

# 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

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Background</b>	<b>3</b>
<b>3</b>	<b>Preliminaries</b>	<b>3</b>
3.1	Permanent of a matrix . . . . .	3
3.1.1	Definition . . . . .	3
3.1.2	Computing the permanent . . . . .	3
<b>4</b>	<b>Describing the paper and specifying the problem</b>	<b>4</b>
4.1	Explaining the problem . . . . .	4
4.1.1	Calculating the size of the sample space . . . . .	5
<b>5</b>	<b>The Boson Sampling Algorithm</b>	<b>5</b>
5.1	The naive approach . . . . .	5
5.2	Algorithm A . . . . .	5
<b>6</b>	<b>Implementation</b>	<b>7</b>
<b>7</b>	<b>Results</b>	<b>7</b>
<b>8</b>	<b>Conclusion</b>	<b>7</b>
	<b>References</b>	<b>7</b>

# 1 Introduction

# 2 Background

# 3 Preliminaries

## 3.1 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.

### 3.1.1 Definition

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

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

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

**Example 3.2.** For a  $2 \times 2$  matrix, the permanent is calculated as follows

$$\text{Per} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad + bc \quad (2)$$

**Example 3.3.** For a  $3 \times 3$  matrix, the permanent is calculated as follows

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

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.  $\text{Per } A = \text{Per } A^T$  [2].

### 3.1.2 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, Ryser's algorithm uses a variant of the inclusion-exclusion principle and can be evaluated in  $\mathcal{O}(2^n n^2)$  time [2]. Nijenhuis and Wilf sped this up to  $\mathcal{O}(2^n n)$  time by iterating over the sum in Gray Code order [4].

Another formula that is as fast as Ryser's 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

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

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

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

## 4 Describing the paper and specifying the problem

Clifford 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<sup>1</sup> of size  $n$  with elements in  $[m]$ , where  $[m] = \{1, \dots, m\}$ . Let  $z = [z_1, z_2, \dots, z_n]$  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  $z$ .

$A = (a_{ij})$  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  $k^{\text{th}}$  row of  $A_z$  is the  $z_k^{\text{th}}$  row in  $A$ , for  $k = 1, \dots, n$ . Finally, define a probability mass function over  $\Phi_{m,n}$  as

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

---

<sup>1</sup>A multiset is a special kind of set in which elements can be repeated

where  $\text{Per } A_z$  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  $\mathcal{O}(n 2^n)$  time (Section 3.1.2). 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  $\mathbf{r} = (r_1, r_2, \dots, r_n)$  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

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

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

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

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

We first sample  $r_1$  from  $p(r_1), r_1 \in [m]$ . Then for  $k = 2, \dots, n$ , we sample  $r_k$  from the conditional pmf  $p(r_k | r_1, r_2, \dots, r_{k-1})$  with  $r_1, \dots, r_{k-1}$  fixed. After sampling all values of  $r_k$ , sort  $(r_1, r_2, \dots, r_n)$  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  $(r_1, \dots, r_k)$  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:

$$p(r_1, \dots, r_k) = \frac{(n-k)!}{n!} \sum_{c \in \mathcal{C}_k} |\text{Per } A_{r_1, \dots, r_k}^c|^2, \quad k = 1, \dots, n \quad (8)$$

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

Notice in equation 7, we need to sample  $r_k$  from the conditional probability distribution  $p(r_k | r_1, \dots, r_{k-1})$ . This can be rewritten as  $p(r_1, \dots, r_k) / p(r_1 \dots r_{k-1})$ . 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  $(r_1, \dots, r_{k-1})$  are fixed, known values at this stage in the algorithm. Therefore, the formula in 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  $k \leftarrow 1$  to  $n$  do
3:    $w_i \leftarrow \sum_{c \in \mathcal{C}_k} |\text{Per } A_{(\mathbf{r}, i)}^c|^2, i \in [m]$ 
4:    $x \leftarrow \text{Sample}(w)$ 
5:    $\mathbf{r} \leftarrow (\mathbf{r}, x)$ 
6: end for
7:  $\mathbf{z} \leftarrow \text{IncSort}(\mathbf{r})$ 
8: return  $\mathbf{z}$ 
```

---

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  $|\mathcal{C}_k| \times k$  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  $k^{\text{th}}$  iteration is  $\mathcal{O}(m \binom{n}{k} k 2^k)$ , 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,

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

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

## 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 ALBERT NIJENHUIS and HERBERT S. WILF, editors, *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.