Theoretical Computer Science 8 (1979) 189–201.

THE COMPLEXITY OF COMPUTING THE PERMANENT

L.G. VALIANT
Computer Science Department, University of Edinburgh, Edinburgh EH9 3JZ, Scotland

Communicated by M.S. Paterson
Received October 1977

Abstract. It is shown that the permanent function of (0, 1)-matrices is a complete problem for the class of counting problems associated with nondeterministic polynomial time computations. Related counting problems are also considered. The reductions used are characterized by their nontrivial use of arithmetic.

1. Introduction

Let A be an $n \times n$ matrix. The *permanent* of A is defined as

$$\text{Perm } A = \sum_{\sigma} \prod_{i=1}^{n} A_{i,\sigma(i)}$$

where the summation is over the $n!$ permutations of $(1, 2, \ldots, n)$. It is the same as the *determinant* except that all the terms have positive sign. Despite this similarity, while there are efficient algorithms for computing the determinant all known methods for evaluating the permanent take exponential time. This discrepancy is annoyingly obvious even for small matrices, and has been noted repeatedly in the literature since the last century [15]. Several attempts have been made to determine whether the permanent could be reduced to the determinant via some simple matrix transformation. The results have always been negative, except in certain special cases [12, 13, 16].

The aim of this paper is to explain the apparent intractability of the permanent by showing that it is "complete" as far as counting problems. The results can be summarized informally as follows:

Theorem 1. *The complexity of computing the permanent of $n \times n$ (0, 1)-matrices is NP-hard* [3, 11] *and, in fact, of at least as great difficulty (to within a polynomial factor) as that of counting the number of accepting computations of any nondeterministic polynomial time Turing machine.*

Theorem 2. *For any integer K that is not an exact power of two the complexity of computing the permanent of a* $(0, 1)$*-matrix* mod K *is UP-hard* [24] (*i.e. a polynomial time algorithm for it would imply that any single-valued function that can be checked fast can also be evaluated fast*).

Theorem 3. *For any integer k the permanent of an integer matrix* mod 2^k *can be computed in* $O(n^{4k-3})$ *steps if* $k \geq 2$ (*and* $O(n^{2.81})$ *steps if* $k = 1$ *since the permanent and determinant are equal* mod 2).

To express Theorem 1 precisely we define the class $\#P$ of all problems computed by nondeterministic polynomial time Turing machines that have the additional facility of outputting the number of accepting computations. (N.B. This class is essentially equivalent to the polynomial time "probabilistic" TMs of Gill [4] and the "threshold" TMs of Simon [19]). For *NP*-complete problems counting the number of "solutions" (e.g. for satisfiability of propositional formulae we would count the number of satisfying assignments) is usually "$\#P$-complete" for trivial reasons. This is also the case for some polynomial computable problems that can be related to an *NP*-complete one in some direct fashion. The permanent function is a more surprising member of the $\#P$-complete class since the natural correspondence is with counting sets of distinct representatives (or equivalently perfect matchings in bipartite graphs) and for these the naturally related detection problems are polynomial time computable [5, 10].

On a more general level we establish a framework for classifying counting problems and define a hierarchy for this purpose. Examples of problems that one may attempt to classify are the graph-enumeration problems [6]. An example of the immediate consequences of our results is the following: If R is a polynomial computable predicate, then the number of labelled n-node graphs having property R can be expressed "explicitly" as the permanent of a matrix that can be computed in time polynomial in n. Unfortunately the best method known for evaluating the permanent of an $n \times n$ matrix takes $2^{n+O(\log n)}$ steps [17, p. 26].

We do not know of any pair of functions, other than the permanent and determinant, for which the explicit algebraic expressions are so similar, and yet the computational complexities are apparently so different. With this example we can understand better the difficulties that beset certain approaches to proving lower bounds on the complexity of unrestricted arithmetic (and Boolean) computations. An optimistic hope, that is fulfilled in certain restricted cases [18], is that lower bounds can be proved by assigning complexity measures to intermediate results according to syntactic criteria on their expressions. Clearly any such measure that could so distinguish P from NP would have to distinguish pairs of expressions that resemble each other as closely as the permanent and determinant.

2. The complexity of counting

For most non-deterministic algorithms each accepting computation corresponds in a natural way to a "solution" to the problem. It has been widely observed (e.g. [8, 19, 23]) that for almost all pairs of *NP*-complete problems there exist polynomial transformations between them that preserve the number of solutions. These problems are therefore equivalent not only as far as the existence of solutions but also as far as the problem of counting the solutions.

Definition 2.1. A *counting Turing machine* is a standard nondeterministic TM with an auxiliary output device that (magically) prints in binary notation on a special tape the number of accepting computations induced by the input. It has (worst-case) *time-complexity* $f(n)$ if the longest accepting computation induced by the set of all inputs of size n takes $f(n)$ steps (when the TM is regarded as a standard nondeterministic machine with no auxiliary device).

Definition 2.2. $\# P$ is the class of functions that can be computed by counting TMs of polynomial time complexity.

We denote the class of functions computed by deterministic polynomial time TMs by FP, and the class of predicates by P or DP. For convenience we shall often identify a class of machines with the class of functions it computes. It will be assumed that objects are represented in some standard economical manner as words over an alphabet Σ (say $\{0, 1\}$). $|x|$ will denote the size of x if x is a set, and its length if x is a string. A function $f: \Sigma^* \times \Sigma^* \to \Sigma^*$ (or a relation $R \subseteq \Sigma^* \times \Sigma^*$) is *polynomial bounded* iff there is a polynomial p such that for all x $|f(x)| < p(|x|)$ (or such that $R(x, y) \Rightarrow |y| < p(|x|)$).

The notion of reduction used is one by oracles, in a similar sense to Cook [3] except that the oracles cannot only be predicates but also arbitrary polynomial bounded functions. An oracle TM is a TM with a query tape, an answer tape, and some working tapes. To consult the oracle the TM prints a word on the query tape and, on going into a special query state an answer is returned in unit time on the answer tape, and a special answer state entered. An oracle TM is said to be in DP (or FP, or NP, or $\# P$, etc.) iff for all polynomial bounded oracles it behaves like a machine in P (or FP, or NP, or $\# P$, etc.).

If α is a class of oracle-TMs and x an appropriate function for it (i.e. polynomial bounded in the present context) then we denote the class of function (or predicates) that can be computed by oracle-TMs from α with oracles for x by α^x. A problem y is *$\# P$-hard* iff $\# P \subseteq FP^y$. It is *$\# P$-complete* iff $\# P \subseteq FP^y$ and $y \in \# P$.

The following notation is useful for unifying the necessary concepts with those from [3, 11, 14]. Let $\Gamma = \{N, \#, D, F\}^+\{P\}$. Each element of Γ will denote a complexity class defined inductively starting from DP, NP, FP or $\# P$: (i) If $\alpha \in \Gamma$

and $Z \in \{N, \#, D, F\}$, then $Z\alpha = \bigcup_{x \in \alpha} ZP^x$. (ii) If $\alpha \in \Gamma$ is a class of predicates, then co-α is the class of complements of elements of α.

Clearly any occurrence of D or F in $\alpha \in \Gamma$ is redundant except if it is the first letter. No other equivalences among the defined classes are known however. In the polynomial hierarchy [11, 14, 21] Σ_i^p, Π_i^p and Δ_i^p respectively become N^iP, co-N^iP and $DN^{i-1}P$. The $\#$ symbols define a potentially infinite tree-hierarchy of classes which collapses if $\# P \subseteq FP$. (N.B. $FP \subseteq \# P$). However, we know of only such trivial containments as the following which hold for all α.

$$\alpha \cup \text{co-}\alpha \subseteq D\alpha \subseteq N\alpha \cap \text{co-}N\alpha \subseteq N\alpha \cup \text{co-}N\alpha \subseteq D \# \alpha$$

In general a problem x is *α-hard via β* iff $\alpha \subseteq \beta^x$, and is *α-complete via β* iff in addition $x \in \alpha$. We assume throughout that β is P or FP as appropriate. In this paper we are concerned mainly with $\# P$-completeness. We shall also observe that there are natural problems complete in $\# NP$, and also that certain well-known problems occur lower in the N-$\#$-P hierarchy than is immediately apparent.

For NP-complete problems proving $\# P$-completeness for their natural counterparts is usually easy. Finding related polynomial time problems for which counting is still $\# P$-complete is also easy. As an artificial example consider a Boolean conjunctive normal form formula F. Finding an assignment that makes F false is trivial, yet counting them is $\# P$-complete. As a better example consider a monotone Boolean formula. Detecting the existence of a satisfying assignment is trivial. However, since any F can be rewritten, without changing the number of solutions as $G \wedge \neg H$ (where G, H are monotone), counting the solutions of $G \wedge H$ and of G would give the number of solutions of F. The rewriting consists of replacing each unnegated variable x_i in F by new variable y_i, and each negated variable $\bar{x}_i$ by z_i, and conjoining F with

$$\bigwedge_i (y_i \vee z_i) \wedge \neg \bigvee_i (y_i \wedge z_i).$$

A nondeterministic Turing machine is *unambiguous* [24] if for any input there is at most one accepting computation. UP is the class of predicates computed by unambiguous polynomial time TMs. The significance of the question $P = ?\ UP$ is that a positive answer would imply that the problems of checking and evaluating are polynomially related for all single-valued functions [24]. There are some problems (e.g. computing the prime decomposition of an integer) for which this is currently conjectured by some not to be true. Theorem 2 can be interpreted therefore either as giving evidence of intractability in a circumstance where NP-hardness is in doubt, or alternatively as a possible route to a powerful positive result. In particular, a fast algorithm for computing permanents mod 3 would imply that such schemes for cryptography as those proposed in [26] are nonexistent.

It is relevant to observe that among $\# P$-complete problems several finer distinctions can be made. A problem is *P-enumerable* if all the solutions can be listed

in time $p(n) \cdot N$ where N is the number of solutions found and $p(n)$ some polynomial in the input size. While matchings in bipartite graphs and satisfying assignments in monotone circuits are p-enumerable, NP-complete problems will not be in general unless $P = UP$. Another distinction involves defining $\#_K P$ to be $\# P$ but with arithmetic modulo K. Our results show that the permanent mod K is $\#_K P$-complete if K is not a power of 2, but is polynomial time computable otherwise.

3. Results and proofs

The main reduction used is the following:

Lemma 3.1. *There is a function $f \in FP$ from propositional formulae in conjunctive normal form to matrices with entries from $\{-1, 0, 1, 2, 3\}$ such that*

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

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

To prove $\# P$-hardness for the permanent of general integer matrices we need only the following additional fact.

Lemma 3.2. *There is a function $g \in FP$ that maps an arbitrary NP TM M and an input x for it to a propositional formula in 3-conjunctive normal form such that the number of satisfying assignments of $g(M, x)$ is equal to the number of accepting computations of M on x.*

Proof. By a modification of Cook's construction [3] using the idea of [2]. See [19] or [25].

To obtain results for (0, 1)-matrices we also need the following:

Lemma 3.3. *There is a transformation h, computable in time polynomial in m and the order of the matrix, that maps matrices with elements from the set $\{0, 1, \ldots, m\}$ to (0, 1)-matrices such that*

$$\forall A \quad \text{Perm } A = \text{Perm } h(A).$$

Before proving Lemmas 3.1 and 3.3 we observe that Theorems 1 and 2 follow from them.

Proposition 3.4. *For some positive constant d the problem of computing the permanent* mod r, *given an $n \times n$ (0, 1)-matrix and a positive integer $r < dn \log_2 n$, is $\# P$-hard.*

Proof. If C is an integer matrix in which no entry is larger than μ in magnitude then $|\text{Perm } C| \leq \mu^n \cdot n!$ To compute Perm C it is sufficient to compute its value mod p_i for each p_i in some set $\{p_1, \ldots, p_t\}$ of distinct prime numbers whose product exceeds $2\mu^n \cdot n!$ For some constant d' it is always sufficient that each $p_i < d'n(\log_2 \mu n)$ [7, p. 342]. But by Lemma 3.3, for each p_i, C mod p_i can be transformed in polynomial time into a (0, 1)-matrix with the same permanent. The result therefore follows from Lemmas 3.1 and 3.2.

Theorem 1. *Computing the permanent of a (0, 1)-matrix is $\# P$-complete.*

Proof. Proposition 3.4 implies that the problem is $\# P$-hard. That it also belongs to $\# P$ is immediate.

Theorem 2. *For any fixed positive integer K that is not an exact power of two, computing the permanent* mod K *of (0, 1)-matrices is UP-hard.*

Proof. From Lemmas 3.1 and 3.2 given any *UP* machine M and an input x the permanent of the matrix $f(g(M, x))$ will equal either $4^{t(F)}$ or zero according to whether M accepts x. Hence Perm $f(g(M, x))$ will be divisible by K if and only if M does not accept x.

Proof of Lemma 3.1. Any $n \times n$ matrix A can be regarded as the adjacency matrix of an n-node weighted directed graph G where A_{ij} gives the weight of the edge from node i to node j. Each additive term in Perm A corresponds to the product of the weights of the edges in some set of node-disjoint directed cycles that cover all the nodes of G (called "cycle covers" for short). To prove our result we exhibit a function f such that in the graph associated with $f(F)$ the cycle covers that correspond to satisfying assignments of F will each contribute $4^{t(F)}$ to the permanent, while the contributions of all the "spurious" cycle covers will cancel each other out.

Let $F = C_1 \wedge C_2 \wedge \cdots \wedge C_r$ where $C_i = (y_{i1} \vee y_{i2} \vee y_{i3})$ with $y_{ij} \in \{x_1, \bar{x}_1, x_2, \bar{x}_2, \ldots, x_m, \bar{x}_m\}$. (N.B. This assumption of 3-form is not essential.) We construct the graph $G = f(F)$ by superposing the following structures: a *track* T_k for each variable x_k, an *interchange* R_i for each clause C_i, and, for each literal $y_{i,j}$ such that $y_{i,j}$ is x_k or $\bar{x}_k$, a *junction* $J_{i,k}$ at which R_i and T_k meet. Interchanges also have *internal junctions* of the same structure as junctions. The construction of the tracks and interchanges is taken from a proof in [23] which itself is adapted from one in [9]. We describe them in Fig. 1 by an example fragment. T_5 and R_3 are shown for

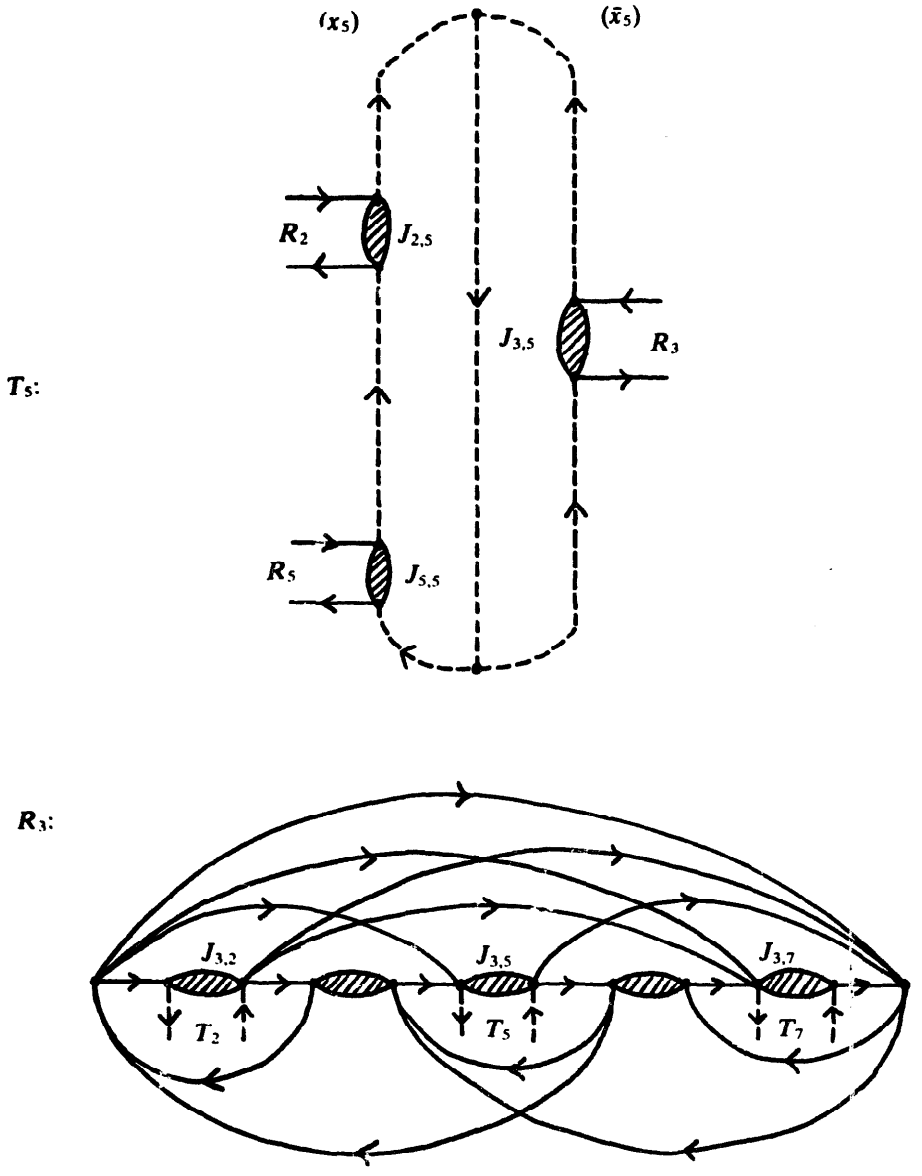


Fig. 1. A track and an interchange.

the case where $C_3 = (x_2 \vee \bar{x}_5 \vee x_7)$, and where x_5 occurs in C_2 and C_5, and $\bar{x}_5$ in C_3.

We assume that all the edges outside junctions or internal junctions are weighted one.

The crucial part of the construction is the structure of the junctions. The junctions and internal junctions are all identical four-node weighted digraphs corresponding to the following 4×4 matrix X.

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

Each one has external connections only via nodes 1 and 4 and not via 2 or 3.

Denoting by $X(\gamma;\delta)$ the matrix X with rows γ and columns δ removed, the following properties of X can be verified:

(i) Perm $X = 0$
(ii) Perm $X(1;1) = 0$
(iii) Perm $X(4;4) = 0$
(iv) Perm $X(1,4;1,4) = 0$
(v) Perm $X(1;4)$ = Perm $X(4;1)$ = nonzero constant (=4)

[N.B. The given X is about the simplest possible among all matrices with properties (i)–(v) if $\{1,4\}$ is taken to denote an arbitrary pair of indices, and any nonzero constant is allowed in (v). This can be seen from the following easily proved facts: (a) any such matrix has to be at least 4×4, and if it is 4×4 then (b) it is not symmetric and (c) at least two entries are greater than one in magnitude. Also (d) no matrix can have the same set of properties for the determinant, and hence if it has them for the permanent the constant in (v) must be even.]

Let a *route* in the graph $f(F)$ be the set of all cycle covers that have the same set of edges outside the junctions. A route is *good* if every junction and internal junction is entered exactly once the left exactly once and at the opposite end. Routes may fail to be good either because (a) some junction or internal junction is not entered and left, or (b) because it is entered and left at the same end, or (c) because it is entered twice and left twice. By virtue of conditions (i) for (a), (ii) and (iii) for (b) and (iv) for (c) any route that is not good contributes zero to the permanent. Condition (v) ensures that any good route contributes exactly $4^{t(F)}$.

It is clear that in any track T_k of any good route either all junctions on the left are "picked up" by the track and all the ones on the right by interchanges, or vice versa (corresponding to x_k and $\bar{x}_k$ respectively). The interchanges are so constructed that any route can pick up any subset of them, except for the whole set itself. Furthermore it can do so in exactly *one* way for each subset. Thus if for some R_i at least one of the junctions is picked up by the tracks, then all the remaining ones will be picked up by R_i in the unique good route available.

Using the obvious correspondence between good routes and assignments of truth values, we conclude that there is a one–one correspondence between good routes in the graph, each of which contributes $4^{t(F)}$ to the permanent, and satisfying assignment of F. The result follows.

Proof of Lemma 3.3. To obtain $h(A)$ we replace each edge of weight $k > 1$ in A by a subgraph. The subgraph is illustrated in Fig. 2 for the case $k = 5$. It replaces an edge of weight 5 from node x to node y. All the other nodes shown are new additions to A.

Now if (x, y) is not covered by a cycle in A then there is just one way to cover the corresponding new nodes in $h(A)$. On the other hand, if (x, y) is covered by a cycle in A, then so must be also the chain of these edges from x to y in $h(A)$. Then there

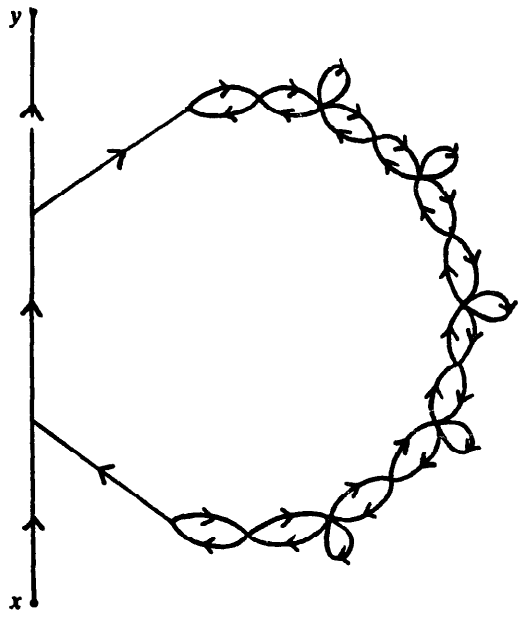


Fig. 2.

are five ways of covering the remainder, each corresponding to the inclusion of a different self-loop.

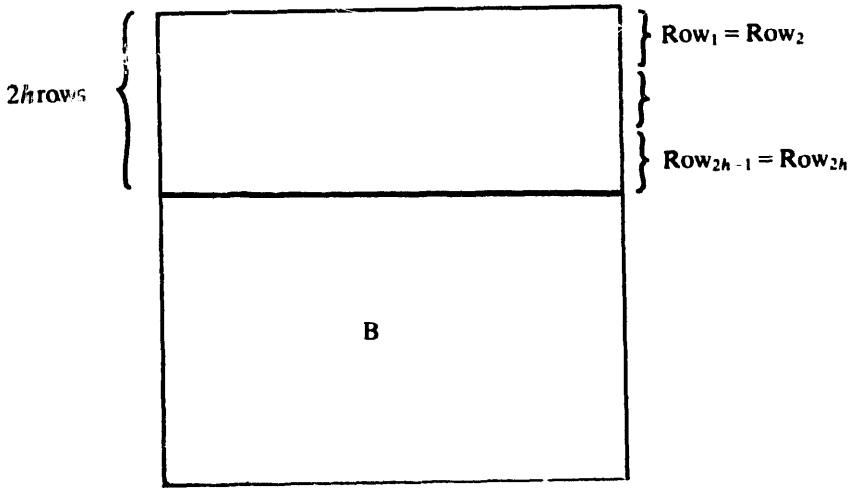
To obtain our main positive result we generalise the Gaussian elimination technique.

Theorem 3. *Let k be any positive integer and let $Q_k(n)$ be the number of bitwise operations required to evaluate the permanent* mod 2^k, *for $n \times n$ integer matrices. Then $Q_k(n) = O(n^{4k-3})$ for $k \geq 2$ and $Q_1(n) = O(n^{2.81})$.*

Proof. The proof is by induction on matrices of the two types shown in Fig. 3. $F_h(n)$ and $G_g(n)$ are the complexities of evaluating the permanent of the first and second types respectively. We shall assume arithmetic mod 2^k throughout.

To evaluate the permanent of a matrix of the second type we scan the columns from left to right for $i = 1, \ldots, n-g$. For each value of i, for every set of i of the first $i+g$ rows, we compute the permanent of the submatrix formed by these rows and the first i columns Since, for each i, there are fewer than n^g such submatrices, we have fewer than n^g values to compute at each stage. Each value at stage i is the sum of some of the values at stage $i-1$. Furthermore, each value at stage $i-1$ influences at most $g+1$ values at stage i (corresponding to the possible choices of rows omitted at stage $i-1$ but used at stage i). Hence the total complexity is $O(n^{g+1})$. The task is completed in the same fashion for the last g columns, giving $G_g(n) = O(n^{g+1})$.

A Matrix of the 1st Type



A Matrix of the 2nd Type

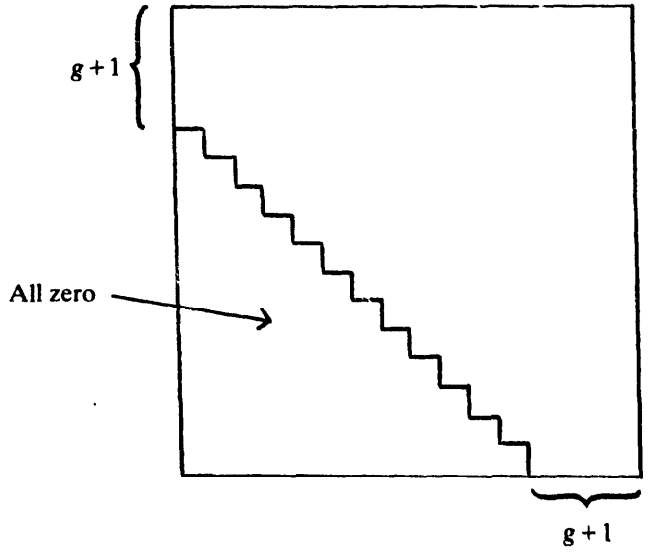


Fig. 3.

To evaluate the permanent of a matrix of the first kind we perform "elimination" on it to give a matrix of the second kind, plus n^2 matrices of the first kind with h increased by one.

The elimination is based on the fact that if by adding row i to row j in an integer matrix A we obtain A', then

$$\text{Perm } A' = \text{Perm } A + \text{Perm } A''$$

where A'' is the same as A but with row j replaced by row i. A'' must then have two equal rows and hence Perm $A'' = 0 (\mathrm{mod}\ 2)$. (Note that $F_k(n) = 0$ for a similar reason: the permanent of any matrix with k disjoint pairs of equal rows is a multiple of 2^k.)

We can therefore reduce the $(n-2h)\times n$ submatrix B (i.e. the bottom $n-2h$ rows) so that E_{ij} becomes zero for $i > j$ and hence the overall matrix becomes of the second type. The reduction consists of computing $O(n^2)$ linear combinations of pairs of rows in the manner of standard Gaussian elimination except that now we get an additive term of the first kind (with h increased by one) at each step. As pivot we always choose from among the elements of the relevant submatrix one that is odd, or if no odd element exists, then one that has the fewest factors of two. This ensures that we can work in arithmetic mod 2^k. Hence we have

$$F_h(n) = O(n^3) + O(n^2) F_{h+1}(n) + G_{2h}(n),$$

and

$$F_k(n) = 0.$$

Solving this gives $F_0(n) = O(n^{4k-3})$ for all $k \geq 2$. (The case $k = 1$ gives $O(n^3)$ Gaussian elimination, which can be improved to $O(n^{2.81})$ [22]).

4. Remarks

For many predicates in P the corresponding counting problem is in FP. This is evident, for example, for problems with a dynamic programming flavour. More interesting examples are spanning trees in graphs, and Eulerian paths in diagraphs, either of which can be counted by evaluating appropriate determinants [6].

For many other easily computed predicates the counting problem is $\# P$-complete. This is true for satisfiability even in the very restricted case of monotone CNF formulae with two disjuncts per conjunct. (A collection of such complete problems appears in [25].) Note that good proofs of $\# P$-completeness appear to be characterized by the necessity for nontrivial arithmetic.

Isomorphism of graphs has a seemingly intermediate position: R. Mathon has recently observed that counting isomorphic embeddings belongs to FNP, and hence that the full power of counting (i.e. $\# P$) is not apparently required.

The above result is relevant in classifying enumeration problems [6]: Let $x \in NP$ and consider the problem C_x: "given a graph G count the number of subgraphs with property x that are distinct under relabellings". Many natural enumeration problems are of this flavour for the special case that G is the complete graph.

Fact. $C_x \in F\#NP$.

Proof. Run a $\#P$ machine M of which each nondeterministic branch performs the following: guess a subgraph G', test it for x, and if this is true then run a nondeterministic computation with g accepting branches for

$$g = h^{-1} (\mathrm{mod}\, p)$$

where h is the number of automorphisms of G' and p is a computed prime number larger than the expected answer. Finding p and h can be done within *FNP* [27].

As a final remark we note that the following problem NSAT can be easily shown to be complete in $\#NP$: A propositional formula F can be regarded as being nondeterministic if its variables are partitioned into set X and Y. A *solution* to the NSAT problem is an assignment of values to the elements of X for which there exists an assignment of values to Y that together satisfy F. The counting problem for NSAT is that of counting the number of such solutions.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] M. Bauer, D. Brand, M. Fischer, A. Meyer and M. Paterson, A note on disjunctive form tautologies, *SIGACT News* **52** (1973) 17–20.

[3] S.A. Cook, The complexity of theorem proving procedures, *Proc. 3rd ACM Symp. on Theory of Computing* (1971) 151–158.

[4] J. Gill, Computational complexity of probabilistic Turing machines, *Proc. 6th ACM Symp. on Theory of Computing* (1974) 91–95.

[5] M. Hall Jr., An algorithm for distinct representatives, *Am. Math. Monthly* **63** (1956) 716–717.

[6] F. Harary and E.M. Palmer, *Graphical Enumeration* (Academic Press, New York, 1973).

[7] G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers* (Oxford University Press, 4th Ed., 1960).

[8] J. Hartmanis and L. Berman, On Isomorphisms and density of *NP* and other complete sets, *Proc. 8th ACM Symp. on Theory of Computing* (1976) 30–40.

[9] P.P. Herrmann, On reducibility among combinatorial problems, MAC TR-113, MIT (1973).

[10] J.E. Hopcroft and R.M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, *SIAM J. Comput.* **2** (1973) 225–231.

[11] R.M. Karp, Reducibility among combinatorial problems, in: R.E. Miller and J.W. Thatcher, Eds., *Complexity of Computer Computations* (Plenum Press, New York, 1972).

[12] C.H.C. Little, A characterization of convertible (0, 1)-matrices, *J. Combin. Theory* **18** (B) (1975) 187–208.

[13] M. Marcus and H. Minc, On the relation between the determinant and the permanent, *Illinois J. Math.* **5** (1961) 376–381.

[14] A.R. Meyer and L.J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, *Proc. 13th Symp. on Switching and Automata Theory* (1972) 125–129.

[15] T. Muir, On a class of permanent symmetric functions, *Proc. Roy. Soc. Edinburgh* **11** (1882) 409–418.

[16] G. Polya, Aufgabe 424, *Arch. Math. Phys.* (3) **20** (1913) 27.

[17] H.J. Ryser, *Combinatorial Mathematics*, Carus Math. Monograph no. 14 (1963).

[18] C.P. Schnorr, A lower bound on the number of additions in monotone computations, *Theor. Comput. Sci.* **2** (1976) 305–315.

[19] J. Simon, On some central problems in computational complexity, Ph.D. Thesis, TR75-224, Cornell Univ. (1975).
[20] J. Simon, On the difference between one and many, *Proc. 4th Colloq. on Automata, Languages and Programming*, Turku, 1977.
[21] L.J. Stockmeyer, The polynomial-time hierarchy, *Theoret. Comput. Sci.* **3** (1977) 1–22.
[22] V. Strassen, Gaussian elimination is not optimal, *Numer. Math.* **13** (1969) 354–356.
[23] L.G. Valiant, A polynomial reduction of satisfiability to Hamiltonian circuits that preserves the number of solutions, Manuscript, Univ. of Leeds (1974).
[24] L.G. Valiant, The relative complexity of checking and evaluating, *Information Processing Lett.* **5**(1) (1976) 20–23.
[25] L.G. Valiant, The complexity of enumeration and reliability problems, *SIAM J. Comput.* (to appear).
[26] W. Diffie and M.E. Hellman, New directions in cryptography, *IEEE. Trans Information Theory* (New York (Nov. 1976).
[27] V.R. Pratt, Every prime has a succint certificate, *SIAM J. Comput.* **4**(1975) 214–220.
[28] R.L. Rivest, A. Shamir and L. Adleman, On digitial signatures and public-key cryptosystems, TM 82, Lab. for Comput. Sci., MIT (April 1977).