JASS-2006 Proofs and computers

$\sharp P$ Complexity of the permanent An interactive proof for $P^{\sharp P}$ (Lecture note)

Dmitry Itsykson

June 24, 2006

Abstract

In this note we overview class $\sharp\mathbf{P}$. The main attention is devoted to the permanent computing problem. The proof of it's $\sharp P$-completeness (Valiant's theorem) is given. And finally we give an interactive protocol for $\mathbf{P}^{\sharp\mathbf{P}}$ with prover from the same class.

Contents

1 Introduction **3**
1.1 $\sharp$**P** definition 3

2 PERMANENT **3**
2.1 PERMANENT vs Perfect Matching 4
2.2 PERMANENT vs Cycle Cover 4
2.3 The $0-1$ PERMANENT is $\sharp P$ complete 4

3 Interactive proof **8**

1 Introduction

So far we have studied two related styles of problems: One asks whether a desired solution exists; the other requires that a solution be produced. But there is a third important, natural, and fundamentally different kind of problem: The one that asks how many solutions exist.

Consider the following examples:

- ♯**SAT**: Given a boolean expression, compute the number of different assignments that satisfy it
- ♯**Hamilton Path**: compute the number of Hamilton paths in given graph
- ♯**Clique**: compute the number of cliques of size k or larger

1.1 ♯P definition

Now we give formal definition of the ♯**P** language. It is similar to the **NP** definition.

Definition 1.1 ([Pap94]) *We now define a powerful class of functions called ♯**P** (pronounced "number **P**" or "sharp **P**," or even "pound **P**"). Let Q be a polynomially balanced, polynomial-time decidable binary relation. The counting problem associated with Q is the following: Given x, how many y are there such that $(x, y) \in Q$? The output required is an integer in binary, say ♯**P** is the class of all counting problems associated with polynomially balanced polynomial-time decidable relations.*

As usual with function problems, a reduction between two counting problems A and B consists of two parts: A part R mapping instances x of A to instances $R(x)$ of B, and a part S recovering from the answer N of $R(x)$ the answer $S(N)$ of x.

Theorem 1 *$\sharp SAT$, $\sharp 3 - SAT$ are ♯**P**-complete*

Proof (sketch) Reduction from Cook's theorem preserves the number of solutions. I.e. function R is from Cook's theorem, function $S = Id$. □

Using the same arguments we can conclude $\sharp P$ completeness of ♯**Hamilton Path** and ♯**Clique**.

2 PERMANENT

Let A be a matrix $n \times n$ under the Ring R. Recall definition of the determinant:

$$\det A = \sum_{\pi \in S_n} (-1)^{\sigma(\pi)} \prod_{i=1}^{n} A_{i,\pi(i)},$$

where S_n is a set of all permutations of n elements and $\sigma(\pi)$ is a parity of permutation π. *The permanent* of matrix A define by the following way:

$$perm(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} A_{i,\pi(i)}.$$

The main difference between this two definitions is (-1)s. However the determinant can be computed in polynomial time using Gauss elimination. And we show that permanent computing is at least **NP**-hard.

2.1 PERMANENT vs Perfect Matching

Consider the MATCHING problem. Although telling whether a bipartite graph has a perfect matching can be done in polynomial time ([CLR90]), computing the number of different perfect matchings in a bipartite graph is an important and notoriously difficult problem. Suppose that $G = (U, V, E)$ is a bipartite graph with $U = \{u_1, ..., u_n\}$ and $V = \{V_1, ..., V_n\}$, and $E \in UxV$. Consider the adjacency matrix A a of the graph, the $n \times n$ matrix whose (i, j)th element is 1 if $(u_i, v_j) \in E$, and 0 otherwise. The permanent of matrix A is

$$perm(A) = \sum_{\pi \in S_n} \prod_{i=1}^{n} A_{i,\pi(i)}$$

where the summation is over all perfect matchings of G. The permanent of A a is precisely the number of perfect matchings in G. This is why the problem of counting perfect matchings of bipartite graphs is known not as ♯MATCHING, but as PERMANENT. We shall see soon that it is a very hard problem.

Cororllary 2.1 *PERMANENT is in* $\sharp P$.

2.2 PERMANENT vs Cycle Cover

Let us consider another PERMANENT graph interpretation.

Consider directed graph $G = (V = \{v_1, v_2, \ldots, v_n\}, E)$. The set of disjoint cycles $\{C_1, C_2, \ldots, C_n)$ we call as *Cycle Cover* if it covers all vertices. Consider the $0-1$ matrix A $n \times n$ associated with G:

$$A_{i,j} = 1 \iff (v_i, v_j) \in E.$$

It is not hard to understand, that the $permA$ is the number of cycle covers in the graph G (cycles in permutations corresponds to cycles in G).

In case graph $G = (V, E, w)$ is a weighted graph, we define weight of Cycle Cover $\{C_1, C_2, \ldots, C_n)$ as a product of weights of all edges included in Cycle Covering. The matrix A associated with G have the following property: $A_{i,j} = w(v_i, v_j)$. In this case $permA$ is sum of all weights of cycle covering.

2.3 The $0-1$ PERMANENT is $\sharp P$ complete

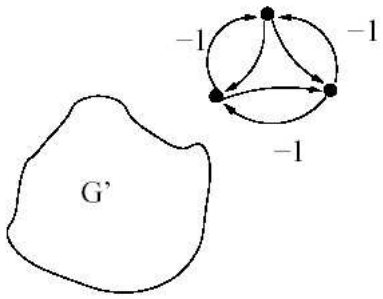

Figure 1: This graph has permanent 0

Example 2.2 *Consider the graph in Figure 1. (Unmarked edges have unit weight. We follow this convention through out this chapter.) Even without knowing what the subgraph G' is, we show that the permanent of the whole graph is* 0. *For each cycle cover in G' there are exactly two cycle covers for the three nodes, one with weight 1 and one with weight -1. Any non-zero weight cycle cover of the whole graph is composed of a cycle cover for G' and one of these two cycle covers. Thus the sum of the weights of all cycle covers of G is 0.*

Theorem 2 (Valiant) *0-1 PERMANENT is $\sharp\boldsymbol{P}$ complete.*

Proof([Aro06]) We shall reduce the $\sharp$**P**-complete problem $\sharp 3SAT$ to PERMANENT. Given a boolean formula ϕ with n variables and m clauses, first we shall show how to construct an integer matrix A' with negative entries such that $perm(A') = 4^{3m} \cdot (\sharp\phi)$. ($\sharp\phi$ stands for the number of satisfying assignments of ϕ). Later we shall show how to to get a $0-1$ matrix A from A' such that knowing $perm(A)$ allows us to compute $perm(A')$.

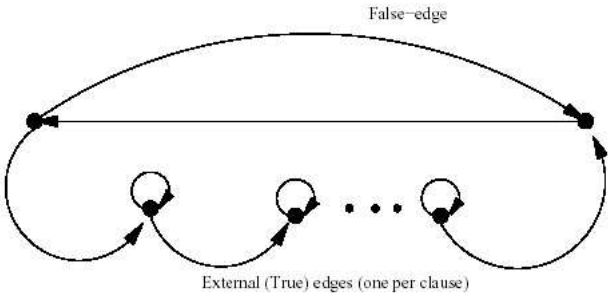

Figure 2: Variable gadget

The main idea is that there are two kinds of cycle covers in the digraph G' associated with A': those that correspond to satisfying assignments (we will make this precise) and those that don't. Recall that perm(A') is the sum of weights of all cycle covers of the associated digraph, where the weight of a cycle cover is the product of all edge weights in it. Since A' has negative entries, some of these cycle covers may have negative weight. Cycle covers of negative weight are crucial in the reduction, since they help cancel out contributions from cycle covers that do not correspond to satisfying assignments. (The reasoning to prove this will be similar to that in Example 2.2) On the other hand, each satisfying assignment contributes $4m$ to $perm(A')$, so $perm(A') = 4^{3m} \cdot (\sharp\phi)$.

To construct G' from ϕ, we use three kinds of gadgets as shown in Figures 2, 3 and 4.

There are two possible cycle covers of a variable gadget (fig. 2), corresponding to an assignment of 0 or 1 to that variable. Assigning 1 corresponds to a single cycle taking all the external edges ("true-edges", one true edge per clause containing variable without negation), and assigning 0 correspond to taking all the self-loops and taking the "false-edges" (one false edge per clause containing variable with negation)).

The clause gadget (fig. 3) is such that the only possible cycle covers exclude at least one external edge. Also for a given (proper) subset of external edges used there is a unique cycle cover (of weight 1). Each external edge is associated with a variable appearing in the clause. The clause gadget has exact 7 cycle covers. Each cycle cover corresponds to satisfying assignment of it, the falue of literal can be compute by the following rule: "cycle cover does not traverse my external" edge.

We will also use a graph called the XOR gadget (Figure 4) which has the following purpose: we want to ensure that exactly one of the edges uu' and vv' (see the schematic representation

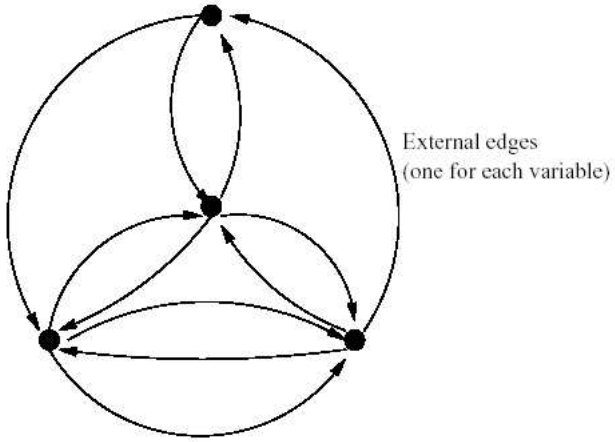

Figure 3: Clause gadget

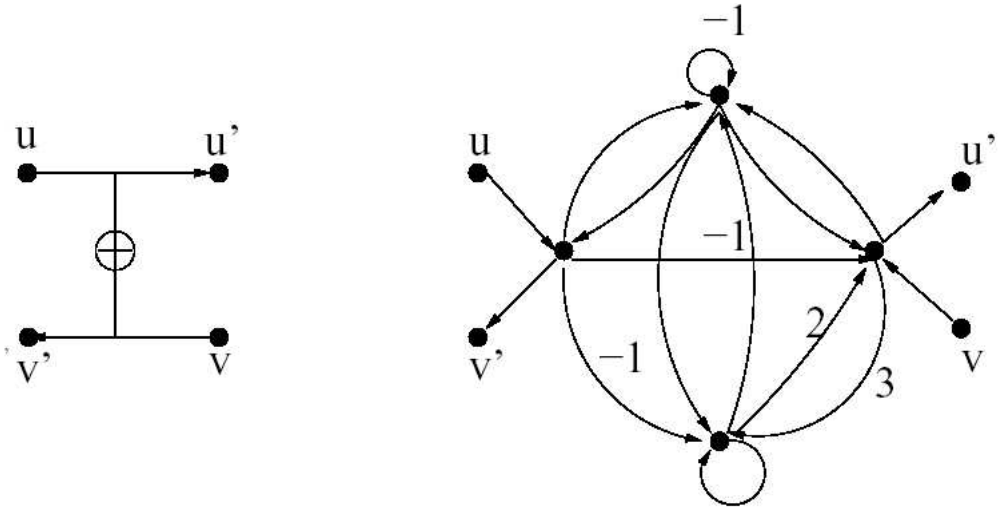

Figure 4: XOR gadget

in Figure 4) is used in a cycle cover that contributes to the total count. So after inserting the gadget, we want to count only those cycle covers which either enter the gadget at u and leave it at u' or enter it at v and leave it at v'. This is exactly what the gadget guarantees: one can check that the following cycle covers have total weight of 0: those that do not enter or leave the gadget; those that enter at u and leave at v', or those that enter at v and leave at u'. In other words, the only cycle covers that have a nonzero contribution are those that either (a) enter at u and leave at u' (which we refer to as using "edge" uu') or (b) enter at v and leave at v' (referred to as using edge vv'). These are cycle covers in the "schematic graph" (which has edges as shown in Figure 4) which respect the XOR gadget.

The XOR gadgets are used to connect the variable gadgets to the corresponding clause gadgets so that only cycle covers corresponding to a satisfying assignment need be counted towards the total number of cycle covers. Consider a clause, and a variable appearing in it. Each has an external edge corresponding to the other, connected by an XOR gadget (Figure general:construction). If the external edge in the clause is not taken (and XOR is respected)

the external edge in the variable must be taken (and the variable is true). Since at least one external edge of each clause gadget has to be omitted, each cycle cover respecting all the XOR gadgets corresponds to a satisfying assignment. (If the XOR is not respected, we need not count such a cycle cover as its weight will be cancelled by another cover, as we argued above). Conversely, for each satisfying assignment, there is a cycle cover (unique, in the schematic graph) which respects all the XOR gadgets.

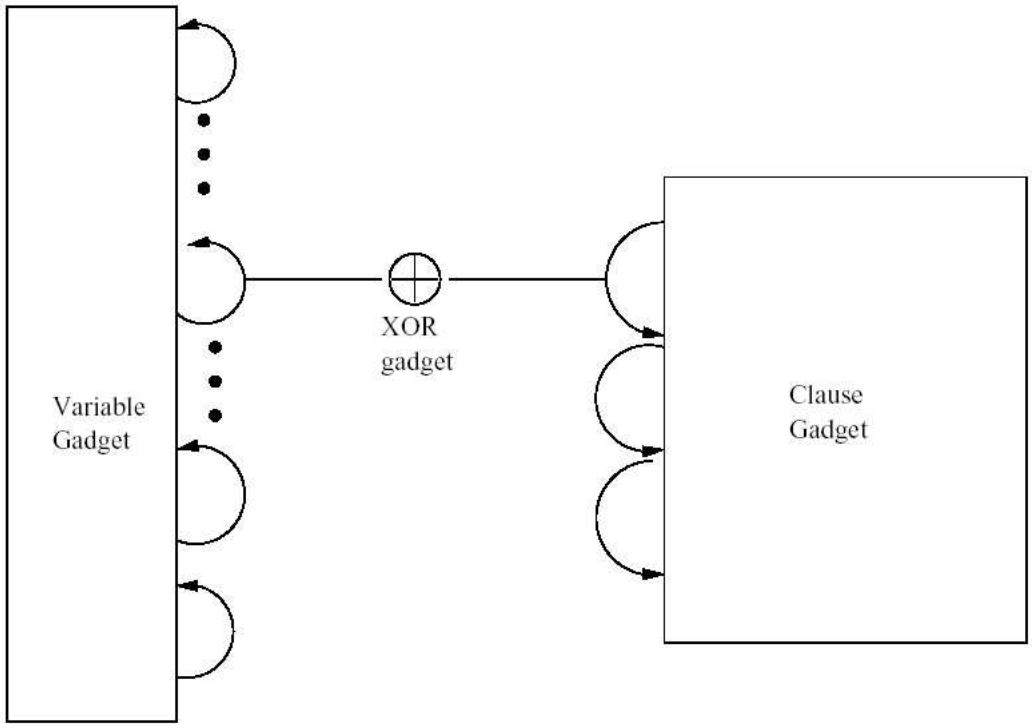

Figure 5: General construction

Now, consider a satisfying assignment and the corresponding cycle cover in the schematic graph. Passing (exactly one of) the external edges through the XOR gadget multiplies the weight of each such cover by 4. Since there are $3m$ XOR gadgets, corresponding to each satisfying assignment there are cycle covers with a total weight of 4^{3m} (and all other cycle covers total to 0). So $perm(G') = 4^{3m}$.

Finally we have to reduce finding $perm(G')$ to finding $perm(G)$, where G is an unweighted graph. Changing edges of (small) positive integral weights (i.e., multiple or parallel edges) to unweighted edges is as follows: cut each (repeated) edge into two and insert a node to connect them; add a self loop to the node (Figure 6 (a)-(b)). Then we reduce it to finding PERM(G) modulo $2^N + 1$ for a large enough N (but still polynomial in $|G'|$). For this, we can replace -1 edges with edges of weight 2^N, which can be converted to N edges of weight 2 in series (Figure 6 (c)). This does not change the permanent, and the new graph has only unweighted edges. □

Cororllary 2.3 (from the proof) *Integer PERMANENT modulo N (N is polynomially bounded in terms of size of the matrix) with polinomially bounded coefficients is in* $\sharp\boldsymbol{P}$.

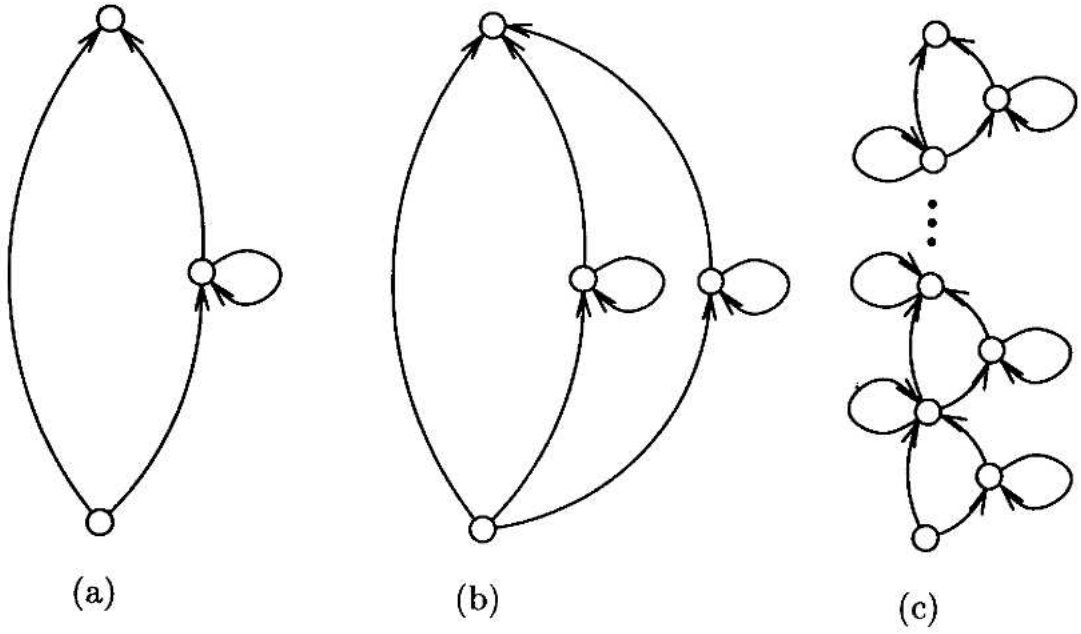

Figure 6: Simulating integer weights

3 Interactive proof

It is not hard to see that $\mathbf{P}^{\sharp\mathbf{P}} = \mathbf{P}^{PERMANENT} \subset \mathbf{PSPACE}$. Shamir's theorem (**IP**=**PSPACE**) states, that every language from **PSPACE** has Interactive proof with prover from **PSPACE**. We will prove that every language from $\mathbf{P}^{\sharp\mathbf{P}}$ has Interactive proof with prover from $\mathbf{P}^{\sharp\mathbf{P}}$ (smaller than **PSPACE**).

Theorem 3 *There exists interactive proof for language $\boldsymbol{P}^{\sharp\boldsymbol{P}}$ (with PERMANENET as an oracle) with prover from $\boldsymbol{P}^{\sharp\boldsymbol{P}}$.*

Proof([Hir04]) Consider language L from $\mathbf{P}^{\sharp\mathbf{P}}$. Let M be polynomial time Turing Machine with PERMANENT as an oracle, deciding L. The Verifier simulates M and uses Interactive Protocol for permanent computing. The Verifier asks to compute $perm(A)$ of $0-1$ matrix A $n \times n$, prover's answer is b. Instead of verifying statement $perm(A) = b$ the Verifier will verify statement $perm(A) \equiv b (\mathrm{mod} p_i)$ for polynomially many primes p_i.

Let $\mathbb{F}$ be the field $\mathbb{Z}_{p_i}$. All further evaluations will be done in $\mathbb{F}$.

Let's denote $(1,j)$th minor of matrix A as $A^{(j)}$. The following expression holds since it holds for the determinant:

$$perm(A) = A_{1,1} perm(A^{(1)}) + A_{1,2} perm(A^{(2)}) + \ldots + A_{1,n} perm(A^{(n)}).$$

The Verifier asks to compute $perm(A^{(1)}), perm(A^{(1)}), \ldots, perm(A^{(n)})$. The Prover answer is $b_1, b_2, \ldots, b_n$. The Verifier verifies the statement $b = A_{1,1}b_1 + A_{1,2}b_2 + \ldots + A_{n,n}b_n$.

If $perm(A) \neq b$, then exists i: $permA^{(i)} \neq b_i$.

The Verifiers has to verify the following list S of pairs: $S = (A^{(1)}, b_1), (A^{(2)}, b_2), \ldots, (A^{(n)}, b_n)$. The Verifier takes (C, d) and (E, f) from S a and asks Prover to compute polynomial: $perm(Cx + E(1-x))$ (this is polynomial of degree at most n and the honest Prover from $P^{\sharp}P$ is able to compute its coefficients using interpolation). The Prover answers the polynomial

$q(x)$. The Verifier verifies statements: $d = q(1)$ and $f = q(0)$ (therefore incorrectness of pair (C, d) (or (E, f)) implies incorrectness of $q(x)$)

The Verifier takes y from $\mathbb{F}$ at random, replace (C, d) and (E, f) by $(Cy + E(1-y), q(y))$ in S. If $perm(Cx + E(1-x))$ is not $q(x)$ then

$$Pr_y\{perm(Cx + E(1-x)) = q(y)\} \leq \frac{n}{|\mathbb{F}|},$$

since two different polynomials of degree n can npt have more then n common points.

Repeat last step $(n-1)$ times. S will contain only one pair (A_1, b'). We can estimate probability, that we "lost incorrectness" during our steps:

$$Pr\{perm(A_1) = b')\} \leq \frac{n^2}{|\mathbb{F}|}.$$

Matrix A_1 is $(n-1) \times (n-1)$. Repeat this procedure and we have matrix A_2 $(n-2) \times (n-2)$. After (n-1) steps we have A_{n-1} 1×1. And the Verifier should verify statement $perm(A_{n-1} = b^{(n-1)})$. It is possible to do in polynomial time. After $(n-1)$ steps we have $(n-1)$ times to "lost incorrectness, i.e.:

$$Pr\{perm(A_{n-1}) = b^{(n-1)})\} \leq \frac{n^3}{|\mathbb{F}|}.$$

So, we are to claim $|\mathbb{F}| \geq n^4$. □

References

[Aro06] S. Arora. Computational complexity: a modern approach. 2006.

[CLR90] T. Cormen, C.E. Leiserson, and R.L Rivest. Introduction to algorithms. 1990.

[Hir04] E.A. Hirsch. Computational complexity, lecture notes (in russian). 2004.

[Pap94] C.H. Papadimitriu. Computational complexity. 1994.