The Complexity of the Permanent and Related Problems

Tim Abbott and Alex Schwendner

May 9, 2007

Contents

1 The Permanent

In this paper, we discuss and present a proof of Valiant's result that the problem of computing the permanent of matrix with integer entries is #P-complete [Val79]. We proceed to discuss a number of related results to try to understand why computing the permanent is hard. The results discussed in this paper, unless specifically attributed otherwise, are due to Valiant.

1.1 Variations on the Determinant

This section presents an elementary result that we did not find in any published work. The result that computing the permanent is hard may seem surprising, given that the determinant is easily computed in polynomial time. But considering that the determinant is defined as a sum of $n!$ terms, perhaps the surprise is instead that the determinant is easy. The key difference between these important problems arises from the fact that there are no interesting operations that one can perform on a matrix that in general preserve the permanent. The determinant's sign function coefficient is precisely what one needs for adding a multiple of one row to another to preserve the determinant. Since this is the key operation used in the polynomial time algorithm for computing the determinant, one might then suspect that other functions that are defined similarly to the determinant would also be hard. Such a function would be of the form

$$\sum_{\sigma \in S_n} \prod_{i=1}^{n} f(\sigma) A_{i,\sigma(i)}$$

for some function $f : S_n \to \mathbb{C}^\times$, the multiplicative subgroup of $\mathbb{C}$. The permanent and determinant correspond to the function $f(\sigma) = 1$ and $f(\sigma) = \mathrm{sign}(\sigma)$. Because S_n is a finite group, all elements of the image of S_n under a group homomorphism f must have finite multiplicative order. Thus, the homomorphism in fact maps S_n into Φ_N, the Nth roots of unity, for some N. Since A_n is a simple group for all $n \geq 5$, it is the unique proper normal subgroup of S_n for $n \geq 5$. Thus, sign and the identity are the only homomorphisms from S_n into $\mathbb{C}^\times$. Consequently, the permanent and the determinant are the only functions from matrices into $\mathbb{C}$ that can be defined this way.

1.2 Graph Interpretation

The permanent has an interesting interpretation when we think of the matrix as the adjacency matrix of a weighted directed graph. Each of the $n!$ terms in the sum defining the permanent corresponds to a permutation. Since any permutation can be written as a union of disjoint cycles, in our graph representation this corresponds to a union of directed cycles covering all the vertices of the graph. The term in the permanent from this permutation and is equal to the product of the weights of the edge in the cycles.

Thus, if the matrix were a $\{0, 1\}$ matrix, then the permanent of the matrix counts the number of ways to partition the graph into a union of directed cycles. Valiant proved in his seminal 1979 paper that computing the permanent is #P-complete, even in this seemingly simpler case [Val79].

1.2.1 #(Hamiltonian Cycle)

The hardness of the permanent for $\{0, 1\}$ matrices may be more intuitive if one considers that the NP-complete problem Hamiltonian Cycle is the problem of finding a single cycle that covers all the vertices of a graph. Thus, computing the permanent of a $\{0, 1\}$ matrix is similar to the #P-complete problem of counting the number of Hamiltonian Cycles in a graph. The difference between #(Hamiltonian Cycle) and the permanent of an unweighted graph's adjacency matrix is that the sum defining the permanent includes terms for all $n!$ permutations, whereas that for #(Hamiltonian Cycle) sums over only the $(n-1)!$ n-cycles.

2 Main Proof Structure and Lemmas

2.1 Reduction to 3-CNF

Cook's Theorem provides a polynomial time reduction g mapping an arbitrary NP TM M and an input x to a propositional formula φ in 3-conjunctive normal form such that the there exists a satisfying assignment of φ if and only if there exists a valid accepting computation history for M on x. It is easy to see that a minor modification of Cook's reduction will also provide a 3-CNF formula φ with the number of satisfying assignments equal to the number of accepting computations of M on x. This implies that the problem #3-SAT of computing the number of such satisfying assignments of a 3-CNF formula φ is #P-hard. We shall provide a reduction from #3-SAT to the permanent, thereby showing that the permanent is #P-hard. This reduction is the content of Valiant's main lemma.

2.2 Main Lemma

There is a function f computable in polynomial time mapping propositional formula in conjunctive normal form to matrices with entries from $\{-1, 0, 1, 2, 3\}$ such that

$$\forall \varphi \quad \operatorname{Perm}(f(\varphi)) = 4^{t(\varphi)} \cdot s(\varphi)$$

where $t(\varphi)$ denotes "twice the number of occurrences of literals in φ, minus the number of clauses in φ", and $s(\varphi)$ is the number of assignments that satisfy φ.

Our approach in constructing this reduction is to use the graph interpretation of the permanent discussed in Section 1.2. We shall construct a graph such that each satisfying assignment of φ corresponds to a collection of cycle covers contributing $4^{t(\varphi)}$ to the permanent.

3 Graph Construction

Write $\varphi = C_1 \wedge C_1 \wedge \cdots \wedge C_l$ where $C_i = (y_{i1} \vee y_{i2} \vee y_{i3})$ for literals $y_{ij} \in \{x_k\} \cup \{\overline{x}_k\}$. We construct widgets to represent φ by constructing a *track* T_k for each variable x_k and an *interchange* R_i for each clause C_i. A track T_k and an interchange R_i will connect if one of the literals y_{ij} in C_i is either x_k or $\overline{x}_k$, and we shall connect them by with the construction of a *junction*. Each junction is a small fixed four-node subgraph which forces the track and the interchange to be connected correctly. Additionally, each interchange will have several *internal junctions*, each with the same structure as a junction.

Refer to Figure 1 for the structure of a track and an interchange as illustrated with an example.

3.1 Tracks & Interchanges

Refer to Figure 1 to see the structure of a track and an interchange. If a variable x_k is true in some assignment, then this assignment corresponds to a cycle cover in which the left path of the track is taken; it x_k is false, then it corresponds to one in which the right path is taken.

It is not difficult to verify that an assignment of variables satisfying a clause corresponds to a valid path through an interchange, and that if all of the literals in the clause are false, then no such valid path exists.

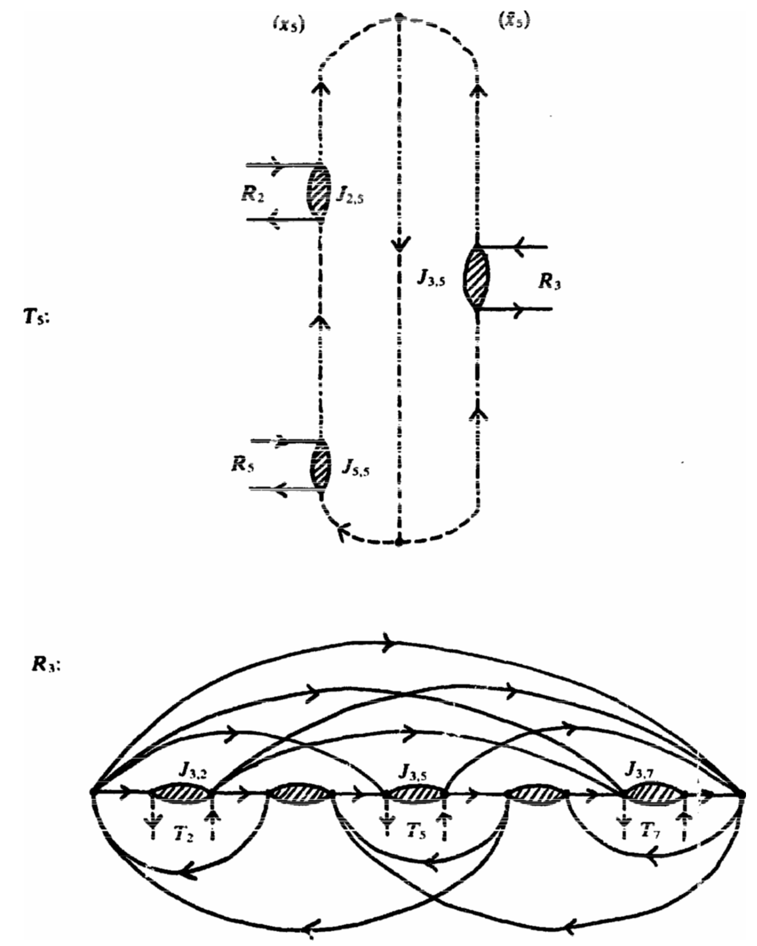

Figure 1: An example track T_5 for the variable x_5 and an interchange R_3 for a clause $C_3 = (x_2 \vee \overline{x}_5 \vee x_7)$. The dark ovals are junctions. [Figure copied from Valiant's paper.]

3.2 Junctions

The novel and key part of this construction is the structure of the junctions. The junctions and internal junctions are identical four-node weighted subgraphs represented by the following 4×4 matrix X:

$$X = \begin{pmatrix} 0 & 1 & -1 & -1 \\ 1 & -1 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 3 & 0 \end{pmatrix}$$

Labeling the nodes of a junction $1, 2, 3, 4$ — in the same order as the rows and columns of X — each junction is connected to the rest of the graph at nodes 1 and 4 only. Let $X(\Gamma; \Delta)$ denote the matrix X with rows in Γ and columns in Δ removed. The matrix X has the following properties:

1. $\operatorname{Perm} X = 0$
2. $\operatorname{Perm} X(1; 1) = 0$
3. $\operatorname{Perm} X(4; 4) = 0$
4. $\operatorname{Perm} X(1, 4; 1, 4) = 0$
5. $\operatorname{Perm} X(1; 4) = \operatorname{Perm} X(4; 1) = 4$

These properties are essential to the construction, and ensure that the cycle covers which enter each junction exactly once and leave at the opposite end contribute $4^{t(\varphi)}$ to the permanent, and the other cycle covers contribute zero. Property 1 implies that each junction is both entered and left. Properties 2 and 3 imply that the junction is entered and left at both ends. Property 4 implies that a cycle cover actually passes through the junction, instead of two cycle covers touching each end. Lastly, property 5 means that a proper cycle cover which enters each junction exactly once and leaves at the opposite end contributes $4^{t(\varphi)}$ as desired.

3.3 Comments on the Junctions

All of the edge weights in the tracks and interchanges are in $\{0, 1\}$. Only the junctions have edge weights of 2, 3, or -1. It would seem desirable to remove the other edge weights from our construction, especially since we would then not need to separately prove that computing the permanent of $\{0, 1\}$ matrices is #P-hard. However, if we could directly reduce #3-SAT to the permanent of a $\{0, 1\}$ matrix with some reduction g such that every nonzero contribution to $\operatorname{Perm} g(\varphi)$ corresponded with a distinct satisfying assignment of φ, then a uniquely satisfiable formula φ would have $\operatorname{Perm} g(\varphi) = \det g(\varphi) = 1$. Since computing a determinant is easy, this would imply that P = UP, which seems unlikely. In this proof, the junctions provide a factor of $4^{t(\varphi)}$ in $\operatorname{Perm} f(\varphi)$ which prevents $\det f(\varphi)$ from computing a #P-hard problem.

4 #P Hardness with entries in $\{0, 1\}$

Valiant's reduction that we have presented relied fundamentally on the use of negative edge weights. We will now prove Valiant's corollary that while negative edge weights are fundamental to Valiant's initial reduction, they are not fundamental to the complexity of the problem. Indeed, computing the permanent is #P-complete even in the unweighted ($\{0, 1\}$) case.

4.1 #P Hardness mod *some* small prime

If M is an integer matrix with no entry of absolute value greater than c, then $|perm(M)| \leq c^n n!$. Thus, to compute the permanent, it suffices to compute the permanent modulo some integer larger than $2c^n n!$, since there is then only one integer satisfying both the congruence and the inequality.

Let $p_1, \ldots, p_t$ be the primes less than or equal to $r = dn \log n$. Then if $\theta(x)$ is Chebyshev's theta function, by the Prime Number Theorem, $\prod_{i=1}^{r} p_i = 2^{\theta(r)} \geq 2^{d'r} \geq n^{ndd'} > 2c^n n!$ for a sufficiently large choice of the constant d. By the Chinese Remainder Theorem, we can compute the permanent modulo $\prod_{i=1}^{r} p_i$ by computing it modulo each prime p_i. Since there are only polynomially many p_i, this means that computing the permanent of an input matrix M modulo input prime $p < r = dn \log n$ is #P-hard.

4.2 #P Hardness with entries in $\{0, 1, \ldots, r = O(n \log n)\}$

The result we just proved readily allows us to reduce the problem of computing the permanent to that for a matrix that does contain any negative numbers. For, when computing the permanent of a matrix modulo p, one can replace each -1 with $p - 1$ without affecting the result modulo p, and compute the permanent of this new integer matrix with nonnegative entries, and reduce the result modulo p. It follows that computing the permanent on a matrix with entries in $\{0, 1, \ldots, p-1\}$ is #P-hard. Since $r \geq p$, it must be hard to compute the permanent of a matrix with entries in $\{0, 1, \ldots, r - 1\}$.

4.3 Transformation $\{0, 1, \ldots, r - 1\} \to \{0, 1\}$

In order to show that computing the permanent is hard for $\{0, 1\}$ matrices, it remains to show how to reduce an integer matrix whose entries fall in $\{0, 1, \ldots, r - 1\}$ to a $\{0, 1\}$ matrix without changing the value of the permanent.

Thinking of the permanent in the graph representation, we replace each edge (x, y) of weight $k > 1$ with a subgraph that can be covered by cycles in k different ways if (x, y) is used in the union of disjoint cycles, but in only one way otherwise. See Figure 2 (the figure for $k = 5$ from Valiant's paper). When (x, y) is used, the distinct cycle coverings correspond to the various choices of self-loops to include; while if it is not, the clockwise cycle is the only possible covering.

Every term in the sum defining the permanent either uses the edge (x, y) or does not. Those terms that use the edge are correctly counted k times because of the k difference choices for how to cycle cover the subgraph, while those that do not use the edge are counted precisely once, as desired. Thus, we have replaced an edge of weight k with $O(k)$ new vertices connected by edges of weight 1, while preserving the permanent of the corresponding matrix.

This transformation blows up the size of the matrix by a factor linear in the maximum entry in the matrix, which for our reduced case is at most r. Since $r = O(dn \log n)$, this transformation increases the matrix dimension to at most $O(n^2 \log n)$, while preserving the permanent. Thus, computing the permanent of a $\{0, 1\}$ matrix is #P-hard. Since computing the permanent of a $\{0, 1\}$ matrix is clearly in #P, we have shown the following theorem: Computing the permanent of a $\{0, 1\}$ matrix is #P-complete.

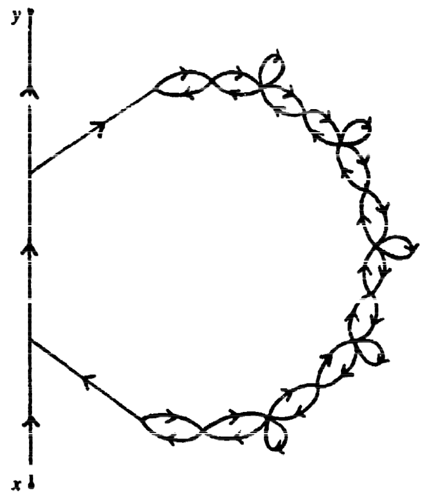

Figure 2: An example of the de-weighting transformation for k=5. If the from x to y is used, there are exactly k ways to cycle cover the new subgraph, giving effective weight k. If it is not used, there is only one way. [Figure copied from Valiant's paper.]

5 Hardness of Variants

5.1 UP hardness mod all $k \neq 2^k$

Suppose that $k \neq 2^k$. Define the class $\mathrm{Mod}_k\, P$ to be the class of decision problems solvable by an NP machine such that the number of accepting paths is divisible by k if and only if the answer is 'no'. Then the problem is computing the permanent modulo k is $\mathrm{Mod}_k\, P$-complete (this follows immediately from the #P-completeness of computing the permanent). It seems to be an open problem as to whether $\mathrm{Mod}_k\, P$ is as hard as #P [Bei91], [KM97]. This is interesting to contrast with the earlier result that computing the permanent modulo a prime less than $O(n \log n)$ is #P-hard (which does not tell us anything about the problem modulo any particular prime).

5.2 Low Rank Matrices

Many problems associated with matrices are simpler when the rank of the matrices involved is low. The determinant of a matrix of constant rank can be computed in constant time (since the answer is 0 for any n greater than that constant), while algorithms for the determinant in general require de-weighting time. As a less trivial example, the complexity of computing a Nash equilibrium in a general 2-player matrix game is known to be $PPAD$-complete [CD06]. However, when the payoff matrices have constant rank, there is a polynomial-time algorithm for computing a Nash equilibrium [LMM]. A similar result is true for computing the permanent of a matrix. There is a polynomial-time algorithm for computing the permanent for matrices of constant rank (but whose running time is exponential in the rank of the matrix) [Bar96].

5.3 Approximating the Permanent

In 2004, a FPRAS for the permanent was found for the case where all the entries in the matrix are nonnegative [JSV01]. This algorithms presents a certain sense in which the difficulty of computing the permanent may depend on the complexity of the numbers of the matrix: negative entries make it more difficult to approximate the permanent with good relative precision.

6 Conclusion

Our project focused on understanding why it is that the permanent is a hard problem.

References

[Val79] L. G. Valiant, The complexity of computing the permanent, Theoretical Computer Science, Volume 8, Issue 2, 1979, Pages 189-201.

[JSV01] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. ACM Symposium on the Theory of Computation 2001, pages 712-721.

[Bei91] R. Beigel. Relativized counting classes: relations among thresholds, parity and mods. Journal of Computer and System Sciences, 42(1):76–96, 1991.

[KM97] Grigory P. Kogan, Johann A. Makowsky. Computing Permanents over Fields of Characteristic 3: Where and Why it Becomes Difficult. IEEE Symposium on Foundations of Computer Science 1997.

[CD06] Chen, X. and Deng, X. 2006. Settling the Complexity of Two-Player Nash Equilibrium. In Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS '06) - Volume 00 (October 21 - 24, 2006). IEEE Computer Society, Washington, DC, 261-272.

[LMM] Richard J. Lipton, Evangelos Markakis, Aranyak Mehta. Playing Large Games Using Simple Strategies. Proceedings of the 4th ACM conference on Electronic commerce, p.36-41, June 09-12, 2003, San Diego, CA, USA

[Bar96] Barvinok, Alexander: Two Algorithmic Results for the Traveling Salesman Problem. Mathematics of Operations Research; February 1996, Vol. 21 Issue 1, p65-84, 20p