BosonSampling.jl: A Julia package for quantum multi-photon interferometry

Benoit Seron and Antoine Restivo

Quantum Information and Communication, Ecole polytechnique de Bruxelles, CP 165/59, Université libre de Bruxelles (ULB), 1050 Brussels, Belgium

We present a free open source package for high performance simulation and numerical investigation of boson samplers and, more generally, multi-photon interferometry. Our package is written in Julia, allowing C-like performance with easy notations and fast, high-level coding. Underlying building blocks can easily be modified without complicated low-level language modifications. We present a great variety of routines for tasks related to boson sampling, such as statistical tools, optimization methods and classical samplers. Special emphasis is put on validation of experiments, where we present novel algorithms. This package goes beyond the boson sampling paradigm, allowing for the investigation of new interferometric behaviours such as bosonic bunching.

1 Introduction

Quantum computing holds strong hopes for a profound change of computational paradigm and power in the coming years [1, 2, 3]. A large number of experimental and theoretical efforts took place in the last decades, leading to the fast growth in experiments' complexity. While a general purpose, universal quantum computer seems so far out of reach, non-universal models already claim a quantum advantage for specialized tasks on a quantum system that cannot be simulated in a reasonable amount of time with the most advanced classical super-computers [4, 5]. One of these restricted models is that of boson sampling. In their seminal paper [6], Aaronson and Arkhipov describe a quantum version of Galton's board where n bosons - generally photons - are sent through a m-modes linear interferometer implementing a unitary transformation U. The task consists in sampling output mode patterns of particles, such as finding 2 photons in the first mode, none in the second, etc. Depending on the importance of the noise sources from the experimental components, the output distribution $\mathcal{D}$ can be hard or easy to sample from [1]. By hard we understand that it takes a super-polynomial number of steps to generate a sample from $\mathcal{D}$ on a classical computer. Indeed, for modest experimental noise, AA prove that this task remains hard, under widely believed conjectures in complexity theory. However, when sufficiently strong noise is present, e.g. due to partial distinguishability or particle loss, then classical algorithms can sample from $\mathcal{D}$ efficiently [7, 8, 9, 10, 11, 12, 13].

Boson sampling sparked a great interest both from theoreticians and experimentalists. Various alternative schemes were constructed, such as

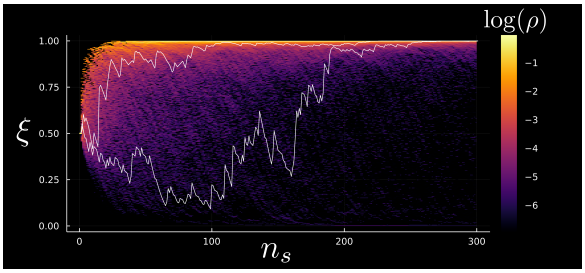

Figure 1: Validating boson sampling using Bayesian methods (see Sec. 3.4). Is displayed the confidence ξ that the input is made of 10 indistinguishable photons, averaged over $n_{trials} = 500$, as more samples n_s are provided to the validation protocol. The color scale represents the density of ξ curves. White curves are examples of validation runs.

[1] One specific requirement for the proof of hardness is that the number of modes scales as n^6, while the hardness is strongly thought to hold even with $m \approx n^2$. We refer to Boson Sampling as the sampling task described above, whether it is in the quantum advantage regime or not.

Benoit Seron: benoitseron@gmail.com
Antoine Restivo: antoine.restivo@ulb.be

Scattershot and Gaussian boson sampling (GBS) [14, 15] to alleviate the technical difficulties in generating single photons. These efforts culminated in multiple claims of quantum advantage in GBS [16, 17, 18, 19], therefore questioning the validity of the extended Church-Turing thesis [20]. Standard boson sampling saw experimental implementations with $n = 20$ photons in $m = 60$ modes [21]. Other experimental platforms than photonics were also considered [22, 23].

These factors imply a great need for optimized, high performance numerical simulations of multi-photon interferometry. As the fight for quantum advantage is played between quantum devices and classical computers, scalable, versatile and fast implementations of the boson sampling algorithms become even more important. In particular, an adaptive framework for the boson sampling community is of great interest given the speed of growth of this field of research. Indeed, many new boson sampling paradigms were introduced theoretically, and a great variety of experimental techniques were refined, enhancing the demand for evolutive code bases.

In this paper we expose our package, BosonSampling.jl, written in the Julia programming language to tackle classical tasks related to boson sampling. It intends to solve the challenges discussed above by providing an adaptable and extendable framework for multi-photon interferometry designed to study new boson sampling paradigms. The choice of Julia solves the two-language problem: instead of using a high-level wrapper (e.g. Python) with time-critical functions written in a low-level languages (e.g. C, Fortran), Julia is fast out-of-the-box while being easy to write. As a byproduct, all functions are fast, and not only time-critical ones. This is especially relevant for time-crunched researchers who are not professional-programmers but still require a fast execution time for this type of computationally intensive research. The package is evolutive and obeys the Open Source Software standards. It welcomes contributions from the community through its GitHub portal [24]. A complete documentation as well as a pedagogical tutorial are provided [25].

The paper is structured as follows. We first explain how the specific choice of the Julia programming language is, in the eyes of the authors, optimal for both theoreticians and experimentalists and how it affects the structure of our package and benefits users. In the second section of this paper, we explore the capabilities of our codebase and expose how our package is structured, including more specific descriptions and examples in the third section. It contains worked-through examples of well-known physical phenomena such as the Hong-Ou-Mandel effect. We also show how one can make use of the package architecture to take into account new sources of noise for instance. Finally, we compare our package to other existing software and discuss possible future features.

1.1 Julia

Julia is a high-level and dynamic programming language originally designed for technical computing, in particular, the simulation of physical systems. It is catching more and more attention from the scientific community and has been adopted in several high class projects such as working with dynamical systems [26], satellites simulations [27], the Celeste project (1.54 petaFLOPS) [28], the Climate Modelling Alliance [29], as well as NASA who saw a 15.000 times speed improvement over their previous Simulink/MATLAB code [30]. The field of quantum information and computation is already well covered, with several packages such as Yao.jl and Quantum Optics.jl rivalling in scope and efficiency with the leaders of the market [31, 32, 33, 34].

The main strength of Julia, in particular for scientists, is its goal to solve the "two-languages problem". It is very common that code is first prototyped in a high-level, easy to write, language such as Python and then made fast to execute in low-level, and hard to write languages such as C++. Julia allows the coding to be convenient and efficient, while being as fast as lower level languages (such as C and Fortran) out of the box. This performance comes from its just-in-time (JIT) compilation and specific type architecture, more akin to functional programming than Object Oriented Programming (OOP) found in most languages (C++, Java, Python,...). This bottom-up approach is especially suited for scientists, whose code evolve organically as results are found, without a set roadmap ("proofs then theorems") as would be preferred for OOP programming [35].

Another strength of Julia's type structure is multiple dispatch: functions are written without strong restrictions to the type of their arguments, and can be reused with custom types with no modifications [36] (provided that the function makes sense for these new types). For instance, a function computing the square root would be restricted to specific types (say, `Float64`) in strongly typed languages. A new type, say `BigFloat`, would require a rewriting of the function, while in Julia the same square root algorithm will work automatically. This design philosophy, is, once again, a blessing for scientists who may not know in advance what the future of their code looks like.

2 Contents and architecture

2.1 Package overview

Our package, BosonSampling.jl together with its paired package Permanents.jl, provides tools to classically simulate multi-photon interferometry experiments. This includes the well-known cases of standard boson sampling, scattershot and Gaussian boson sampling.

Our platform is aimed at high-performance computations for scientists. This is reflected in the choice of the language, Julia, and the modular architecture. Our goal is to make it both easy and fast to implement new models while maintaining all the provided methods and tools adapted to the new features with fast execution time.

The package is evolutive, and welcomes outside contributions from the community. It is intended to be a toolkit for both experimentalists and theoreticians, allowing to easily bridge between new theoretical models and their realistic implementation.

We provide state-of-the-art tools addressing relevant boson sampling problems. Let us now give an overview of these tools made available.

1. **Boson-samplers**, which allow to get samples as if performing an experiment. These samplers are based on the best known algorithms from the literature. Those samplers, include experimental noise such as photons' partial distinguishability and loss. This noise is critical as it will affect the classical hardness of simulating a given experiment.

2. **Event probability** computation routines, allow to observe specific input/output patterns. Special interest is given to noise, and in particular to a fast implementation in the case of partial distinguishability.

3. **Bunching** tools, aimed at investigating the tendency of bosons to bunch and of fermions to anti-bunch, as exemplified by the Hong-Ou-Mandel effect. We provide functions to investigate the link between bunching and matrix permanent conjectures [37, 38].

4. **Validation** tools for boson-samplers. These allow, given experimental as well as black-box samples, to discuss the degree of multi-photon interference observed in a boson sampler device, and to estimate the level of noise. This gives indications as to whether the device outputs samples that are classically hard to obtain, i.e. if quantum advantage can be claimed. Standard tools from the literature are unified in a recent validation protocol [39] along with the common method of correlators [40, 41], suppression laws [42, 43, 44, 45], and full bunching [38, 46] and also allowing to recover marginal probabilities [47].

5. **Optical circuits** built from optical elements. While in the standard boson sampling problem, networks, described by a unitary matrix U, are chosen from the Haar measure, specific interferometers are also implemented such as the Fourier and Hadamard transformations. In addition, a suite of linear-optical elements is provided, so that networks can be constructed as in an experimental setting.

6. **Optimization** routines to maximize cost functions over unitary matrices. Maximizing certain properties of interferometers, such as photonic bunching, can be convenient for the design of experiments. We include algorithms from Ref [48] that can optimize cost-functions on the Haar measure.

7. **Time-bin interferometry** types and circuits to simulate the experimental boson sampling proposal of [49], where photons are separated in time bins and sent through a time-dependant beam-splitter and delay lines.

8. **Applications of boson sampling** such as quantum cryptographic tools [50, 51, 52] and

cryptocurrency proof-of-work miners [53].

9. **Fundamental many-photon interference** investigation tools, such as generic (potentially entangled) input/output multiphoton interferometry [38], generalized matrix function analysis [54] and counter-example search [55] and quantum photo-thermodynamics tools [56, 57]. Fermion sampling is also implemented.

2.2 Code architecture

The two main procedures achievable with this package are the simulation of photonic experiments and validation for multi-photon quantum experiments. We will first describe the simulation architecture and then the validation procedure.

2.2.1 Simulation

The present package takes advantage of Julia's type system through a hierarchy used to classify the building blocks of a quantum interferometry experiment, namely: a photonic `Input` state sent through an `Interferometer` and measured according to an `OutputMeasurement`. This last category also includes (classical) sampling. Those three elements are represented by abstract types, that is, they cannot be instantiated and form the backbone of the package structure. They can be seen as containers of sets of related concrete types, which can be instantiated. Concrete types will represent the specific building blocks of a given interferometric setup. For instance, an interferometer defined from a Haar distributed unitary, `RandHaar`, is a concrete type of the abstract type `Interferometer` as displayed in Fig. 2.

The latter structure is at the core of the code efficiency and tunability. Indeed, the generic programming allowed by Julia combined to the inherent type hierarchy makes the package easy to adapt depending on the user's needs. As a new model can simply be embedded as a concrete type of an existing abstract type, implementing new elements can be done easily without modifying the overall package structure, or any function that acts in the same conceptual fashion on the new type. For instance, a function adding noise could work similarly on any type of `Interferometer`, be it `RandHaar` or `Hadamard`. We will now describe the three main abtract building blocks used in simulating an experiment.

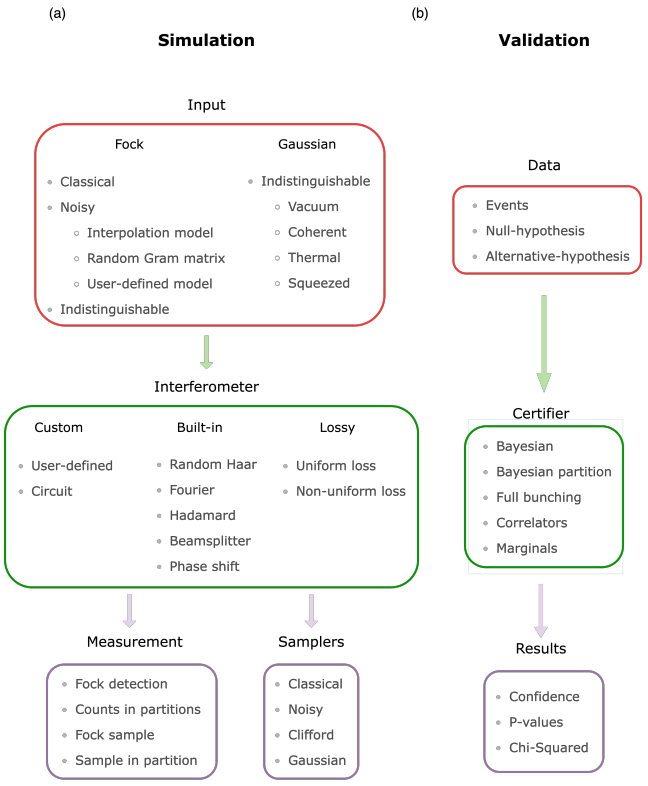

Figure 2: Architecture of BosonSampling.jl (a) Simulation framework: the abstract types representing the experimental setup are displayed as the colored boxes, containing they related concrete types. (b) Validation framework. An extensive description of every type and their usage can be found in the documentation and the associated tutorials.

Input When considering photons in Fock states at the input, BosonSampling.jl offers three families of input types depending on the partial distinguishability of the particles. We refer to indistinguishable photons (identical arrival time, polarization) using `Bosonic`. When they are partially distinguishable, they fall under `PartDist`. Finally, we use `Distinguishable` for completely distinguishable input photons. The type `PartDist` allows for a general description of the partial distinguishability. Common models used in the literature are also implemented as subtypes [58].

The input state can also be `Gaussian`. In order to take into account partial distinguishability, we adopt the model introduced in [59]. More precisely, any Gaussian input will be model of fully indistinguishable states while, partial distinguishability can be introduced at the level of the interferometer. The input state is decomposed

into interfering and non-interfering modes which evolve independently through the interferometer.

Interferometers The second building block of a quantum multi-photon experiment is the interferometer. A common practice when simulating boson sampling is sample a unitary matrix according to the Haar random measure. Specific interferometers, such as the discrete Fourier and the Hadamard transforms are built-in, together with elementary linear optical circuit elements such as beamsplitters or phase shifts. This allows the user to build networks from scratch. Generic interferometers are supported. Loss can be accounted for in generality.

Measurement/Sample This object represents the output of the experiment, which we classify between a measurement that can be the probability of a specific output pattern, property or a randomly generated sample. For Fock states, a possibility is to observe a given output mode occupation $\mathbf{s} = (s_1, s_2, ...)$, where $0 \leq s_i \leq n$ is the number of photons in mode i. Likewise, we can observe a specific partition photon count $\mathbf{k} = (k_1, k_2, ...)$, where $0 \leq k_i \leq n$ is the number photons in a bin K_i gathering multiple output modes.

Different samplers are available depending on the level of distinguishability of the input particles or the use of a lossy interferometer. For exact sampling, we provide Clifford-Clifford's algorithm [60]. For imperfect sampling, we rely on the works of Moylett et al. [61]. Their detailed implementations are discussed in the package documentation.

2.2.2 Validation

Experimental data can be analysed within this package to validate boson sampling devices. The procedure is straightforward for the user: data is held in form of `Events`, comprising all known experimental parameters, such as the detector outputs. Hypotheses are provided, such as that the input is made of indistinguishable particles. An alternative hypothesis could be that they are distinguishable. Another task that can be addressed with this package is to estimate the amount of noise, e.g due to partial distinguishability.

Common validation protocols are implemented, with a comprehensive statistical analysis, providing p-values, etc. A detailed description of these protocols can be found in [62, 39, 13, 63].

3 Examples

This section presents some basic experiments that can be simulated by BosonSampling.jl. Further details about the package, as well as a complete, step-by-step tutorial, can be found in the documentation, hosted on the GitHub page of the package. As for any registered Julia package, BosonSampling.jl can be installed by simply executing the following commands in the Julia REPL:

```
using Pkg
Pkg.add("BosonSampling")
```

Code sample 1: *Installing BosonSampling.jl.*

3.1 Hong-Ou-Mandel

It is well known since the experimental work of Hong, Ou and Mandel [64] that two indistinguishable photons impinging on the two input modes of a balanced beamsplitter will bunch in one of the two output modes, leaving a zero probability to observe one photon at each output mode (the coincidence probability). As an introductory example, we review the effect of partial distinguishability in the HOM effect. In this context, it is a common practice to use the time delay $\Delta\tau$ between the two incoming beams as a source of partial distinguishability, while the distinguishability parameter itself is $\Delta\omega\Delta\tau$ with $\Delta\omega$ the uncertainty of the frequency distribution. In order to make the parallel with the built-in interpolating model, we substitute its linear parameter x by $e^{-\Delta\omega^2\Delta\tau^2}$. In this way, a distinguishable input is recovered for $\Delta\omega\Delta\tau \to \infty$ and a bosonic input for $\Delta\tau = 0$.

We reproduce in the code sample 2 the HOM experiment as described in Fig. 3(a), with a variable overlap between the two initial wave functions (Fig. 3(b)) and compute the coincidence probability.

```
# Set experimental parameters
delta_omega = 1
# Set the model of partial distinguishability
T = OneParameterInterpolation
# Define the balanced beams-plitter
B = BeamSplitter(1/sqrt(2))
# Set each particle in a different mode
r_i = ModeOccupation([1,1])

# Define the output as detecting a coincidence
r_f = ModeOccupation([1,1])
o = FockDetection(r_f)

# Will store the events probability
events = []

for delta_t in -4:0.01:4
    # distinguishability
    dist = exp(-(delta_omega * delta_t)^2)
    i = Input{T}(r_i,dist)

    # Create the event
    ev = Event(i,o,B)
    # Compute its probability to occur
    compute_probability!(ev)

    # Store the event and its probability
    push!(events, ev)
end
```

Code sample 2: *Effect of Partial distinguishability in the Hong-Ou-Mandel effect.*

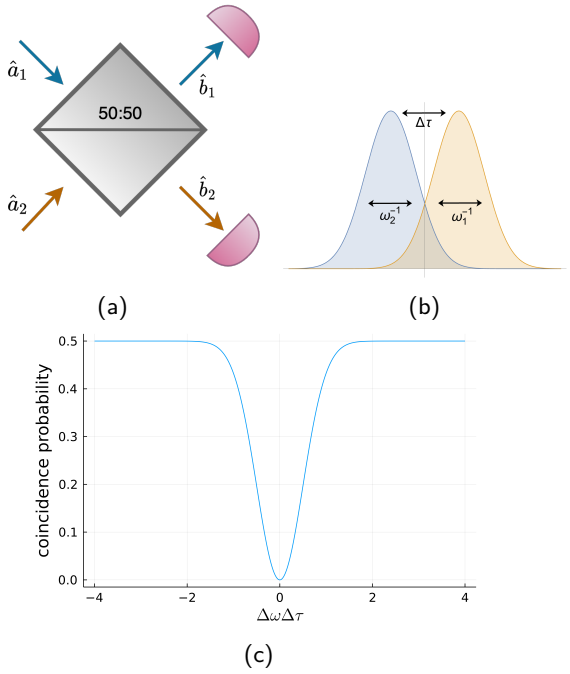

Figure 3: The Hong-Ou-Mandel effect: (a) experimental setup (b) time delay between the incoming wave packets (c) HOM dip translating the bosonic interference appears when the overlap between the beams is perfect.

3.2 Sampling

As described in Sec. 2.1, several classical algorithms for boson sampling are implemented, such as the Clifford and Clifford's algorithm [60] for exact sampling and algorithms taking into account typical sources of noise when considering realistic schemes [61]. To simulate a boson sampling experiment, one defines an input, the interferometer and a measurement. As in the previous example, an `Event` type is used as a container to hold all interferometric parameters.

```
n = 20
m = 400

# Define an input of 20 photons among 400 modes
i = Input{Bosonic}(first_modes(n,m))

# Define the interferometer
interf = RandHaar(m)

# Set the output measurement
o = FockSample()

# Create the event
ev = Event(i, o, interf)

# Simulate
sample!(ev)
# output:
# state = [0,1,0,...]
```

Code sample 3: *Simulate a perfect boson sampling experiment involving 20 single-photon Fock states in 400 modes via Clifford and Clifford's algorithm.*

Given n photons and m modes, the package proposes methods to compute the entire photon-counting distribution exactly or approximately when different sources of noise are included [12] through the `noisy_distribution` function. It returns by default the exact distribution, an approximated distribution in which the computation of the probabilities are truncated and a distribution reconstructed from a Metropolis Independent Sampler (MIS). Below we compute those three distributions for $n = 3$ photons in $m = 5$ modes with a distinguishability parameter $x = 0.74$ and a loss $l = 0.63$ (see Fig. 4).

```
# Set the model of partial distinguishability
T = OneParameterInterpolation

# Define an input of 3 photons placed
# among 5 modes
x = 0.74
i = Input{T}(first_modes(3,5), x)

# Interferometer
l = 0.63
U = RandHaar(i.m)

# Compute the full output statistics
p_exact, p_truncated, p_sampled =
noisy_distribution(input=i,loss=l,interf=U)
```

Code sample 4: *Computing the output mode occupation statistics with partial distinguishability and loss for 3 photons at the input of a (5,5) Haar distributed unitary.*

Both computing `p_truncated` and `p_sampled` have by default a probability of failure and a maximal error equal to 10^{-4}. It is still possible to refine their computations by setting the keywords `error` and `failure_probability` when calling `noisy_distribution`.

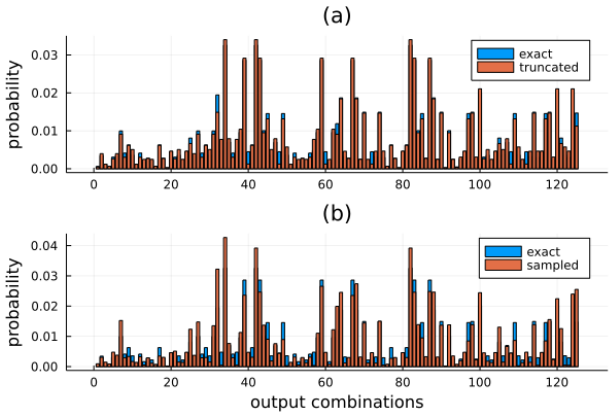

Figure 4: Photon-counting statistics at the output of a 5-mode Haar distributed unitary matrix. (a) Comparison between the ideal distribution and the approximated one. (b) Comparison between the theoretical distribution and a sampled distribution from 10^5 samples.

3.3 Photon counting in partitions

One of the novel theoretical tools implemented in this package is the ability to efficiently compute photon counting probabilities in partitions of the output modes of the interferometer. A detailed theoretical and numerical investigation can be found in Ref. [39].

Let us now show how we can obtain the probability of finding $k = 0, ..., n$ photons in a subset consisting of the first half of the modes of a Haar-randomly chosen an interferometer with $n = 25$ photons and $m = 400$ modes. With these parameters, a brute force calculation would be pratically impossible, while the new methods used in this package allow for a fast computation.

```
n = 25
m = 400

# Experiment parameters
input_state = first_modes(n,m)
interf = RandHaar(m)
i = Input{Bosonic}(input_state)

# Subset selection
s = Subset(first_modes(Int(m/2),m))
part = Partition(s)

# Want to find all photon counting probabilities
o = PartitionCountsAll(part)

# Define the event and compute probabilities
ev = Event(i,o,interf)
compute_probability!(ev)
# About 30s execution time on a single core
#
# output:
#
# 0 in subset = [1, 2,..., 200]
# p = 4.650035467008141e-8
# ...
```

Code sample 5: This snippet retrieves the probability of finding $k = 0, ..., n$ photons in the top half of the output modes, with $n = 25$ and $m = 400$. Note that this calculation would be completely out of reach using a brute-force approach.

3.4 Validation

We now show how experimental data can be validated. In this example, we compare the null hypothesis $H_0 =$ (bosonic input) to the alternative $H_a =$ (distinguishable input). Multiple validation protocols are implemented in this package, as shown for instance in Fig. 2. We choose to use the new formalism developed by some of the authors of photon counting in binned output modes.

In Fig. 1, we display an example of validation protocol. Data is assessed using a Bayesian procedure on the event probabilities (instead of partition photon counting in the code sample). As explained in detail in [63, 39], each sample updates the confidence level ξ. To be satisfied that the null hypothesis is true, we could require $\xi = 95\%$ for instance. We plot this curve averaged over $n_{trials} = 500$ with $n_{samples} = 300$ events for each run, in a randomly chosen interferometer with $m = 30$ modes and $n = 10$ photons. The input photons are set to be indistinguishable. The

heatmap shows the density of the ξ-curves, on a logarithmic scale.

3.5 Incorporating new detectors: dark counts

We now show one of the main advantages of this package and Julia: the ability to create new models while keeping most of the structure untouched.

Consider a simple model of threshold detector. With a probability p the detector clicks even though no photon was present, so-called dark count.

We start by defining a new type of output measurement that we label `DarkCountFockSample`, which is a sub-type of `OutputMeasurement`. It takes as arguments the probability p to observe a dark count. A variable holds the sampled `ModeOccupation`, but is first initialised empty at creation of `DarkCountFockSample`. It will be filled later. This is an interesting feature of Julia: the absence of an argument is integrated efficiently by the compiler while a typical, variable-size array would not.

```
mutable struct DarkCountFockSample
    <: OutputMeasurement

    s::Union{ModeOccupation, Nothing}
    p::Real

    function DarkCountFockSample(p::Real)
        if isa_probability(p)
            new(nothing, p)
        else
            error("invalid probability")
        end
    end
end
```

Code sample 6: Implementation of dark counts in the detectors in the BosonSampling framework.

The second step is to define a new method `sample!` that implements a sampling algorithm for this specific type of detector.

As mentioned before, Julia enables multiple dispatch, which means that the new method will not overwrite the already existing ones but will extend the method to the new detectors. That is, when calling `sample!`, Julia will automatically apply the relevant method for a given type (`FockSample`, `DarkCountFockSample`,...). The new algorithm first extracts a sample without dark counts, then adds their influence on the photon counts:

```
function sample!(ev::Event{TIn,TOut})
    where {TIn <: InputType,
           TOut <: DarkCountFockSample}

    # sample without dark counts
    ev_no_dark = Event(ev.input_state,
                       FockSample(),
                       ev.interferometer)

    sample!(ev_no_dark)
    sample_no_dark =
    ev_no_dark.output_measurement.s

    # add count with probablity p
    observe_dark_count(p) =
    Int(do_with_probability(p))
    dark_counts =
    [observe_dark_count(ev.output_measurement.p)
    for i in 1: ev.input_state.m]

    ev.output_measurement.s =
    sample_no_dark + dark_counts
end
```

Code sample 7: Defining a new sampling algorithm.

Now, our new measurement can be used like any other in the previous examples

```
n = 10
m = 10
p_dark = 0.1
input_state = first_modes(n,m)
interf = RandHaar(m)
i = Input{Bosonic}(input_state)
o = DarkCountFockSample(p_dark)
ev = Event(i,o,interf)

sample!(ev)
# output:
# state = [3, 1, 0, 3, 0, 1, 2, 0, 0, 1]
```

Code sample 8: Usage of the newly added detectors.

4 Benchmarks

4.1 Julia is fast!

When simulating a boson sampling experiment, the most time consuming part comes from the computation of the transition probabilities, since they are related to the evaluation of the permanent. The best known exact algorithm for its computation is due to Ryser [65] and runs in $\mathcal{O}\left(2^{n-1}n\right)$ time for a matrix of size n (using Gray ordering). Having an intensive usage of Ryser's algorithm, we compare the running time of its implementation in Julia with two other interpreted languages: Matlab and Python. Fig. 5 shows how Julia outperforms the two other implementations by two orders of magnitude in computation time.

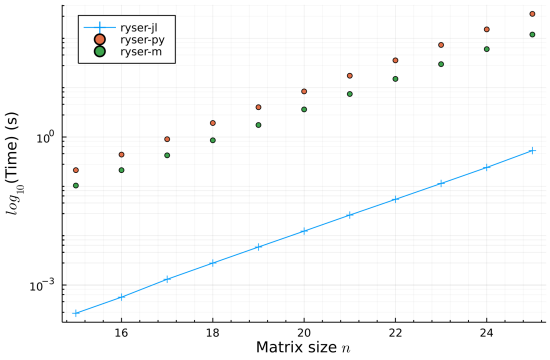

Figure 5: Benchmarks for the single-core running time of Ryser's algorithm in Julia, Matlab and Python on a 3.2GHz Apple M1 processor.

4.2 Comparison to existing softwares

The present package is designed for multi-photon interference and boson sampling experiments, taking into account sources of noise and providing a wide variety of validation tools while remaining highly flexible to new paradigms. Other packages for the numerical evaluations of matrix functions and simulation bosonic systems were published in the last few years. Among the most popular softwares are The Walrus [66] and Perceval [67], both written in Python with a C++ backend.

We illustrate how the Julia programming language is efficient and how, indeed, it can reach C-like performance while keeping the coding as easy, fast and convenient as in Python. We benchmark common functionalities in the three softwares. We first compute permanents of Haar distributed unitary matrices[2] (Fig. 6.a) showing that our implementation of Ryser's algorithm remains faster even on single-core. Our simulations of perfect boson sampling through Clifford and Clifford's algorithm achieves comparable performance to what is possible on Perceval, and is more performant only via multithreading (Fig. 6.b).

[2]Benchmarks inspired from https://the-walrus.readthedocs.io/en/latest/gallery/permanent_tutorial.html and https://github.com/Quandela/Perceval/blob/main/scripts/performance.py

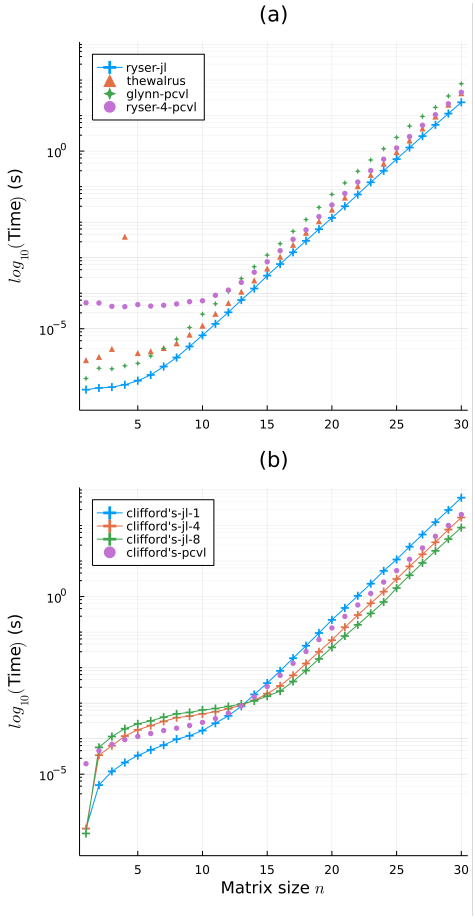

Figure 6: (a) Benchmarks measuring the average time elapsed when computing the permanent of matrices with increasing size n. Are displayed the running times of (orange curve) The Walrus, version 0.19, on a single-core, (blue curve) Ryser implementation of Permanent.jl, version 0.1.1, single-core, (green dots) Glynn implementation of Perceval, version 0.6.2 (Quandelibc version 0.5.3), on a single-core, (purple dots) Ryser implementation of Perceval, 4 threads. (b) Average time elapsed when simulating perfectly indistinguishable bosons via Clifford and Clifford's algorithm in Julia on 1, 4 and 8 threads compared to Perceval's implementation. Both benchmarks are realized with the same configuration as in Fig. 5.

In the final stages of writing this paper, SOQCS [68] was released, a C++ library centered around performance at cost of handiness and ease of modification.

5 Development path

This package is a continuously expanding, and many more features are expected to be implemented in the coming years. Current development areas are Gaussian boson sampling related functionalities, in particular simulating

GBS with partial distinguishability in a lossy channel. Many of the functions used in this package will be made faster, such as computing permanents and hafnians via GPU computing, and HPC-friendly deployment through multithreading and distributed computing.

Potential users are welcome to contact the authors for more details or specific development paths, either through the emails listed above or directly through the Github interface.

Outside contributions are welcome, and multiple improvement paths are outlined on the Github discussion page.

6 Conclusion

In this paper we introduced a new, free open source Julia package for the high-performance simulation of multi-photon interference. We motivated how the choice of language is particularly relevant for scientists, and allows for a package structure that is easy to modify while keeping execution time as fast as in low-level languages. This package is therefore aimed at experimentalists, who need to implement specifics of their hardware and at theoreticians, who want to develop new models and boson sampling paradigms. We showed how the performance positively compares to similar packages written in other languages.

Acknowledgments

The authors would like to thank Fulvio Flamini for critical reading of the manuscript. We thank Christoph Hotter and David Plankensteiner for discussions and providing the LaTeX template to display Julia code. Leonardo Banchi is also thanked for discussions and his contribution to the Permanents.jl package, a basis for the package presented in this article. Leonardo Novo and Nicolas Cerf are thanked for discussions.

B.S. is a Research Fellow of the Fonds National de la Recherche Scientifique – FNRS. A.R is part of the AppQInfo MSCA ITN which received funding from the EU Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 956071.

References

[1] Richard P Feynman. Simulating physics with computers. In *Feynman and computation*, pages 133–153. CRC Press, 2018. DOI: 10.1007/BF02650179. URL https://doi.org/10.1007/BF02650179.

[2] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994. DOI: 10.1109/SFCS.1994.365700.

[3] Max Tillmann, Borivoje Dakić, René Heilmann, Stefan Nolte, Alexander Szameit, and Philip Walther. Experimental boson sampling. *Nature Photonics*, 7(7): 540–544, may 2013. DOI: 10.1038/nphoton.2013.102. URL https://doi.org/10.1038%2Fnphoton.2013.102.

[4] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018. DOI: 10.22331/q-2018-08-06-79.

[5] John Preskill. Quantum computing 40 years later. *arXiv preprint arXiv:2106.10522*, 2021. DOI: 10.1201/9781003358817-7.

[6] Scott Aaronson and Alex Arkhipov. The computational complexity of linear optics. In *Proceedings of the forty-third annual ACM symposium on Theory of computing*, pages 333–342, 2011. DOI: 10.1145/1993636.1993682.

[7] Saleh Rahimi-Keshari, Timothy C Ralph, and Carlton M Caves. Sufficient conditions for efficient classical simulation of quantum optics. *Physical Review X*, 6(2):021039, 2016. DOI: 10.1103/PhysRevX.6.021039.

[8] Jelmer J Renema, Adrian Menssen, William R Clements, Gil Triginer, William S Kolthammer, and Ian A Walmsley. Efficient classical algorithm for boson sampling with partially distinguishable photons. *Physical review letters*, 120(22):220502, 2018. DOI: 10.1103/PhysRevLett.120.220502.

[9] Michał Oszmaniec and Daniel J Brod. Classical simulation of photonic linear optics with lost particles. *New Journal of Physics*, 20(9):092002, 2018. DOI: 10.1088/1367-2630/aadfa8.

[10] Raúl García-Patrón, Jelmer J Renema, and Valery Shchesnovich. Simulating boson sam-

pling in lossy architectures. *Quantum*, 3:169, 2019. DOI: 10.22331/q-2019-08-05-169.

[11] Daniel Jost Brod and Michał Oszmaniec. Classical simulation of linear optics subject to nonuniform losses. *Quantum*, 4:267, 2020. DOI: 10.22331/q-2020-05-14-267.

[12] Jelmer Renema, Valery Shchesnovich, and Raul Garcia-Patron. Classical simulability of noisy boson sampling. *arXiv preprint arXiv:1809.01953*, 2018. DOI: 10.48550/arXiv.1809.01953.

[13] Valery S Shchesnovich. Noise in boson sampling and the threshold of efficient classical simulatability. *Physical Review A*, 100(1):012340, 2019. DOI: 10.1103/PhysRevA.100.012340.

[14] Marco Bentivegna, Nicolò Spagnolo, Chiara Vitelli, Fulvio Flamini, Niko Viggianiello, Ludovico Latmiral, Paolo Mataloni, Daniel J. Brod, Ernesto F. Galvão, Andrea Crespi, Roberta Ramponi, Roberto Osellame, and Fabio Sciarrino. Experimental scattershot boson sampling. *Science Advances*, 1(3), apr 2015. DOI: 10.1126/sciadv.1400255. URL https://doi.org/10.1126%2Fsciadv.1400255.

[15] Craig S. Hamilton, Regina Kruse, Linda Sansoni, Sonja Barkhofen, Christine Silberhorn, and Igor Jex. Gaussian boson sampling. *Physical Review Letters*, 119(17), oct 2017. DOI: 10.1103/physrevlett.119.170501. URL https://doi.org/10.1103%2Fphysrevlett.119.170501.

[16] Zhong, Han-Sen and Wang, Hui and Deng, Yu-Hao and Chen, Ming-Cheng and Peng, Li-Chao and Luo, Yi-Han and Qin, Jian and Wu, Dian and Ding, Xing and Hu, Yi and others. Quantum computational advantage using photons. *Science*, 370(6523):1460–1463, 2020. DOI: 10.1126/science.abe8770.

[17] Han-Sen Zhong, Yu-Hao Deng, Jian Qin, Hui Wang, Ming-Cheng Chen, Li-Chao Peng, Yi-Han Luo, Dian Wu, Si-Qiu Gong, Hao Su, et al. Phase-programmable gaussian boson sampling using stimulated squeezed light. *Physical review letters*, 127(18):180502, 2021. DOI: 10.1364/IPRSN.2022.ITu3B.1.

[18] Madsen, Lars S and Laudenbach, Fabian and Askarani, Mohsen Falamarzi and Rortais, Fabien and Vincent, Trevor and Bulmer, Jacob FF and Miatto, Filippo M and Neuhaus, Leonhard and Helt, Lukas G and Collins, Matthew J and others. Quantum computational advantage with a programmable photonic processor. *Nature*, 606(7912):75–81, 2022. DOI: 10.1038/s41586-022-04725-x.

[19] Yu-Hao Deng, Yi-Chao Gu, Hua-Liang Liu, Si-Qiu Gong, Hao Su, Zhi-Jiong Zhang, Hao-Yang Tang, Meng-Hao Jia, Jia-Min Xu, Ming-Cheng Chen, et al. Gaussian boson sampling with pseudo-photon-number resolving detectors and quantum computational advantage. *arXiv preprint arXiv:2304.12240*, 2023. DOI: 10.1103/PhysRevLett.131.150601.

[20] Dominik Hangleiter and Jens Eisert. Computational advantage of quantum random sampling. *Reviews of Modern Physics*, 95 (3):035001, 2023. DOI: 10.1103/RevModPhys.95.035001.

[21] Hui Wang, Jian Qin, Xing Ding, Ming-Cheng Chen, Si Chen, Xiang You, Yu-Ming He, Xiao Jiang, L You, Z Wang, et al. Boson sampling with 20 input photons and a 60-mode interferometer in a 10^{14}-dimensional Hilbert space. *Physical review letters*, 123 (25):250503, 2019. DOI: 10.1103/PhysRevLett.123.250503.

[22] Carsten Robens, Iñigo Arrazola, Wolfgang Alt, Dieter Meschede, Lucas Lamata, Enrique Solano, and Andrea Alberti. Boson sampling with ultracold atoms. *arXiv preprint arXiv:2208.12253*, 2022. DOI: 10.48550/arXiv.2208.12253.

[23] Aaron W. Young, Shawn Geller, William J. Eckner, Nathan Schine, Scott Glancy, Emanuel Knill, and Adam M. Kaufman. An atomic boson sampler, 2023.

[24] Seron, Benoit and Restivo, Antoine. BosonSampling.jl. URL https://github.com/benoitseron/BosonSampling.jl.git.

[25] Benoît Seron and Antoine Restivo. Bosonsampling.jl documentation. URL https://docs.juliahub.com/BosonSampling/olGSq/1.0.1/.

[26] George Datseris. Dynamicalsystems.jl: A julia software library for chaos and nonlinear dynamics. *Journal of Open Source Software*, 3(23):598, 2018. DOI: 10.21105/joss.00598. URL https://doi.org/10.21105/joss.00598.

[27] Satellitetoolbox. URL https://juliaspace.github.io/SatelliteToolbox.jl/dev/.

[28] Jeffrey Regier, Kiran Pamnany, Ryan Giordano, Rollin Thomas, David Schlegel, Jon McAuliffe, et al. Learning an astronomical catalog of the visible universe through scalable bayesian inference. *arXiv preprint arXiv:1611.03404*, 2016. DOI: 10.48550/arXiv.1611.03404.

[29] Ali Ramadhan, Gregory LeClaire Wagner, Chris Hill, Jean-Michel Campin, Valentin Churavy, Tim Besard, Andre Souza, Alan Edelman, Raffaele Ferrari, and John Marshall. Oceananigans.jl: Fast and friendly geophysical fluid dynamics on gpus. *Journal of Open Source Software*, 5(53):2018, 2020. DOI: 10.21105/joss.02018. URL https://doi.org/10.21105/joss.02018.

[30] Ranjan Anantharaman, Kimberly Hall, Viral B. Shah, and Alan Edelman. Circuitscape in julia: High performance connectivity modelling to support conservation decisions. *Proceedings of the JuliaCon Conferences*, 1(1):58, 2020. DOI: 10.21105/jcon.00058. URL https://doi.org/10.21105/jcon.00058.

[31] Piotr Gawron, Dariusz Kurzyk, and Łukasz Pawela. QuantumInformation.jl—a julia package for numerical computation in quantum information theory. *PLOS ONE*, 13 (12):e0209358, dec 2018. DOI: 10.1371/journal.pone.0209358. URL https://doi.org/10.1371/journal.pone.0209358.

[32] Xiu-Zhe Luo, Jin-Guo Liu, Pan Zhang, and Lei Wang. Yao.jl: Extensible, efficient framework for quantum algorithm design. *Quantum*, 4:341, oct 2020. DOI: 10.22331/q-2020-10-11-341.

[33] Sebastian Krämer, David Plankensteiner, Laurin Ostermann, and Helmut Ritsch. QuantumOptics.jl: A julia framework for simulating open quantum systems. *Computer Physics Communications*, 227:109–116, jun 2018. DOI: 10.1016/j.cpc.2018.02.004.

[34] David Plankensteiner, Christoph Hotter, and Helmut Ritsch. QuantumCumulants.jl: A julia framework for generalized mean-field equations in open quantum systems. *Quantum*, 6:617, jan 2022. DOI: 10.22331/q-2022-01-04-617.

[35] Jeff Bezanson, Stefan Karpinski, Viral B. Shah, and Alan Edelman. Julia: A fast dynamic language for technical computing. 2012. DOI: 10.48550/ARXIV.1209.5145. URL https://arxiv.org/abs/1209.5145.

[36] Avik Sengupta. *Julia High Performance*. Packt, 2019. ISBN 978-1-78829-811-7.

[37] Benoît Seron, Leonardo Novo, and Nicolas J. Cerf. Boson bunching is not maximized by indistinguishable particles, 2022. URL https://arxiv.org/abs/2203.01306.

[38] VS Shchesnovich. Universality of generalized bunching and efficient assessment of boson sampling. *Physical review letters*, 116(12):123601, 2016. DOI: 10.1103/PhysRevLett.116.123601.

[39] Benoit Seron, Leonardo Novo, Alex Arkhipov, and Nicolas J Cerf. Efficient validation of boson sampling from binned photon-number distributions. *arXiv preprint arXiv:2212.09643*, 2022. DOI: 10.48550/arXiv.2212.09643.

[40] Mattia Walschaers, Jack Kuipers, Juan-Diego Urbina, Klaus Mayer, Malte Christopher Tichy, Klaus Richter, and Andreas Buchleitner. Statistical benchmark for bosonsampling. *New Journal of Physics*, 18(3):032001, 2016. DOI: 10.1088/1367-2630/18/3/032001.

[41] Mattia Walschaers. *Statistical Benchmarks for Quantum Transport in Complex Systems: From Characterisation to Design*. Springer, 2018. DOI: 10.1007/978-3-319-93151-7.

[42] Malte C Tichy, Klaus Mayer, Andreas Buchleitner, and Klaus Mølmer. Stringent and efficient assessment of boson-sampling devices. *Physical review letters*, 113(2):020502, 2014. DOI: 10.1103/PhysRevLett.113.020502.

[43] Christoph Dittel, Gabriel Dufour, Mattia Walschaers, Gregor Weihs, Andreas Buchleitner, and Robert Keil. Totally destructive many-particle interference. *Physical Review Letters*, 120(24):240404, 2018. DOI: 10.1103/PhysRevLett.120.240404.

[44] Niko Viggianiello, Fulvio Flamini, Luca Innocenti, Daniele Cozzolino, Marco Bentivegna, Nicolò Spagnolo, Andrea Crespi, Daniel J Brod, Ernesto F Galvão, Roberto Osellame, et al. Experimental generalized quantum suppression law in sylvester interferometers. *New Journal of Physics*,

20(3):033017, 2018. DOI: 10.1088/1367-2630/aaad92.

[45] Andrea Crespi. Suppression laws for multiparticle interference in sylvester interferometers. *Physical Review A*, 91(1):013811, 2015. DOI: 10.1103/PhysRevA.91.013811.

[46] Valery Shchesnovich. Distinguishing noisy boson sampling from classical simulations. *Quantum*, 5:423, March 2021. ISSN 2521-327X. DOI: 10.22331/q-2021-03-29-423. URL https://doi.org/10.22331/q-2021-03-29-423.

[47] Jelmer J Renema. Marginal probabilities in boson samplers with arbitrary input states. *arXiv preprint arXiv:2012.14917*, 2020. DOI: 10.48550/arXiv.2012.14917.

[48] Traian Abrudan, Jan Eriksson, and Visa Koivunen. Conjugate gradient algorithm for optimization under unitary matrix constraint. *Signal Processing*, 89(9):1704–1714, 2009. DOI: 10.1016/j.sigpro.2009.03.015.

[49] Keith R Motes, Alexei Gilchrist, Jonathan P Dowling, and Peter P Rohde. Scalable boson sampling with time-bin encoding using a loop-based architecture. *Physical review letters*, 113(12):120501, 2014. DOI: 10.1103/PhysRevLett.113.120501.

[50] Georgios M Nikolopoulos and Thomas Brougham. Decision and function problems based on boson sampling. *Physical Review A*, 94(1):012315, jul 2016. DOI: 10.1103/PhysRevA.94.012315. URL https://doi.org/10.1103%2Fphysreva.94.012315.

[51] Nikolopoulos, Georgios M. Cryptographic one-way function based on boson sampling. *Quantum Information Processing*, 18(8):1–25, jul 2019. DOI: 10.1007/s11128-019-2372-9. URL https://doi.org/10.10072Fs11128-019-2372-9.

[52] Xiao-Wei Wang, Wen-Hao Zhou, Yu-Xuan Fu, Jun Gao, Yong-Heng Lu, Yi-Jun Chang, Lu-Feng Qiao, Ruo-Jing Ren, Ze-Kun Jiang, Zhi-Qiang Jiao, Georgios M. Nikolopoulos, and Xian-Min Jin. Experimental boson sampling enabling cryptographic one-way function. *Phys. Rev. Lett.*, 130:060802, Feb 2023. DOI: 10.1103/PhysRevLett.130.060802. URL https://link.aps.org/doi/10.1103/PhysRevLett.130.060802.

[53] Deepesh Singh, Boxiang Fu, Gopikrishnan Muraleedharan, Chen-Mou Cheng, Nicolas Roussy Newton, Peter P. Rohde, and Gavin K. Brennen. Proof-of-work consensus by quantum sampling. 2023. DOI: 10.48550/arXiv.2305.19865.

[54] Fuzhen Zhang. An update on a few permanent conjectures. *Special Matrices*, 4(1), 2016. DOI: doi:10.1515/spma-2016-0030. URL https://doi.org/10.1515/spma-2016-0030.

[55] Valery S Shchesnovich. The permanent-on-top conjecture is false. *Linear Algebra and its Applications*, 490:196–201, 2016. DOI: 10.1016/j.laa.2015.10.034.

[56] Simon Becker, Nilanjana Datta, Ludovico Lami, and Cambyse Rouzé. Convergence rates for the quantum central limit theorem. *Communications in Mathematical Physics*, 383(1):223–279, feb 2021. DOI: 10.1007/s00220-021-03988-1. URL https://doi.org/10.1007%2Fs00220-021-03988-1.

[57] M Correa Anguita, FHB Somhorst, R van der Meer, R Schadow, HJ Snijders, M de Goede, B Kassenberg, P Venderbosch, C Taballione, JP Epping, et al. Quantum photo-thermodynamics on a programmable photonic quantum processor. In *Quantum 2.0*, pages QTu3A–3. Optica Publishing Group, 2022. DOI: 10.1364/QUANTUM.2022.QTu3A.3.

[58] Malte C Tichy. Sampling of partially distinguishable bosons and the relation to the multidimensional permanent. *Physical Review A*, 91(2):022316, 2015. DOI: 10.1103/PhysRevA.91.022316.

[59] Shi, Junheng and Byrnes, Tim. Gaussian boson sampling with partial distinguishability. 2021. DOI: 10.48550/ARXIV.2105.09583. URL https://arxiv.org/abs/2105.09583.

[60] Peter Clifford and Raphaël Clifford. The classical complexity of boson sampling, 2017. URL https://arxiv.org/abs/1706.01260.

[61] Alexandra E Moylett, Raú l García-Patrón, Jelmer J Renema, and Peter S Turner. Classically simulating near-term partially-distinguishable and lossy boson sampling. *Quantum Science and Technology*, 5(1):015001, nov 2019. DOI: 10.1088/2058-9565/ab5555. URL https://doi.org/10.1088%2F2058-9565%2Fab5555.

[62] Mattia Walschaers. Signatures of many-particle interference. *Journal of Physics*

B: Atomic, Molecular and Optical Physics, 53(4):043001, 2020. DOI: 10.1088/1361-6455/ab5c30.

[63] Fulvio Flamini, Mattia Walschaers, Nicolò Spagnolo, Nathan Wiebe, Andreas Buchleitner, and Fabio Sciarrino. Validating multi-photon quantum interference with finite data. *Quantum Science and Technology*, 5(4):045005, 2020. DOI: 10.1088/2058-9565/aba03a.

[64] C. K. Hong, Z. Y. Ou, and L. Mandel. Measurement of subpicosecond time intervals between two photons by interference. *Phys. Rev. Lett.*, 59:2044–2046, Nov 1987. DOI: 10.1103/PhysRevLett.59.2044. URL https://link.aps.org/doi/10.1103/PhysRevLett.59.2044.

[65] John Leech. H. j. ryser, combinatorial mathematics (carus mathematical monographs, no. 14; published by the mathematical association of america, distributed by john wiley and sons, 1963), xiv 154 pp., 30s. *Proceedings of the Edinburgh Mathematical Society*, 14(1):82–83, 1964. DOI: 10.1017/S0013091500011299.

[66] Brajesh Gupt, Josh Izaac, and Nicolás Quesada. The walrus: a library for the calculation of hafnians, hermite polynomials and gaussian boson sampling. *Journal of Open Source Software*, 4(44):1705, 2019. DOI: 10.21105/joss.01705. URL https://doi.org/10.21105/joss.01705.

[67] Nicolas Heurtel, Andreas Fyrillas, Grégoire de Gliniasty, Raphaël Le Bihan, Sébastien Malherbe, Marceau Pailhas, Boris Bourdoncle, Pierre-Emmanuel Emeriau, Rawad Mezher, Luka Music, Nadia Belabas, Benoît Valiron, Pascale Senellart, Shane Mansfield, and Jean Senellart. Perceval: A software platform for discrete variable photonic quantum computing, 2022. URL https://arxiv.org/abs/2204.00602.

[68] Javier Osca and Jiri Vala. Implementation of photon partial distinguishability in a quantum optical circuit simulation. 2022. DOI: 10.48550/ARXIV.2208.03250. URL https://arxiv.org/abs/2208.03250.