The Classical Complexity of Boson Sampling : A multi-core CPU implementation

Manan Vaswani

Level I
40cp project

Supervisor: Dr. Raphaël Clifford
Date: 6th May, 2019

Acknowledgement of Sources

Contents

1 Introduction

2 Background

3 Preliminaries

3.1 Binary Gray Code

3.2 Permanent of a matrix

Computing the permanent of large matrices is one of the key parts of the Boson Sampling problem, as will be discussed in detail while explaining the problem in the subsequent sections of the paper.

3.2.1 Definition

Definition 3.1. *[1] The permanent of an $n \times n$ matrix $A = (a_{ij})$ is defined as*

$$\operatorname{Per} A = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)} \tag{1}$$

where the sum is over all elements of the symmetric group S_n i.e. over all permutations of the numbers in $[n] = \{1, 2, ..., n\}$

Example 3.2. For a 2×2 matrix, the permanent is calculated as follows

$$\operatorname{Per} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad + bc \tag{2}$$

Example 3.3. For a 3×3 matrix, the permanent is calculated as follows

$$\operatorname{Per} \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = aei + bfg + cdh + ceg + bdi + afh \tag{3}$$

One can observe that the definition of the permanent is similar to the more commonly used determinant function, differing in the fact that the permanent definition lacks the alternating signs. An important property to note about the permanent function is that it is invariant to transposition i.e. $\operatorname{Per} A = \operatorname{Per} A^T$ [2].

3.2.2 Computing the permanent

Valiant showed that the problem of computing the permanent of a matrix is in the class #P-complete which implies that it is unlikely to have a polynomial time algorithm which implies that it is unlikely to have a polynomial-time algorithm [3]. The naive algorithm obtained by directly translating the formula into an algorithm would run in $\mathcal{O}(n!n)$ time.

A significant improvement on the naive approach, Rysers algorithm uses a variant of the inclusion-exclusion principle and can be evaluated in $\mathcal{O}(n^2 2^n)$ time [2]. Nijenhuis and Wilf sped this up to $\mathcal{O}(n2^n)$ time by iterating over the sum in Gray Code order [4].

Another formula that is as fast as Rysers was independently derived by Balasubramanian[5], Bax[6], Franklin and Bax[7], and Glynn[8], all using different methods. We shall henceforth refer to this as Glynn's formula and it is described as follows.

Let $M = (m_{ij})$ be an $n \times n$ matrix with $m_{ij} \in \mathbb{C}$, then

$$\operatorname{Per} M = \frac{1}{2^{n-1}} \sum_{\delta} \left(\prod_{k=1}^{n} \delta_k \right) \prod_{j=1}^{n} \sum_{i=1}^{n} \delta_i m_{ij} \tag{4}$$

where $\delta \in \{-1, 1\}^n$ with $\delta_1 = 1$. Hence there are 2^{n-1} such values for δ.

Implementing the formula as is would require $\mathcal{O}(2^n n^2)$ time. However, iterating over the δ arrays in Gray code order reduces it to $\mathcal{O}(n2^n)$ time.

In order to exploit this trick, let v_j be the symbol used to denote the innermost sum i.e. $v_j = \sum_{i=1}^{n} \delta_i m_{ij}$. If δ is iterated through in Gray code order, then one can notice that the terms $\{v_j, j \in [n]\}$ can actually be computed in $\mathcal{O}(n)$ time for a given value of δ rather than $\mathcal{O}(n^2)$ which is how long it would take if done naively. This is because successive elements of δ differ in only one position, so each successive v_j differs only by the addition or subtraction of some m_{ij} where i is the position at which δ was changed. Hence, the product of v_j terms can now be calculated in $\mathcal{O}(n)$ time.

4 Describing the paper and specifying the problem

Cliiford and Clifford [9] gave a study and analysis of the classical complexity of the exact Boson Sampling problem and proposed an algorithm that is significantly faster than previous algorithms for the problem. The algorithm is simple to implement, and is able to solve the Boson Sampling problem for system sizes much greater than quantum computing systems currently available, which reduces the likelihood of achieving quantum supremacy in the context of Boson Sampling in the near future rephrase last bit.

4.1 Explaining the problem

A summary of the problem in purely mathematical terms is as follows.

Let m and n be positive integers. Consider all possible multisets[1] of size n with elements in $[m]$, where $[m] = \{1, ..., m\}$. Let $z = [z_1, z_2, ..., z_n]$ be an array representation of such a multiset, with its elements in non-decreasing order. In other words, z is an array of n integers taken from $[m]$ (with repetition) and arranged in non-decreasing order. Define $\Phi_{m,n}$ to be the set of all distinct values that z can take. Define $\mu(z) = \prod_{j=1}^{m} s_j!$ where s_j is the multiplicity of j in the array z i.e. the number of times it appears in z.

[1] A multiset is a special kind of set in which elements can be repeated

$A = (a_{ij})$ is a complex-valued $m \times n$ matrix constructed by taking the first n columns of a given $m \times m$ Haar random unitary matrix. For each z, build an $n \times n$ matrix A_z where the k^{th} row of A_z is the z_k^{th} row in A, for $k = 1, ..., n$. Finally, define a probability mass function over $\Phi_{m,n}$ as

$$q(z) = \frac{1}{\mu(z)} |\text{Per } A_z|^2 = \frac{1}{\mu(z)} \left| \sum_{\sigma} \prod_{k=1}^{n} a_{z_k \sigma_k} \right|^2, \quad z \in \Phi_{m,n} \tag{5}$$

where Per A_z is the permanent of A_z and $\pi[n]$ is the set of all permutations of $[n]$.

The computing task is to simulate random samples from the above pmf $q(z)$.

4.1.1 Calculating the size of the sample space

The size of the sample space $\Phi_{m,n}$ can be calculated using the 'stars and bars' technique from combinatorics [10]. In our problem, we have n 'stars' representing the elements of the array z, and m 'buckets' representing all the values in $[m]$. Recall that z is a sorted array representation of a multiset, so multiple 'stars' can be in one 'bucket'. Since there are m 'buckets', we need $m - 1$ 'bars' to divide the 'stars' into 'buckets'. Therefore, from a total of $m - 1 + n$ objects, we need to pick n of these to be the 'stars'. There are $\binom{m+n-1}{n}$ ways to do this. Hence, there are $\binom{m+n-1}{n}$ possible values of z.

5 The Boson Sampling Algorithm

5.1 The naive approach

Translating the formula above (5) directly to an algorithm would require $\mathcal{O}(\binom{m+n-1}{n} n 2^n)$ time to evaluate. The $\binom{m+n-1}{n}$ term comes from the size of the sample space of the pmf, and calculating the value of the permanent using the fastest known methods takes $\mathcal{O}(n2^n)$ time (Section 3.2.2). With $m = \mathcal{O}(n^2)$ as suggested in Section????, the total running time is $\mathcal{O}(\binom{n^2+n-1}{n} n 2^n) =$???. Hence, even for relatively small values of n ((give some actual numbers), computing the Boson Sampling problem would be intractable for even very powerful supercomputers.

In the literature [9], two new algorithms are proposed for exact Boson Sampling. These are referred to as Algorithm A and Algorithm B, and both provide a significant speed up on the naive algorithm. The following subsections summarise the approach taken to obtain them.

5.2 Algorithm A

The approach to Algorithm A starts by expanding the sample space to a much larger size, which seems counterintuitive at first, but it allows us to express the pmf (Equation 5) in a form that is much easier to compute. The sample space is expanded to the space of all arrays $\mathbf{r} = (r_1, r_2, ..., r_n)$ where each element r_k is in $[m]$, which implies that we

are considering a distribution on the product space $[m]^n$. It is stated and proved that sampling from $q(\mathbf{z})$ is equivalent to sampling from the pmf

$$p(\mathbf{r}) = \frac{1}{n!} |\text{Per } A_{\mathbf{r}}|^2 = \frac{1}{n!} \left| \sum_{\sigma} \prod_{i=1}^{n} a_{r_i \sigma_i} \right|^2, \quad \mathbf{r} \in [m]^n \tag{6}$$

where as before, σ is the set of all permutations of $[n]$.

This method requires $p(\mathbf{r}) = p(r_1, ..., r_n)$ to be rewritten as a product of conditional probabilities using the chain rule i.e.

$$p(\mathbf{r}) = p(r_1)p(r_2|r_1)p(r_3|r_1, r_2)...p(r_n|r_1, r_2, ..., r_{n-1}) \tag{7}$$

We first sample r_1 from $p(r_1), r_1 \in [m]$. Then for $k = 2, ..n$, we sample r_k from the conditional pmf $p(r_k|r_1, r_2, ..., r_{k-1})$ with $r_1, ..., r_{k-1}$ fixed. After sampling all values of r_k, sort $(r_1, r_2, ..., r_n)$ in non-decreasing order, and that results in the array representation of a multiset sampled from the Boson Sampling distribution $q(\mathbf{z})$. In order to obtain the conditional probabilities, a formula for the joint pmf of the leading subsequences of $(r_1, ..., r_k)$ is given in Lemma 1 of the paper, and proved using arithmetic techniques and facts about probability measures. The formula in Lemma 1 is as follows:

$$p(r_1, ..., r_k) = \frac{(n-k)!}{n!} \sum_{c \in \mathcal{C}_k} \left| \text{Per } A^c_{r_1,...,r_k} \right|^2, \quad k = 1, ..., n \tag{8}$$

where $\mathcal{C}_k$ is the set of $k-$combinations taken without replacement from $[n]$ and $A^c_{r_1,...,r_k}$ is the matrix formed by taking only columns $c \in \mathcal{C}_k$ of the rows $(r_1, ..., r_k)$ of A.

Notice in equation 7, we need to sample r_k from the conditional probability distribution $p(r_k|r_1, ..., r_{k-1})$. This can be rewritten as $p(r_1, ..., r_k)/p(r_1...r_{k-1})$. Since r_k does not appear in the denominator, in order to sample r_k from the conditional pmf, we can equivalently sample from the pmf proportional to the numerator, as $(r_1, ...r_{k-1})$ are fixed, known values at this stage in the algorithm. Therefore, the formula is Lemma 1 is used to calculate the conditional pmfs at each stage, giving way to the following algorithm.

Algorithm 1 Boson Sampler: Single sample $\mathbf{z}$ from $q(\mathbf{z})$ in $\mathcal{O}(mn3^n)$ time

Require: $m, n \in \mathbb{Z}_+$; A formed by first n columns of $m \times m$ Haar random unitary matrix

1: $\mathbf{r} \leftarrow \emptyset$
2: **for** $k \leftarrow 1 \rightarrow n$ **do**
3: $\quad w_i \leftarrow \sum_{c \in \mathcal{C}_k} \left| \text{Per } A^c_{(\mathbf{r},i)} \right|^2, i \in [m]$
4: $\quad x \leftarrow \text{Sample}(w)$
5: $\quad \mathbf{r} \leftarrow (\mathbf{r}, x)$
6: $\mathbf{z} \leftarrow \text{IncSort}(\mathbf{r})$
7: **return z**

The correctness of the algorithm is clear as it is simply an evaluation of the required pmf using the chain rule of probability. The literature gives a mathematical proof to derive the runtime, however here we shall give an informal explanation to obtain the runtime using the algorithm above. The runtime is dominated by the **for** loop in lines 2 to 6, as the sorting step in line 7 takes only $\mathcal{O}(n \log n)$ using the fastest methods for sorting (insert reference). In each iteration of the **for** loop, we first construct the conditional pmf on the sample space $[m]$. This is represented by a weighted array $\mathbf{w}$. This involves calculating the permanents of $|\mathcal{C}_k|$ $k \times k$ matrices for each $i \in [m]$, where k is the loop index variable. Sampling from this distribution (line 4) takes $\mathcal{O}(m)$ time [11], but this too is dominated by the calculations in line 3, so we can ignore it. Therefore, the total time taken in the k^{th} iteration is $\mathcal{O}(m\binom{n}{k}k2^k)$, as $|\mathcal{C}_k| = \binom{n}{k}$ and calculating the permanent of a $k \times k$ matrix takes $\mathcal{O}(k2^k)$ using the fastest methods. So for n iterations,

$$\sum_{k=1}^{n} mk2^k \binom{n}{k} = m\frac{2}{3}n3^n = \mathcal{O}(mn3^n) \tag{9}$$

Therefore the total time taken is $\mathcal{O}(mn3^n)$. In terms of space complexity, we only need to store one permanent calculation at a time, and the weight array for each pmf is of size m. These can also be stored only one at a time, so $\mathcal{O}(m)$ additional space is required.

5.3 Algorithm B

The main theorem of the paper(Theorem 1) states that the time complexity of the Boson Sampling problem is $\mathcal{O}(n2^n + \text{poly}(m, n))$ where $\text{poly}(m, n) = \mathcal{O}(mn^2)$, with $\mathcal{O}(m)$ additional space required. This is achieved using the second algorithm proposed in the paper i.e. Algorithm B.

Algorithm B makes use of the Laplace Expansion [1] of permanents to obtain a significant speed up. The Laplace expansion allows us to calculate the permanent of a matrix using its permanent minors (the permanent of a submatrix with a column and a row removed). For any $k \times k$ matrix $B = (b_{i,j})$,

$$\text{Per}\, B = \sum_{l=1}^{k} b_{k,l} \,\text{Per}\, B^{\diamond}_{k,l}, \tag{10}$$

where $B^{\diamond}_{k,l}$ is the submatrix of B with row k and column l removed. Hence the permanent of B can be calculated in $\mathcal{O}(k)$ steps provided the permanent minors i.e. the values $\{\text{Per}\, B^{\diamond}_{k,l}\}$ are known.

Computing $\{\text{Per}\, B^{\diamond}_{k,l}\}$ by calculating each of the permanents independently would collectively take $\mathcal{O}(k^2 2^k)$ time as there are k permanents to compute and to compute the value of each one takes $\mathcal{O}(k2^k)$ time. Lemma 2 of the paper states that the collection $\{\text{Per}\, B^{\diamond}_{k,l}\}$ can be collectively evaluated in $\mathcal{O}(k2^k)$ time and $\mathcal{O}(k)$ extra space. using Glynn's formula to evaluate a permanent minor for a given value of l, we have

$$\text{Per}\, B^{\diamond}_{k,l} = \frac{1}{2^{k-2}} \sum_{\delta} \left(\prod_{i=1}^{k-1} \delta_i \right) \prod_{j \in [k] \setminus l} v_j(\delta), \quad l \in [k], \tag{11}$$

where $\delta \in \{-1,1\}^{k-1}$ with $\delta_1 = 1$ and $v_j(\delta) = \sum_{i=1}^{k-1} \delta_i b_{ij}$. The usual trick of iterating over the values of δ in Gray code order is required to compute $\operatorname{Per} B^{\diamond}_{k,l}$ in $\mathcal{O}(k2^k)$ time for each value of l, but this would still have to be repeated k times to cover all $l \in [k]$. The second and more novel trick used to achieve an additional speed up is used while calculating the products $\prod_{j\in[k]\setminus l} v_j(\delta)$. Observe that for each l, $\prod_{j\in[k]\setminus l} v_j(\delta) = \prod_{j\in[k]} v_j(\delta)/v_l(\delta)$. However, we cannot compute this method to compute the partial product as it would not work for cases when $v_l(\delta) = 0$. Instead, we first precompute two arrays $\mathbf{f}$ and $\mathbf{b}$ containing the forwards and backwards cumulative products of $v_j(\delta)$ respectively as follows,

$$f_i = \prod_{j=1}^{i} v_j(\delta) \quad b_i = \prod_{j=i}^{k} v_j(\delta), \quad i \in [k]. \tag{12}$$

Then each partial product $\prod_{j\in[k]\setminus l} v_j(\delta)$ can be expressed as a product of two terms from $\mathbf{f}$ and $\mathbf{b}$, $f_{l-1}b_{l+1}$ for any l. Since it takes $\mathcal{O}(k)$ time to compute the forward and backward cumulative product arrays and $\mathcal{O}(1)$ time to obtain each partial product $\prod_{j\in[k]\setminus l} v_j(\delta)$, for each value of δ, the total time taken to simultaneously calculate the values of $\{\operatorname{Per} B^{\diamond}_{k,l}\}$ is $\mathcal{O}(k2^k)$. Additional space required to store the partial products is $\mathcal{O}(k)$.

Coming back to formulating the second Boson Sampling algorithm, the sample space is now expanded even further with an auxiliary array $\alpha = (\alpha_1, \alpha_2, ..., \alpha_n)$ where $\alpha \in \pi(n)$, which is the set of permutation of $[n]$. The approach is similar to that of Algorithm A in that the goal is to create a succession of pmfs for leading subsequences of $\mathbf{r}$ and then progressively sampling r_i from conditional pmfs on $(r_1, ..., r_{i-1})$. The conditioning variable α helps us reformulate the pmf in such a way that we can use the permanent minors that we are able to calculate efficiently, while still ensuring that it is equivalent to sampling from the pmf $p(\mathbf{r})$ in equation 6. Define

$$\phi(r_1, ...r_k|\alpha) = \frac{1}{k!} \left|\operatorname{Per} A^{[n]\setminus\{\alpha_{k+1},...,\alpha_n\}}_{r_1,...r_k}\right|^2, \quad k = 1, ..., n-1. \tag{13}$$

In Lemma 3 of the paper, it is shown that $p(\mathbf{r}) = \mathbb{E}_\alpha\{\phi(\mathbf{r}|\alpha)\}$, which is the expectation taken over α, uniformly distributed over $\pi(n)$ for fixed $\mathbf{r}$. The chain rule of expectation is used to prove the equality, and this full proof is provided in the literature. As in Algorithm A, a chain of conditional pmfs as follows is used,

$$\phi(\mathbf{r}|\alpha) = \phi(r_1|\alpha)\phi(r_2|r_1, \alpha)\phi(r_3|r_1, r_2, \alpha)...\phi(r_n|r_1, ...r_{n-1}, \alpha), \tag{14}$$

where $\phi(r_k|r_1, ...r_{k-1}, \alpha) = \phi(r_1, r_2, ...r_k|\alpha)/\sum_{r_k} \phi(r_1, ...r_k|\alpha)$, using conditional probabilities and the law of total probability. The algorithm starts off by sampling r_1 from the pmf $\phi(r_1|\alpha)$. Then for stages $k = 2, ..., n$, r_k is sampled from $\phi(r_k|r_1, ...r_{k-1}, \alpha)$ with $(r_1, ..., r_{k-1})$ fixed. Since the terms in the denominator are fixed, known values, sampling from $\phi(r_k|r_1, ...r_{k-1}$ is equivalent to sampling from the numerator, which can be evaluated by taking advantage of the Laplace Expansion of permanents. At stage n, the array $(r_1, ..., r_n)$ will have been sampled from $p(\mathbf{r})$. Sorting $(r_1, ..., r_n)$ in non-decreasing

Algorithm 2 Boson Sampler: Single sample $\mathbf{z}$ from $q(\mathbf{z})$ in $\mathcal{O}(n2^n + \text{poly}(m,n))$ time

Require: $m, n \in \mathbb{Z}_+$; A formed by first n columns of $m \times m$ Haar random unitary matrix

$\mathbf{r} \leftarrow \emptyset$ ▷ Empty array
$A \leftarrow \text{Permute}(A)$ ▷ Randomly permute columns of A
$w_i \leftarrow |a_{i,1}|^2, i \in [m]$ ▷ Make indexed weighted array w
$x \leftarrow \text{Sample}(w)$ ▷ Sample index x from w
$\mathbf{r} \leftarrow (\mathbf{r}, x)$ ▷ Append x to $\mathbf{r}$
for $k \leftarrow 2$ to n **do**
    $B_k^{\diamond} \leftarrow A_{\mathbf{r}}^{[k]}$
    Compute $\{\text{Per}\, B_{k,l}^{\diamond}, l \in [k]\}$ ▷ From Lemma 2
    $w_i \leftarrow \left|\sum_{l=1}^{k} \text{Per}\, B_{k,l}^{\diamond}\right|^2, i \in [m]$ ▷ Using Laplace Expansion
    $x \leftarrow \text{Sample}(w)$
    $\mathbf{r} \leftarrow (\mathbf{r}, x)$
$\mathbf{z} \leftarrow \text{IncSort}(\mathbf{r})$ ▷ Sort $\mathbf{r}$ in non-decreasing order
return $\mathbf{z}$

order gives $\mathbf{z}$, the array representation of the multiset sampled from the Boson Sampling distribution $q(\mathbf{z})$.

Correctness of the algorithm ...

6 Implementation

7 Results

8 Conclusion

References

[1] Henryk Minc Marvin Marcus. Permanents. *The American Mathematical Monthly*, vol. 72:577–591, 1965.

[2] Herbert John Ryser. *Combinatorial Mathematics*. Mathematical Association of America, 1963.

[3] L.G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189 – 201, 1979.

[4] Albert Nijenhuis and Herbert S. Wilf. 23 - the permanent function (perman). In *Combinatorial Algorithms (Second Edition)*, pages 217 – 225. Academic Press, second edition edition, 1978.

[5] K Balasubramanian. *Combinatorics and diagonals of matrices.* PhD thesis, Indian Statistical Institute, Calcutta, Dec 1980.

[6] Eric Bax. *Finite-difference algorithms for counting problems.* PhD thesis, California Institute of Technology, February 1998.

[7] Eric Bax and Joel Franklin. A finite-difference sieve to count paths and cycles by length. *Information Processing Letters*, 60(4):171 – 176, 1996.

[8] David G. Glynn. The permanent of a square matrix. *European Journal of Combinatorics*, 31(7):1887 – 1891, 2010.

[9] Peter Clifford and Raphaël Clifford. The classical complexity of boson sampling. *CoRR*, abs/1706.01260, 2017.

[10] William Feller. *An Introduction to Probability Theory and its Applications*, volume 1. Wiley, 1968.

[11] A. J. Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, April 1974.