that $P{\ne}NP$, we have the following **Approximability Hierarchy** with 4 types of problems.

(1) **TSP** (without triangle inequality): There is no finite approximation possible.

(2) **Vertex Cover**, **TSP** (with triangle inequality), **MAX-CUT**, **MAX-SAT**: There is a limit to the approximation ratio achievable.

(3) **Subset Sum Problem**, **TSP** (with points in Euclidean space): There exist algorithm with approximation ratio arbitrarily close to 1, i.e., $1+\varepsilon$. The **running time** is within $poly(n)$ treating ε as fixed, e.g., $n^{O(1/\varepsilon)}$ and $n^5(1/\varepsilon)^2$ are polynomial time complexities.

(4) Problems like **Set Cover**: The approximation ratio is about $\log(n)$ and it's the best possible, e.g., approximation ratios of $\log\log n$ and $\log n/\log\log n$ are impossible.

Notes: One can reduce the **Hamiltonian Cycle Problem** to the problem of approximating **TSP**. Given a graph (not necessary to be complete), the **Hamiltonian Cycle Problem** aims to find a cycle that go through all the vertices exactly once, which is an **NP-Complete Problem**.

Notes: In TSP with points in Euclidean space, a complete graph with the Euclidean distance between each pair of vertices as the corresponding edge weight. Our goal is to find a TSP tour with minimum cost on the given complete graph.

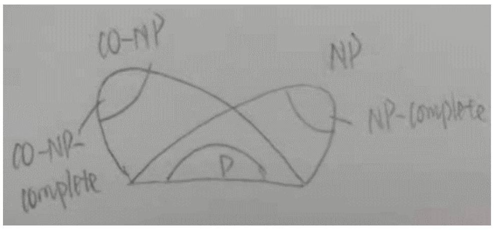
(**Definition 6**) (**Class** *NP*) Class of **decision problems**, for which there is a '**Yes-Certificate**'. More precisely, for **yes-instances**, there exists a **short proof/certificate** that the answer is yes.

(**Definition 7**) (**Class** *CO-NP*) Class of **decision problems**, for which there is a '**No-Certificate**'. More precisely, for **no-instances**, there exists a **short proof/certificate** that the answer is no.

(**Example 4**) (1) (**Unsatisfiability Problem**) Given a Boolean formula, is the formula unsatisfied? Such a **decision problem** is in *CO-NP*. Given an assignment that makes the Boolean formula satisfied, one can finish the **No-Certificate** in polynomial time by checking whether the given assignment can make the Boolean formula satisfied.

(2) (**TSP**) Given a complete graph with non-negative weights and an integer k, do all TSP tours have cost$>k$? Such a **decision problem** is in Class *CO-NP*. Given a TSP tour with cost$\le k$, one can finish the **No-Certificate** in polynomial time by checking whether the given TSP tour is valid and whether the cost is $\le k$.

(**Example 5**) The conjecture sketch of *P*, *NP*, and *CO-NP* is as follow:



We have $P \subseteq NP$, $P \subseteq CO\text{-}NP$, and $P \subseteq NP \cap CO\text{-}NP$. Is $P = NP \cap CO\text{-}NP$? No one knows the answer so far.

(**Definition 8**) Problems that have both **Yes-** and **No-Certificate** (i.e., lies in $NP \cap CO\text{-}NP$) are said to be **well-characterized** or said to have a **good characterization**.

Notes: Since the **MST problem** is in *P*, **MST** is well-characterized.

Notes: (**Decision Problem** of **Bipartite Matching**) Given a bipartite graph and an integer *k*, is there a **matching** with size at least *k*? Since there exist polynomial-time algorithm for the Bipartite Matching problem (i.e., **Bipartite Matching** is in *P*), it's well-characterized.

Notes: Once a problem has been shown to have a good characterization, this leads to a search for a **polynomial-time algorithm** for it. Many examples known where a problem is first found to have a non-trivial(非平凡) good characterization, but only years later, this problem is discovered to be in *P*, e.g., Linear Programming.

Notes: Problems that are well-characterized typically have an associated **Min-Max relation** (e.g., maximum matching and minimum vertex cover). The Min-Max relation is beautiful and powerful combinatorial result. Polynomial-time exact algorithms are designed around these Min-Max relations, e.g., special case of LP-Duality.

(**Example 6**) Assume that you don't know that **Bipartite Matching** is in *P*. Is **Bipartite Matching** well characterized?

Given a valid matching with size at least *k*, one can finish the **Yes-Certificate** in polynomial time by checking whether the given matching is valid and whether its size is at least k. Hence, Bipartite Matching is in *NP*.

Given a valid vertex cover with size less than *k*, one can finish the **No-Certificate** in polynomial time by checking whether the given vertex cover is valid and whether its size is less than *k*. Since there exist a vertex cover with size less than *k*, the size of **minimum vertex cover** must be less than *k*. According to the **Konig's Theorem**, the size of **maximum matching** equals the size of **minimum vertex cover** in a bipartite graph. The size of **maximum matching** must be less then *k*, so there's no matching with size at least *k*. Hence, Bipartite Matching is in *CO-NP*.

In summary, Bipartite Matching is well-characterized.

(**Example 7**) Let Π denote the following **decision problem** of **Max Flow**. Given a flow network, is there a feasible flow with value at least ($\geq$) *k*? Prove that Π is well-characterized without using the fact that $\Pi \in P$.

Given a valid flow with value at least *k*, one can finish the **Yes-Certificate** in polynomial time by checking whether the given flow is valid and whether its value is at least *k*. Hence, $\Pi \in NP$.

Given a valid *s-t* cut with value less than *k*, one can finish the **No-Certificate** in polynomial time by checking whether the given cut is valid and whether its value is less than *k*. Since there exist an *s-t* cut with value less than k, the value of **minimum cut** must be less than k. By the **Max-Flow Min-Cut Theorem**, the value of **maximum flow** equals the value of **minimum cut**, so the value of **maximum flow** must be less than k. The value of any feasible flow must be less than k, i.e., there's no feasible flow with value at least k. Hence, $\Pi \in CO\text{-}NP$.

In summary, Π is well-characterized.

(**Example 8**) Given a feasible and bounded **maximization LP**, is there a feasible solution whose value is at least k?

Treat the given maximization LP as the **Primal LP**, so its **Dual** is a minimization LP.

Given a feasible solution to the **Primal LP** with objective value at least k, one can finish the **Yes-Certificate** in polynomial time by checking whether the given primal solution is feasible and whether its objective value is at least k. Hence, **LP** is in *NP*.

Given a feasible solution to the **Dual minimization LP** with objective value less than k, one can finish the **No-Certificate** in polynomial time by checking whether the given dual solution is feasible and whether its objective value is less than k. Since there exist a feasible dual solution with objective value less than k, the **optimal dual solution** must have objective value less than k. By **Strong Duality**, the optimal dual value equals the optimal primal value, so the value of optimal primal solution must be less than k. Any feasible solution to the Primal LP must be less than k, i.e., there's no feasible solution with value at least k. Hence, **LP** is in *CO-NP*.

In summary, **LP** is well-characterized.

(**Example 9**) (**Factorization Problem**) Given an integer n, express the **prime factorization** of n, e.g., 24=2×2×2×3.

(**Factorization Decision Problem**) Given two integers x and y, does x have a **factor** less than (<) y and larger than (>) 1?

The **Factorization Problem** and **Factorization Decision Problem** are equivalent up to polynomial-time. If we can solve the **Factorization Decision Problem** in polynomial time, we can also solve the **Factorization Problem** in polynomial time.

Initially, let $x=n$ and $y=n/2$ for the Factorization Decision Problem and solve the problem using the polynomial-time algorithm. If the decision problem outputs yes, then we further set $y=n/4$ and we set $y=3n/4$, otherwise (like the **binary search**). Hence, for the given integer n, the input size of the **Factorization Problem** is $\log(n)$.

Factorization Problem (the decision problem) is well-characterized.

Given a prime factor $p<y$, one can finish the **Yes-Certificate** in polynomial time by checking whether $p>1$, $p<y$, and p is a prime factor of x.

Given a **prime factorization** $\{p_1, p_2, \cdots, p_k\}$ of x, one can finish the **No-Certificate** in polynomial time by checking that they are all primes and comparing them with y to make sure they are all at least y.

The **Factorization Decision Problem** is well-characterized (i.e., in $NP \cap CO\text{-}NP$), but it's not known whether the Factoring Problem is in *P* or not.

(**Fact 1**) The **Primality Problem** (i.e., determine whether a given integer n is a prime) is in *P*, which was proved in 2002.