Permanents

The permanent of an N-by-N square matrix $A = (a_{i,j})$ is defined as

$$\operatorname{per}(A) = \sum_{\sigma \in P_N} \prod_{i=1}^{N} a_{i,\sigma(i)} \tag{1}$$

where P_N is the set of permutations of the N-set $\{1, \ldots, N\}$ [1]. Equation (1) is similar to the definition of the determinant,

$$\det(A) = \sum_{\sigma \in P_N} \operatorname{sgn}(\sigma) \prod_{i=1}^{N} a_{i,\sigma(i)}, \tag{2}$$

except that the signatures of the permutations are not taken into account; this exclusion notably makes the permanent much more difficult to compute than the determinant, because it means that decomposition methods can no longer be used [2]. In 1979, Valiant published his theorem stating that the computation of the permanent is in the complexity class #P-hard, i.e. neither in P nor in NP [3]. The permanent, however, commonly appears in problems related to quantum mechanics, so it is still worthwhile to pursue more efficient algorithms than the naive ones suggested by the permanent's definition, which has time complexity $\mathcal{O}(N!N)$.

To date, there are two general algorithms considered to be the fastest; one due to Ryser (1963), [4] and one due to Glynn (2010, although it was found independently by several others prior to this) [2, 5]. The Ryser algorithm is based on the inclusion-exclusion principle and is given by the formula

$$\operatorname{per}(A) = \sum_{S \subseteq \{1,\ldots,N\}} (-1)^{|S|} \prod_{i=1}^{N} \sum_{j=1}^{|S|} a_{i,S(j)}. \tag{3}$$

The Glynn algorithm is based on invariant theory, derived from the polarization identity for a symmetric tensor, and is given by the formula

$$\operatorname{per}(A) = \frac{1}{2^{N-1}} \cdot \sum_{\delta} \left(\sum_{k=1}^{N} \delta_k \right) \prod_{j=1}^{N} \sum_{i=1}^{N} \delta_i a_{i,j} \tag{4}$$

where the outer sum is over all 2^{N-1} vectors $\delta = \{\pm 1_1, \ldots, \pm 1_N\}$. It is still not clear which algorithm is faster, but they both scale in time with $\mathcal{O}(2^N N^2)$ if implemented naively, and with $\mathcal{O}(2^N N)$ if the sets S in Equation (3) and δ in Equation (4) are iterated over in Gray code order [2, 6].

At small values of N, the naive algorithm is actually the fastest, but it is outsped by the other two algorithms at around $N \approx 10$, according to my testing, and the difference in time between the algorithms at large values of N is several orders of magnitude. We haven't been able to find any fast C/++ code for these algorithms, though, in order to properly determine which algorithm is best at which values of N. It would be nice to have a good C/++ implementation of the permanent function which automatically deploys the fastest algorithm for the size of the input matrix.

Permanents of rectangular matrices

The definition of the permanent can be generalized to work with M-by-N rectangular matrices with $M \leq N$, giving us the formula

$$\mathrm{per}(A) = \sum_{\sigma \in P_{NM}} \prod_{i=1}^{M} a_{i,\sigma(i)} \tag{5}$$

where P_{NM} is the set of M-permutations of the N-set $\{1, \ldots, N\}$ [1]. Since $\mathrm{per}(A) = \mathrm{per}(A^T)$, this is sufficient for computing the permanent of all rectangular matrices. Ryser also extended his own algorithm to rectangular permanents in the same original 1963 article [2, 4].

The Glynn algorithm can be generalized to work with rectangular permanents by use of the identity (shown here for $M \geq N$),

$$\mathrm{per}\begin{pmatrix} a_{1,1} & \cdots & a_{1,N} & 1_{1,N+1} & \cdots & 1_{1,M} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{M,1} & \cdots & a_{M,N} & 1_{M,N+1} & \cdots & 1_{M,M} \end{pmatrix} = \frac{1}{(M-N+1)!} \cdot \mathrm{per}\begin{pmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{M,1} & \cdots & a_{M,N} \end{pmatrix}, \tag{6}$$

which can be trivially fit into Equation (4) by extending the sum over δ:

$$\mathrm{per}(A) = \frac{1}{2^{N-1}} \cdot \frac{1}{(N-M+1)!} \cdot \sum_{\delta} \left(\sum_{k=1}^{N} \delta_k \right) \prod_{j=1}^{M} \left(\sum_{i=1}^{M} \delta_i a_{i,j} + \sum_{i=M+1}^{N} \delta_i \right). \tag{7}$$

Now, our problem is more complicated, since we want to determine the best algorithm for each value of M and N.

References

[1] Wikipedia, "Permanent (mathematics) — Wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Permanent\%20(mathematics)&oldid=1046043878`, 2021.

[2] Wikipedia, "Computing the permanent — Wikipedia, the free encyclopedia." `http://en.wikipedia.org/w/index.php?title=Computing\%20the\%20permanent&oldid=1037768944`, 2021.

[3] L. G. Valiant, "The complexity of computing the permanent," *Theoretical computer science*, vol. 8, no. 2, pp. 189–201, 1979.

[4] H. J. Ryser, *Combinatorial mathematics*, vol. 14. American Mathematical Soc., 1963.

[5] D. G. Glynn, "The permanent of a square matrix," *European Journal of Combinatorics*, vol. 31, no. 7, pp. 1887–1891, 2010.

[6] D. E. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 2: Generating All Tuples and Permutations (Art of Computer Programming)*. Addison-Wesley Professional, 2005.