The Journal of Open Source Software

VeriQuEST.jl: Emulating verification of quantum computations with QuEST

Jonathan Miller [1*], **Dominik Leichtle**[2*], **Elham Kashefi**[2], **and Cica Gustiani**[2]

1 School of Informatics, University of Edinburgh, 10 Crichton Street, Edinburgh EH8 9AB, United Kingdom **2** Laboratoire d'Informatique de Paris 6, CNRS, Sorbonne Université, 4 Place Jussieu, Paris 75005, France * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Within the practical context of quantum computing, the *client-server* framework is anticipated to become predominant in the future due to the high costs, intensive maintenance, and operational complexity of quantum computers. In this framework, a client with a very limited quantum power, so-called Alice, delegates her quantum computation to a powerful quantum server, so-called Bob, who then provides Alice her computation results. While this scenario appears practical, it raises significant security concern: how do we ensure Bob follows Alice's instructions and confirm the reliability of his quantum computer? These questions undeniably pose significant challenges and remain under active investigation; *verifiable quantum computations* is a subfield of quantum computing that aims to address these concerns.

This paper introduces `VeriQuEST.jl`, a Julia package equipped with functionalities to emulate verification protocols for quantum computation (Gheorghiu et al., 2019) within the framework of measurement-based quantum computations (Raussendorf & Briegel, 2001). The package utilizes QuEST (Jones et al., 2019) as its backend, a versatile quantum computer emulator (including noise simulation) written in C, while supporting high-performance computations. VeriQuEST.jl is designed to aid researchers in testing their verification protocols or concepts on emulated quantum systems. This approach allows them to assess performance estimates without the need for actual hardware, which, at the time of writing this paper, is significantly limited.

Statement of need

Our package `VeriQuEST.jl` is aimed at assisting researchers in exploring and designing their verification protocols within the paradigm of *measurement-based quantum computation* (MBQC). The conventional way to express quantum algorithms is through a series of unitary operations and measurements, so-called *gate-based* computations (Nielsen & Chuang, 2011). On the other hand, MBQC uses a *graph state* as the resource and a series of adaptive single-qubit measurements to realise a quantum algorithm. While gate-based emulators – such as `QuEST` – are predomonant, MBQC emulators remain scarce. Our package, VeriQuEST.jl, offers an interface that enables users to express their verification protocols and quantum algorithms in the native MBQC language, supporting interactive protocols, such as verifications, that require a quantum internet, powered by a performant backend `QuEST` (Jones et al., 2019). Moreover, to this date, there is no emulator that is specifically aimed at simulating verification protocols.

Core features and functionality

`VeriQuEST.jl` is equipped with fundamental functionalities to support the testing of various verification protocols of quantum computations based on MBQC. The package `VeriQuEST.jl` contains three fundamental elements to support emulation of verification protocols: MBQC computations, multi-round interactions between client and server, and Hilbert space partition to realize the explicit client-server separation. The package allows for (noiseless) state vector and (noisy) density matrix simulations. In the current release, we provide several well-known MBQCs and verification protocols ready for the users to use. If one wishes to become acquainted with the details, see the public GitHub repository for `VeriQuEST`.

Software architecture

[JON:Explain how the package works, where does it stand**: function call, core link (if there is any), compilation]

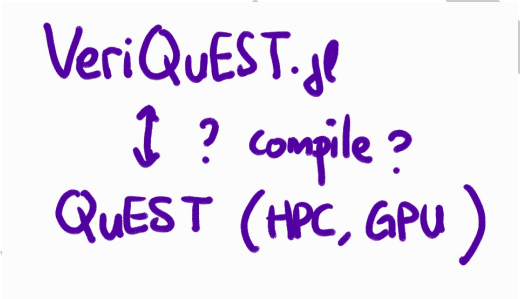

Figure 1: Architecture: here is a sketch. Some diagram would be cool.

Measurement-Based Quantum Computation (MBQC)

In the MBQC framework, a quantum algorithm can be represented as a set $\{(G, I, O), \vec{\phi}\}$, where (G, I, O) denotes the quantum resource and $\vec{\phi}$ is a set of measurement angles. The triplet (G, I, O) signifies an *open graph*, i.e., graphs characterised by the presence of *flow* (Danos & Kashefi, 2006). Single-qubit adaptive measurements are performed sequentially on the vertices with measurement operator $\{|+_{\theta_j}\rangle\langle+_{\theta_j}|, |-_{\theta_j}\rangle\langle-_{\theta_j}|\}$, measured on vertex j, where $|\pm_\theta\rangle := (|0\rangle \pm e^{i\theta}|1\rangle)/\sqrt{2}$ and $\theta_j = \phi_j + \delta$ for a correction δ that is calculated by our tool. For an illustration, see Figure 2.

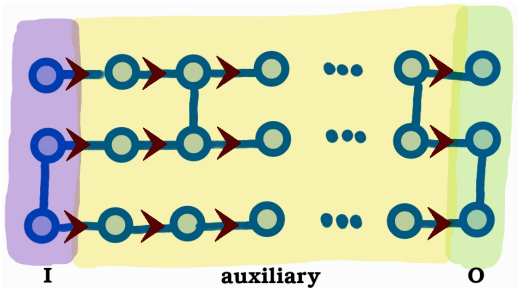

Figure 2: The triplet (G, I, O) representing graph state for the quantum resource, where $G = (V, E)$ is the graph, I is a set of input nodes, and O is a set of output nodes. Each node represents qubit with state $(|0\rangle + |1\rangle)/\sqrt{2}$ and each edge represent controlled-Z operation operated to the corresponding nodes. Measurements in the XY-plane of Bloch sphere are performed from the left to the right. The arrows indicate the *flow* $f : O^c \to I^c$ which induces partial ordering of the measurements. Input nodes I may be initialised to an arbitrary quantum state ρ and the output nodes O is the final output that can be classical or quantum – if left unmeasured.

[JON:CODE: performing an MBQC]

Blind Quantum Computation (BQC)

Blind Quantum Computation (BQC) is a type of cryptographic protocol that enables a client, referred to as Alice, to securely delegate her quantum computing tasks to a powerful quantum server, called Bob, while ensuring privacy. Within this setup, Bob cannot infer Alice's algorithm nor her measurement outcomes. The most notable BQC protocol is the Universal Blind Quantum Computation (UBQC)(Broadbent et al., 2009). This framework allows Alice to privately delegate her MBQC algorithm to Bob through a series of quantum and classical interactions between them.

Remarkably, the UBQC promises composable security that does not rely on traditional computational assumptions (Dunjko et al., 2014). Instead, the privacy stems directly from the intrinsic properties of quantum measurement. In this model, Alice requires only limited quantum resources: the ability to prepare specific quantum states, $|+_\theta\rangle$, and to send them to Bob, underscoring the necessity for a robust quantum network. The essence of maintaining Alice's privacy lies in obfuscating the measurement angles $\vec{\phi}$ and the associated outcomes through the strategic use of randomness, as illustrated in Figure 3.

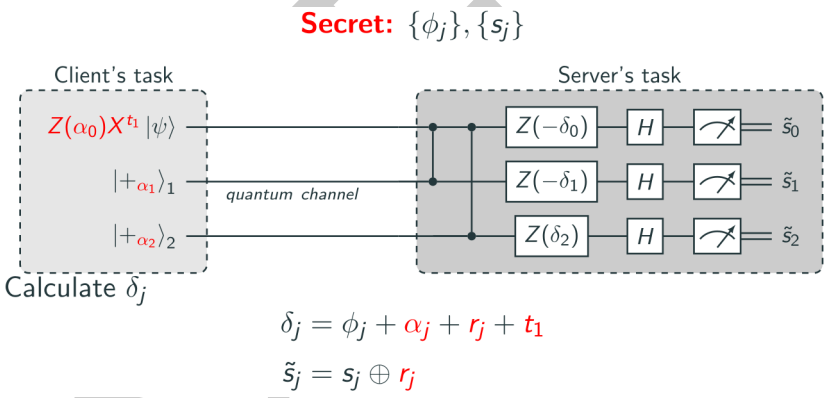

Figure 3: This circuit shows task separation between the client and the hiding strategy, where measurement angles $\{\phi_j\}$ and initial outcomes $\{s_j\}$ are obscured. Additional randomness (highlighted in red) is introduced for hiding the secrets, which later neutralised by adjustments in δ_j to hide the measurement angles ϕ_j. Notice that the actual measurement outcomes s_j remain exclusively accessible to Alice due to random binary r_j. Note that the first state $|\psi\rangle$ indicates an arbitrary input state that is encrypted by random phase α_0 and random bit flip t_1; thus, $\delta_0 = (-1)^{t_1} + \alpha_0 + r_0$. In the MBQC language, this circuit is equivalent with the three nodes path graph with angles $\{\alpha_0, \alpha_1, \alpha_2\}$.

In BQC, delegating quantum tasks privately highlights the importance of client state preparation, state transfer, accessibility, correctness, and security. To address these critical aspects efficiently, end users are presented with two approaches for emulating BQC algorithms:

- **Implicit network emulation**. In this option, we do not explicitly emulate the quantum state of the client or the quantum network. Instead, the states prepared on the server side already take into account the state transfer. This approach is useful for studying the noise effect on the computation. For an example, see the code below. [JON:CODE SNIPPET with implicit client]
- **Explicit network emulation**. In this option, the quantum state of the client and the state transfer are explicitly emulated. The quantum network is simulated using remote entanglement operators, which can also be specified by the end user. This is made possible by operators operating between the client and the server in the joint Hilbert space of the client and the server, denoted as $\mathcal{H}_c \otimes \mathcal{H}_s$. This approach is useful for

studying the effects of noise on the protocol as well as examining security in greater detail. Additionally, users can access the state of each party: the client's state in $\mathcal{H}_c$ and the server's state in $\mathcal{H}_s$, enabled by the partial trace operation. For an example, see the code below. [JON:CODE SNIPPET with explicit client]

Verification protocols [DOM]

The verification protocol can be added on top of BQC, certifying the executed quantum algorithm.

Introduce verification and.

Our tool provides several verification protocols that are ready for users to use:

- Dominik's BQP verification (Leichtle et al., 2021)
- Maybe Fitsimons verification with hidden trap?

[JON:Example: Code BQP verification]

Noiseless and noisy simulations

In the study of quantum verification protocols, it is crucial to be able to simulate both state vector and density matrix simulations. State vector simulations require less computational space, e.g., n-qubit system has dimension 2^n, making it easier to check if the protocols are correct. On the other hand, density matrix simulations requires more computational space, e.g., n-qubit system has dimension $2^n \times 2^n$, however, they are excellent for understanding how noise impacts the whole protocol, affecting both its certification and security. Such features help researcher to ensuring their protocols are both correct and secure in the presence of noise. Some examples on the state-vector and density matrix simulations are given below.

[JON:Example:CODE SNIPPET]

Future plans

Adding the upcoming verification protocols in the library. Multi-parties computation as per. Graph with gflow, more choice on the measurement plane. Minimal resource of MBQC (Lazy 1WQC). Integrating realistic noise. Mathematica integration.

Acknowledgements

We acknowledge contributions from QSL, NQCC,...???

References

Broadbent, A., Fitzsimons, J., & Kashefi, E. (2009). Universal blind quantum computation. *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, 517–526. https://doi.org/10.1109/FOCS.2009.36

Danos, V., & Kashefi, E. (2006). Determinism in the one-way model. *Physical Review A*, *74*(5), 052310. https://doi.org/10.1103/PhysRevA.74.052310

Dunjko, V., Fitzsimons, J. F., Portmann, C., & Renner, R. (2014). Composable security of delegated quantum computation. *International Conference on the Theory and Application of Cryptology and Information Security*, 406–425. https://doi.org/10.1007/978-3-662-45608-8_22

Gheorghiu, A., Kapourniotis, T., & Kashefi, E. (2019). Verification of quantum computation: An overview of existing approaches. *Theory of Computing Systems*, *63*, 715–808. https://doi.org/10.1007/s00224-018-9872-3

Jones, T., Brown, A., Bush, I., & Benjamin, S. C. (2019). QuEST and high performance simulation of quantum computers. *Scientific Reports*, *9*(1), 10736. https://doi.org/10.1038/s41598-019-47174-9

Leichtle, D., Music, L., Kashefi, E., & Ollivier, H. (2021). Verifying BQP computations on noisy devices with minimal overhead. *PRX Quantum*, *2*, 040302. https://doi.org/10.1103/PRXQuantum.2.040302

Nielsen, M. A., & Chuang, I. L. (2011). *Quantum computation and quantum information: 10th anniversary edition*. Cambridge University Press. ISBN: 9781107002173

Raussendorf, R., & Briegel, H. J. (2001). A one-way quantum computer. *Physical Review Letters*, *86*(22), 5188. https://doi.org/10.1103/PhysRevLett.86.5188