

Lecture 2: April 5

Lecturer: Eric Vigoda

Scribe: Tom Hayes & Daniel Štefankovič

Today we will prove that computing the permanent is #P-complete. Let $\chi : \Sigma^* \times \Sigma^* \rightarrow \{0, 1\}$ be a predicate. We say that χ is an *NP-predicate* if the following two conditions hold:

1. $\chi(x, w)$ can be computed in time polynomial in $|x|$, and
2. there exists a polynomial p such that if $\chi(x, w) = 1$ then $|w| \leq p(|x|)$.

The language associated to χ is $L_\chi = \{x \mid \exists w; \chi(x, w) = 1\}$. A string w such that $\chi(x, w) = 1$ is called a *witness* of $x \in L$. The *counting function associated to χ* counts the number of witnesses.

$$f_\chi(x) = |\{w \mid \chi(x, w) = 1\}|.$$

The class #P is the set of functions associated to NP-predicates.

Exercise 2.1 Let $f, g \in \#P$. Let $\phi : \mathbb{N} \rightarrow \mathbb{N}$ where $\phi \in FP$. Show that $f + g$, fg and $\phi \circ f$ are in #P.

Let f, g be two functions. We say that $f \leq_K g$ (f is *many-one* or *Karp* reducible to g) if there are polynomial-time computable functions $\phi, \psi \in FP$ such that $f(x) = \psi(g(\phi(x)))$ for all x . We say that $f \leq_T g$ (f is *Turing* or *Cook* reducible to g) if f can be computed in polynomial time given an oracle for g .

We say that a function f is *#P-hard* if any function $f' \in \#P$ reduces to f . If moreover $f \in \#P$ then we say that f is *#P-complete*. We can define #P-hardness using either many-one or Turing reductions. We will show that the permanent is #P-complete under Turing reductions (for a proof of #P-completeness under many-one reductions see [Z91]).

Let ϕ be a reduction from a language L_χ to L_ν . If the ϕ preserves the number of witnesses (i.e. $f_\chi(x) = f_\nu(\phi(x))$) then we say that ϕ is *parsimonious*. The standard proof of NP-completeness of SAT is parsimonious and hence it also proves that #SAT is #P-complete.

Definition 2.2 Let A be an $n \times n$ matrix. The permanent of A is

$$\text{per } A = \sum_{\pi \in S_n} \prod_{i=1}^n a_{i, \pi(i)}.$$

The problem of computing the permanent of an integer matrix will be called #PERM. For matrices with entries 0 and 1 it will be called #(0,1)-PERM. The problem of counting the number of perfect matchings in a bipartite graph #BI-PER-MAT is clearly equivalent to #(0,1)-PERM. Similarly the problem of computing the total weight of perfect matchings (where weight of a matching is the product of the edges in the matching) of a weighted bipartite graph #W-BI-PER-MAT is equivalent to #PERM.

Theorem 2.3 (Valiant '79) #(0,1)-PERM is #P-complete (using Turing reductions).

We will prove Theorem 2.3 in a sequence of reductions in which we will reduce the following #P-complete problem to #(0,1)-PERM.

#EXACT 3-COVER:

INPUT: A finite set $X = \{1, \dots, n\}$ and a collection $Y \subseteq \binom{X}{3}$.

OUTPUT: The number of $Z \subseteq Y$ such that each $i \in X$ is covered exactly once by Z .

We will also use the following intermediate problem.

#W-BI-MAT (Weighted Bipartite Matching):

INPUT: bipartite graph G with integer edge weights

OUTPUT: the sum of weights of matchings of G

$$P_{match}(G) = \sum_{M \in \mathcal{M}} w(M) = \sum_{M \in \mathcal{M}} \prod_{e \in M} w(e),$$

where $\mathcal{M} = \mathcal{M}(G)$ denotes the set of matchings of G .

Lemma 2.4 $\#EXACT\ 3-COVER \leq_K \#W-BI-MAT$.

Proof: Consider the following gadget H . It is easily verified by inspection that the total weight of the matchings of the gadget H is $4(1 + x^3)$.

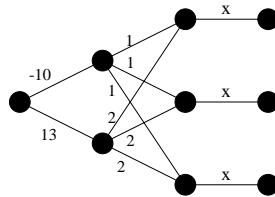


Figure 2.1: Gadget H .

Let $X = \{1, \dots, n\}$ and $Y \subseteq \binom{X}{3}$ be an instance of #EXACT 3-COVER. Let S be the value of #EXACT 3-COVER on this instance. Construct a graph G as follows. For each $i \in X$ there will be an edge $\{u_i, w_i\}$ of weight -1 . For each $A = \{a_1, a_2, a_3\} \in Y$ add a new copy H_A of the gadget H and identify vertex v_i of the gadget with the vertex u_{a_i} . We will show that the total weight of matchings in G with $x = 1$ is $4^{|Y|}S$.

Let $\mathcal{M} = \mathcal{M}(G)$ be the set of all matchings of G . Let $\mathcal{M}' \subseteq \mathcal{M}$ be the set of matchings of G which cover all the u_i but none of the w_i . The total weight of matchings in $\mathcal{M} \setminus \mathcal{M}'$ is 0, because there exists bijection $\phi : \mathcal{M} \setminus \mathcal{M}' \rightarrow \mathcal{M} \setminus \mathcal{M}'$ which switches the sign of the weight of the matching. The bijection ϕ is defined as follows. For $M \in \mathcal{M} \setminus \mathcal{M}'$ let i be the smallest index such that u_i is not covered or w_i is covered. Let $\phi : M \mapsto M \oplus \{u_i, w_i\}$.

Let B be the subgraph of G induced by the v_i and their neighbors. If we choose a partial matching m of B (which covers all the v_i) then the choices of edges in the H_A are independent. Hence

$$P_{match}(G) = \sum_{M \in \mathcal{M}'} w(M) = \sum_{M_B \in \mathcal{M}(B)} \prod_{A \in Y} \sum_{\substack{M_A \in \mathcal{M}(H_A) \\ M_A \cap B = M_B \cap H_A}} w(M_A).$$

The innermost sum is 4 if $|M_B \cap H_A| \in \{0, 3\}$ and 0 otherwise. Hence the value of the product is $4^{|Y|}$ if M_B corresponds to an exact 3-cover and 0 otherwise. ■

Lemma 2.5 $\#W\text{-BI-MAT}_{\leq T} \#PERM$.

Proof: Let G be a bipartite graph with parts A and B of size a and b . We will compute the sum of weights of matchings of size k in G for each $0 \leq k \leq \min(a, b)$ separately. To do so, we construct a new graph H , by adding $a - k$ new vertices, each adjacent to every vertex in A and no others, and adding $b - k$ new vertices, each adjacent to every vertex in B and no others. All the new edges are given weight 1.

Every matching of size k in G corresponds to $(a - k)!(b - k)!$ perfect matchings of the same weight in H . Hence it is enough to compute the permanent of the “bipartite” adjacency matrix of H and divide the result by $(a - k)!(b - k)!$. ■

Exercise 2.6 Show that $\#W\text{-BI-MAT}_{\leq K} \#PERM$. Hint: add a vertices A_0 adjacent to all vertices in A and b vertices B_0 adjacent to all vertices in B . Connect vertices in A_0 to vertices in B_0 with edges of large weight. Compute the permanent and from the resulting number extract the sum of weights of matchings of size k in G for each $0 \leq k \leq \min(a, b)$.

Lemma 2.7 $\#PERM_{\leq K} \#(0,1)\text{-PERM}$.

The proof of Lemma 2.7 involves gadgets which can simulate any integer weight w and have size $O(\log |w|)$. The gadgets have edges of weights $-1, 0, 1$. To get rid of the -1 the permanent is computed modulo a sufficiently large number N and the -1 are replaced by $N - 1$.

We will not prove the Lemma 2.7. Instead we will prove a weaker result which is sufficient for our purposes and has more interesting proof. Note that the matrix constructed during the reduction has only constantly many different entries. For matrices that have at most d different entries other than 0, 1 the problem of computing the permanent will be called $\#(0,1)\&d\text{-PERM}$. The following result implies $\#(0,1)\&d\text{-PERM}_{\leq T} \#(0,1)\text{-PERM}$ for any constant d .

Lemma 2.8 For $d \geq 1$ $\#(0,1)\&d\text{-PERM}_{\leq T} \#(0,1)\&(d-1)\text{-PERM}$.

Proof: First we show how to simulate small positive numbers. The following gadget has k matchings in which u, v are covered and 1 matching in which u, v are not covered. Hence if we replace an edge of weight

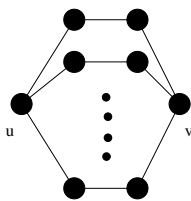


Figure 2.2: Gadget F_k .

k with F_k (which has only edges of weight 1) the total weight of perfect matchings is preserved.

Now suppose that α is a value which occurs in a matrix A . If we replace all occurrences of α by a variable x then $\text{Per } A$ is a polynomial $p(x)$ of degree at most n in x . We want to evaluate $p(\alpha)$. To do this we only need to evaluate p at $n + 1$ different places and use interpolation. We can evaluate p at point $k \in \{0, \dots, n\}$ using the gadget F_k and an oracle for $\#(0,1)\&(d-1)\text{-PERM}$. ■

References

- [V79] L. G. Valiant *The complexity of computing the permanent*, Theoretical Computer Science, 8, pp. 189-201, 1979.
- [Z91] V. Zankó *#P-completeness via many-one reductions*, International Journal of Foundations of Computer Science, 2, pp. 77-82, 1991.