GraphiQ: Quantum circuit design for photonic graph states

Jie Lin*[1,2], Benjamin MacLellan*[3,4,5], Sobhan Ghanbari*[1,6], Julie Belleville[5], Khuong Tran[5], Luc Robichaud[1,2], Roger G. Melko[3,7], Hoi-Kwong Lo[1,2], and Piotr Roztocki[5]

[1]Quantum Bridge Technologies Inc., 108 College St., Toronto, ON, Canada

[2]Department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, Toronto, ON, Canada

[3]University of Waterloo, Department of Physics & Astronomy, 200 University Ave., Waterloo, ON, Canada

[4]Institute for Quantum Computing, 200 University Ave., Waterloo, ON, Canada

[5]Ki3 Photonics Technologies, 2547 Rue Sicard, Montreal, QC, Canada

[6]Department of Physics, University of Toronto, 60 St George St., Toronto, ON, Canada

[7]Perimeter Institute for Theoretical Physics, 31 Caroline St N., Waterloo, ON, Canada

Friday 23rd August, 2024

GraphiQ is a versatile open-source framework for designing photonic graph state generation schemes, with a particular emphasis on photon-emitter hybrid circuits. Built in Python, GraphiQ consists of a suite of design tools, including multiple simulation backends and optimization methods. The library supports scheme optimization in the presence of circuit imperfections, as well as user-defined optimization goals. Our framework thus represents a valuable tool for the development of practical schemes adhering to experimentally-relevant constraints. As graph states are a key resource for measurement-based quantum computing, all-photonic quantum repeaters, and robust quantum metrology, among others, we envision GraphiQ's broad impact for advancing quantum technologies.

1 Introduction

GraphiQ is an open-source software framework for the design of photonic quantum circuits, with a particular focus on the realistic generation of entangled graph states in hybrid photon-emitter platforms. Graph states, a family of entangled quantum states, are a key resource in a variety of quantum information applications, including entanglement distribution and quantum networking [1, 2], quantum error correction [3, 4], measurement-based quantum computing [5–7], fusion-based quantum computing [8–10], and quantum metrology [11]. Among the various platforms leveraged for quantum technologies, photonics has unique advantages towards graph state generation, such as qubits with long coherence times, compatibility with well-developed integrated and fiber optic infrastructures, and the ability to be distributed spatially (e.g., between different network nodes).

Generating photonic graph states can be realized with two main approaches: probabilistic and deterministic. In the probabilistic case, fusion gates [12, 13], implemented with linear optics, detectors and post-selection, are used to build up graph states from small entangled states. However, given the probabilistic nature of the fusion, the resources required scale exponentially with the graph state size [14]. On the other hand, deterministic approaches leverage entanglement operations between emitters, such as quantum dots, trapped ions or nitrogen-vacancy centers in diamond [15], for direct graph state generation without the need for probabilistic fusion. Recent experimental demonstrations using such architectures [16] have reached impressive milestones, e.g., the generation of 10-qubit linear cluster states [17, 18] and 14-qubit Greenberger–Horne–Zeilinger (GHZ) states [19]. There are also various theoretical proposals on quantum-emitter-based methods for the generation of 2-dimensional graph states [15, 20–22]. Due to limitations on the coherence time of quantum emitters and challenges in their coupling, graph states generated by such approaches are still too small to be useful for many practical ap-

---

*These authors contributed equally to this work
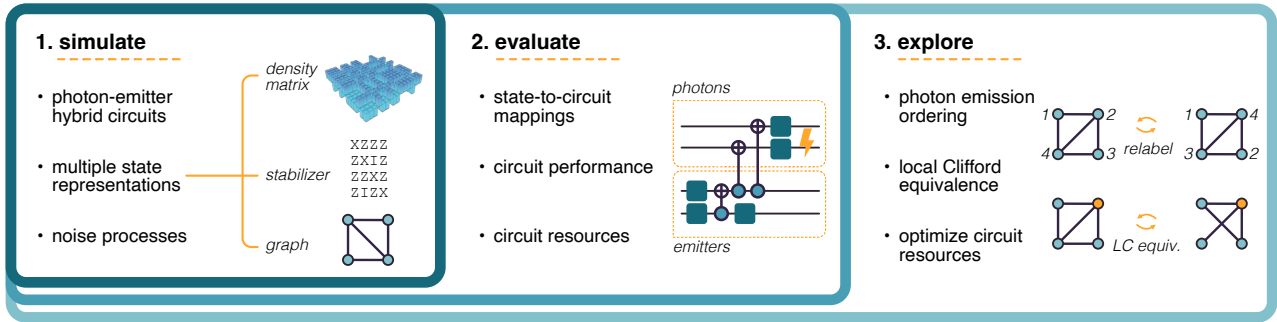
Figure 1: *GraphiQ software framework and its three core use cases: the simulation, evaluation, and exploration of quantum circuits that generate photonic graph states.*

plications. Searching for resource-efficient generation schemes obeying realistic experimental constraints is thus an active goal for the research community.

Nevertheless, designing the quantum circuit(s) appropriate for the generation of a particular graph state is a non-trivial task, especially as the size of the graph state grows. This is in part due to the lack of usable photon-photon interactions; using linear optics and post-selection to perform entangling operations, after photon emission, is impractical due to the rapid scaling in the number of required components and the drastic decrease in success probability. Rather, entanglement can be created through interactions between emitter qubits, and subsequently transferred to photonic qubits through the emission process (represented as a CNOT gate between emitter and photon) [15]. However, identifying the required sequence of quantum operations, for both the emitter and photonic qubits, to realize a particular entangled state is challenging.

In recent years, a rich ecosystem of software packages for simulating and designing quantum circuits has been developed [23–28]. Such open-source toolboxes have been central to progress in quantum information science, allowing a broad research community to use and collaborate using a common set of tool-chains. While these toolboxes may have similar basic components for simulating quantum circuits and quantum states, they typically focus on specific qubit architectures or computation models. For example, PennyLane [23] is for differential programming of quantum computers and its core feature is to compute gradients of variational quantum circuits; Quimb [24] is designed for quantum information and many-body calculations; Strawberry Fields [25] focuses on continuous-variable systems; Perceval [28] is for discrete-variable photonic quantum computing, focusing on quantum circuit simulation. No packages thus far have been tailored to designing and optimizing hybrid photonic quantum circuits for photonic graph state generation. Very recently, there is some recent progress in optimizing protocols for graph state generation [29, 30]. Ref. [30] introduces `OptGraphState`, a Python software to study fusion-based graph state generation. In terms of deterministic approaches, Ref. [29] uncovers a method for constructing circuits that minimize the number of required emitter qubits. It should be noted that further optimizations are still possible to address different experimental challenges. Nevertheless, the task of designing optimized photon-emitter circuits remains challenging, especially if the constraints of real experimental devices (e.g., photon loss, noise) are to be considered. In this paper, we focus on the deterministic approaches.

Here, we introduce `GraphiQ`[1], an open-source toolbox for hybrid photon-emitter schemes, implemented in Python, particularly suited for the study and design of practical graph state generation circuits. The framework (Figure 1) comprises a suite of tools towards this end, centered on three main use-cases:

- **Simulation** of photon-emitter quantum circuits, with multiple backend representations for quantum states (density matrix, stabilizer, and graph);

- **Evaluation** of circuit performance and required resources, alongside benchmarking tools;

---

[1]This paper refers to `GraphiQ` version 0.1.0.

- **Exploration** and optimization of photonic circuits, governed by experimental constraints, user-defined metrics, and target applications.

Our use of the framework for investigating circuit optimization via local operations [31] has yielded significant improvements in generation scheme designs for the photonic-emitter platform. The package is, moreover, readily adaptable to new hardware platforms, applications, and optimization methods.

The remainder of this paper is outlined below. In Section 2, we summarize the architecture of `GraphiQ` and its key features. We then discuss the circuit simulation module of `GraphiQ` in Section 3. In Section 4, we discuss evaluating the performance of a given quantum circuit using various circuit- and graph-based metrics. In Section 5, we present optimization schemes currently implemented in `GraphiQ`. Lastly, we provide concluding remarks in Section 6. Additional background and implementation details are provided in Appendix A for reference. We include examples accomplished using `GraphiQ` in Appendix B.
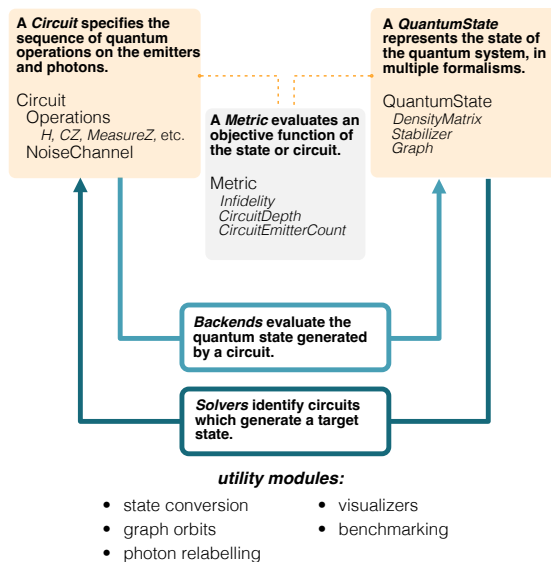
2 Overview of `GraphiQ`

Figure 2 gives an overview of the package interface and highlights its main classes and modules. The development of `GraphiQ` is focused on four design pillars:

- **Versatility:** Support for a broad range of tasks related to circuit design, spanning simulation through to visualization and benchmarking;

- **Accuracy:** Experimental realism is upheld by an extensive library of circuit noise models, as well as by imposing hardware constraints on designed circuits;

- **Modularity:** Module independence enables fast tailoring of software for custom use cases;

- **Extensibility:** Users can rapidly prototype and include new, custom optimization algorithms, physical circuit constraints, and state/circuit metrics.

What can `GraphiQ` do?

- Simulate the output of noisy quantum circuits, with emitted quantum states represented as density matrices, stabilizer tableaux, or graphs. See Section 3.

- Convert quantum states from one representation to another.

- Find quantum circuits that produce a target graph state and optimize them with respect to custom state/circuit performance metrics. A set of physical rules and experimental constraints can be imposed on the resultant circuits. See Section 5.

- Find alternative circuits for the generation of the same target graph state and/or the graphs isomorphic to it.

- Check for the local Clifford equivalency of graph states.

- Find the local Clifford equivalency class (LC orbit) for a given graph.

- Evaluate state, graph theoretical, and circuit metrics.

- Visualize quantum states and circuits.

Figure 2: `GraphiQ` interface overview. It consists of several modules including state/circuit representation, noise simulation, metric evaluation and various utility modules in addition to optimization solvers.

Tutorials and examples of `GraphiQ` are available on https://github.com/graphiq-dev/graphiq. More information about the full documentation can be found in https://graphiq.readthedocs.io/en/latest/.

3 Simulation of quantum circuits

Simulating the quantum state produced by a given circuit is prerequisite to its optimization. `GraphiQ`'s simulation modules comprise representations of quantum states, circuits, and noise.

3.1 States

The underlying quantum state of emitters and photons, represented by the evolution of an initial state through the circuit, can be modelled with different classical simulation techniques. `GraphiQ` supports multiple state representations, providing flexibility for different use cases. The state representation module comes with a wrapper class `QuantumState` that acts as a common application programming interface (API) for users. Currently it supports several backends: **density matrix**, **stabilizer**, **mixed stabilizer**, or **graph**. Codeblock 1 shows one way to initialize `QuantumState` objects for initial data in the form of a `Graph` object from the `NetworkX` package.

```
1 data = networkx.Graph([(0,1),(1,2)])
2 state_dm = QuantumState(data,"dm")
3 state_graph = QuantumState(data,"g")
```

Codeblock 1: *Using a `NetworkX` graph to initialize two `QuantumState` objects: one with the density matrix backend and the other with the graph backend.*

We discuss briefly each representation, their use cases, and limitations below.

A quantum state in a finite-dimensional Hilbert space can be represented as a positive semi-definite matrix with unit trace. The **density matrix** ρ of an n-qubit state is an element of $\mathbb{C}^{2^n \times 2^n}$ (see Ref. [32, Section 2.4] for review). The density matrix formalism captures all information about the underlying quantum state and enables the state to evolve under arbitrary unitary operations, quantum measurements, and general noise processes. However, due to the exponential scaling of the size of the density matrix with the number of qubits, it becomes an impractical representation to describe large states as both the

memory storage and running time for state simulations scale as $O(2^n)$. Consequently, the density matrix backend is only suitable for states with a small number of qubits (e.g., $n \leq 10$).

In turn, the **stabilizer** formalism [33, 34] enables the efficient representation of so-called stabilizer states, a subset of quantum states. As graph states form a subset of stabilizer states, this representation is suitable for their study. An n-qubit stabilizer state is uniquely defined by its stabilizer group which has n independent generators. More background information about the stabilizer formalism can be found in Appendix A.2. See also Refs. [34, 35]. In particular, `GraphiQ` adopts the formalism in Ref. [34], keeping in memory the stabilizer and destabilizer generators[2] of the n-qubit stabilizer state, as an $2n \times 2n$ binary matrix, which scales much better relative to density matrices. For each generator, `GraphiQ` also uses 2 bits to keep track of four possible phases. We note that generators corresponding to the stabilizer part can take only ± 1 phases, while the destabilizer generators can take on both ± 1 and $\pm i$ phases.

Graph states are, by construction, defined by a **graph** where the nodes represent qubits and the connecting edges represent entanglement between the qubits. In this representation, graph states correspond to simple, undirected graphs, which can be stored in memory as adjacency matrices, or node and edge sets. As the size of node-edge sets grows with the connectivity of the graph while adjacency matrices remain constant in size for a given qubit number, node-edge sets are more favorable for graphs with low connectivity, while adjacency matrices are more favorable for high connectivity cases.

Given that the graph representation is very visual, it is useful for understanding the entanglement structure of a given state, which is not always so obvious in the case of the density matrix or stabilizer representations. A graph representation also makes it more intuitive to look at graph actions like local complementation. The action of local complementation on a graph corresponds to local Clifford operations on the corresponding graph state (see Appendix A).

---

[2]As defined in [34], destabilizer generators are Pauli operators such that destabilizer and stabilizer generators together generate the entire n-qubit Pauli group up to four global phases $\pm 1, \pm i$.

Since graph states form a proper subset of stabilizer states, to empower this representation to represent all stabilizer states, it is necessary to amend this representation. As any stabilizer state can be converted to a graph state by local Clifford operations [36], `GraphiQ` accomplishes this task by including a record of required local Clifford operations in each node of the graph.

It is worth noting that the stabilizer and graph representations discussed so far can represent only pure states. However, when noise is present, it is usually necessary to consider **mixed states**. Unlike the density matrix representation, which can naturally represent mixed states, the stabilizer and graph representations must be amended to support mixed states. A simple approach, implemented in `GraphiQ`, is to keep track of an ensemble of pure states that can form the mixed state of interest.

`GraphiQ` also offers functions to convert different state representations. More information can be found in Appendix A.3.

3.2 Circuits

Quantum circuits describe a sequence of operations applied to qubits. `GraphiQ` is developed with a particular focus on emitter-photonic hybrid circuits, meaning that circuits are composed of three register blocks: emitter qubits, photonic qubits, and classical bits. For example, Figure 3(b) illustrates an emitter-photonic circuit that produces the 3-qubit linear cluster state shown in Figure 3(a). In `GraphiQ`, the `CircuitDAG` class can be used to store a quantum circuit (internally represented as a directed acyclic graph, or DAG), enabling easy modification. Codeblock 2 demonstrates how quantum gates can be added into a circuit one by one. See Appendix B.1 for further code examples, including circuit simulation.

```
1  c = CircuitDAG(n_emitter=1, n_photon=1)
2  c.add(Hadamard(register=0, reg_type="e"))
3  c.add(CNOT(control=0, control_type="e",
   ↪  target=0, target_type="p"))
```

Codeblock 2: *Construction of a quantum circuit by adding gates.*

Additionally, users can also insert, remove, or replace a quantum gate at a specified location. In addition to direct manipulation of quantum circuits, it is also possible to import a quantum

(a)

(b)
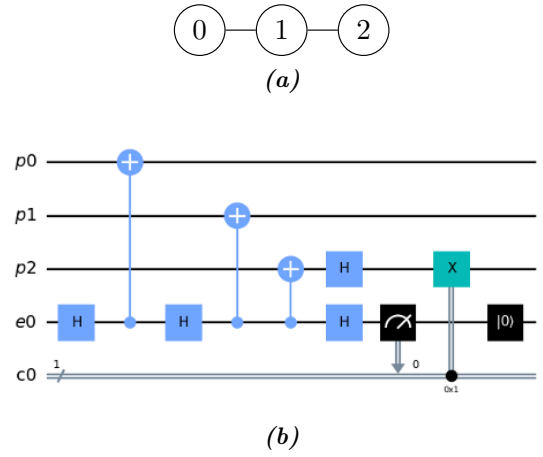
Figure 3: *(a) Graph representation for a 3-qubit linear cluster state, where each node represents a qubit initialized in a $|+\rangle$ state, and each edge represents a CZ gate. Each node is assigned a unique ID. (b) A quantum circuit that generates the above cluster state by using one emitter. p0-p2 represent 3 photonic qubits corresponding to those in (a), e0 represents the emitter qubit, and c0 represents a classical bit.*

circuit from Open Quantum Assembly Language (OpenQASM) files [37]. Likewise, `GraphiQ` circuit objects can be serialized to OpenQASM files for export to other toolchains, such as external visualizers and simulators [27].

3.3 Noise simulation

In practice, noise and experimental imperfections affect all quantum information tasks. The capacity to simulate noise is prerequisite to modeling schemes resembling true experimental conditions.

`GraphiQ` is flexible in its modeling, allowing users to simulate noise within and/or between quantum gates. This respectively corresponds to replacing circuit gates with their noisy versions, or to inserting noise before/after quantum gates (i.e., noise between operations, like during photon propagation or storage).

The software presently supports noise types common and relevant to photonic graph state generation, e.g., depolarizing noise and photon loss. Codeblock 3 illustrates an example of applying photon loss and depolarizing noise to a state.

```
1  import graphiq.noise.noise_models as nm
2  # create a photon loss with loss
   ↪  probability = 0.1
3  loss = nm.PhotonLoss(0.1)
4  # create a depolarizing noise with error
   ↪  probability = 0.02
```

```
5  noise = nm.DepolarizingNoise(0.02)
6  # apply the loss and depolarizing noise to
   ↪   the first qubit (index 0) of a 2-qubit
   ↪   state
7  loss.apply(state_dm, 2, 0)
8  noise.apply(state_dm, 2, 0)
```

Codeblock 3: *Applying photon loss with a loss probability of* 0.1 *to the first qubit of a 2-qubit state and then applying a depolarizing channel with a depolarizing probability of* 0.02 *to the same qubit.*

Moreover, the included noise models can be readily extended and customized to account for user-specific hardware considerations. Extended discussion and specific examples on noise simulation in `GraphiQ` can be found in Appendix A.4.

4 Evaluation of circuits

When optimizing a quantum circuit for the generation of a target graph state, it is important to be able to quantify the circuit performance using experimentally relevant figures of merit, which are often specific to the user hardware.

`GraphiQ` enables great flexibility in evaluating circuits with figures of merit (defined by the user) that can be based on the output quantum state and/or the circuit itself. The software also allows defining a weighted average of a list of metrics or custom functions. Below, we discuss already-implemented metrics based on quantum circuits and output states, as well as the algorithms that leverage them to converge on performant generation schemes for targeted graph states (i.e., state-to-circuit mappings).

4.1 Circuit metrics

Reducing the total number of gates or the longest path in a circuit (i.e., the circuit depth) is of general interest as it corresponds to a reduction in experimental complexity and cost. However, one may care about reducing specific gate types due to the experimental difficulty of implementing them. For example, in the context of generating photonic graph states using solid-state emitters, operations on emitter qubits are typically more challenging to realize in practice. Photonic-emitter circuit designers may thus want to minimize the circuit depth on emitter qubits, the number of CNOT between emitter qubits, as well as the number of emitter qubits, among others.

`GraphiQ` thus supports circuit metrics such as:

- Total number of unitaries,
- Number of emitter qubits,
- Total number of emitter-emitter CNOT gates,
- Circuit depth: the length of the longest path in the quantum circuit,
- Emitter depth: the maximum number of gates applied on an emitter in the circuit between two consecutive resets of that emitter,
- Emitter history: the maximum number of gates applied on a single emitter.

The implementation of custom user-defined metrics of the generation circuit is supported in `GraphiQ` and straightforward.

Codeblock 4 shows an example of evaluating a metric, which has a common interface for both circuit and state metrics.

```
1  metric = CircuitMaxEmitDepth()
2  metric.evaluate(state, circuit)
```

Codeblock 4: *Creating and evaluating the maximum emitter depth metric on a circuit.*

4.2 State metrics

Fidelity quantifies how alike two quantum states are to one another. Thus, for the task of designing circuits that output a specific target quantum state, circuit performance can effectively be quantified by comparing the actual versus targeted output state via fidelity.

For two states ρ_o and ρ_t, corresponding here to the output and target states respectively, the fidelity is defined as

$$F(\rho_o, \rho_t) = \mathrm{tr}\left(\sqrt{\sqrt{\rho_o}\rho_t\sqrt{\rho_o}} \right)^2. \qquad (1)$$

As it is natural to minimize a cost function in an optimization algorithm rather than maximizing, `GraphiQ` implements infidelity as one of its state metrics, defined as $1 - F(\rho_o, \rho_t)$. While calculating fidelity is straightforward in the density matrix representation, it is more complicated to do so in the stabilizer representation. Here, `GraphiQ` leverages an algorithm based on the one presented in Ref. [38].

Other state metrics such as the trace distance are also included in `GraphiQ`.

4.3 State-to-circuit mapping

In `GraphiQ`, a solver is an algorithm that maps a user-defined graph state to a circuit that can generate it, with its performance assessed by a cost function also chosen by the user. The cost function can be composed of state and circuit metrics, e.g., maximizing the output graph state fidelity or minimizing the number of CNOT gates in the circuit. Additionally, each solver accounts for physical/platform constraints to guarantee that any output quantum circuit is realizable. For photonic-emitter circuits, the main constraints relate to the lack of photon-photon interactions; entanglement between photons is generated indirectly by entangling emitter qubits and emitting the photons, creating photon-emitter entanglement. This means that any two-qubit gates (except CNOT gates used to model photon emission) are only allowed between two *emitter* qubits.

To use a solver, a user specifies the target graph state, cost function, as well as the preferred state representation/compiler. The solver outputs a list of potential quantum circuits, ranked by cost function value. In other words, each solver gives a state-to-circuit mapping for a given choice of cost function. The quality of the mapping will depend on the choice of solver and cost function. From a given state-to-circuit mapping, additional techniques can be applied to derive more advanced solvers; we discuss these further in Section 5 and demonstrate an application in Ref. [31].

The current version of `GraphiQ` has multiple solvers implemented. Solvers can loosely be grouped into probabilistic, deterministic, or hybrid approaches. Probabilistic approaches are based on random perturbations to successive generations of quantum circuits, e.g., `EvolutionarySolver`, which leverages evolutionary algorithms. In turn, deterministic approaches execute rule sets without randomness. An example of this is `TimeReversedSolver`, which deterministically finds a quantum circuit for a target graph state. This solver is based on the method of Li et al. presented in Ref. [29]. It uses time-reversed processes to fully disentangle the target graph state with the aid of so-called height function to guide the optimization. The resulting quantum circuit is obtained by the inverse ordering of gates found in this process. Noteworthily, for a given photon emission ordering, it outputs a single unique circuit which uses the minimum number of quantum emitters required to generate said graph state (making it a great basis for further exploration/optimization, see Section 5). Finally, hybrid approaches use a combination of deterministic and probabilistic algorithms, e.g., using deterministic solver outputs as a starting point for random perturbations.

In terms of probabilistic solvers, `GraphiQ` currently has `RandomSearchSolver` and `EvolutionarySolver`, which are suitable for exploring graph states of small size. They are based on applying random operations (which includes addition, removal, or replacement of a quantum gate) to a population of candidate circuits, evaluating the relevant metrics to assign a score to each circuit, and selecting circuits with the highest scores in each iteration. Deterministic solvers include `TimeReversedSolver`, which is the implementation of Ref. [29]'s algorithm that allows the use of minimal number of emitters. `HybridEvolutionarySolver` is an example of a hybrid solver, which uses the resulting circuit from a deterministic solver (e.g. `TimeReversedSolver`) as a seed, and then applies the evolutionary algorithm similar to `EvolutionarySolver`. Finally, there is `AlternateTargetSolver` that we will discuss in Section 5. More details about each solver can be found in the documentation. Community contributions are welcome in `GraphiQ` to centralize more sophisticated solver algorithms as they develop.

5 Exploration and optimization of quantum circuits

The circuits that generate a target graph state are generally not unique, with many potentially suited for the task to varying degree. Users often aim to simultaneously satisfy several objectives within a design: experimental platform constraints, performance for a specific application, noise robustness, cost of the realization, among others. A central goal of `GraphiQ` is enabling exploration across the space of candidate circuits towards practical and optimized designs. The framework includes several tools to support this search including modules for photon emission reordering and local Clifford equivalency, as well as solvers that integrate such modules into optimization workflows.

5.1 Photon emission ordering

If the photons of a graph can be considered indistinguishable, it becomes possible to consider different ways to label the photons in the graph. The nodes are numerically labeled, corresponding to the order of photon emission, with different photon emission orderings naturally corresponding to different quantum circuits that produce states with the same entanglement structure [29]. For applications that are not sensitive to photon emission ordering, it is thus possible to utilize this degree of freedom for exploration and optimization.

GraphiQ provides users with the option to automatically consider emission reordering in the search for suitable circuits, generally leading to more circuit candidates for users to rank and choose from. Further details on the emission ordering module can be found in Appendix A.5.

5.2 Local Clifford equivalency

Local Clifford gates are tensor products of one-qubit Clifford gates and are typically easy and inexpensive to apply experimentally. Interestingly, applying local Clifford gates to a graph state is closely related to local complementation actions on the corresponding graph [36]. By applying local Clifford operations onto it, a target graph state can be made to span a larger space of states. These additional target states differ, but are locally equivalent to the original target graph state via local Clifford operations. As these can be efficiently transformed into the original target state through single-qubit gates on the photons, the overhead to consider the alternative graphs is minimal. However, using local Clifford equivalent graph states as targets may lead some solvers to produce alternative quantum circuits, useful for optimization.

GraphiQ includes modules for exploring local Clifford equivalency to complement the search for quantum circuits, also appending the necessary local Clifford gates. Further details on the module can be found in Appendix A.6.

5.3 Optimization workflows

GraphiQ supports user-defined optimization workflows built up of the solvers and tools described in the preceding sections, as well as those added by users. In particular, as discussed in Section 4.3, existing solvers can be used as a building block for more advanced exploration and optimization algorithms.

As an example, we describe AlternateTargetSolver, available in GraphiQ, which leverages photon emission ordering, local Clifford equivalency, and deterministic state-to-circuit mapping to generate an array of circuit candidates. Users can then rank these candidate circuits by their preferred metric and select the best one to use. The workflow of this solver is shown in Figure 4. First, emission reordering and local Clifford complementation are used to create a list of alternatives to the initial target graph state. A deterministic solver like TimeReversedSolver is then used as a state-to-circuit mapping to run a noise-free optimization, outputting a set of alternative circuits. The local Clifford gates necessary to complete the quantum circuits (to produce the original target state) are then found. A post-selection procedure eliminates redundant circuits from the set. Finally, circuit noise is simulated prior to a final evaluation of the circuits' performance metrics and ranking. An example based on AlternateTargetSolver is showcased in Appendix B.2.3, demonstrating its ability to produce alternative target graph generating circuits. A more detailed study can also be found in [31].

The tools in GraphiQ are well suited to user-customized workflows allowing, as in the case of AlternateTargetSolver, to explore and converge on experimentally-practical circuit candidates.
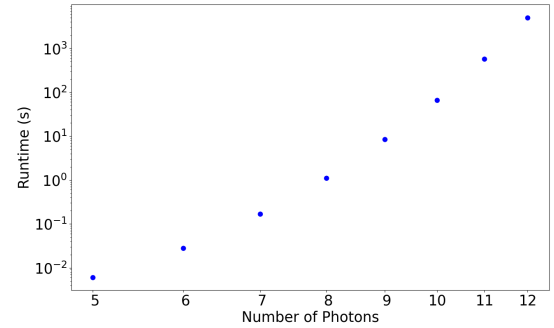
5.4 Runtime

The runtime of AlternateTargetSolver depends on a few choices made by a user. For example, the number of all possible photon emission orderings and the number of local Clifford equivalent graphs can grow exponentially as the number of photons in general. If one attempts to explore every possibility exhaustively, the runtime can scale exponentially. If the user sets a maximum number of alternative graphs for the solver to consider, the runtime will mainly depend on the runtime of the selected state-to-circuit mapping. On the other hand, if a specific family of graphs exhibit certain symmetries,
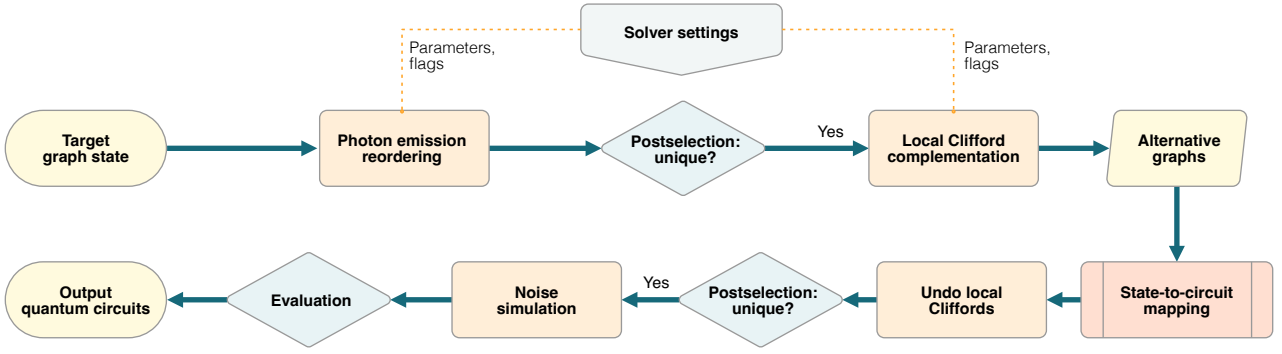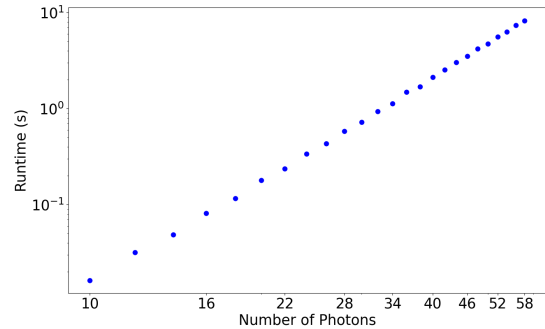
Figure 4: *Workflow of `AlternateTargetSolver`. After a target state is provided, photon emission ordering and local Clifford equivalency modules are used to create alternative graph targets. A state-to-circuit mapping algorithm is used to then obtain a set of candidate quantum circuits. These circuits are post-selected to remove redundancy, simulated with user-specified noise sources, and evaluated in performance by user-defined metrics to identify the best circuits.*

the number of alternative graphs may have a better scaling (e.g. it can grow polynomially with the number of photons). In Figure 5, we demonstrate the scaling of `AlternateTargetSolver`'s runtime (with `TimeReversedSolver` as the state-to-circuit mapping) for linear cluster states and repeater graph states when an exhaustive search of orbits is employed. In this case, the runtime of linear cluster states exhibits exponential scaling, whereas that of repeater graph states scales polynomially due to the symmetry of repeater graph states. It is also possible to use other methods to search local Clifford equivalent graph states. We refer to [31] for a correlation-assisted method that avoids the potentially poor scaling of an exhaustive search and achieves performance (in terms of circuit metrics) similar to that of the exhaustive search.

(a) *Linear cluster state*

(b) *Repeater graph state*

Figure 5: *Runtime of `AlternateTargetSolver` versus the number of photons for (a) linear cluster state (in semi-log scale for y-axis) and (b) repeater graph state (in log-log scale) using an exhaustive search of the graph orbit. A linear line in the semi-log plot indicates the runtime of linear cluster states scales exponentially, whereas a linear line in the log-log plot shows the runtime for repeater graph states scales polynomially. `TimeReversedSolver` is used as the state-to-circuit mapping in the `AlternateTargetSolver`.*

6 Conclusion

`GraphiQ` is a user-friendly Python framework for optimizing photonic graph state generation circuits. It supports circuit simulation and optimization in the presence of imperfections/noise, flexible cost function definition, and tools for producing several circuits per target output state. Easily extended and customised, `GraphiQ` can thus be a valuable tool for researchers studying graph state generation schemes obeying experimentally-relevant constraints. Further information is available in the Appendices as well as `GraphiQ`'s documentation.

Code availability

`GraphiQ` is available on https://github.com/graphiq-dev/graphiq under Apache 2.0 license. The full documentation can be found in https://graphiq.readthedocs.io/en/latest/.

Acknowledgement

A Background and implementation details

In this appendix, we provide a short theoretical background on graph states, the stabilizer formalism, conversion between state representations, noise models, photon emission ordering, and local complementation on graphs,.

A.1 Graph state

A graph state can be represented by a graph $G = (V, E)$ consisting of the vertex set V and the edge set E. Each vertex (or node) in V represents a qubit in the $|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ state. Each edge between two vertices represents a Controlled-Z (CZ) gate acting on those two qubits. From a given graph, it is easy to write down the corresponding graph state $|\psi\rangle$ as

$$|\psi\rangle = \prod_{(i,j)\in E} CZ_{i,j} |+\rangle^{\otimes|V|}, \qquad (2)$$

where $CZ_{i,j}$ denotes the CZ gate with the control qubit indexed by i and the target qubit indexed by j.

A.2 Stabilizer formalism

A.2.1 Stabilizer group and symplectic representation

A stabilizer group S on n qubits is a subgroup of the n-qubit Pauli group, P_n, such that it is Abelian and it does not contain $-I$. An n-qubit stabilizer state $|\psi\rangle$ is defined as a simultaneous eigenvector with eigenvalue 1 of a stabilizer group, S. The number of independent generators in the stabilizer group corresponding to an n-qubit stabilizer state is equal to n.

The binary symplectic representation of each Pauli operator is listed in Table 1. The set of n generators for the stabilizer group is then translated to two $n \times n$ matrices, one to represent Pauli X matrix components and the other to represent Pauli Z matrix components. The first digit in the translation becomes an entry in the X component matrix and the second digit becomes an entry in the Z component matrix.

Pauli operator	Binary representation
I	00
X	10
Y	11
Z	01

Table 1: *Translation between Pauli operators and their binary symplectic representation.*

The stabilizer formalism allows us to efficiently represent so-called stabilizer states, a subset of quantum states, by tracking the generators of the stabilizer group that stabilizes the quantum state [34, 35]. As `GraphiQ` is tailored for photonic graph state generation and graph states form a subset of stabilizer states, it is natural to consider

the stabilizer formalism. The number of generators needed to represent a stabilizer state grows linearly with the number of qubits, n. Since each generator consists of n Pauli operators, a stabilizer state can be represented by writing down n^2 Pauli operators. This stabilizer representation scales well with the number of qubits, making it more efficient for the study of large systems relative to the density matrix formalism. In practice, we adopt a binary symplectic representation to the stabilizer tableau. Specifically, we adopt the formalism in Ref. [34] to improve the efficiency of simulating measurements at a cost of doubling the size of the tableau. We keep track of destabilizer generators in addition to stabilizer generators. Destabilizer generators are Pauli operators such that destabilizer and stabilizer generators together generate the entire n-qubit Pauli group (modular four phases).

A.2.2 Tableau

In the `CliffordTableau` class implemented in `GraphiQ`, for an n-qubit stabilizer state, the state is represented by an $2n \times 2n$ binary matrix as below:

$$
\left(
\begin{array}{ccc|ccc}
x_{11} & \cdots & x_{1n} & z_{11} & \cdots & z_{1n} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
x_{n1} & \cdots & x_{nn} & z_{n1} & \cdots & z_{nn} \\
\hline
x_{(n+1)1} & \cdots & x_{(n+1)n} & z_{(n+1)1} & \cdots & z_{(n+1)n} \\
\vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
x_{(2n)1} & \cdots & x_{(2n)n} & z_{(2n)1} & \cdots & z_{(2n)n}
\end{array}
\right)
$$

where rows 1 to n represent the destabilizer generators, and rows $n+1$ to $2n$ represent the stabilizer generators. Two additional binary vectors (each of size $2n$) are used represent the phases (1 out of 4 possible values in the set $\{\pm 1, \pm i\}$) of those $2n$ generators:

$$
\left(
\begin{array}{c}
r_1 \\
\vdots \\
r_n \\
\hline
r_{n+1} \\
\vdots \\
r_{2n}
\end{array}
\right),
\left(
\begin{array}{c}
i_1 \\
\vdots \\
i_n \\
\hline
i_{n+1} \\
\vdots \\
i_{2n}
\end{array}
\right).
$$

We remark that the second phase vector is needed to properly handle all Clifford gates in the circuit simulation. While the phases for stabilizer

generators can be either 1 or -1, the destabilizer generators can take $\pm i$ phases in addition to ± 1 phases. Due to the need to keep track of the phases of destabilizer generators to guarantee the correct simulation of measurement gates, we choose to include this second phase vector. As such, it is different from Ref. [34] in this aspect.

As an example, the tableau for $|0\rangle^{\otimes n}$ is

$$
\left(
\begin{array}{c|c}
I_n & 0_n \\
\hline
0_n & I_n
\end{array}
\right), \tag{3}
$$

where I_n is $n \times n$ identity matrix and 0_n is the $n \times n$ matrix with all zeros. The two phase vectors of this state are initialized to all zeros.

A.2.3 Mixed stabilizer

The stabilizer formalism is suitable for graph state generation since graph states can be generated with only Clifford operations. However, only a subset of quantum operations can be efficiently simulated within the stabilizer representation, limiting the scope for circuit optimization, especially when noise sources are included. This poses the challenge that this representation is unsuited for the simulation of noise without judicious changes to the formalism. We amend this representation with a couple of approaches to handle mixed states.

A mixed state can be written as an ensemble of pure states. To handle mixed states in the stabilizer representation, one idea is to keep track of all pure states in the ensemble with their associated probability distribution. The `MixedStabilizer` class implements this idea. This representation is suitable when the number of pure states in the ensemble is relatively small. In the simulation, each tableau needs to be updated independently and thus the running time of the simulation also grows with the number of pure states in the ensemble. In practice, if the number of possible events can grow fast when considering a noise model that introduces probabilistic noises (e.g. a depolarizing noise), the memory requirement and the simulation running time for this representation can be very demanding. However, it is not always necessary to keep track of all possible events. For example, if one is interested in a bound on a given metric instead of a precise value, ignoring states with very low probabilities does not loosen the bound significantly while saving substantial computational resources. For this reason,

the `MixedStabilizer` class allows the probability distribution to be subnormalized, that is, the sum of all probabilities is smaller or equal to 1.

A.3 Conversion between state representations

Given the clear advantages and disadvantages of various state representations, it is useful to be able to convert one representation to another as necessary. As the set of all graph states is a proper subset of stabilizer states, and the set of all stabilizer states is a proper subset of all quantum states representable by density matrices, it is expected that not all conversions are allowed. Here, we restrict to the cases where a given state is a stabilizer state or a probabilistic mixture of stabilizer states.

Conversion between representations is mostly straightforward with the exception of converting a density matrix representation to a graph representation.

From a graph $G = (V, E)$ whose adjacency matrix is θ, the stabilizer generators R_j's corresponding to the graph state are given by

$$R_j = X_j \prod_{k=1}^{n} Z_k^{\theta_{kj}}, \qquad (4)$$

for $j = 1, \ldots, n$, where X_j is the Pauli X operator acting on the j-th qubit, Z_k is the Pauli Z operator acting on the k-th qubit and θ_{kj} is the (k, j)-entry of the adjacency matrix θ.

From the set of generators $\{R_j : j = 1, \ldots, n\}$, one can construct the density matrix ρ of the stabilizer state by

$$\rho = \frac{1}{2^n} \prod_{j=1}^{n} (I_n + R_j). \qquad (5)$$

From a graph $G = (V, E)$, it is straightforward to write down the corresponding density matrix $\rho := |\psi\rangle \langle\psi|$, where ψ is given in Eq. (2). From the graph, the stabilizer generators can be found by Eq. (2). In our stabilizer representation, we also need to find destabilizer generators. `GraphiQ` includes a function to find gates that can disentangle an n-qubit stabilizer state into $|0\rangle^{\otimes n}$. As the `CliffordTableau` for $|0\rangle^{\otimes n}$ is given in Eq. (3), one can construct the `CliffordTableau` for the stabilizer state by applying those gates in the reversed order. The conversion from a stabilizer formalism to a density matrix simply follows Eq. (5).

A less straightforward conversion is from a density matrix representation to a graph representation. For this conversion, we have developed a procedure that constructs an adjacency matrix from a density matrix. To construct the graph, we must verify the existence of an edge between each pair of qubits in the state. To do so, we apply Pauli Z measurements to all qubits except those indexed by i and j. All these measurements effectively disentangle all other qubits and keep these two qubits indexed by i and j. After obtaining the reduced density matrix of these two remaining qubits, we calculate the negativity of the reduced density matrix [17, 39] to verify if an edge exists between these two qubits. Since the space is reduced to two qubits, the negativity serves as an accurate entanglement metric and gives a positive value when two nodes are connected [40, 41]. By applying this procedure to every pair of nodes, we can construct the adjacency matrix from the density matrix. We note that this procedure works under the assumption that our state is a graph state.

Algorithm 1 Converting the density matrix of a graph state to its adjacency matrix graph representation.

Inputs:

ρ	A density matrix that corresponds to an n-qubit graph state
$\delta \in [0.49, 0.5]$	A threshold for negativity

Output:

  θ An adjacency matrix

Algorithm:

1. Initialize $\theta := 0_n$;

2. for $i = 1$ to $n - 1$:
    for $j = i + 1$ to n:

   (a) Obtain $\rho^{(i,j)}$ by measuring all qubits, except qubits indexed by i, j, in the Z basis;

   (b) Calculate negativity of the two qubit system $\rho^{(i,j)}$;

   (c) If $\rho^{(i,j)} \geq \delta$, set $\theta(i, j) = \theta(j, i) = 1$.

3. Return θ.

We note that one approach to translate a density matrix $\rho = |\psi\rangle \langle\psi|$ corresponding to a pure

stabilizer state to a stabilizer representation is as follows. First, we find the stabilizer group corresponding to the stabilizer state $|\psi\rangle$ by iterating through each element M in the n-qubit Pauli group and recording M if it stabilizes the state $|\psi\rangle$, that is, $M|\psi\rangle = |\psi\rangle$. After obtaining the stabilizer group, we can find a set of generators that generate the whole group. Then, using that set of generators, we can write down the binary symplectic representation. While this procedure can return a correct answer, it is very inefficient due to the size of the n-qubit Pauli group. As we have an efficient ($O(n^2)$) approach to convert a density matrix to the graph representation and also an efficient way to convert a graph representation to the stabilizer representation, we can combine these two methods to move from the density matrix to the stabilizer representation for graph stabilizer states.

A.4 Noise models

In `GraphiQ`, we allow for the flexible placement of noise. A noise source can arise before or after a quantum gate. It can also replace a gate with another gate. The noise models presently implemented in `GraphiQ` focus on common noise types for photonic graph state generation, particularly, depolarizing noise and photon loss. We also include additional noise models that can be useful for versatile circuit simulation purposes.

As discussed before, each state representation has a varying capacity for representing mixed states. Since noise sources often turn pure states into mixed states, each state representation will have a different ability to handle noise. Both the density matrix and stabilizer backends support depolarizing noise and photon loss as well as Pauli errors. The density matrix backend also supports arbitrary unitary or mixed unitary errors.

We discuss depolarizing noise and photon loss below.

A.4.1 Depolarizing noise

Depolarizing noise is often used to model qubit errors. A depolarizing channel acting on a density matrix ρ is defined as

$$\mathcal{D}_p(\rho) = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z),$$

(6)

where X, Y, Z are Pauli matrices, and p is the depolarizing probability. The resulting state $\mathcal{D}_p(\rho)$ is in general a mixed state. The density matrix is the most versatile as it can inherently and easily include any type of noise. However, it soon becomes impractical when the number of qubits increases. The stabilizer formalism works well for pure states but cannot directly represent mixed states. To handle noises like depolarizing noise in the stabilizer representation, it is necessary to extend the representation, as in the mixed stabilizer representation implemented in the `MixedStabilizer` class. Alternatively, one may also consider Monte Carlo simulation to simulate noisy events. `GraphiQ` has an implementation of Monte Carlo simulation for this purpose. Another approach is to keep track of the most probable event(s) using the `MixedStabilizer` class.

A.4.2 Photon loss

When a photon is lost, the corresponding qubit moves outside the computational space. One approach to represent photon loss is to use a qutrit description where the first two dimensions are the original qubit space and the third dimension represents the vacuum state. This extension naturally allows the density matrix representation to handle loss. Another approach is to calculate the probability of the event where no photon is lost. Arguably, this event is the event of interest for photonic graph state generation. Metrics like fidelity dismiss all other events since the target state has zero overlap with any state that loses one or more photons. In `GraphiQ`, we thus implement the latter idea.

We remark that it is possible to make a protocol loss-tolerant by applying some quantum error correction code to protect logical qubits [7]. The present version of `GraphiQ` (version 0.1.0) is restricted to the case where no such encoding is performed. However, we expect to implement an error correction module in future versions of `GraphiQ`.

A.5 Photon emission ordering

As multiple photons in a graph state can be emitted by one common quantum emitter, it is important to consider in what order these photons should be emitted. Different emission or-

derings are likely to lead to different quantum circuits since the local entanglement structures might be quite different for different photons. Consequently, the user needs to specify the photon emission ordering in the target graph state, which we use to number the graph nodes, i.e, we label nodes numerically where a smaller number means an earlier time slot in the emission process.

As discussed in Ref. [29], different emission orderings require different minimal numbers of quantum emitters to generate the graph state deterministically. However, the task of finding optimal photon emission orderings for an arbitrary graph is an NP-hard problem [29]. Nevertheless, `GraphiQ` provides the users with the option to automatically consider different permutations of the emission ordering while solving for a generating circuit. This search can be either exhaustive or driven by the random sampling of all possible $n!$ permutations. Note, however, that not every permutation yields a distinct new graph since a subset of permutations result in graph automorphisms. `GraphiQ`'s relabelling module removes the automorphic cases so that the returned set of graphs are all distinguishable. A user can specify parameters to choose how many different emission orderings to consider and how to generate a selective subset of orderings.

A.6 Local Clifford and local complementation

For a graph $G = (V, E)$, a local complementation on a node $v \in V$ corresponds to applying the complementation to a subgraph consisting of all neighbors of v. Let $N(v)$ denote all the neighboring nodes of v (i.e., $w \in V$ is in $N(v)$ if and only if $(v, w) \in E$.) Applying the local complementation on the node v generates another graph $G' = (V, E')$ where for each pair of $u, w \in N(v)$, if $(u, w) \in E$, then $(u, w) \notin E'$; if $(u, w) \notin E$, then $(u, w) \in E'$; all other edges in E are also in E'. Figure 6 shows an example of applying local complementation on node 1.

The relation between local complementation actions on a graph and local Clifford gates applied to the corresponding graph state is discussed in Ref. [36]. Each local complementation action corresponds to a sequence of local Clifford gates. If a graph can be transformed into another graph via local complementation actions, then the corresponding graph states are local Clifford equivalent. An efficient algorithm to find local Clifford

(a) *Initial graph* **(b)** *Apply local complementation on node 1*
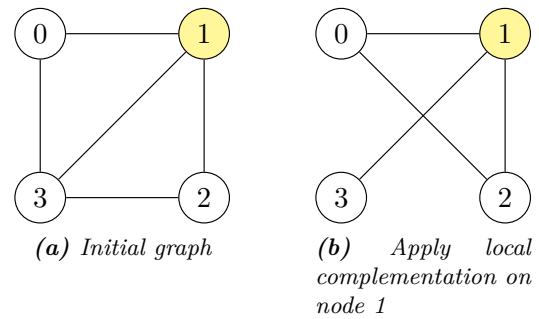
Figure 6: *Illustration of local complementation. By applying the local complementation operation on node 1, neighbors of node 1 that were connected to each other in the initial graph are now disconnected and those that were disconnected are now connected.*

operations between two local Clifford equivalent states is given in Ref. [42]. The overall complexity of this algorithm is $O(n^4)$.

B Examples

B.1 Code examples

Codeblock 5 demonstrates an example of simulating a quantum circuit that generates a photonic 3-qubit linear cluster using the density matrix backend.

```
from graphiq.circuit.circuit_dag import CircuitDAG
from graphiq.circuit.ops import Hadamard, CNOT, MeasurementCNOTandReset
from graphiq.backends.density_matrix.compiler import DensityMatrixCompiler

# initialize a quantum circuit with 1 emitter and 3 photonic qubits
circuit = CircuitDAG(n_emitter=1, n_photon=3)

circuit.add(Hadamard(register=0, reg_type="e"))
circuit.add(CNOT(control=0, control_type="e", target=0, target_type="p"))
```

```
10  circuit.add(Hadamard(register=0, reg_type="e"))
11  circuit.add(CNOT(control=0, control_type="e", target=1, target_type="p"))
12  circuit.add(CNOT(control=0, control_type="e", target=2, target_type="p"))
13  circuit.add(Hadamard(register=2, reg_type="p"))
14  circuit.add(Hadamard(register=0, reg_type="e"))
15  circuit.add(
16      MeasurementCNOTandReset(control=0, control_type="e", target=2, target_type="p")
17  )
18
19  # plot the circuit as a directed, acyclic graph
20  circuit.draw_circuit()
21
22  # simulate the circuit with density matrix compiler
23  compiler = DensityMatrixCompiler()
24  state = compiler.compile(circuit)
25
26  # trace out the emitter qubit
27  state.partial_trace(keep=(0, 1, 2), dims=4 * [2])
```

Codeblock 5: *Simulating a quantum circuit that produces the three-qubit linear cluster state in Figure 3(a) using the density matrix backend.*

Codeblock 6 presents an example of evaluating the quantum circuit found by `TimeReversedSolver` using the infidelity and the circuit depth metrics.

```
1   import networkx as nx
2   from graphiq.backends.stabilizer.compiler import StabilizerCompiler
3   from graphiq.metrics import Infidelity, CircuitDepth
4   from graphiq.solvers.time_reversed_solver import TimeReversedSolver
5   from graphiq.state import QuantumState
6
7   # a 3-qubit linear cluster state as the target
8   graph = nx.Graph([(0, 1), (1, 2)])
9   target = QuantumState(graph, rep_type="graph")
10
11  # use the infidelity as the metric
12  metric = Infidelity(target)
13
14  # use the stabilizer backend
15  compiler = StabilizerCompiler()
16
17  # use the time reversed solver by Li et al.
18  solver = TimeReversedSolver(
19      target=target,
20      metric=metric,
21      compiler=compiler,
22  )
23
24  solver.solve()
25  score, circuit = solver.result
26  print(f"The fidelity between the target and generated states is {1 - score}")
27  depth = CircuitDepth()
28  print(f"Circuit depth is {depth.evaluate(state=None, circuit=circuit)}")
29  circuit.draw_circuit()
```

Codeblock 6: *Using the `TimeReversedSolver` to find a quantum circuit that generates the 3-qubit linear cluster state, followed by evaluating the output state's infidelity and circuit depth.*

Codeblock 7 presents an example of using `AlternateTargetSolver`. This example uses the stabilizer

backend for the noise-free simulation and uses the density matrix backend for the noise simulation. The noise model used is depolarizing channels. More examples can be found in the Jupyter notebooks in the `GraphiQ`'s code repository.

```python
from graphiq.solvers.alternate_target_solver import AlternateTargetSolver,
    AlternateTargetSolverSetting
from graphiq.backends.stabilizer.compiler import StabilizerCompiler
from graphiq.backends.density_matrix.compiler import DensityMatrixCompiler
from graphiq.metrics import Infidelity

# assume target_graph is a valid QuantumState object
setting = AlternateTargetSolverSetting()
solver = AlternateTargetSolver(
    target=target_graph, solver_setting=setting, noise_model_mapping="depolarizing"
)

# solver options
solver.metric = Infidelity
solver.compiler = StabilizerCompiler()
solver.noise_compiler = DensityMatrixCompiler()
solver.io = None
solver.noise_simulation = True
solver.seed = 1

# obtain results
results = solver.solve()
```

Codeblock 7: *Using the `AlternateTargetSolver` to find quantum circuits that generate a target graph state.*

B.2 Sample results using `GraphiQ`

B.2.1 3-qubit linear cluster state

Figure 7 shows an example of finding alternative quantum circuits for the three-qubit linear cluster state using our probabilistic solvers, compared against the output of the deterministic solver based on Ref. [29]. We see that our probabilistic solvers can potentially produce quantum circuits with reduced circuit depth.

(a) *Using* `TimeReversedSolver`   (b) *Using* `EvolutionarySolver`
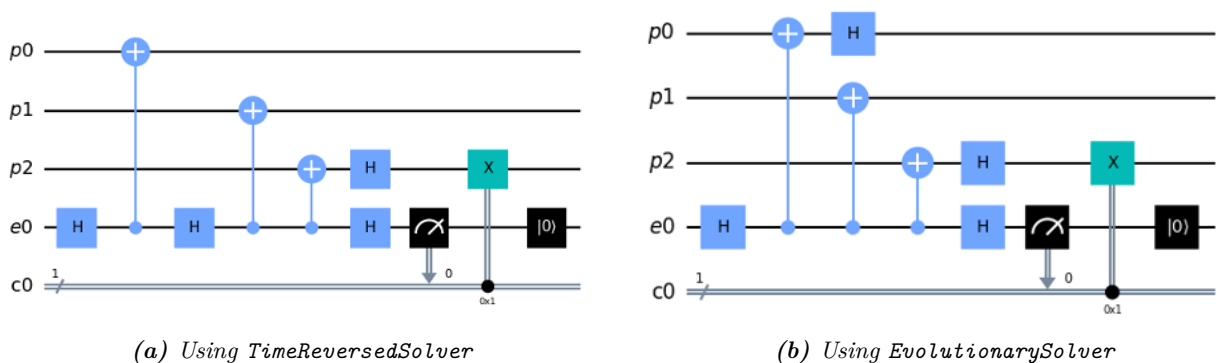
Figure 7: *Optimal circuits for generating the three-qubit linear cluster state when the cost function is the infidelity function, using (a) the `TimeReversedSolver` based on Ref. [29] and (b) using `EvolutionarySolver` with the random search option.*

B.2.2 Improving fidelity

(a) *original*   **(b)** *alternative*
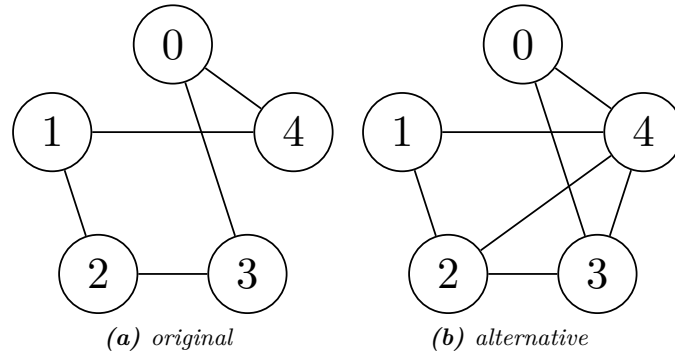
Figure 8: *Original target graph and its alternative, locally equivalent shape found using the* `AlternateTargetSolver`*.*

Using the graph shown in Figure 8(a) as the target graph for `AlternateTargetSolver`, we search for a generation circuit that offers a better fidelity for the final state with respect to when the circuit is obtained using the `TimeReversedSolver`.
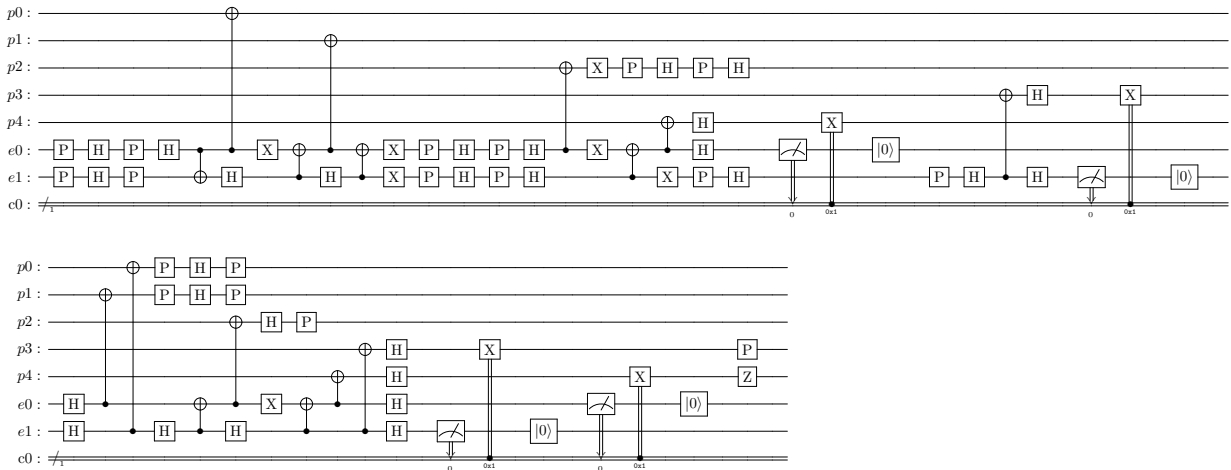

Figure 9: *The original (top) and alternative (bottom) circuits, corresponding to the use of* `TimeReversedSolver` *and* `AlternateTargetSolver` *respectively.*

The circuits for the original graph (found via `TimeReversedSolver`) and the alternative graph (found via `AlternateTargetSolver`) are depicted in Figure 9. When applying a depolarizing noise model on the emitter gates with a 0.01 depolarization rate, the fidelity is increased from 0.66 to 0.81 for the alternative case. The reduction in the size of the circuit is also evident in Figure 9.

B.2.3 3D cluster state $2 \times 2 \times 3$

This example demonstrates the application of `AlternateTargetSolver` to explore different generation circuits using the local Clifford equivalency orbit of a 3D cluster state (Figure 10). Table 2 shows the range of values for a set of circuit metrics, encompassing the total number of unitary gates, the number of CNOT gates between emitters, and the circuit depth, across all available options.
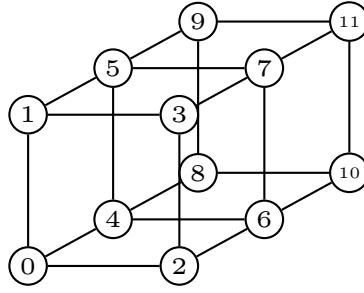
Figure 10: *A 12-qubit $2 \times 2 \times 3$ 3D cluster state.*

Metrics	Unitaries	CNOTs	Depth
Best case	46	12	16
Worst case	203	28	59

Table 2: *The best and worst values for a set of circuit metrics over 5160 members of the LC equivalency class of a three-dimensional $(2 \times 2 \times 3)$ cluster state. The metrics correspond to the number of unitary gates, number of emitter-emitter CNOT gates, and the depth of the corresponding quantum circuits.*

References

[1] K. Azuma, K. Tamaki, and H.-K. Lo. All-photonic quantum repeaters. *Nature Communications*, 6(1):6787, 2015. DOI: https://doi.org/10.1038/ncomms7787.

[2] P. Hilaire, E. Barnes, and S. E. Economou. Resource requirements for efficient quantum communication using all-photonic graph states generated from a few matter qubits. *Quantum*, 5:397, 2021. DOI: https://doi.org/10.22331/q-2021-02-15-397. arXiv:2005.07198.

[3] S. Y. Looi, L. Yu, V. Gheorghiu, and R. B. Griffiths. Quantum-error-correcting codes using qudit graph states. *Phys. Rev. A*, 78(4):042303, 2008. DOI: https://doi.org/10.1103/PhysRevA.78.042303.

[4] B. A. Bell, D. A. Herrera-Martí, M. S. Tame, D. Markham, W. J. Wadsworth, and J. G. Rarity. Experimental demonstration of a graph state quantum error-correction code. *Nature Communications*, 5(1):3658, 2014. DOI: https://doi.org/10.1038/ncomms4658.

[5] R. Raussendorf and H. J. Briegel. A One-Way Quantum Computer. *Phys. Rev. Lett.*, 86(22): 5188–5191, 2001. DOI: https://doi.org/10.1103/PhysRevLett.86.5188.

[6] R. Raussendorf, E. D. Browne, and H. J. Briegel. Measurement-based quantum computation on cluster states. *Phys. Rev. A*, 68(2):022312, 2003. DOI: https://doi.org/10.1103/PhysRevA.68.022312.

[7] M. Varnava, D. E. Browne, and T. Rudolph. Loss tolerance in one-way quantum computation via counterfactual error correction. *Phys. Rev. Lett.*, 97:120501, 2006. DOI: https://doi.org/10.1103/PhysRevLett.97.120501.

[8] M. A. Nielsen. Optical quantum computation using cluster states. *Phys. Rev. Lett.*, 93:040503, 2004. DOI: https://doi.org/10.1103/PhysRevLett.93.040503.

[9] M. Gimeno-Segovia, P. Shadbolt, D. E. Browne, and T. Rudolph. From three-photon greenberger-horne-zeilinger states to ballistic universal quantum computation. *Phys. Rev. Lett.*, 115:020502, 2015. DOI: https://doi.org/10.1103/PhysRevLett.115.020502.

[10] S. Bartolucci, P. Birchall, H. Bombin, H. Cable, C. Dawson, M. Gimeno-Segovia, E. Johnston, K. Kieling, N. Nickerson, M. Pant, F. Pastawski, T. Rudolph, and C. Sparrow. Fusion-based quantum computation. *Nature Communications*, 19:912, 2023. DOI: https://doi.org/10.1038/s41467-023-36493-1.

[11] N. Shettell and D. Markham. Graph States as a Resource for Quantum Metrology. *Phys. Rev. Letters*, 124(11):110502, 2020. DOI: https://doi.org/10.1103/PhysRevLett.124.110502.

[12] D. E. Browne and T. Rudolph. Resource-Efficient Linear Optical Quantum Computation. *Phys. Rev. Lett.*, 95(1):010501, 2005. DOI: https://doi.org/10.1103/PhysRevLett.95.010501.

[13] F. Ewert and P. van Loock. 3/4-efficient bell measurement with passive linear optics and unentangled ancillae. *Phys. Rev. Lett.*, 113:140403, 2014. DOI: https://doi.org/10.1103/PhysRevLett.113.140403.

[14] M. Pant, H. Krovi, D. Englund, and S. Guha. Rate-distance tradeoff and resource costs for all-optical quantum repeaters. *Phys. Rev. A*, 95:012304, 2017. DOI: https://doi.org/10.1103/PhysRevA.95.012304.

[15] A. Russo, E. Barnes, and S. E. Economou. Generation of arbitrary all-photonic graph states from quantum emitters. *New Journal of Physics*, 21(5):055002, 2019. DOI: https://doi.org/10.1088/1367-2630/ab193d.

[16] N. H. Lindner and T. Rudolph. Proposal for pulsed on-demand sources of photonic cluster state strings. *Phys. Rev. Lett.*, 103(11):113602, 2009. DOI: https://doi.org/10.1103/PhysRevLett.103.113602.

[17] I. Schwartz, D. Cogan, E. R. Schmidgall, Y. Don, L. Gantz, O. Kenneth, N. H. Lindner, and D. Gershoni. Deterministic generation of a cluster state of entangled photons. *Science*, 354(6311): 434–437, 2016. DOI: https://doi.org/10.1126/science.aah4758.

[18] D. Cogan, Z.-E. Su, O. Kenneth, and D. Gershoni. Deterministic source of indistinguishable photons in a cluster state. *Nat. Photon.*, pages 324–329, 2023. DOI: https://doi.org/10.1038/s41566-022-01152-2.

[19] P. Thomas, L. Ruscio, O. Morin, and G. Rempe. Efficient generation of entangled multiphoton graph states from a single atom. *Nature*, 608(7924):677–681, 2022. DOI: https://doi.org/10.1038/s41586-022-04987-5.

[20] S. E. Economou, N. Lindner, and T. Rudolph. Optically generated 2-dimensional photonic cluster state from coupled quantum dots. *Phys. Rev. Lett.*, 105:093601, 2010. DOI: https://doi.org/10.1103/PhysRevLett.105.093601.

[21] D. Buterakos, E. Barnes, and S. E. Economou. Deterministic generation of all-photonic quantum repeaters from solid-state emitters. *Phys. Rev. X*, 7:041023, 2017. DOI: https://doi.org/10.1103/PhysRevX.7.041023.

[22] M. Gimeno-Segovia, T. Rudolph, and S. E. Economou. Deterministic generation of large-scale entangled photonic cluster state from interacting solid state emitters. *Phys. Rev. Lett.*, 123:070501, 2019. DOI: https://doi.org/10.1103/PhysRevLett.123.070501.

[23] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, S. Ahmed, V. Ajith, M. S. Alam, G. Alonso-Linaje, B. AkashNarayanan, A. Asadi, J. M. Arrazola, U. Azad, S. Banning, C. Blank, T. R. Bromley, B. A. Cordier, J. Ceroni, A. Delgado, O. D. Matteo, A. Dusko, T. Garg, D. Guala, A. Hayes, R. Hill, A. Ijaz, T. Isacsson, D. Ittah, S. Jahangiri, P. Jain, E. Jiang, A. Khandelwal, K. Kottmann, R. A. Lang, C. Lee, T. Loke, A. Lowe, K. McKiernan, J. J. Meyer, J. A. Montañez-Barrera, R. Moyard, Z. Niu, L. J. O'Riordan, S. Oud, A. Panigrahi, C.-Y. Park, D. Polatajko, N. Quesada, C. Roberts, N. Sá, I. Schoch, B. Shi, S. Shu, S. Sim, A. Singh, I. Strandberg, J. Soni, A. Száva, S. Thabet, R. A. Vargas-Hernández, T. Vincent, N. Vitucci, M. Weber, D. Wierichs, R. Wiersema, M. Willmann, V. Wong, S. Zhang, and N. Killoran. Pennylane: Automatic differentiation of hybrid quantum-classical computations. *arXiv preprint arXiv:1811.04968v4*, 2018. DOI: https://doi.org/10.48550/arXiv.1811.04968.

[24] J. Gray. quimb: A python package for quantum information and many-body calculations. *The Journal of Open Source Software*, 3(29):819, 2018. DOI: https://doi.org/10.21105/joss.00819.

[25] N. Killoran, J. Izaac, N. Quesada, V. Bergholm, M. Amy, and C. Weedbrook. Strawberry fields: A software platform for photonic quantum computing. *Quantum*, 3:129, 2019. DOI: https://doi.org/10.22331/q-2019-03-11-129.

[26] X.-Z. Luo, J.-G. Liu, P. Zhang, and L. Wang. Yao.jl: Extensible, efficient framework for quantum algorithm design. *Quantum*, 4:341, 2020. DOI: https://doi.org/10.22331/q-2020-10-11-341.

[27] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta. Quantum computing with Qiskit. *arXiv preprint arXiv:2405.08810*, 2024. DOI: https://doi.org/10.48550/arXiv.2405.08810.

[28] N. Heurtel, A. Fyrillas, G. de Gliniasty, R. L. Bihan, S. Malherbe, M. Pailhas, E. Bertasi, B. Bourdoncle, P.-E. Emeriau, R. Mezher, L. Music, N. Belabas, B. Valiron, P. Senellart, S. Mansfield, and J. Senellart. Perceval: A Software Platform for Discrete Variable Photonic Quantum Computing. *Quantum*, 7:931, 2023. DOI: https://doi.org/10.22331/q-2023-02-21-931.

[29] B. Li, S. E. Economou, and E. Barnes. Photonic resource state generation from a minimal number of quantum emitters. *npj. Quantum Information*, 8(11):1–7, 2022. DOI: https://doi.org/10.1038/s41534-022-00522-6.

[30] S.-H. Lee and H. Jeong. Graph-theoretical optimization of fusion-based graph state generation. *Quantum*, 7:1212, 2023. DOI: https://doi.org/10.22331/q-2023-12-20-1212.

[31] S. Ghanbari, J. Lin, B. MacLellan, L. Robichaud, P. Roztocki, and H.-K. Lo. Optimization of deterministic photonic graph state generation via local operations. *arXiv preprint arXiv:2401.00635*, 2024. DOI: https://doi.org/10.48550/arXiv.2401.00635.

[32] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition.* Cambridge University Press, 2010. DOI: https://doi.org/10.1017/CBO9780511976667.

[33] D. Gottesman. Stabilizer codes and quantum error correction. *arXiv preprint arXiv:quant-ph/9705052*, 1997. DOI: https://doi.org/10.48550/arXiv.quant-ph/9705052.

[34] S. Aaronson and D. Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70(5): 052328, 2004. DOI: https://doi.org/10.1103/PhysRevA.70.052328.

[35] K. M. R. Audenaert and M. B. Plenio. Entanglement on mixed stabilizer states: normal forms and reduction procedures. *New Journal of Physics*, 7(1):170, 2005. ISSN 1367-2630. DOI: https://doi.org/10.1088/1367-2630/7/1/170.

[36] M. Van den Nest, J. Dehaene, and B. De Moor. Graphical description of the action of local Clifford transformations on graph states. *Phys. Rev. A*, 69(2):022316, 2004. DOI: https://doi.org/10.1103/PhysRevA.69.022316.

[37] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta. Open Quantum Assembly Language. *arXiv preprint arXiv:1707.03429*, 2017. DOI: https://doi.org/10.48550/arXiv.1707.03429. arXiv:1707.03429.

[38] H. J. Garcia, I. L. Markov, and A. W. Cross. Efficient inner-product algorithm for stabilizer states. *arXiv preprint arXiv:1210.6646*, 2012. DOI: https://doi.org/10.48550/arXiv.1210.6646.

[39] M. B. Plenio and S. Virmani. An introduction to entanglement measures. *Quantum Inf. Comput.*, 7(1):1–51, 2007. DOI: https://doi.org/10.5555/2011706.2011707.

[40] A. Peres. Separability criterion for density matrices. *Phys. Rev. Lett.*, 77:1413–1415, 1996. DOI: https://doi.org/10.1103/PhysRevLett.77.1413.

[41] M. Horodecki, P. Horodecki, and R. Horodecki. Separability of mixed states: necessary and sufficient conditions. *Physics Letters A*, 223(1):1–8, 1996. DOI: https://doi.org/10.1016/S0375-9601(96)00706-2.

[42] M. Van den Nest, J. Dehaene, and B. De Moor. Efficient algorithm to recognize the local Clifford equivalence of graph states. *Phys. Rev. A*, 70(3):034302, 2004. DOI: https://doi.org/10.1103/PhysRevA.70.034302.