18.338 report: RMT and boson computers

John Napp

May 11, 2016

Abstract

In 2011, Aaronson and Arkhipov [1] proposed a simple model of quantum computation based on the statistics of noninteracting bosons. Physically, this model of computation could be realized as a linear optics experiment in which n photons are sent through a linear-optical network consisting of phaseshifters and beamsplitters before arriving at m possible photodetectors. The output of the computer is simply how many photons are measured at each photodetector, and is a sample from a probability distribution $\mathcal{D}_A$ which is a function of how the optical network is laid out. Aaronson and Arkhipov prove that, *assuming two very plausible conjectures about random matrices*, if a classical algorithm can approximately sample from $\mathcal{D}_A$, then there would be consequences for computational complexity theory that are widely believed to not be true. Hence, if the conjectures are true, this work provides strong evidence that quantum computers can efficiently solve certain problems that classical computers cannot. This report is mainly an exposition of the original paper.

1 Introduction

Randomness plays a central role in classical information theory, and it also appears in contexts of quantum information theory as well. And since the time evolution of quantum states is encoded by unitary matrices, one can see that it's not surprising that random matrix theory appears in quantum information theory. In this report, I will give an exposition of one particular instance in which random matrix theory appears in the context of quantum computing and quantum complexity theory. Namely, I will consider a certain model of quantum computing proposed by Aaronson and Arkhipov in 2011 [1] which is based on noninteracting bosons. In practice, this type of computer could be realized as a linear optics experiment, since photons are bosons. This model is interesting because despite its simplicity, one can show that it is impossible for a classical computer to simulate a boson computer without drastic consequences for complexity theory, *assuming two very plausible conjectures from random matrix theory.*

The model is "simple" because such a boson computer would be much easier to build than a universal quantum computer. Indeed, the boson computer's operation simply consists

of sending photons through a linear-optical network of beamsplitters and phaseshifters, and then measuring the number of photons in each mode at the end. It's believed that such a computer could not even do universal classical computation, let alone quantum computation. However, assuming the random matrix conjectures are true, [1] arguably provides the strongest evidence to date that quantum computers possess capabilities beyond those of probabilistic classical computers. For example, Shor's algorithm [3] is an efficient quantum algorithm for factoring, a problem not believed to be efficiently solvable classically. However, few believe that the classical factoring algorithms we have cannot be improved, and it is not even totally beyond reasonable doubt that a polynomial-time classical factoring algorithm exists. In particular, we have sub-exponential-time classical algorithms for factoring, and factoring is not believed to be NP-hard. In contrast, the Aaronson-Arkhipov result directly shows that efficient simulation of boson computers would imply a drastic consequence for complexity theory – namely, that $\mathsf{P}^{\#\mathsf{P}} = \mathsf{BPP}^{\mathsf{NP}}$, which implies that the polynomial hierarchy collapses to the third level. This is not quite as drastic as $\mathsf{P} = \mathsf{NP}$, but in some sense is not too far away. Now we state some definitions and the conjectures that this result relies on. Note that $z \sim \mathcal{N}(0,1)_{\mathbb{C}}$ means that z is chosen over the complex Gaussian distribution with $\mathrm{E}[z] = 0$ and $\mathrm{E}[|z|^2] = 1$.

Problem 1.1 ($|\mathrm{GPE}|^2_{\pm}$)**.** *Given as input a matrix $X \sim \mathcal{N}(0,1)^{n \times n}_{\mathbb{C}}$ of iid Gaussians, together with error bounds $\varepsilon, \delta > 0$, estimate $|\mathrm{Per}(X)|^2$ to within additive error $\pm\varepsilon \cdot n!$, with probability at least $1 - \delta$ over X, in poly$(n, 1/\varepsilon, 1/\delta)$ time.*

Problem 1.2 ($\mathrm{GPE}_{\times}$)**.** *Given as input a matrix $X \sim \mathcal{N}(0,1)^{n \times n}_{\mathbb{C}}$ of iid Gaussians, together with error bounds $\varepsilon, \delta > 0$, estimate $\mathrm{Per}(X)$ to within $\pm\varepsilon \cdot |\mathrm{Per}(X)|$, with probability at least $1 - \delta$ over X, in poly$(n, 1/\varepsilon, 1/\delta)$ time.*

Conjecture 1.3 (Permanent-of-Gaussians Conjecture (PGC))**.** $\mathrm{GPE}_{\times}$ *is $\#\mathsf{P}$-hard.*

Conjecture 1.4 (Permanent Anti-Concentration Conjecture (PACC))**.** *There exists a polynomial p such that for all n and $\delta > 0$,*

$$\Pr_{X \sim \mathcal{N}(0,1)^{n \times n}_{\mathbb{C}}} \left[|\mathrm{Per}(X)| < \frac{\sqrt{n!}}{p(n, 1/\delta)} \right] < \delta$$

Section 2 gives relevant background in quantum computing. Section 3 gives relevant background in complexity theory. Section 4 outlines the central Aaronson-Arkhipov result. Section 5 discusses the RMT conjectures in more detail.

2 Quantum computing preliminaries

We now give some essential quantum computing preliminaries. A state of a quantum computer is described by a ray in a Hilbert space. A Hilbert space can be thought of as a vector space over $\mathbb{C}$ with some additional properties. We will denote vectors by $|\psi\rangle$ (bra-ket notation). A Hilbert space has an inner product which maps a pair of vectors $\langle\psi|\varphi\rangle$ to $\mathbb{C}$, and respects positivity, linearity, and skew symmetry:

- $\langle\psi|\psi\rangle > 0$ for all $|\psi\rangle \neq 0$
- $\langle\varphi| (a|\psi_1\rangle + b|\psi_2\rangle) = a\langle\varphi|\psi_1\rangle + b\langle\varphi|\psi_2\rangle$
- $\langle\varphi|\psi\rangle = \langle\psi|\varphi\rangle^*$

where $*$ denotes complex conjugation. These are the only properties of a Hilbert space that are relevant here. Since quantum states correspond to rays in the Hilbert space, the vectors $|\psi\rangle$ and $a|\psi\rangle$ correspond to the same physical state, for any nonzero $a \in \mathbb{C}$.

Observables correspond to Hermitian operators acting on the Hilbert space. Any Hermitian operator A can be decomposed as $A = \sum_n a_n P_n$ where a_n are the eigenvalues of A and P_n are the corresponding projectors. Let the system be in the normalized state $|\psi\rangle$ (that is, $\langle\psi|\psi\rangle = 1$). Then if the observable A is measured, the measurement outcome will be a_n with probability $\langle\psi| P_n |\psi\rangle$, and the resulting post-measurement state after obtaining the outcome a_n will be $P_n|\psi\rangle$.

Evolution of a quantum system corresponds to a unitary transformation U on the state: $|\psi\rangle \to U|\psi\rangle$.

The general procedure for a quantum computation is as follows. First, prepare some easy-to-prepare input state. For example, our initial state could be a row of spins with each spin pointing either up or down: $|\uparrow\cdots\uparrow\rangle$. Now, perform a series of *quantum gates* $U_1, \ldots, U_t$ on the initial state. Each quantum gate should be some easy-to-implement operation which acts on only a small number of the spins. The state after the series of gates is given by $U_t \cdots U_1 |\uparrow\cdots\uparrow\rangle$. Finally, perform a measurement on the final state in the *computational basis*. That is, the projectors are the operators $|\uparrow\cdots\downarrow\rangle\langle\uparrow\cdots\downarrow|$ for the possible sequences $\uparrow\cdots\downarrow$. Now, if our algorithm is good, the resulting state corresponds to the solution of the problem we were trying to solve with high probability.

3 Complexity theory preliminaries

We now give some essential ingredients from complexity theory, assuming basic familiarity with the subject. Recall that P is the class of languages decidable in time polynomially-large in the input length by a deterministic Turing Machine (TM). Essentially, this is the class of decision problems which can be solved efficiently by a deterministic algorithm. NP essentially corresponds to the class of decision problems for which, if the answer is 'yes', it can be efficiently verified that the answer is 'yes' given access to a proof, but it is not necessarily possible to efficiently determine if the answer is 'yes' or 'no' without help from a proof.

Another class we will use is BPP, which stands for Bounded-Error Probabilistic Polynomial-Time. This is essentially just the probabilistic analog of P. It is the class of languages for which there exists a probabilistic TM which decides the language with success probability at least 2/3 on each input. Note that this 2/3 is arbitrary, as the success probability can be boosted to a number exponentially close to 1 by running TM some constant number of times and using a majority voting procedure to determine if the correct answer is 0 or 1.

We will also encounter the class #P.

Definition 3.1 (#P)**.** *A function $f : \{0,1\}^* \rightarrow \mathbb{N}$ is in #P if there exists a polynomial $p : \mathbb{N} \rightarrow \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$:*

$$f(x) = \left|\left\{y \in \{0,1\}^{p(|x|)} : M(x,y) = 1\right\}\right|$$

Essentially, #P is the counting version of NP. For example, an example of an NP problem is to determine if there is a perfect matching in a bipartite graph. The #P version of this question is to determine *how many* perfect matchings there are in a given bipartite graph. In some sense, it is believed that #P is a much more powerful class than NP. A function is #P-complete if it is in #P, and if every other function in #P can be reduced to it in polynomial time. Valiant [7] famously proved that computing the permanent of a 0,1-matrix is a #P-complete problem.

Theorem 3.2 (Valiant [7])**.** *The following problem is #P-complete: given a matrix $X \in \{0,1\}^{n \times n}$, compute $\mathrm{Per}(X)$.*

We will need a variant of this result, proved by Aaronson-Arkhipov.

Theorem 3.3 (Aaronson-Arkhipov [1])**.** *The following problem is #P-hard, for any $g \in [1, poly(n)]$: given a matrix $X \in \mathbb{R}^{n \times n}$, approximate $\mathrm{Per}(X)^2$ to within a multiplicative factor of g.*

We will need yet another classic result from complexity theory due to Stockmeyer.

Theorem 3.4 (Stockmeyer [4])**.** *Given an efficiently-computable Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, let*

$$p = \Pr_{x \in \{0,1\}^n}[f(x) = 1] = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} f(x).$$

Then for all $g \geq 1 + \frac{1}{poly(n)}$, there exists a $\mathsf{BPP}^{\mathsf{NP}}$ machine that approximates p to within a multiplicative factor of g.

The notation $\mathsf{BPP}^{\mathsf{NP}}$ denotes a BPP machine with access to an NP *oracle*, where an NP oracle is essentially a black box that can decide membership in any language $L \in \mathsf{NP}$ in a single computational step. Equivalently, one can think of the oracle as a black box which takes as input a 3CNF formula and in a single computational step outputs 1 if the formula is satisfiable and 0 if it is not satisfiable (recall that determining the satisfiability of 3CNF formulas is an NP-complete problem). So, a $\mathsf{BPP}^{\mathsf{NP}}$ machine is a probabilistic TM with access to an NP oracle, which runs in time polynomial in the length of the input and outputs a correct answer with probability at least 2/3. Note that Theorem 3.4 can be interpreted as saying that a $\mathsf{BPP}^{\mathsf{NP}}$ machine can estimate the probability of acceptance of any BPP machine.

4 BosonSampling

We are now ready to define the computational model of the boson computer, following [1]. The computer is built out of linear-optical elements. The particles involved in the computation are photons, which are a type of boson. There are n particles in m possible modes (a mode can simply be thought of as a place that a photon can be). We assume that $n \leq m \leq poly(n)$. Photons are never created or destroyed, and any mode can have any nonnegative integer number of photons. (This feature is unique to bosons as compared to fermions – by the Pauli exclusion principle there can be no more than one identical fermion in the same state, but there can be an unlimited number of bosons in the same state). The computational basis states of the computer can be written as $|S\rangle = |s_1, \ldots, s_m\rangle$ where s_i is the number of photons in mode i. Further, S must satisfy $S \in \Phi_{m,n}$ where $\Phi_{m,n}$ is the set of tuples $S = (s_1, \ldots, s_m)$ such that each $s_i \geq 0$ and $s_1 + \cdots + s_m = n$. A measurement consists of counting how many photons there are in each mode (eg with a photodetector), which projects the state onto a computational basis state.

The boson computer works as follows. It starts in the basis state $|1_n\rangle \equiv |1, \ldots, 1, 0, \ldots, 0\rangle$ where there is exactly one photon in modes 1 through n, and modes $n+1$ through m have no photons. It then applies a unitary transformation by applying some sequence of phaseshifters and beamsplitters. Finally, a projective measurement in the computational basis is performed as described above.

Let $\mathcal{U}$ be the unitary matrix corresponding to this sequence of phaseshifters and beamsplitters for the case of 1 photon. Note that in this case, there are m computational basis states and so $\mathcal{U}$ is $m \times m$. Define the matrix A to be the $m \times n$ matrix obtained by keeping only the first n columns of $\mathcal{U}$. Now, let $S \in \Phi_{m,n}$, and define the matrix A_S as follows. If $S = (s_1, \ldots, s_m)$, take s_i copies of the i'th row of $\mathcal{U}$ for all $i \in [m]$. Hence, A_S is an $n \times n$ matrix. Now, if $\mathcal{D}_A$ is the probability distribution corresponding to the outputs of the boson computer upon measuring in the computational basis, we have

$$\Pr_{\mathcal{D}_A}[S] = \frac{|\mathrm{Per}(A_S)|^2}{s_1! \cdots s_m!} \tag{1}$$

The derivation of this result is beyond the scope of this overview, but is essentially a consequence of the statistics of identical bosons. See [1] for details.

4.1 Exact BosonSampling

The BosonSampling problem is to sample from $\mathcal{D}_A$, given A as input (note that A fully specifies the distribution of the boson computer). We now give an outline of the proof of how a BosonSampling oracle $\mathcal{O}$ would allow one to compute the square of an arbitrary $\mathbb{C}^n \times \mathbb{C}^n$ permanent in $\mathsf{BPP}^{\mathsf{NP}}$, but first we record a few ingredients which are necessary for the proof. By BosonSampling oracle, we mean an oracle which takes a string $r \in \{0,1\}^{poly(n)}$ and an $m \times n$ matrix A specifying the boson computer whose distribution over r chosen uniformly at random is equal to $\mathcal{D}_A$. Note that the oracle is a deterministic function of r and A – repeatedly querying the oracle with the same values of r and A will result in the same

output. For the purposes of this report, the phrases "BosonSampling oracle" and "efficient classical algorithm for sampling from the output distribution of a boson computer" can be used interchangeably.

We also need the following lemma, also proved in the original BosonSampling paper:

Lemma 4.1 (Aaronson-Arkhipov [1]). *Let $X \in \mathbb{C}^{n \times n}$. Then for all $m \geq 2n$ and $\varepsilon \leq 1/\|X\|$, there exists an $m \times m$ unitary matrix U which can be computed in polynomial time that contains εX as a submatrix.*

Given all of the requisite ingredients, the desired result follows fairly straightforwardly:

Theorem 4.2. *For any* BosonSampling *oracle $\mathcal{O}$, $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{BPP}^{\mathsf{NP}^{\mathcal{O}}}$. [In other words, if there exists an efficient classical algorithm for sampling from the output distribution of a boson computer, then $\mathsf{P}^{\#\mathsf{P}} \subseteq \mathsf{BPP}^{\mathsf{NP}}$.]*

Proof. Given a matrix $X \in \mathbb{R}^{n \times n}$, we show how to approximate the squared permanent of X in $\mathsf{BPP}^{\mathsf{NP}^{\mathcal{O}}}$. By Theorem 3.3 this is a #P-hard task, and so the result follows. The strategy is to embed X into a unitary matrix corresponding to a boson computer, so that the squared permanent of X corresponds to the probability of the boson computer outputting $|1_n\rangle$. Then by Stockmeyer's result, if one has an oracle for BosonSampling, then one can approximate this probability in $\mathsf{BPP}^{\mathsf{NP}}$ and hence approximate the squared permanent of X.

More explicitly, let $m \equiv 2n$ and $\varepsilon \equiv 1/\|X\|$. Let U be a $m \times m$ unitary matrix whose $n \times n$ upper-left submatrix $U_{n,n}$ is equal to εX. Such a U exists by Lemma 4.1. Let A be the $m \times n$ submatrix of U corresponding to selecting the first n columns. Note that A can be interpreted as a description of a boson computer. Let p_A be the probability that the boson computer outputs $|1_n\rangle$. Now we have

$$p_A = \Pr_r[\mathcal{O}(A, r) = 1_n] = |\mathrm{Per}(U_{n,n})|^2 = \varepsilon^{2n} |\mathrm{Per}(X)|^2$$

The first equality is by definition of the BosonSampling oracle, the second follows from Equation 1, and the third follows from the embedding of X into U. But by Theorem 3.3, we can approximate p_A in $\mathsf{BPP}^{\mathsf{NP}^{\mathcal{O}}}$, and hence we can approximate $|\mathrm{Per}(X)|^2$ in the same. But by Theorem 3.3 approximating this quantity is #P-complete, and so the desired result follows. □

This result does *not* imply that a boson computer can solve a #P-complete problem. In particular, the permanent that one is trying to compute corresponds to a probability that is exponentially small. Hence, one would need an exponential number of samples to get a good estimate of it. This is why the deterministic nature of the BosonSampling oracle is crucial for the above proof.

However, there is actually a problem in the above proof. Namely, the BosonSampling oracle $\mathcal{O}$ is assumed to be able to exactly sample from the true distribution, $\mathcal{D}_A$. But this is not a reasonable requirement for the BosonSampling oracle, because in reality there will be at least a small amount of noise in the boson computer, so even the boson computer itself cannot sample exactly from $\mathcal{D}_A$. Hence, we should allow the BosonSampling oracle to sample

from some distribution $\mathcal{D}'_A$ that is close to $\mathcal{D}_A$ in total variation distance: $\|\mathcal{D}'_A - \mathcal{D}_A\| \leq \varepsilon$ for some small error bound ε. But unfortunately, the above proof completely breaks down when we allow even a tiny amount of error in the distribution that the oracle samples from. This is because the proof relies on estimating probabilities that are potentially exponentially small, and therefore if the oracle is adversarial and could figure out where in A we embedded the matrix whose permanent we want to estimate, it could concentrate its error on the particular measurement whose probability corresponds to that permanent! In the next section we see how this problem can be remedied with random matrices.

4.2 Approximate BosonSampling and RMT

We now move onto the much more challenging case of approximate BosonSampling. Specifically, [1] shows that if one had a fast classical algorithm for approximate BosonSampling, then a $\mathsf{BPP}^{\mathsf{NP}}$ machine would be able to solve the $|\mathrm{GPE}|^2_{\pm}$ problem. Note that assuming the two conjectures stated in the introduction are true, this would imply that a $\mathsf{BPP}^{\mathsf{NP}}$ machine could solve #P-hard problems, which is widely believed by complexity theorists to not be the case. The high-level proof strategy is as follows. One has a matrix X drawn from the distribution $\mathcal{G}^{n\times n}$, where $\mathcal{G}^{n\times n}$ is the probability distribution over $n \times n$ complex matrices whose entries are independent Gaussians with mean 0 and variance 1. One would like to compute $|\mathrm{Per}(X)|^2$ to within additive error $\pm\varepsilon \cdot n!$ with probability at least $1-\delta$ over X, in $poly(n, 1/\varepsilon, 1/\delta)$ time. We can use a $\mathsf{BPP}^{\mathsf{NP}}$ machine to take X as input, and output a matrix $A \in \mathcal{U}_{m,n}$, where m is a polynomial function of n, such that X appears as a random $n \times n$ submatrix of A, and the distribution over A is $\mathcal{H}_{m,n}$. Here, $\mathcal{U}_{m,n}$ denotes the set of $m \times n$ column-orthogonal complex matrices and $\mathcal{H}_{m,n}$ denotes the Haar measure over $m \times n$ column-orthogonal matrices. Actually, the above procedure will fail with some small probability. But since the $|\mathrm{GPE}|^2_{\pm}$ problem allows failure on some small fraction of instances, this is not a problem. From this point, one can use the approximate BosonSampling oracle to approximate the permanent of the embedded matrix X to within the needed additive error. Since A is distributed as $\mathcal{H}_{m,n}$, even an 'adversarial' oracle could not in general cause a substantial amount of error in the estimation of the permanent of the embedded matrix.

We omit most of the proofs because many of them are quite lengthy and the point of this report is to just give a taste of the BosonSampling problem (the original paper is about 100 pages). One RMT result that is needed concerns truncations of Haar-random unitaries. Specifically, *any $m^{1/6} \times m^{1/6}$ submatrix of an $m \times m$ Haar-random unitary matrix is close, in variation distance, to a matrix of iid Gaussians.* Despite the fact that truncations of Haar-random unitaries have appeared frequently in the RMT literature, this result was first proved by Aaronson-Arkhipov. More formally, define $\mathcal{S}_{m,n}$ to be the distribution over $n \times n$ matrices obtained by first drawing a unitary U from $\mathcal{H}_{m,m}$, and then outputting $\sqrt{m}U_{n,n}$, where $U_{n,n}$ is defined to be the top $n \times n$ submatrix of U. So essentially, $\mathcal{S}_{m,n}$ is just the distribution of $n \times n$ truncations of $m \times m$ Haar-unitaries, appropriately scaled up. Formally,

Theorem 4.3 ([1]). *Let $m \geq \frac{n^5}{\delta} \log^2 \frac{n}{\delta}$, for any $\delta > 0$. Then $\|\mathcal{S}_{m,n} - \mathcal{G}^{n\times n}\| = O(\delta)$.*

Proof Idea. The strategy is that they pick some value k, and show that for both distributions, the probability of a matrix having its maximal eigenvalue $\lambda_{\max}$ be larger than k is very small. They also show that amongst matrices for which $\lambda_{\max} < k$, the distributions are very close. The value of k is $O(n^2 \log \frac{n}{\delta})$. □

Actually, they need a stronger variant of this result, which they go on to prove next.

Theorem 4.4 (Haar-Unitary Hiding Theorem [1])**.** *For all $X \in \mathbb{C}^{n \times n}$ and $m \geq \frac{n^5}{\delta} \log^2 \frac{n}{\delta}$,*

$$p_S(X) \leq (1 + O(\delta)) p_G(X)$$

where p_S and p_G are the probability density functions of $\mathcal{S}_{m,n}$ and $\mathcal{G}^{n \times n}$.

Furthermore, for any $X \in \mathbb{C}^{n \times n}$, one can efficiently calculate $\zeta p_S(X)/p_G(X)$ where ζ is a constant close to 1. This allows us to generate samples from $p_S(X)$ using samples from $p_G(X)$. More formally, let p_x and q_x be two distributions over some set, and suppose that one can efficiently compute $\zeta q_x/p_x$ given x. Further, suppose that $q_x/p_x \leq 1 + \delta$ for all x, and that $|\zeta - 1| \leq \delta$. Now say we can generate samples from p, but we want samples from q. To do this, generate some sample x from p. Now, compute $\zeta q_x/p_x$, and accept with probability $\frac{\zeta q_x/p_x}{(1+\delta)^2}$. One can check that conditioned on acceptance, the output distribution is q_x. Also, the probability that the algorithm rejects is $O(\delta)$. This procedure is known as *rejection sampling*. These results lead to the Hiding Lemma, which informally says that given $X \sim \mathcal{G}^{n \times n}$, with high success probability a $\mathsf{BPP}^{\mathsf{NP}}$ machine can embed X as a submatrix of a larger $m \times n$ matrix A, such that, conditioned on the algorithm being successful, the distribution over A is the Haar measure over $m \times n$ column-orthogonal matrices $\mathcal{H}_{m,n}$. The formal statement is below, copied from the original paper.

Lemma 4.5 (Hiding Lemma ([1]))**.** *Let $m \geq \frac{n^5}{\delta} \log^2 \frac{n}{\delta}$ for some $\delta > 0$. Then there exists a $\mathsf{BPP}^{\mathsf{NP}}$ algorithm $\mathcal{A}$ that takes as input a matrix $X \sim \mathcal{G}^{n \times n}$, that "succeeds" with probability $1 - O(\delta)$ over X, and that, conditioned on succeeding, samples a matrix $A \in \mathcal{U}_{m,n}$ from a probability distribution $\mathcal{D}_X$, such that the following properties hold:*

i) $X/\sqrt{m}$ occurs as a uniformly-random $n \times n$ submatrix of $A \sim \mathcal{D}_X$, for every X such that $\Pr[\mathcal{A}(X)$ succeeds$] > 0$.

ii) The distribution over $A \in \mathbb{C}^{m \times n}$ induced by drawing $X \sim \mathcal{G}^{n \times n}$, running $\mathcal{A}(X)$, and conditioning on $\mathcal{A}(X)$ succeeding is simply $\mathcal{H}_{m,n}$.

Proof Idea. We have a sample $X \sim \mathcal{G}^{n \times n}$ that we would like to convert to a sample from $\mathcal{S}_{m,n}$. This can be accomplished by applying the rejection sampling procedure described above. Hence, after doing the rejection sampling procedure, with only a small probability of failure, we have produced a sample from $\mathcal{S}_{m,n}$. Now we claim that a candidate for $\mathcal{D}_X$ is the distribution obtained by sampling from $\mathcal{H}_{m,n}$, and then conditioning on the requirement that $X/\sqrt{m}$ appear as a submatrix. Clearly $\mathcal{D}_X$ satisfies requirement (i) – $X/\sqrt{m}$ appears as a uniformly-random $n \times n$ submatrix by symmetry. To see that (ii) is satisfied, note that

since X is a sample from $\mathcal{S}_{m,n}$, a truncation of a sample from $\mathcal{H}_{m,n}$, we can equivalently generate $\mathcal{D}_X$ by embedding the submatrix into a random location of an $m \times n$ matrix, and then randomly filling in the remaining entries up to the column-orthogonal constraint. From this perspective, it is evident that (ii) is satisfied as well.

The reason why a sample from $\mathcal{D}_X$ can be produced by a $\mathsf{BPP}^{\mathsf{NP}}$ machine given $X \sim \mathcal{S}_{m,n}$ is simple, but is beyond the scope of this report. (If you have a complexity theory background, it is simply because the machine can efficiently generate a Haar-random sample $Y \sim \mathcal{H}_{m,n}$ and then postselect on Y containing $X/\sqrt{m}$ as a submatrix. Since $\mathsf{PostBPP} \subseteq \mathsf{BPP}^{\mathsf{NP}}$, the result follows.)

□

With the Hiding Lemma, one can prove the main result – that if there exists an efficient classical algorithm for approximate BosonSampling, then $|\mathrm{GPE}|^2_{\pm} \in \mathsf{BPP}^{\mathsf{NP}}$.

Theorem 4.6 ([1])**.** *Assume there is a deterministic classical algorithm that, for all $\varepsilon > 0$, takes as input $A \in \mathcal{U}_{m,n}$ and a string $r \in \{0,1\}^{p(n)}$ where p is a polynomial, runs in time $poly(n, 1/\varepsilon)$, and outputs a tuple $S \in \Phi_{m,n}$ with distribution $\mathcal{D}'_A$ over r such that $\|\mathcal{D}'_A - \mathcal{D}_A\| < \varepsilon$ in variation distance. Then $|\mathrm{GPE}|^2_{\pm} \in \mathsf{BPP}^{\mathsf{NP}}$.*

Proof Idea. The proof is somewhat long, but the high level intuition for why this is true is as follows. Recall in Theorem 4.2, we wanted to approximate the permanent of a matrix X using a $\mathsf{BPP}^{\mathsf{NP}}$ machine, assuming we could exactly sample from the distribution of a BosonSampling computer. To do so, we simply defined a boson computer $A \in \mathcal{U}_{m,n}$ which had X embedded in it. Then the (exponentially small) probability of the computer measuring the state $(1, \ldots, 1, 0, \ldots, 0)$ was proportional to the squared permanent of X. We could estimate this probability using the $\mathsf{BPP}^{\mathsf{NP}}$ machine and the exact BosonSampling algorithm.

This fails in the approximate BosonSampling case because the approximate BosonSampling algorithm could be 'adversarial' in the sense that if it could tell where we had embedded X in A, it could concentrate the error on the probability corresponding to the squared permanent of X!

Going back to the random matrix case, using a $\mathsf{BPP}^{\mathsf{NP}}$ machine and an efficient algorithm for approximate BosonSampling, we can solve the $|\mathrm{GPE}|^2_{\pm}$ problem by first sampling a matrix $X \sim \mathcal{G}^{n \times n}$, and then use the Hiding Lemma to embed X in some matrix $A \in \mathcal{U}_{m,n}$, with only a tiny failure probability. Note that the squared permanent of X corresponds to the probability of a certain output of the boson computer corresponding to A. The key point now is that, since A is distributed as the Haar-random distribution $\mathcal{H}_{m,n}$, the approximate BosonSampling algorithm has absolutely no way of knowing where X is embedded in A. Essentially, X is smuggled into A. No matter what X is or where it is embedded in A, from the perspective of the approximate BosonSampling algorithm, A always is distributed as the Haar measure on column-orthogonal matrices. So even if the algorithm is 'adversarial', trying its best to ruin the approximation of $|\mathrm{Per}(X)|^2$, one can prove that it is always possible to approximate $|\mathrm{Per}(X)|^2$ efficiently, with only a small failure probability, up to additive error $\varepsilon \cdot n!$.

□

5 The PACC and PGC

We have seen that it is possible to solve the $|\text{GPE}|^2_{\pm}$ problem with a $\mathsf{BPP}^{\mathsf{NP}}$ machine if there exists an efficient classical algorithm for approximately sampling from the distribution of outputs of a boson computer. Now, Conjecture 1.3 (the PGC) states that it is a #P-hard problem to efficiently estimate the permanent of a random matrix $X \sim \mathcal{G}^{n\times n}$ with high success probability up to some multiplicative error (the $\text{GPE}_{\times}$ problem). Furthermore, it is proved in the original paper that assuming Conjecture 1.4 (the PACC), one can "bridge the gap" between the $\text{GPE}_{\times}$ problem and the $|\text{GPE}|^2_{\pm}$ problem. That is, if the PACC is true, then if one can solve the $|\text{GPE}|^2_{\pm}$ problem, one can solve the $\text{GPE}_{\times}$ problem. Since the PGC states that $\text{GPE}_{\times}$ is #P-hard, then if the PGC and the PACC are true, then an approximate $\mathsf{BosonSampling}$ oracle would allow a $\mathsf{BPP}^{\mathsf{NP}}$ machine to solve #P-hard problems. Specifically, it would imply $\mathsf{P}^{\#\mathsf{P}} = \mathsf{BPP}^{\mathsf{NP}}$, which would imply that the polynomial hierarchy collapses to the third level. This is a drastic consequence for complexity theory – most believe that the polynomial hierarchy does not collapse to the third level. Hence, assuming the PGC and the PACC, this is among the strongest evidence we have that quantum computers can solve problems which are intractable for classical computers. I now say just a few words about the PGC and the PACC, following [1]. For a very detailed analysis of these conjectures, consult the original paper.

5.1 PGC

A discussion of the hardness of Gaussian permanents is mostly beyond the scope of this report, requiring more complexity theory than is appropriate here. But perhaps the greatest reason to believe in the PGC is that an analogous version for finite fields is known to be true. In particular, for all $\alpha \geq 1/poly(n)$ and primes $p > (3n/\alpha)^2$, the following problem is #P-hard: given a uniform random matrix $M \in \mathbb{F}_p^{n\times n}$, output $\text{Per}(M)$ with probability at least α over M.

The PGC can be viewed as an analogous statement involving complex numbers rather than finite fields.

5.2 PACC

Recall the statement of the PACC: there exists a polynomial p such that for all n and $\delta > 0$,

$$\Pr_{X\sim\mathcal{N}(0,1)_{\mathbb{C}}^{n\times n}}\left[|\text{Per}(X)| < \frac{\sqrt{n!}}{p(n,1/\delta)}\right] < \delta$$

To understand this more intuitively, first note that the expected value of $\text{Per}(X)$ is clearly 0. Also, it is not difficult to show that

$$\mathrm{E}_{X\sim\mathcal{N}(0,1)_{\mathbb{C}}^{n\times n}}[|\text{Per}(X)|^2] = n!$$

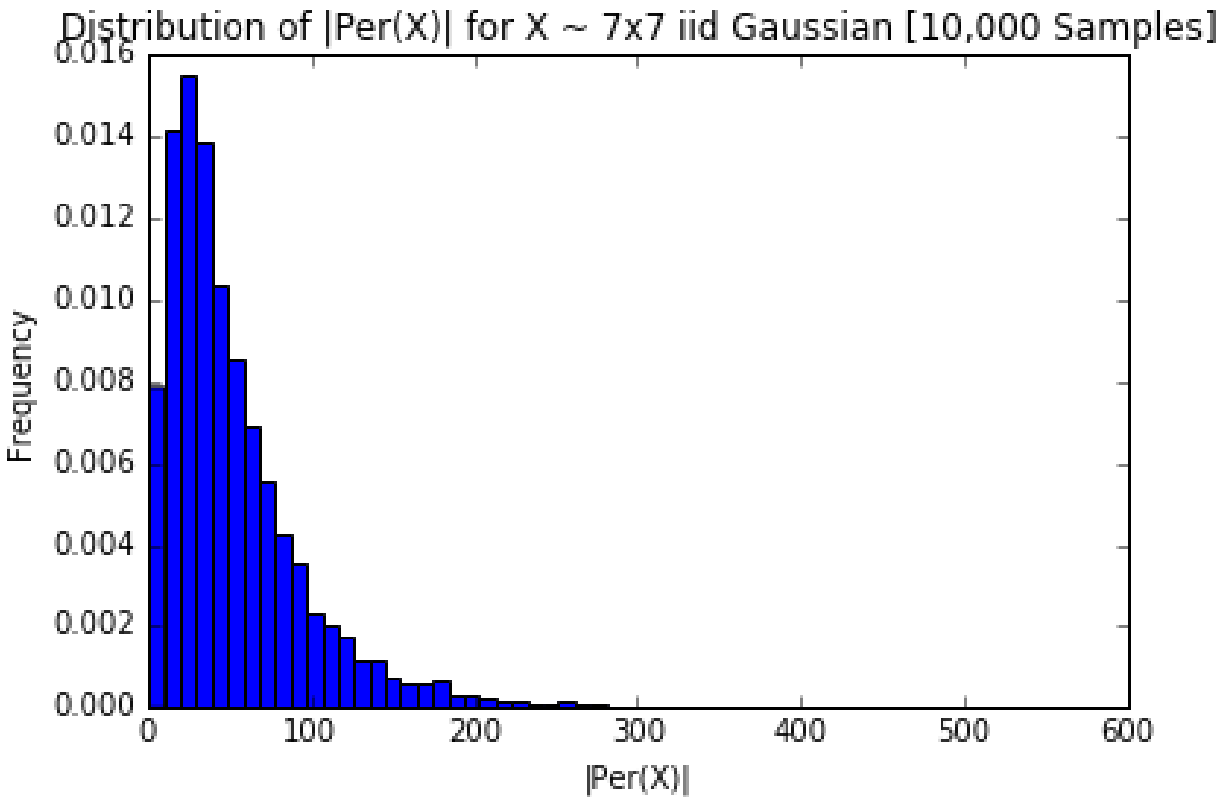

Figure 1: Distribution of $|\mathrm{Per}(X)|$ for $X \sim \mathcal{G}^{7\times 7}$. 10,000 samples were taken. Note that $\sqrt{7!} \approx 71$.

Hence, the PACC is essentially claiming that the distribution of permanents is not too close to 0 relative to the standard deviation. In other words, it is claiming that if X is an $n \times n$ random matrix with iid Gaussian entries, then with probability at least $1 - \frac{1}{poly(n)}$, the permanent of X has magnitude at least $\frac{\sqrt{n!}}{poly(n)}$ for some polynomials. See Figure 1 for a histogram of $|\mathrm{Per}(X)|$ for $X \sim \mathcal{G}^{7\times 7}$.

There is a 2009 result by Tao and Vu that is similar in spirit:

Theorem 5.1 (Tao-Vu [6])**.** *For all $\varepsilon > 0$ and sufficiently large n,*

$$\Pr_{X \in \{-1,1\}^{n\times n}}\left[|\mathrm{Per}(X)| < \frac{\sqrt{n!}}{n^{\varepsilon n}}\right] < \frac{1}{n^{0.1}}$$

While suggestive, this result is not enough to prove the PACC.

Another piece of evidence in favor of the PACC is that, if you consider the determinant instead of the permanent, then the conjecture is true, as proved in [1]. They are able to prove the result in the determinant case by exactly computing its moments. This is possible due to nice geometrical properties of the determinant which are not there for the permanent case.

As another piece of evidence in favor of the PACC, they are able to prove the following weaker version of the conjecture:

Theorem 5.2 (Weak Anti-Concentration of the Permanent [1])**.** *For all* $\alpha < 1$*,*

$$\Pr_{X \sim \mathcal{G}^{n \times n}} \left[|\mathrm{Per}(X)|^2 \geq \alpha \cdot n! \right] > \frac{(1-\alpha)^2}{n+1}.$$

Although this statement is weaker than the PACC, it at least tells us that there is a non-negligible probability of $|\mathrm{Per}(X)|$ being sufficiently large relative to its standard deviation. For example, one could imagine that perhaps almost all of the probability density is extremely close to 0, and an exponentially small amount of the probability density lies far past 0, so that the standard deviation is $\sqrt{n!}$. Theorem 5.2 tells us that this is *not* the case, which is good news for the PACC.

The proof of Theorem 5.2 is a bit too long to include in this report, but the interested reader should refer to page 78 of [1] (no physics or complexity theory is involved).

Yet another bit of evidence for the PACC is that it is supported by the numerics. For a detailed study of the numerics and how they provide evidence for the PACC, see the original paper [1].

References

[1] S. Aaronson and A. Arkhipov. The computational complexity of linear optics. In *Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 333–342, 2011.

[2] M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with non-negative entries. *J. ACM*, 51(4):671–697, 2004. Earlier version in STOC?2001.

[3] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput., 26(5):1484?1509, 1997. Earlier version in IEEE FOCS 1994. quant-ph/9508027.

[4] L. J. Stockmeyer. The complexity of approximate counting. In *Proc. ACM STOC*, pages 118–126, 1983.

[5] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[6] T. Tao and V. Vu. On the permanent of random Bernoulli matrices. *Advances in Mathematics*, 220(3):657?669, 2009. arXiv:0804.2362.

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