Information Processing Letters 125 (2017) 20–25

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Computing the permanent modulo a prime power

Andreas Björklund [a], Thore Husfeldt [a,b,*], Isak Lyckberg [a]

[a] Lund University, Box 118, 22100 Lund, Sweden
[b] ITU Copenhagen, Rued Langgaards Vej 7, 2300 København S, Denmark

ARTICLE INFO

Article history:
Received 16 May 2016
Received in revised form 17 April 2017
Accepted 28 April 2017
Available online 4 May 2017
Communicated by Ł. Kowalik

Keywords:
Algorithms
Graph algorithms
Randomized algorithms

ABSTRACT

We show how to compute the permanent of an $n \times n$ integer matrix modulo p^k in time $n^{k+O(1)}$ if $p = 2$ and in time $2^n / \exp\{\Omega(\gamma^2 n/p \log p)\}$ if p is an odd prime with $kp < n$, where $\gamma = 1 - kp/n$. Our algorithms are based on Ryser's formula, a randomized algorithm of Bax and Franklin, and exponential-space tabulation.

Using the Chinese remainder theorem, we conclude that for each $\delta > 0$ we can compute the permanent of an $n \times n$ integer matrix in time $2^n / \exp\{\Omega(\delta^2 n/\beta^{1/(1-\delta)} \log \beta)\}$, provided there exists a real number β such that $|\text{per}\, A| \le \beta^n$ and $\beta \le (\frac{1}{44}\delta n)^{1-\delta}$.

© 2017 Elsevier B.V. All rights reserved.

1. Introduction

The permanent of an $n \times n$-matrix $A = (a_{ij})$ is defined as

$$\text{per}\, A = \sum_{\sigma} \prod_{i=1}^{n} a_{i,\sigma(i)} \tag{1}$$

where the sum is over all permutations σ of the elements $1, \dots, n$. From the definition, per A can be computed in $O(n!n)$ arithmetic operations. Using Ryser's classic formula [9], per A can be computed in $O(2^n n)$ arithmetic operations. More recently it was shown that when the entries of A are $n^{O(1)}$-bit integers, then per A can be computed in time $2^n / \exp\{\Omega(\sqrt{n/\log n})\}$ [4].

These exponential running times seem particularly disappointing when compared to the polynomial-time algorithms for computing the determinant. This discrepancy was famously explained by Valiant's seminal result [12], which showed that the permanent is hard for the complexity class #P.

We present here some improved algorithms for computing per A modulo a prime power.

2. Results

Theorem 1. *Given an $n \times n$ integer matrix A and a positive integer k, the value* per A mod 2^k *can be computed in time $n^{k+O(1)}$ and $n^{O(1)}$ space.*

This result is established by Algorithm A in Section 4. This improves Valiant's algorithm [12] for per A mod 2^k, which runs in time $O(n^{4k-3})$. It is crucial here that computation is performed modulo a power of 2: There is little hope of finding, say, an algorithm for per A mod 3^k in time $n^{O(k)}$, since already the computation of per A mod 3 requires time $\exp(\Omega(n))$ under the randomized exponential time hypothesis [6].

Instead, for larger primes $p > 2$, we present an algorithm for per A mod p^k with running time $O\big((2 - \epsilon_p)^n\big)$, where ϵ_p is positive and depends on p:

* Corresponding author at: Lund University, Box 118, 22100 Lund, Sweden.
E-mail address: thore.husfeldt@cs.lth.se (T. Husfeldt).

http://dx.doi.org/10.1016/j.ipl.2017.04.015
0020-0190/© 2017 Elsevier B.V. All rights reserved.

Theorem 2. *Given an $n \times n$ integer matrix A, a positive integer k, and a prime p such that $kp < n$, the value* $\operatorname{per} A \bmod p^k$ *can be computed in time within a polynomial factor of*

$$2^n \Big/ \exp\left\{\Omega(\gamma^2 n/p \log p)\right\},$$

where $\gamma = 1 - kp/n$.

In a setting where the product kp can be bounded away from n, say $kp \le \frac{99}{100}n$ for n sufficiently large, the term γ^2 can be absorbed in the Ω notation for a cleaner bound. This result is established by Algorithm B in Section 5.

Theorem 3 can be applied to permanents whose value is known to be small:

Theorem 3. *Given $\delta > 0$, an $n \times n$ matrix A of integers, and a real number $\beta \le (\frac{1}{44}\delta n)^{1-\delta}$ such that $|\operatorname{per} A| \le \beta^n$, the value* $\operatorname{per} A$ *can be computed in time within a polynomial factor of*

$$2^n \Big/ \exp\left\{\Omega(\delta^2 n/\beta^{1/(1-\delta)} \log \beta)\right\}.$$

In particular, if β is a constant, the bound can be given as $O\big((2-\epsilon_\beta)^n\big)$. This result is established by Algorithm C in Section 6.

An interesting special case is when the entries of A are restricted to $\{0, 1\}$. Then the permanent equals the number of perfect matchings in the bipartite graph whose biadjacency matrix is A. For instance, assume that such a graph contains $\exp\{O(n)\}$ perfect matchings. Apply Theorem 3 with β constant. The resulting running time is $2^n/\exp\{\Omega(n)\}$.

We note that Theorem 3 can be applied even if no bound β is known, given that the input matrix contains only nonnegative integers. For such matrices, a celebrated randomized algorithm by Jerrum, Sinclair, and Vigoda [7] computes for given $\epsilon > 0$ in time polynomial in n and $1/\epsilon$ a value b such that $\Pr\big((1-\epsilon)\operatorname{per} A \le b \le (1+\epsilon)\operatorname{per} A\big) \ge \frac{1}{2}$. We can then take $\beta = b^{1/n}$, which is only a factor $(1+\epsilon)^{1/n}$ off the best possible bound. Provided that $\operatorname{per} A \le n^n$, the size restriction on β applies for all $\delta > 0$ and n sufficiently large, so we can apply Theorem 3.

An algorithm by Cygan and Pilipczuk [5] computes the permanent in time

$$2^n/\exp\{\Omega(n/d)\},$$

where d is the average number of nonzero entries per row. Their algorithm requires no bound on the size of the permanent. We can compare Theorem 3 to the result of [5] by looking at matrices over $\{-1, 0, 1\}$ with at most d nonzero entries per row. For such matrices, we have $|\operatorname{per} A| \le \prod_{i=1}^n \sum_{j=1}^n |a_{ij}| \le d^n$, so that Theorem 3 applies with $\beta = d$ for the weaker bound $2^n/\exp\{\Omega(n/d^{1/(1-\delta)} \log d)\}$. On the other hand, Theorem 3 outperforms [5] on families of matrices with many nonzero entries but small permanents. For instance, consider an $n \times n$ matrix A over $\{-1, 0, 1\}$ constructed by taking d nonzero random entries per row and picking the sign on each 1 uniformly at random. It is known that $|\operatorname{per} A| \le (\lambda_{\max})^n$, where $\lambda_{\max}$ is the spectral norm of the matrix A [1, Sec. 2]. By the Bai–Yin theorem [2, Thm. 2], the spectral norm of a random matrix whose elements have mean 0 and variance σ^2 is concentrated around $2\sigma\sqrt{n}$. Since the variance of the elements in A is $\sigma^2 = d/n$, the absolute value of the permanent of A is almost surely less than $(\frac{201}{100}\sqrt{d})^n$. Now Theorem 3 with $\beta = \frac{201}{100}\sqrt{d}$ gives $2^n/\exp\{\Omega(n/d^{1/(2-\delta)} \log d)\}$ for any $\delta > 0$.

3. Preliminaries

Our starting point is *Ryser's formula* [9] for the permanent. It is based on the principle of inclusion–exclusion and can be given as follows:

$$\operatorname{per} A = (-1)^n \sum_{x \in \{0,1\}^n} (-1)^{x_1 + \cdots + x_n} \prod_{i=1}^n (Ax)_i. \tag{2}$$

We now review an idea of Bax and Franklin [3].

Lemma 4. *Let A be an $n \times n$ integer matrix. Then for every vector $r \in \mathbf{Z}^n$,*

$$\operatorname{per} A = (-1)^{n+1} \sum_{x \in \{0,1\}^n} (-1)^{x_1 + \cdots + x_n} \prod_{i=1}^n (Ax + r)_i. \tag{3}$$

Proof. Define the matrix $A' \in \mathbf{Z}^{n+1 \times n+1}$ as

$$A' = \begin{pmatrix} a_{11} & \dots & a_{1n} & r_1 \\ \vdots & \ddots & \vdots & \vdots \\ a_{n1} & \dots & a_{nn} & r_n \\ 0 & \dots & 0 & 1 \end{pmatrix}.$$

First, we observe $\operatorname{per} A = \operatorname{per} A'$, because in the Laplace expansion of the permanent of A' along the last row, all terms vanish except $a'_{n+1,n+1} \operatorname{per} A = 1 \cdot \operatorname{per} A$.

Now consider evaluating $\operatorname{per} A'$ with Ryser's formula (2). The factor

$$(A'x)_{n+1} = 0 \cdot x_1 + \cdots + 0 \cdot x_n + 1 \cdot x_{n+1} = x_{n+1}$$

vanishes unless $x_{n+1} = 1$. Thus, we can restrict our attention to vectors of the form $x' = (x_1, \dots, x_n, 1)$. For such a vector, we have $A'x' = A(x_1, \dots, x_n) + r$. Ryser's formula now gives

$$\operatorname{per} A' = (-1)^{n+1} \sum_{x \in \{0,1\}^n} (-1)^{x_1 + \cdots + x_n} \prod_{i=1}^n (Ax + r)_i. \quad \square$$

We turn to modular computation. Fix a positive integer k and let p be a prime. Let $\mathrm{GF}(p)$ denote the finite field of order p. Let X be the set of vectors $x \in \{0, 1\}^n$ such that the vector $Ax + r$ has fewer than k zeros in $\mathrm{GF}(p)$. The crucial observation is that we can restrict our attention to X:

Lemma 5. *Let A be an $n \times n$ integer matrix. Then,*

$$\operatorname{per} A = (-1)^{n+1} \sum_{x \in X} (-1)^{x_1 + \cdots + x_n} \prod_{i=1}^n (Ax + r)_i \pmod{p^k}.$$

Proof. If x does not belong to X then at least k entries in the vector $Ax+r \pmod p$ vanish. Thus, the product

$$\prod_{i=1}^{n}(Ax+r)_i$$

in (3) vanishes modulo p^k and x does not contribute to the permanent. □

Our algorithms work by evaluating the sum in the above lemma. This sum has $|X|$ terms, which can be much fewer than 2^n, so our ambition is to list the members of X in time significantly faster than 2^n.

4. Modulo a power of two

We consider first the case $p=2$. Here, we can calculate X precisely. Let W be the set of vectors $w\in\{0,1\}^n$ with fewer than k zeros. For $w\in W$ let X_w denote the set of vectors $x\in X$ such that $Ax+r=w$ in GF(2). By definition, $X=\bigcup_{w\in W}X_w$. Moreover, if $w\neq w'$ then X_w and $X_{w'}$ are disjoint, so

$$|X|=\sum_{w\in W}|X_w|\,. \tag{4}$$

Every set X_w is an affine subspace, which we list using Gaussian elimination.

Algorithm G *(Gaussian elimination).* Given an $n\times n$ matrix A and an n-dimensional vector b, this algorithm computes a vector u and d linearly independent vectors $v_1,\ldots,v_d$ such that the set of solutions x to the system of linear equations given by $Ax=b$ equals the affine space $u+\mathrm{span}(v_1,\ldots,v_d)$. □

Gaussian elimination runs in polynomial time and works over finite fields. Finally, the total size of X is easily bounded in expectation:

Lemma 6. *Construct the vector* $r=(r_1,\ldots,r_n)$ *by choosing the values* $r_1,\ldots,r_n$ *independently and uniformly at random from* $\{0,1\}$. *Then,* $\mathrm{E}[|X|]=|W|$.

Proof. Fix $w\in W$ and consider the system of equations $Ax=y-r$ in the variables $x_1,\ldots,x_n$. For $x\in\{0,1\}^n$, the ith equation,

$$\sum_{j=0}^{n}a_{ij}x_j=w_i+r_i$$

is satisfied with probability $\frac{1}{2}$. By independence of the random choices, the vector x satisfies the whole system of equations with probability 2^{-n}. By linearity of expectation, the expected number of $x\in\{0,1\}^n$ that satisfy the system is 1. The result follows by summing over all $w\in W$. □

This gives rise to the following randomized algorithm:

Algorithm A *(Permanent modulo a power of two).* Given a matrix $A\in\mathbf{Z}^{n\times n}$, and an integer $k>1$, this algorithm computes $\mathrm{per}\,A \bmod 2^k$. The algorithm works by iterating over all $w\in W$, tallying the contribution of each X_w in the variable t.

A1. [Choose random r.] Choose $r_1,\ldots,r_n\in\{0,1\}$ uniformly and independently at random.
A2. [Initialize.] Let $w=(1,\ldots,1)$ and $t=0$.
A3. [Construct X_y.] Run Algorithm G on input A and $w+r$. The output $u,v_1,\ldots,v_d$ describes the set X of solutions to the system of linear equations $Ax=w+r$ over the field GF(2).
A4. [Tally contribution of X_y.] For each $\alpha\in\{0,1\}^d$ set $x=u+\alpha_1v_1+\cdots\alpha_dv_d$ and increase t by

$$(-1)^{x_1+\cdots+x_n}\prod_{i=1}^{n}(Ax+r)_i\,.$$

A5. [Next w.] Let w be the next vector in W.
A6. [Done.] Return $(-1)^{n+1}t \bmod 2^k$. □

The efficient generation of all $w\in W$ in step A5 in total time $O(|W|n)$ is standard and not considered here, see *e.g.* [8, Sec. 7.2.1.3].

To analyze the running time, the total running time of Step A3 is $|W|\,\mathrm{poly}(n)$ because Gaussian elimination runs in polynomial time. The total running time of Step A4 is within a polynomial factor of $\sum_{w\in W}|X_w|$, which equals $|W|$ by (4) and Lemma 6. Finally, we observe $|W|=\sum_{i=0}^{k-1}\binom{n}{i}=n^{k+O(1)}$.

Algorithm A can be derandomized by applying the method of conditional expectations, which we can sketch as follows. The quantity $|X|$ is a function of the random choices $r_1,\ldots,r_n$. In the ith round of the method, we determine a value for r_i that does not decrease the expected value of $|X|$. Given fixed values for $r_1,\ldots,r_{i-1}$, the conditional expectations of $|X|$ for each choice of $r_i\in\{0,1\}$ can be computed exactly by iterating over the resulting W, again using Algorithm G in polynomial time. This establishes Theorem 1.

5. Modulo a prime power

We turn to the case where p is a prime greater than 2. Even though the approach from the last section would work also over the finite field GF(p), it would be inefficient, because the set corresponding to W, the vectors in $\{0,\ldots,p-1\}^n$ with fewer than k zeros, is much larger than 2^n.

Thus, we need a different approach. We will break down the problem into several blocks of rectangular matrices, so we consider this case first.

In this section, and the following, we use log for the logarithm with base 2, and ln for the natural logarithm.

5.1. Rectangular matrices

Let b be an integer and consider a $b\times n$ matrix B, a random vector $s\in\{0,\ldots,p-1\}^b$, and a number l with

$0 < l < b/p$. Let Y be the set of vectors $x \in \{0,1\}^n$ such that $Bx + s$ contains fewer than l zeros in GF(p).

We begin by showing that each Y is significantly smaller than $\{0,1\}^n$:

Lemma 7. $\mathrm{E}[|Y|] \le 2^n / \exp\{\gamma^2 b/2p\}$ *where* $\gamma = 1 - lp/b$.

Proof. Let $x \in \{0,1\}^n$. Let Z denote the number of zeros in $Bx + s$. Then x belongs to Y if $Z < l$.

For every $j \in \{1,\dots,b\}$, we have

$$\Pr((Bx+s)_j = 0 \bmod p) = \frac{1}{p}$$

where the probability is over the choices of $s_j \in \{0,\dots,p-1\}$. Thus,

$$\mathrm{E}[Z] = \frac{b}{p} \ge \frac{l}{1-\gamma}\,.$$

Using the Chernoff bound

$$\Pr\left(Z \le (1-\epsilon)E[Z]\right) \le \exp\left\{-\tfrac{1}{2}\epsilon^2 E[Z]\right\} \qquad (0 < \epsilon < 1)\,,$$

we can now compute

$$\begin{aligned}\Pr(x \in Y) &= \Pr(Z < l) \le \Pr(Z \le (1-\gamma)\mathrm{E}[Z]) \\ &\le \exp\{-\gamma^2 b/2p\}\,.\end{aligned}$$

By linearity of expectation, $E[|Y|] = \sum \Pr(x \in Y)$, summed over all $x \in \{0,1\}^n$. □

We proceed to show how we can list the vectors in Y in time corresponding to its size. This is our main technical challenge. We are inspired by the fact that

$$Bx = B\tbinom{y}{0} + B\tbinom{0}{z}\,,$$

where $\binom{y}{0} = (x_1,\dots,x_{n/2},0,\dots,0)^{\mathrm{T}}$ and $\binom{0}{z} = (0,\dots,0,x_{(n/2)+1},\dots,x_n)^{\mathrm{T}}$. Our strategy is to first build a table containing useful information for all $2^{n/2}$ choices of y, and then iterate over all $2^{n/2}$ choices of z, using the table to complete the sum.

To this end, we build a table $T(J,v)$ with entries for each index set $J \subseteq \{1,\dots,b\}$ and each vector $v \in \{0,\dots,p-1\}^b$. The table entry $T(J,v)$ is defined as the set of vectors $y \in \{0,1\}^{n/2}$ for which

$$\left(B\tbinom{y}{0}\right)_j = v_j \text{ if and only if } j \in J. \tag{5}$$

The entire table T can be built as follows:

Algorithm T *(Table T).* For given $b \times n$ matrix B, this algorithm builds the table $T(J,v)$ for all $J \subseteq \{1,\dots,b\}$ and $v \in \{0,\dots,p-1\}^b$.

T1 [Initialize.] For each $J \subseteq \{1,\dots,b\}$ and $v \in \{0,\dots,p-1\}^b$, let $T(J,v) = \emptyset$.
T2 [Conditionally append each y.] For each $y \in \{0,1\}^{n/2}$, for each $v \in \{0,\dots,p-1\}^b$, and for each $J \subseteq \{1,\dots,b\}$, insert y into $T(J,v)$ if (5) is satisfied. □

Having built T, we proceed to list Y. The idea is as follows. For given $z \in \{0,1\}^{n/2}$ we set $v = -B\binom{0}{z} - s$. Then $T(J,v)$ consists of all y for which $B\binom{y}{0}$ equals $-B\binom{0}{z} - s$ in exactly the positions that belong to J. In particular, there are exactly $|J|$ zeros in the vector

$$B\tbinom{y}{0} + B\tbinom{0}{z} + s = B\tbinom{y}{z} + s\,.$$

Algorithm Y *(List Y).* For given $b \times n$ matrix B, vector $s \in \{0,\dots,p-1\}^b$, and real number l, this algorithm outputs every vector $x \in \{0,1\}^n$ such that $Bx + s$ contains fewer than l zeros exactly once.

Y1 [Build table.] Build T using Algorithm T.
Y2 [Iterate.] For each $z \in \{0,1\}^{n/2}$, for each nonnegative integer j less than l and each index subset $J \subseteq \{1,\dots,b\}$ of size j, do step Y3.
Y3 [Consult table.] Set $v = -B\binom{0}{z} - s$. Look up $T(J,v)$. [Every $y \in T(J,v)$ is such that $B\binom{y}{z} + s$ has exactly j zeros.] If $T(J,v) \neq \emptyset$ do step Y4.
Y4 [Output.] Output $\binom{y}{z}$ for each $y \in T(J,v)$. □

Note that each $\binom{y}{z} \in Y$ is output exactly once.

Lemma 8. *Algorithm Y runs in time within a polynomial factor of* $|Y| + 2^{(n/2)+b(1+\log p)}$.

Proof. Step Y1 consists of a call to Algorithm T. Up to polynomial factors, Algorithm T is dominated by the number of nested iterations in Step T2, of which there are $2^{n/2}p^b 2^b = 2^{(n/2)+b(1+\log p)}$. The number of times Step Y2 calls Step Y3 is $2^{n/2}\sum_{j\le l}\binom{b}{j} \le 2^{n/2}\sum_j \binom{b}{j} = 2^{(n/2)+b}$; each execution takes a polynomial number of steps for the matrix operations. This takes less time than Step Y1. Finally, the output operation in Step Y4 is executed exactly $|Y|$ times in total. □

5.2. Block decomposition

We return to the case where A is an $n \times n$ matrix.

Our plan is to partition A into submatrices of equal dimension $b \times n$, where b is roughly $n/4\log p$. However, a complication arises when b does not divide n exactly, so we need a more balanced approach.

Select integers $b_1,\dots,b_t$ such that $b_1 + \dots + b_t = n$ and for each $i \in \{1,\dots,t\}$,

$$\frac{n}{8 + 8\log p} \le b_i \le \frac{n}{4 + 4\log p}\,. \tag{6}$$

Note that $t = O(\log p)$.

Partition A into submatrices such that for $i \in \{1,\dots,t\}$, the ith matrix $A^{(i)}$ consists of the jth row of A where $j \in \{b_1 + \dots + b_{i-1} + 1,\dots,b_1 + \dots + b_i\}$. Partition the vector r into blocks $r^{(i)}$ in the corresponding way.

We now formalize an averaging argument that says that if $Ax + r$ contains only few zeros, then for some i, the vector $A^{(i)}x + r^{(i)}$ contains only few zeros as well. For each $i \in \{1,\dots,t\}$, define $X^{(i)}$ as the set of vectors $x \in \{0,1\}^n$ for which

$$A^{(i)}x + r^{(i)} \tag{7}$$

contains fewer than kb_i/n zeros modulo p. Then,

$$X \subseteq X^{(1)} \cup \cdots \cup X^{(t)} . \tag{8}$$

To verify this, assume x is in none of the sets $X^{(i)}$ for $i \in \{1, \ldots, t\}$. Then the number of nonzero elements in the vector in $Ax + r$ is at least

$$\frac{n}{k}(b_1 + \ldots + b_t) = k ,$$

so x does not belong to X either. We conclude that we can list the elements of X by listing the elements of each $X^{(i)}$. This may produce spurious elements, but membership in X can be easily checked by computing $Ax + r$.

5.3. Combining the blocks

With the above considerations we are ready for our algorithm to list the set X. Recall that X is the set of vectors x such that $Ax + r$ contains fewer than k zeros.

Algorithm X *(List X).* For given $n \times n$ matrix A and vector $r \in \{0, \ldots, p-1\}^n$, this algorithm outputs every element in X at least once.

- **X1** [Initialize] Choose integers $b_1, \ldots, b_t$ as described in Section 5.2. Let $i = 1$.
- **X2** [Consider ith block] Let $B = A^{(i)}$, $s = r^{(i)}$, and $Y = X^{(i)}$ as defined in Section 5.2.
- **X3** [List and filter solutions to ith block] Use Algorithm Y on the $b_i \times n$-dimensional problem defined by B, s, and $l = kb_i/n$ to produce every $x \in Y$. For each such x, compute $Ax + r$, and if the result has fewer than k zeros then output x.
- **X4** [Next i] Increment i. If $i \le t$ return to Step X2. Otherwise terminate. □

Lemma 9. *If $kp < n$ then the expected running time of algorithm X is bounded within a polynomial factor of*

$$\frac{2^n}{\exp\{\Omega(\gamma^2 n/(p \log p))\}} ,$$

where $\gamma = 1 - kp/n$.

Proof. Algorithm X is dominated by the calls to Algorithm Y in Step X3 in each round. The total number of rounds is $t = O(\log p) = O(\log n)$, so we can focus on the contribution of a single round.

According to Lemma 8, Step X3 takes time

$$|Y| + 2^{(n/2)+b_i(1+\log p)} .$$

We proceed to bound each of the two terms.

For the first term, Lemma 7 with $\gamma = 1 - lp/b_i = 1 - kp/n$ gives

$$E[|Y|] \le \frac{2^n}{\exp\{\gamma^2 b_i/2p\}} .$$

This fits the claimed bound because the lower bound on b_i in (6) and $p \ge 3$ gives

$$\frac{b_i}{2p} \ge \frac{n}{2p \cdot (8 + 8\log p)} \ge \frac{n}{32 p \log p} .$$

For the second term, consider the exponent

$$\frac{n}{2} + b_i(1 + \log p) \le \frac{n}{2} + \frac{n(1+\log p)}{4 + 4\log p} = \frac{3n}{4} .$$

This fits the claimed bound because a simple calculation shows that

$$\frac{1}{4} \ge \frac{\log e}{2p \log p}$$

for $p \ge 3$. Thus, $2^{n/4} \ge \exp\{n/(2p\log p)\} \ge \exp\{\gamma^2 n/(2p\log p)\}$, and we conclude

$$2^{3n/4} = 2^n/2^{n/4} \le 2^n/\exp\{\gamma^2 n/(2p\log p)\} . \quad \square$$

We are finally ready to present the algorithm for computing $\operatorname{per} A \bmod p^k$.

Algorithm B *(Permanent modulo prime power).* Given a matrix $A \in \mathbf{Z}^{n \times n}$, a prime p, and an integer k such that $kp < n$, this algorithm computes $\operatorname{per} A \bmod p^k$.

- **B1** [Generate r] Generate a random vector $r \in \{0, \ldots, p-1\}^n$ with elements uniformly and independently distributed.
- **B2** [Generate X] Generate the set of vectors X for which $Ax + r$ has fewer than k zeros using Algorithm X.
- **B3** [Calculate permanent] Iterating over each $x \in X$, return

$$(-1)^{n+1} \sum_{x \in X} (-1)^{x_1 + \cdots + x_n} \prod_{i=1}^{n} (Ax + r)_i \bmod p^k . \qquad \square$$

Derandomization can be achieved by the method of conditional probabilities. The running time is dominated by step B2, which we analyzed in Lemma 9. This proves Theorem 2.

A subtlety is that we must avoid iterating over the same $x \in X$ twice in B3, even though Algorithm X may have produced each $x \in X$ several times. Thus, Step B2 needs to store all of X to avoid duplicates. In particular, Algorithm B requires as much space as time. (Algorithm T also uses exponential space to store its table.)

6. Small permanents

We can apply Algorithm B in the non-modular case using some number theory, provided we know an upper bound β^n on $|\operatorname{per} A|$.

Lemma 10. *Let $r, c > 0$. Then*

$$\prod_{p<r} p^{cn/p} \ge (\tfrac{1}{11} r)^{cn} ,$$

where the product ranges over all primes smaller than r.

Proof. Merten's first theorem (see, *e.g.*, [11, Thm. 7]) gives

$$\ln r \le 1 + (\ln 4) + \sum_{p<r} \frac{\ln p}{p}\,.$$

Multiplying both sides by cn and taking exponents, we have

$$r^{cn} \le (4\mathrm{e})^{cn} \prod_{p<r} p^{cn/p}\,,$$

from which the result follows because $4\mathrm{e} < 11$. $\square$

We need the Chinese remainder algorithm (see, *e.g.*, [10, Thm. 4.6]).

Algorithm R *(Chinese remaindering).* Given pairwise coprime integers $q_1, \ldots, q_m$ and integers $s_1, \ldots, s_m$, this algorithm finds an integer s such that $s \equiv s_i \pmod{q_i}$ for all $i \in \{1, \ldots, m\}$. The algorithm runs in time $O((\log \prod_i q_i)^2)$. $\square$

By the Chinese remainder theorem, s is unique with this property among the integers satisfying $0 \le s < q_1 \cdots q_m$. A slight complication arises because $\operatorname{per} A$ can be a negative number. Thus, we use Chinese remaindering to instead compute $\operatorname{per} A + \lceil \beta^n \rceil$, which is positive and at most twice as large. We are now ready to present our algorithm.

Algorithm C *(Small permanent).* Given $\delta > 0$, an $n \times n$ matrix A of integers, a real number $\beta \le (\frac{1}{44}\delta n)^{1-\delta}$ such that $|\operatorname{per} A| \le \beta^n$, this algorithm computes $\operatorname{per} A$.

C1 [Construct primes] Set $r = 11(2\beta)^{1/(1-\delta)}$. Construct the sequence $p_1, \ldots, p_m$ of primes less than r.

C2 [Compute moduli] For each $i \in \{1, \ldots, m\}$, set $k_i = \lceil (1-\delta)n/p_i \rceil$ and compute

$$s_i = (\operatorname{per} A + \lceil \beta^n \rceil) \bmod p_i^{k_i}$$

using Algorithm B to compute $\operatorname{per} A \bmod p_i^{k_i}$.

C3 [Chinese remaindering] Use Algorithm R to compute s from the coprime integers $p_1^{k_1}, \ldots, p_m^{k_m}$ and integers $s_1, \ldots, s_m$. Return $s - \lceil \beta^n \rceil$. $\square$

To see correctness, by Lemma 10, we have

$$\prod_{i=1}^{m} p_i^{k_i} \ge \prod_{i=1}^{m} p_i^{(1-\delta)n/p_i} \ge (\tfrac{1}{11} r)^{(1-\delta)n} = (2\beta)^n \ge 2\beta^n\,,$$

so the choice of moduli suffices to compute $\operatorname{per} A + \lceil \beta^n \rceil$ uniquely in step C3.

For the running time, Step C1 can be performed in time $O(r \log r) = O(n \log n)$ by sieving for primes less than r using standard techniques [10, Sec. 5.4]. The call to Algorithm R in Step C3 takes quadratic time in

$$\log \prod_{i=1}^{m} p_i^{k_i} = \sum_{i=1}^{m} k_i \log p_i \le \sum_i \left(1 + \frac{(1-\delta)n}{p_i}\right) \cdot \log p_i$$
$$= O(n^2)\,.$$

Thus, the running time of algorithm C is dominated by Step C2, which involves no more than $r = O(n)$ calls to Algorithm B. We have for each $i \in \{1, \ldots, m\}$,

$$\begin{aligned}
k_i p_i &= \left\lceil \frac{(1-\delta)n}{p_i} \right\rceil \cdot p_i \le \left(\frac{(1-\delta)n}{p_i} + 1 \right) \cdot p_i \\
&\le (1-\delta)n + p_i \le (1-\delta)n + r \\
&= (1-\delta)n + 11(2\beta)^{1/(1-\delta)} \le (1-\delta)n + 11 \cdot \tfrac{2}{44}\delta n \\
&= (1 - \tfrac{1}{2}\delta)n\,.
\end{aligned}$$

Thus, the condition $k_i p_i < n$ in the call to Algorithm B is satisfied, and the running time of Step C2 is

$$2^n / \exp\{\Omega(\delta^2 n / p_i \log p_i)\}$$

by Lemma 9 with $\gamma = \frac{1}{2}\delta$. Since $p_i \le r = 11(2\beta)^{1/(1-\delta)}$, we have

$$\frac{n}{p_i \log p_i} \ge \frac{n(1-\delta)}{11(2\beta)^{1/(1-\delta)} \log \beta}$$

so we can bound the total running time of Algorithm C to within a polynomial factor of

$$2^n / \exp\{\Omega(\delta^2 n / \beta^{1/(1-\delta)} \log \beta)\}\,,$$

establishing Theorem 3.

Acknowledgements

This work is supported by the Swedish Research Council, VR 2012–4730.

References

[1] S. Aaronsson, T. Hance, Generalizing and derandomizing Gurvits's approximation algorithm for the permanent, Quantum Inf. Comput. 14 (7–8) (2014) 541–559.

[2] Z.D. Bai, Y.Q. Yin, Limit of the smallest eigenvalue of a large-dimensional sample covariance matrix, Ann. Probab. 21 (1993) 1275–1294.

[3] E. Bax, J. Franklin, A permanent algorithm with $\exp[\Omega(n^{1/3}/2\ln n)]$ expected speedup for 0–1 matrices, Algorithmica 32 (1) (2002) 157–162.

[4] A. Björklund, Below all subsets for some permutational counting problems, in: 15th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2016, June 22–24, 2016, Reykjavik, Iceland, in: LIPIcs, vol. 53, Schloss Dagstuhl, Leibniz-Zentrum fuer Informatik, 2016, pp. 17:1–17:11.

[5] M. Cygan, M. Pilipczuk, Faster exponential-time algorithms in graphs of bounded average degree, Inf. Comput. 243 (2015) 75–85.

[6] H. Dell, T. Husfeldt, D. Marx, N. Taslaman, M. Wahlén, Exponential time complexity of the permanent and the Tutte polynomial, ACM Trans. Algorithms 10 (4) (2014) 21.

[7] M. Jerrum, A. Sinclair, E. Vigoda, A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries, J. Assoc. Comput. Mach. 51 (2005) 671–697.

[8] D.E. Knuth, The Art of Computer Programming, vol. 4, Addison-Wesley, 2011.

[9] H.J. Ryser, in: Combinatorial Mathematics, in: Carus Math. Monogr., vol. 14, Mathematical Association of America, 1963.

[10] V. Shoup, A Computational Introduction to Number Theory and Algebra, 2nd edition, Cambridge University Press, 2008.

[11] G. Tenenbaum, Introduction to Analytic and Probabilistic Number Theory, Cambridge University Press, 1995.

[12] L.G. Valiant, The complexity of computing the permanent, Theor. Comput. Sci. 8 (2) (1979) 189–201.