1 Basic Concepts

Given a polynomial-time computable relation $R$, let $L_R$ be the language defined by

$$L_R := \{ x \mid \exists y, (x, y) \in R \}. $$

Then it holds that $L_R \in \text{NP}$. Next we consider a ‘harder’ problem: compute number of $y$’s satisfying $(x, y) \in R$.

**Definition 1.** Define the function $f_R(x) : \{0, 1\}^* \rightarrow \mathbb{N}$ as $f_R(x) = |\{y \mid (x, y) \in R\}|$. Let

$$\#R := \{(x, k) \mid f_R(x) \geq k\}.$$

**Proposition 1.** For all polynomial-time computable $R$, deciding $\#R$ and computing $f_R$ are Turing-reducible to one another.

**Proof.** The reduction from $f_R(x)$ to $\#R$ is obvious. Conversely, if we have the access to an oracle deciding whether $(x, k) \in \#R$, then $f_R(x)$ can be computed by performing a binary search on the value of $k$, costing $\text{poly}(x)$ time. $lacksquare$

**Definition 2.** $\#P$ is defined by the class of languages $L$ such that $L = \#R$ for some polynomial-time computable relation $R$. $L$ is $\#P$-complete if $L \in \#P$ and $R \leq_p L$ for all $R \in \#P$.

For every $\text{NP}$ language decided by NTM $M$, there is a ‘generalized’ problem in $\#P$ which computes the number of certificates that make $M$ accepting a given input $x$. Therefore,

**Fact.** $\text{NP} \leq \#P \subseteq \text{PSPACE}$.

**Definition 3.** We say $f$ is a parsimonious reduction from $\#Q$ to $\#R$, if it is polynomial-time computable and for all $x$, $f_Q(x) = f_R(f(x))$.

**Notation 1.** If $\#R$ is parsimoniously reducible from $Q$, we write $\#Q \leq_{\text{par}} \#R$.

If $f$ is a parsimonious reduction from $\#Q$ to $\#R$, then $L_Q \leq L_R$, since $x \in L_Q$ iff $f(x) \in L_R$. Conversely, if $\#Q \leq_{\text{par}} \#R$, then $(x, k) \in \#Q \iff (f(x), k) \in \#R$.

**Theorem 1.** $\#\text{SAT}$ is $\#P$-complete.
Conjecture. \( \#L \) is \( \#P \)-complete implies that \( L \) is \( \mathsf{NP} \)-complete.

Unfortunately, this conjecture is FALSE:

**Theorem 2.** There exists a polynomial-time computable relation \( R \) such that \( \#R \) is \( \#P \)-complete but \( L_R \in P \).

**Proof.** Define \( R \) as follows:

\[
(x, y') \in R \iff y' = 0 \lor (y' = 1y \land (x, y) \in R_{SAT}),
\]

where \((x, y) \in R_{SAT}\) iff the boolean formula \( \phi \) described by \( x \) is satisfied by assigning the values described by \( y \) to the variables in \( \phi \).

It is obvious that \( L_R \in P \) since \((x, 0) \in R\) for all \( x \in \{0, 1\}^* \), namely \( L_R = \{0, 1\}^* \). On the other hand, \( \#SAT \leq_{par} \#R \), since \((x, k) \in \#SAT \iff (x, k + 1) \in \#R \). Therefore \( \#R \) is \( \#P \)-complete.

**Definition 4.** Given an \( n \times n \) matrix \( A \), its permanent is defined by

\[
\text{perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i, \sigma(i)}.
\]

**Theorem 3** (Valiant). Computing permanent of 0-1 matrices is \( \#P \)-complete.

As shown in Figure 1, given an \( n \times n \) 0-1 matrix \( A \), a bipartite graph \( G(X,Y,E) \) can be built as follows: \( X = \{x_1, x_2, \ldots, x_n\} \), \( Y = \{y_1, y_2, \ldots, y_n\} \), \((x_i, y_j) \in E \iff A_{i,j} = 1\).

![Figure 1: Constructing bipartite graph for 0-1 matrix A](image)

Then it is easy to verify that the permanent of \( A \) equals the number of perfect matchings in \( G \). Therefore, counting the number of perfecting matchings in a bipartite graph is also \( \#P \)-complete.
2 Approximate Counting

Theorem 4. Given any polynomial $p$, there exists a PPT $A$ such that

$$\Pr \left[ \#\text{SAT}(\phi) \cdot \left( 1 - \frac{1}{p(n)} \right) \leq A^{\text{NP}}(\phi) \leq \#\text{SAT}(\phi) \cdot \left( 1 + \frac{1}{p(n)} \right) \right] \geq 1 - 2^{-n}. $$

Basic idea. For all $\phi$, if we can find a rough approximation $A'(\phi)$ such that

$$\#\text{SAT}(\phi) \cdot 2^{-i} \leq A'(\phi) \leq \#\text{SAT}(\phi) \cdot 2^i$$

for some constant $i$, then we are able to obtain a tighter approximation by:

1. construct $\phi'$ from $\phi$ such that $\#\text{SAT}(\phi') = \#\text{SAT}(\phi)k$ for some $k$;
2. output $A'(\phi')^{1/k}$.

Since

$$\#\text{SAT}(\phi)^k \cdot 2^{-i} = \#\text{SAT}(\phi') \cdot 2^{-i} \leq A'(\phi') \leq \#\text{SAT}(\phi') \cdot 2^i = \#\text{SAT}(\phi)^k \cdot 2^i,$$

it holds that

$$\#\text{SAT}(\phi) \cdot 2^{-i/k} \leq A'(\phi')^{1/k} \leq \#\text{SAT}(\phi) \cdot 2^{i/k}.$$  

For step (1), $\phi'$ can be constructed by

$$\phi' = \bigwedge_{i=1}^{k} \phi(\bar{x}_i),$$

where $\phi(\bar{x}_i)$ is a copy of $\phi$ with the variables renamed to $\bar{x}_i$.

Consider GAP-SAT:

$$\Pi_Y = \{ (\phi, k) \mid \#\text{SAT}(\phi) \geq 8k \};$$
$$\Pi_N = \{ (\phi, k) \mid \#\text{SAT}(\phi) \leq k/8 \}. $$

Claim. There exists a polynomial-time TM $A$ such that $A^{O}$ approximates $\#\text{SAT}$ within factor $8^{1.5}$ where $O$ is an oracle that solves GAP-SAT.

Proof. Let $A(\phi)$ work as follows:

1. $i \leftarrow 0$
2. while $O(\phi, 8^i) = 1$ do
3. $i \leftarrow i + 1$
4. end while
5. return $8^{i - \frac{1}{2}}$
After exiting the while loop, it holds that $O(\phi, 8^i) \neq 1$ and $O(\phi, 8^{i-1}) = 1$, which implies that $8^{i-2} < \#SAT(\phi) < 8^{i+1}$. Thus

$$8^{-1.5} < \frac{\#SAT(\phi)}{8^{i-\frac{1}{2}}} < 8^{1.5}.$$  

Next lecture we will show how to solve GAP-SAT (with the power of randomness).