Adaptive framework for failure-aware protocols in fusion-based graph-state generation

Korbinian Staudacher [1,*], Bhilahari Jeevanesan [2,†] and Tobias Guggemos [1,2,3,‡]
[1] *Ludwig-Maximilian-Universität München, MNM-Team, 80539 Munich, Germany*
[2] *Remote Sensing Technology Institute, German Aerospace Center (DLR), 82234 Wessling, Germany*
[3] *University of Vienna, Faculty of Physics, Vienna Center for Quantum Science and Technology (VCQ), 1090 Vienna, Austria*

(Received 21 January 2026; accepted 29 July 2026; published 17 September 2026)

We consider the generation of photonic graph states in a linear optics setting where sequential nondeterministic fusion measurements are used to build large graph states out of small linear clusters and develop a framework to optimize the building process using graph-theoretic characterizations of fusion networks. We present graph-state generation protocols for linear cluster resource states and type-I/type-II fusions which are adaptive to fusion failure, that is, they reuse leftover graph states in the remaining building process. To estimate hardware costs, we interpret our protocols as finite Markov processes. This viewpoint allows us to cast the expected number of fusion measurements until success as a *first passage* problem. We then deploy a pipeline of polynomial algorithms to optimize arbitrary graph states, extract fusion networks, and find beneficial orderings of fusions. We check the effectiveness of these algorithms by verifying that the corresponding *mean first passage times* are decreased. We evaluate our pipeline for different initial resource states and fusion mechanisms with varying success probabilities. Results show reductions in the fusion overhead by several orders of magnitude when compared to simple repeat until success protocols and by up to 40% when compared to state-of-the-art graph-state generation protocols, especially for realistic fusion success probabilities between 50% and 75%.

DOI: 10.1103/2z98-6bsp

I. INTRODUCTION

Graph states play an important role as a fundamental resource in several quantum computing schemes [1,2], as scalable entangled units in quantum networks [3] and for error-correction schemes [4]. They have been realized experimentally on most leading quantum computing implementations such as trapped-ion systems [5,6], linear optics platforms [7,8], superconducting systems [9,10] and neutral-atom devices [11]. For linear optics architectures, the major hurdle in building graph states is the noninteracting nature of photons. Thus entangling operations can only be implemented by using nonlinear elements such as postselecting on measurement outcomes [12]. While linear clusters of entangled photons can be produced deterministically from quantum emitters [13], creating arbitrary graph states requires either interacting emitters [14] or photonic probabilistic entangling measurements, known as fusions [15,16].

In this work, we focus on fusion measurements in an all-photonic architecture where we build arbitrary graph states by sequentially fusing small initial resource states (cf. Fig. 1): At each step we swap qubits and apply a fusion measurement on two adjacent qubits. Based on the fusion outcome, we may then need to add new resource states and again swap qubits and fuse until we have created the desired target graph state. This process of graph-state generation can be captured by so-called *fusion networks*, which give a configuration of resource states and fusions to reach a target graph state [2,17]. In the past few years, there has been a surge of effort to optimize graph-state generation protocols for photonic architectures, including compilation towards fusion networks [17–19] and towards hardware-specific emitter schemes [16,20,21]. These proposals rely on repeat-until-success schemes with or without redundantly encoded qubits. In other words, if a fusion fails, either the graph states involved in the fusion are built up anew from the beginning, or another photon of the redundantly encoded qubit is used to repeat the fusion.

Here we investigate an *adaptive* approach on graph-state generation protocols: After a failed fusion, we reconfigure the network by incorporating new resource states as well as parts of the already existing graph state. We characterize fusion networks by relating fusions to graph-theoretic minor relations and derive procedures to generate fusion networks for arbitrary graphs using either type-I or type-II fusion and linear cluster resource states. We then give adaptive protocols to build graph states from fusion networks with strategies to rebuild networks under fusion failure. To evaluate the efficiency of such protocols, we view fusion success and failure probabilities in the language of finite Markov processes. This allows us to capture the expected number of fusion attempts until successful graph-state generation in terms of the *mean*

*Contact author: staudacher@nm.ifi.lmu.de
†Contact author: bhilahari.jeevanesan@dlr.de
‡Contact author: guggemos@nm.ifi.lmu.de; tobias.guggemos@dlr.de

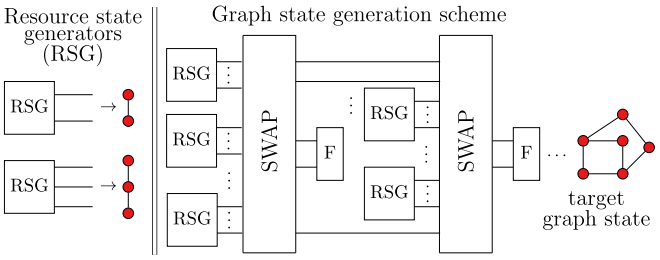

FIG. 1. A schematic overview of the graph-state generation process. We consider resource-generating modules (RSGs), consisting of small entangled units, such as two- or three-qubit linear cluster states serving as input for a sequential buildup process using fusions (F) on adjacent qubits.

first passage time (MFPT). We set up a scheme to calculate the MFPT from the Markov process and provide algorithms to reduce this time for arbitrary graph states when using our protocols, including the search for less costly locally equivalent graph states and optimizing the order in which fusions are applied.

The paper is structured as follows: We start in Sec. II with a discussion of some graph-state preliminaries. In Sec. III, we introduce graph-theoretic fusion networks and show how to obtain fusion networks for arbitrary graphs. We introduce adaptive protocols in Sec. IV and show how to evaluate these protocols using mean first passage times in Sec. V. We incorporate algorithms to optimize graph-state generation towards low mean first passage times in Sec. VI and, finally, we evaluate our strategies in Sec. VII.

II. GRAPH-STATE PRELIMINARIES

Graph states are a family of quantum states where the entanglement structure between qubits can be described as a simple undirected graph $G = (V, E)$, with a vertex set V and edges E [22]. The vertices of V of the graph represent the qubits, and entanglement operations are applied between qubits connected by edges in E. Formally, a graph state $|G\rangle$, with $G = (V, E)$ being an undirected graph, is of the following form:

$$|G\rangle = \prod_{(i,j)\in E} \mathrm{CZ}_{i,j} |+\rangle^{\otimes V}. \tag{1}$$

The CZ operation is a controlled-Z quantum gate which entangles pairs of qubits. Here, they act on qubits in the state $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$. Clearly, such graph states are obtained by acting with Clifford gates only, thus they are stabilizer states. An equivalent formulation of this graph state is given by listing its Clifford group generators

$$K_i = X_i \prod_{j\in N(i)} Z_j. \tag{2}$$

Here, for each qubit i the product extends over all neighbors of i, denoted by $N(i)$. Graph states form only a subclass of stabilizer states, yet any stabilizer state can be transformed into a graph state with just local Clifford unitaries [22]. We next describe some transformations that can be applied to graph states.

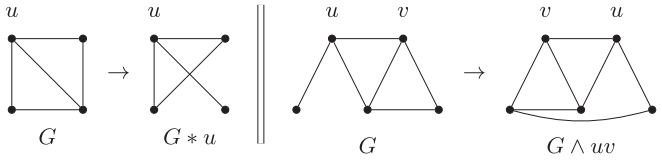

FIG. 2. Examples of graph-theoretic rewrites on two simple graphs. Left: a local complementation operation is applied at vertex u, which toggles the edges between neighbors of u. Right: a pivoting operation on edge $\{u, v\}$ toggling edges between exclusive neighbors of u, v and the shared neighbors of u and v. Further, u and v switch their neighbors.

A. Graph-theoretic rewrites

Given a simple graph $G = (V, E)$ and a vertex $u \in V$, the first graph-rewrite we consider is a so-called *local complementation* on the vertex u that transforms G into a new graph $G * u$. This graph is obtained from G by first considering all neighboring vertices $N(u)$ of u. When two vertices $v, w \in N(u)$ are connected in G, they are disconnected in $G * u$ and vice versa. An example is given in Fig. 2.

An equivalent definition is often given in the literature in terms of the symmetric set difference operator $\triangle$. Given two sets A and B, this is defined as $A \triangle B = (A \setminus B) \cup (B \setminus A)$. Thus $A \triangle B$ consists of precisely those elements that lie in exactly one of the sets.

With this, the graph $G * u$ has the same vertex set V, but the edge set becomes

$$E' = E \triangle K_{N(u)}, \tag{3}$$

where $K_{N(u)}$ is the complete graph on vertices $N(u)$ u [23]. Interestingly, another application of $\triangle$ will undo the operation, because the set difference operator satisfies

$$(A \triangle B) \triangle B = A. \tag{4}$$

Another operation that we will apply below is *pivoting*. This is defined as three alternating local complementations on the end points of an edge $\{u, v\} \in E$:

$$G \wedge uv = [(G * u) * v] * u = [(G * v) * u] * v.$$

This operation toggles edges between neighboring sets $N(u) \setminus N(v)$, $N(v) \setminus N(u)$, and $N(u) \cap N(v)$ and switches neighbors of u and v (cf. Fig. 2).

B. Local equivalence

A particularly useful property of graph states is that local unitary operations acting on just a single qubit can be captured via graph-theoretic rewrites. Local complementation on a graph state corresponds to the following local Clifford operations (up to a global phase) [22]:

$$|G * u\rangle = \left[\sqrt{X_u} \prod_{v\in N(u)} \sqrt{Z_v}\right] |G\rangle.$$

Graph states are equivalent up to local Clifford operations (LC-equivalent) if and only if their graphs are equivalent up to local complementation [24] and equivalent under local unitary operations (LU-equivalent) if and only if their graphs are equivalent under a generalized r-local complementation [25].

Further, pivoting on a graph state corresponds to the following local operations [26]:

$$|G \wedge uv\rangle = H_u H_v Z_{N(u)\cap N(v)} |G\rangle.$$

It has been shown that any real-valued local Clifford equivalence is captured with pivoting [26]. These rewrites are useful in the context of resource minimization in linear optic settings, since local operations are usually easier to realize than entangling operations. To build a graph state on hardware, it is often cheaper to build an LU-equivalent graph state with fewer edges first and then transform it to the desired graph state using appropriate local operations [17].

III. FUSION NETWORKS

In this work, we consider dual-rail encoding, where a qubit is represented as a single photon in two linear optical modes. Fusions are probabilistic measurement operations usually involving two qubits on four modes that serve to glue together different graph states. The pioneering work by Browne and Rudolph [15] introduces two types of fusions based on the detection of one photon for the type-I fusion gate and two photons for type-II fusion. These gates are inherently probabilistic, i.e., upon measurement we may also detect a different number of photons. In such cases we say the fusion has failed because the measurement did not create the desired entanglement. The original fusion gates have a success probability of 50%, yet more recent proposals have shown that both types can be boosted to higher success probabilities using ancilla states [27,28]. By repeatedly applying fusion measurements we can create large entangled graph states from small initial resource states like linear clusters [15]. Fusion networks [2,29] define a configuration of fusion operations on graph states which yield a desired target graph state if every fusion succeeds. We first give a general definition in graph-theoretic terms:

A fusion network is a tuple (H, F) consisting of a simple graph $H = (V, E)$ and a subset $F \subset E$ of edges in H, such that $(V, E \setminus F)$ represents a graph state and the end points of each edge in F are subject to a two-qubit fusion operation [30]. Given a fusion network (H, F) for a target graph state $|G\rangle$, we say that the fusion process is successful if we reach G from H by applying fusions on all edges in F. In all figures of this paper we depict fusion networks as graphs with dashed blue edges for each "fusion" edge in F and black edges for each edge $E \setminus F$ belonging to the actual graph state. We give precise definitions of fusion networks for type-I and type-II fusion in the next section.

A. type-I fusion

Upon success, type-I fusion measures one qubit and leaves the other qubit intact such that it inherits all bonds from the measured qubit (cf. Fig. 3), whereas upon failure, both qubits are measured in the computational Z basis [15]. We note that in our definition of fusion networks, a successful type-I fusion $f \in F$ corresponds to contracting the edge f, i.e., the two end points of the edge are merged into a single vertex [31].

This allows for the following definition of type-I fusion networks where the notation H/e denotes the graph obtained by contracting the edge e in graph H and $H - S$ the graph obtained by deleting from H all vertices appearing in the vertex set S and all edges incident on S:

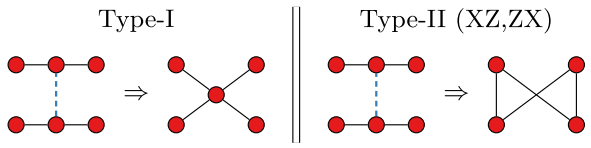

FIG. 3. Successful type-I (left) and type-II fusion (with observables $\{XZ, ZX\}$) on the middle qubits of two linear clusters with three qubits. The fusion is represented as a dashed blue edge with its end points indicating which qubits are inputs of the fusion gate. Upon success, we contract the blue edge in the type-I case, and we apply a Pivot+vertex deletion in the type-II case.

Definition 1. Given a fusion network (H, F). A type-I fusion on two vertices $\{u, v\} \in F$ yields a fusion network $(H', F \setminus \{u, v\})$ with $H' = H/\{u, v\}$ in the success case, that is, we obtain H' from H by contracting the edge $\{u, v\}$. Upon failure we obtain $H' = H - \{u, v\}$.

One can see that the behavior of the actual graph state described by $(V, E \setminus F)$ evolves as formalized in [15]. We can thus define type-I fusion networks such that the target graph is obtained from an initial fusion network by repeated edge contractions:

Definition 2. A type-I fusion network for a graph state $|G\rangle$ is a tuple (H, F) consisting of a simple graph H and a subset F of edges in H, such that we obtain G from H by contracting each edge in F.

In that sense G can also be seen as graph minor of H.

B. Type-II fusion

Type-II fusion consumes both qubits and entangles neighboring qubits. There are multiple ways to implement Type-II fusions, yet, most implementations can be described as Bell-state measurements [28]. The authors of [32] distinguish five Bell-state measurements based on different pairs of observables. In this work, we focus on type-II fusion being implemented as measurement in the $\{XZ, ZX\}$ basis, because the effect on the neighborhood of the fused vertices can be described in an accessible graph-theoretic way using the pivot operation. To see this, we look at the stabilizer generators of a graph state after an $\{XZ, ZX\}$ measurement as described in [32]:

Theorem 1 (Restatement of Eqs. 8–11 in [32]). Given a graph state with two nonadjacent qubits A and B. Measuring parities $\{X_A Z_B, Z_A X_B\}$ yields the following stabilizer generators:

$$\forall a_i \in N(A) \setminus N(B) : X_{a_i} Z_{N(a_i)\triangle N(B)}, \tag{5}$$

$$\forall b_i \in N(B) \setminus N(A) : X_{b_i} Z_{N(b_i)\triangle N(A)}, \tag{6}$$

$$\forall c_i \in N(A) \cap N(B) : X_{c_i} Z_{N(c_i)\triangle N(A)\triangle N(B)}, \tag{7}$$

$$\forall d_i \notin N(A) \cup N(B) : X_{d_i} Z_{N(d_i)}. \tag{8}$$

Recall that in the graph-state formalism, we get one stabilizer term for each qubit with an X operator acting on the qubit and Z operators acting on all neighboring qubits. From Sec. II A, we know that a pivot toggles edges between three sets $S_1 = N(A) \setminus N(B)$, $S_2 = N(B) \setminus N(A)$, and

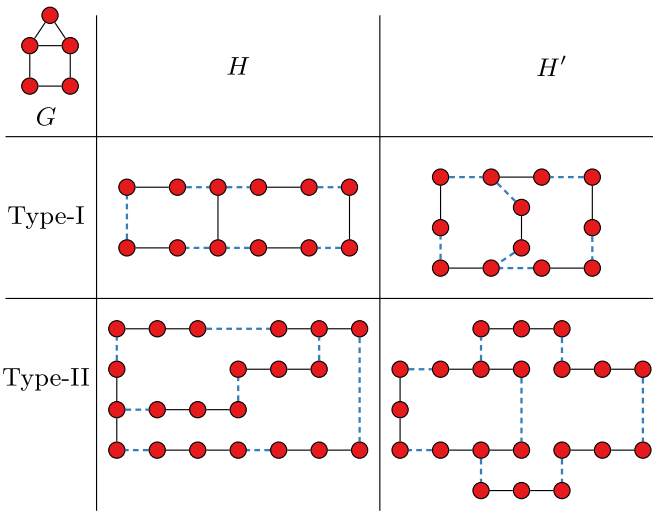

FIG. 4. Different (nonisomorphic) fusion networks H, H' for the same target graph G with either Bell pairs and type-I fusion, or three-qubit linear clusters and type-II ($\{XZ, ZX\}$) fusion.

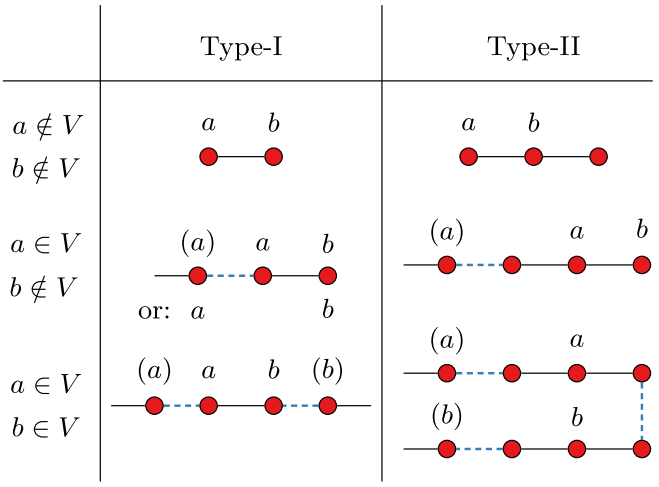

FIG. 5. Adding an edge $\{a, b\} \in G$ to a fusion network for both type-I and type-II networks. Characters in brackets denote the labeling before adding the edge, characters without brackets the updated labeling.

$S_3 = N(A) \cap N(B)$. Since $S_2 \cup S_3 = N(B)$, Eq. (5) describes that edges are toggled between S_1 and $S_2 \cup S_3$. In a similar way, Eqs. (6) and (7) describe that edges are toggled between S_2 and $S_1 \cup S_3$, and S_3 and $S_1 \cup S_2$, respectively. This shows that on fusion success the edges between neighbors change exactly as under a pivot operation. With qubits A and B being consumed in the fusion process, we describe the success case as a pivot followed by vertex deletions.

Compared to type-I fusion, the failure case is more involved since only qubit u gets measured in the Z basis. The other qubit v gets measured in the X basis, which has the effect of a pivot on v and a neighbor $w \in N(v) \neq u$ on the graph. This neighbor can be chosen somewhat arbitrarily based on different local transformations [22].

Definition 3. Given a fusion network (H, F). A $\{XZ, ZX\}$ fusion on two vertices $\{u, v\} \in F$ yields a fusion network $(H', F \setminus \{u, v\})$ with $H' = (H \wedge uv) - \{u, v\}$ on success. Upon failure, we obtain $H' = [(H - \{u\}) \wedge vw] - \{v\}$ for a neighbor $w \in N(v)$.

We can thus define such a network in terms of pivot minors as introduced in [26]:

Definition 4. An $\{XZ, ZX\}$ fusion network for a graph state $|G\rangle$ is a tuple (H, F) of a simple graph H and a subset F of nonadjacent edges in H, such that G is a pivot minor of H, i.e., we obtain G from H by applying a pivot on each edge in F and delete its adjacent vertices.

C. Generating fusion networks

In this section, we consider the problem of finding a suitable fusion network for a given graph state $|G\rangle$ using either two-qubit linear clusters and type-I fusion or three-qubit linear clusters and type-II fusion. Note that in general, fusion networks are not unique for a given target graph. As shown in Fig. 4, there can be many nonisomorphic networks leading to the same graph.

To construct a fusion network (H, F) for a target graph G, we traverse the edges of G and add them sequentially to the fusion network with an internal labeling to keep track of which vertex in H corresponds to which vertex in G. Given a simple graph $H = (V, E)$, we distinguish three cases for adding an edge $\{a, b\}$, see Fig. 5:

(1) $\mathbf{a}, \mathbf{b} \notin \mathbf{V}$: Both end points of the edge are not in the fusion network yet. We add a linear cluster to the fusion network and label two adjacent vertices as a and b. For type-II fusion there remains an unlabeled vertex which does not get consumed by a fusion, yet we may use this vertex when appending future edges.

(2) $\mathbf{a} \in \mathbf{V}; \mathbf{b} \notin \mathbf{V}$: Only one of the end points is present in the fusion network. We add a linear cluster and fuse one of its end points with the existing vertex. For type-II fusion we label the other two vertices of the linear cluster as our new a and b vertices, for type-I fusion we can choose which qubit gets measured upon fusion.

(3) $\mathbf{a}, \mathbf{b} \in \mathbf{V}$: Both end points are present in the fusion network. Here we need to fuse both vertices of the two-qubit cluster with the end points in the type-I case. For type-II fusion we add two new clusters, fuse them together to a four-qubit cluster and fuse their end points with the existing network.

We give an illustrative algorithm for constructing type-I fusion networks in Appendix B. The type-II algorithm is basically the same; however, it needs to manage unused vertices from the three-qubit linear clusters. For type-I fusion and two-qubit linear clusters the traversal strategies result in fusion networks with a minimal number of resource states. Since the unmeasured qubit in a successful type-I fusion just inherits all edges from the measured qubit, it never increases the number of edges in a graph. For generating a graph of n edges with type-I fusion we therefore need at least n two-qubit linear clusters and since our algorithm introduces exactly one resource state for each edge it is optimal in that sense.

Yet, this is not the case for type-II fusion. Our algorithms always fuse in such way, that at least one end point of a resource state is involved, i.e., a node with degree one. Like in the type-I case this results in fusions never changing the number of edges in the target graph. But a pivot may also increase

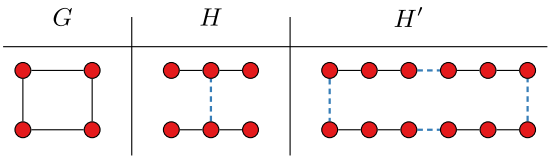

FIG. 6. Two $\{XZ, ZX\}$ fusion networks H, H' for obtaining the target graph G. H needs fewer resource states and fusions, yet our fusion network strategies would only find a fusion network similar to H'.

the number of edges when applied on two vertices with higher degrees (cf. Fig. 6) which shows that there are smaller type-II fusion networks with less resource states which are not found by our traversing strategy. Improving these fusion networks for fixed target graphs is possible (cf. [20]), but it is likely that a general algorithmic approach one has to give up small fusion networks and pay the cost of larger computational complexity.

IV. ADAPTIVE GRAPH-STATE GENERATION

We now give protocols to build graph states from fusion networks that recycle graph states after a fusion failure. As a general procedure, we start with a fusion network (H, F) describing our initial photonic system, sequentially apply fusions in F, and update the fusion network according to the graph-theoretic transformations in Sec. III. In case of fusion failure, we rebuild the network by combining the remaining parts of the graph state with new resource states using the following steps:

(1) We remove some vertices in H which got modified by fusion failure. As a result, H may become disconnected, decomposing into multiple connected components.

(2) We create a graph H' modeling the graph structure of the removed vertices.

(3) We add additional edges to H' where its vertices connect to remaining graph components of H.

(4) We build a fusion network for H' and use the additional edges to fuse it with H.

After the rebuild procedure, we obtain a new fusion network for the target graph state. We illustrate the rebuild protocol for type-I and type-II fusions in Fig. 7.

For the type-I fusion protocol, we again consider two-qubit linear clusters as resource states. Fusion failure on an edge $\{v_1, v_2\} \in F$ performs a Z measurement on both qubits, thereby removing them from the graph state. The subgraph H' thus consists initially only of vertices v_1 and v_2. We then iterate over all connected components C in H and add edges to H' for each edge connecting v_1 or v_2 with C before fusion failure. It can happen that C contains just a single qubit. In this case, we discard the component and just add a new edge to H' since we assume that inserting a new two-qubit linear cluster resource state is cheaper than rebuilding the network with the leftover component. We then use the algorithm from Sec. III C to build a fusion network for H' and integrate it into H. A pseudocode algorithm for the entire type-I protocol can be found in Appendix B.

For type-II fusion, we use three-qubit linear clusters as resource states. Here, fusion failure measures one vertex in Z and the other in X. As an X measurement also affects the entanglement between neighbors of the measured vertex, we cannot just rebuild all edges involving v_1 and v_2 to obtain a valid fusion network. To circumvent this problem, we discard the neighbors of the X measured vertex and thus create a subgraph H' containing vertices v_1, v_2 and all neighbors of the X measured vertex. As a convention, we choose the vertex with the least neighbors to receive an X measurement to make rebuilding less costly [33]. We iterate over connected components just as we did in the type-I case with the distinction that we add two edges to the subgraph for each original edge we want to rebuild. Further, we now discard a component if it contains fewer than three qubits since we have three-qubit linear resource states available.

V. MARKOV PROCESSES AND FIRST PASSAGE TIMES

In this section, we develop a framework for evaluating the efficiency of fusion networks and fusion orders by calculating the *average time* until the attempted fusions lead, after several steps, to the desired target graph state for the first time. The protocols described in the previous section define a *stochastic discrete-time Markov process* on a finite state space of size N. At any given time, the Markov process occupies one of N transient states, each representing a distinct fusion-network configuration.

At each step, a fusion operation succeeds with probability p and fails with probability $q = 1 - p$. The state space of the Markov process consists of all intermediate stages of the

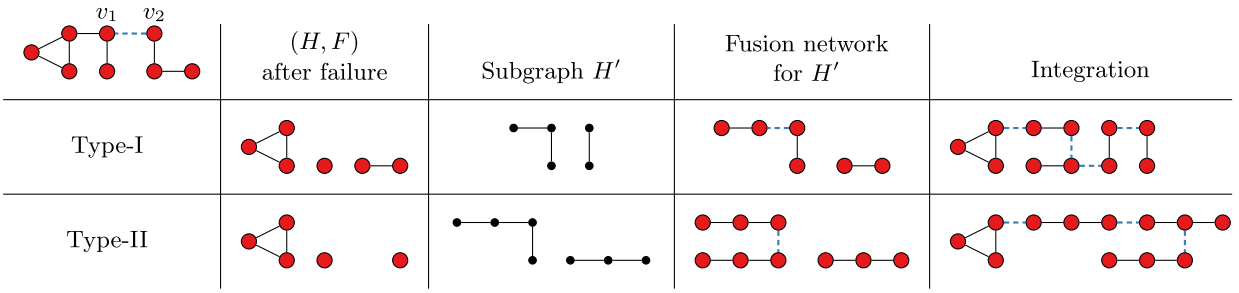

FIG. 7. Example of both type-I and type-II rebuild behaviors. The original fusion network is in the upper left corner. After fusion on the dashed blue edge $\{v_1, v_2\}$ fails, we first update the fusion network, then build a subgraph H' for rebuilding destroyed edges and integrate the resulting fusion network for H' into the existing one. For type-II failure, we consider v_1 to be Z-measured and v_2 to be X-measured.

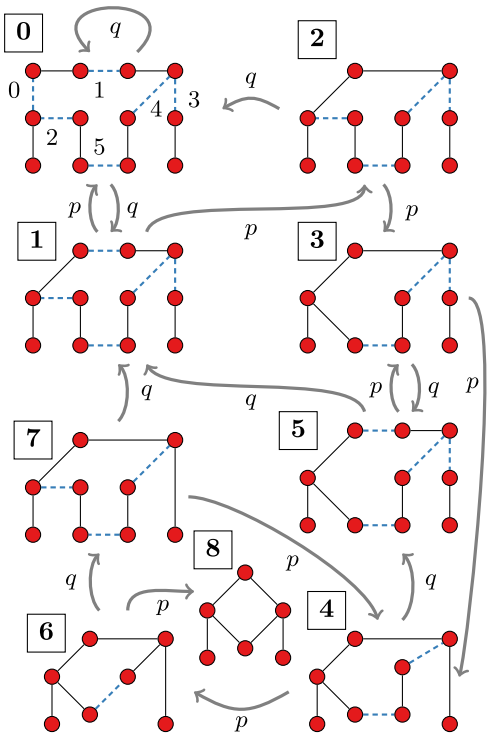

FIG. 8. Exemplary transition graph for creating a six-qubit graph state 8 from two-qubit linear clusters. The order in which the type-I fusions are carried out is indicated next to the edges in state 0.

graph state following the protocol and is fully captured by an $N\times N$ transition matrix P encoding all possible transitions between states with their respective probabilities. More precisely, P_{ij} is equal to the probability to transition to state i in the next step, if the system is currently in state j. We denote this probability also by $p_{i\leftarrow j}$. Note that P is in general not a symmetric matrix since transitions are not necessarily reversible in one step. However, since a transition to *any* state, possibly the same state, must occur with probability 1, P is a stochastic matrix with columns that all add up to 1, i.e., we have $\sum_i P_{ij} = 1$.

Let υ now denote a vector containing the occupation probabilities in state space, that is, the system is found in state i with probability υ_i. After one step of the Markov process the new occupation probabilities are given by $P\upsilon$. The late-time occupation probabilities are found by applying P repeatedly to the initial state υ_0. Since all entries of P are non-negative, there exists an eigenvector π with all positive entries and largest eigenvalue equal to 1 [34]. Under generic conditions (irreducibility and aperiodicity) only this eigenmode survives in the late-time limit. Thus π is the stationary distribution:

$$\lim_{n\to\infty} P^n \upsilon_0 = \pi. \tag{9}$$

To illustrate the construction of a transition matrix, we consider the creation of the five-qubit graph state shown in Fig. 8. We start from the initial resource state labeled configuration 0. The Markov process ends when we eventually reach the finished graph state shown as state 7. The transition matrix P capturing all possible transitions between the states in Fig. 8 is given by

$$P = \begin{pmatrix}
q & p & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
q & 0 & p & 0 & 0 & 0 & 0 & 0 & 0 \\
q & 0 & 0 & p & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & p & q & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & q & p & 0 & 0 \\
0 & q & 0 & p & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & q & p \\
0 & q & 0 & 0 & p & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}.$$

With the underlying Markov process clearly defined, we can now focus on the main quantity that is of interest to us, namely the so-called *mean first passage time* (MFPT). This is the average number of steps that it takes to reach the target state for the first time, when starting from an initial state. This problem has been well studied in the literature, see [34,35] for pedagogical introductions. Here, we confine ourselves to a self-contained introduction to computing the MFPT.

Let the matrix $M_{ij} = M_{i\leftarrow j}$ denote the MFPT to start in state j and arrive in state i for the first time. We want to set up a recursive relation for $M_{i\leftarrow j}$ that will allow us to compute its value. To this end, we argue as follows: Reaching i from j may be possible directly in one step, or alternatively through an intermediate state $k \neq i$. The probability for the former process is $p_{i\leftarrow j}$. The latter process results in a total number of steps $1 + M_{i\leftarrow k}$ and occurs with probability $p_{k\leftarrow j}$, thus we have the recursive relation

$$M_{i\leftarrow j} = p_{i\leftarrow j} + \sum_{k\neq i} p_{k\leftarrow j}(1 + M_{i\leftarrow k}). \tag{10}$$

This allows us to derive the following relation between the transition matrix P and the MFPT matrix M:

$$M = E + M\cdot P - D\cdot P, \tag{11}$$

where E is a matrix with all entries equal to 1, and D is a diagonal matrix with entries

$$D_{ij} = \frac{\delta_{ij}}{\pi_j}. \tag{12}$$

We provide a pedagogical derivation of this result in Appendix A. A straightforward inversion of this relation is not possible, since $\mathbf{1} - P$ is a singular matrix. To see this, note that the columns of P all sum to 1. Thus, the all-one vector $(1, 1, \ldots, 1)$ is a left eigenvector of $\mathbf{1} - P$ with eigenvalue zero, which means $\mathbf{1} - P$ is not invertible. In Appendix A, we show that M is instead given by the following expression:

$$M = D(\mathbf{1} - Z + Z_0 E), \tag{13}$$

where Z is called the fundamental matrix and is defined as

$$Z = (\mathbf{1} - P + \Pi)^{-1}. \tag{14}$$

The matrix Z_0 is the diagonal matrix of Z, i.e., $(Z_0)_{ii} = Z_{ii}$ and $(Z_0)_{ij} = 0$ for $i \neq j$, and the matrix Π consists of N repeated columns of the π vector. The derivation of these results is found in Appendix A.

To summarize, from the knowledge of the transition matrix P, we can calculate the MFPT matrix M explicitly. This will be heavily utilized below to assess the quality of different fusion schemes.

We emphasize that while our pipeline presented in the following sections consists of algorithms with polynomial runtime, the MFPT calculation itself has an exponential runtime. Thus we do not include the latter in the pipeline and instead only use the MFPT to benchmark small instances of fusion networks.

Nevertheless, we point out that the Markov chain is always finite. Let m denote the number of qubits in the initial fusion network. Since our procedure never creates additional qubits, every intermediate fusion network contains at most m qubits. In the case of fusion success, we are left with $m-1$ qubits (type I) or $m-2$ qubits (type 2). In the case of fusion failure, the fusion network has at most m qubits. If a rebuild would require more than m qubits, we start anew from a fusion network with m qubits. Thus the total number of qubits never exceeds m. Recall that we denote by N the number of states in the Markov process. Since every such state contains at most m qubits, the number of possible configurations is finite. Let us denote this number by T_m. Then T_m is upper bounded by the total number of labeled simple graphs on m vertices. This number is easily calculated since on m vertices we have at most $m(m-1)/2$ edges. Since each edge is present or absent, we arrive at the upper bound $T_m \leqslant 2^{m(m-1)/2}$. This is the worst case size of the state space of the Markov process. In other words, the size of the Markov process is bounded by

$$N \leqslant T_m \leqslant 2^{m(m-1)/2}. \tag{15}$$

The building of the Markov chain and construction of the sparse transition matrix requires time $O(N)$. The inversion in Eq. (14) by Gaussian elimination requires $O(N^3)$ operations. The latter dominates the total runtime.

VI. OPTIMIZING FUSION NETWORKS

With mean first passage times as a tool to evaluate fusion network protocols, we now describe algorithms to optimize the graph-state generation process at different stages.

A. Finding locally equivalent graphs with fewer edges

A first step towards minimizing mean first passage times is to find LU-equivalent graph states requiring fewer entangling operations. Since our network generation algorithms require at least one fusion per edge, reducing the number of edges in the target graph directly reduces the number of fusions. We then build this graph state and transform it to the desired graph state with local unitary operations which are comparably much cheaper than fusions.

Given a graph state, finding an LU-equivalent one with minimum edges seems to be a difficult problem even when restricting the unitaries to Clifford gates. If a graph is LC-equivalent to a tree, we can find a sequence of local complementations to transform the graph into the tree in polynomial time [36]. Since local complementation preserves connectivity and we assume our initial graph to be connected, there can be no LC-equivalent graph with fewer edges than a tree. Yet, we are usually interested in graphs not equivalent to trees since the latter can be generated deterministically by a single quantum emitter [13]. For the general case, there are strong indications that the edge minimization problem is NP-complete [37].

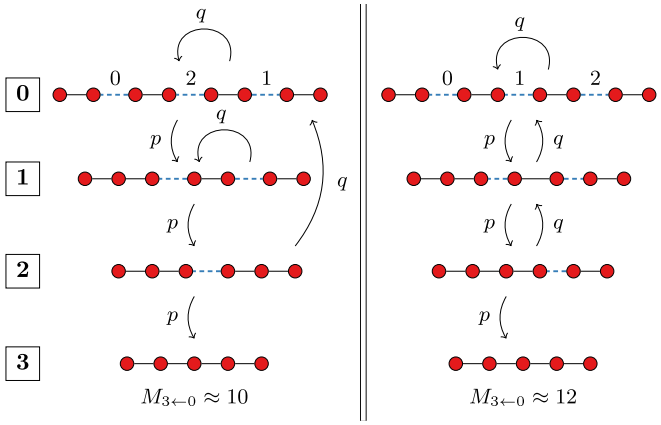

FIG. 9. Transition graphs for two different fusion orders generating a five-qubit linear graph and mean first passage time, respectively, for fusion success probability $p = 0.5$. In the left example, the first and second fusion are independent of each other, that is, they act on separate parts of the graph. Thus, the second fusion does not undo the first fusion, which yields a better mean first passage time than for the right example.

Therefore, given an arbitrary graph we first check whether the graph is equivalent to a tree using the algorithm in [36] and if not, we use a greedy approach to minimize edges with local complementation as introduced in [38].

B. Fusion ordering

For a given fusion network (H, F) we can also optimize the order in which we apply the fusions in F. Depending on the ordering, the mean first passage time until reaching the desired target graph state varies already for small instances (cf. Fig. 9). Formally, the problem can be defined as follows: *Given a fusion network (H, F) with target graph G, find an ordering in F, such that $M_{G\leftarrow H}$ is minimized.* Finding the fusion order with minimal MFPT is challenging since the search space of possible orderings grows exponentially with the number of fusions and it is not immediately clear how the stochastic process evolves from the transition matrix.

Here we introduce a heuristic algorithm inspired from the *min-weight-maximum-matching-first* algorithm introduced in [17]. We motivate this algorithm with the following observation: Each intermediate state in the transition matrix has only two follow up states, namely, one on failure with probability q and the other on success with probability p. The calculation of $M_{G\leftarrow H}$ can therefore be simplified to the following recursion, where H is labeled as starting state 0 and G as target state n:

$$M_{n\leftarrow 0} = 1 + qM_{n\leftarrow 0} + pM_{n\leftarrow 1}, \tag{16}$$

$$M_{n\leftarrow i} = 1 + qM_{n\leftarrow j} + pM_{n\leftarrow k}. \tag{17}$$

For any intermediate state i, j denotes the state being reached if the fusion at state i fails and k the state if the fusion

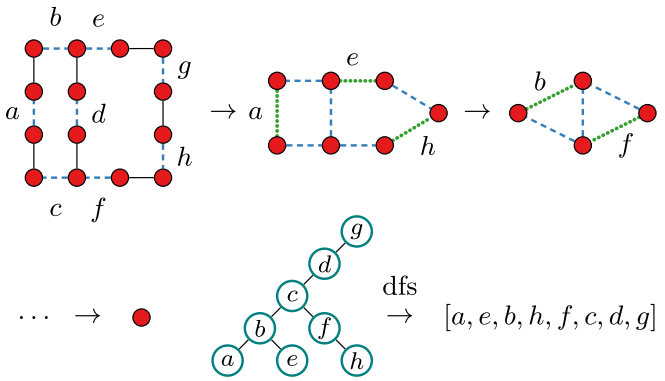

FIG. 10. Example of finding a fusion order with the algorithm described in Sec. VI B. The initial fusion network consists of eight fusions labeled from a to h. We repeatedly contract the network and find the maximal set of disjoint edges highlighted as dotted green edges until the network is contracted into a single node. At each iteration we add the disjoint fusions as vertices to a tree whose inverted dfs order determines the final fusion order.

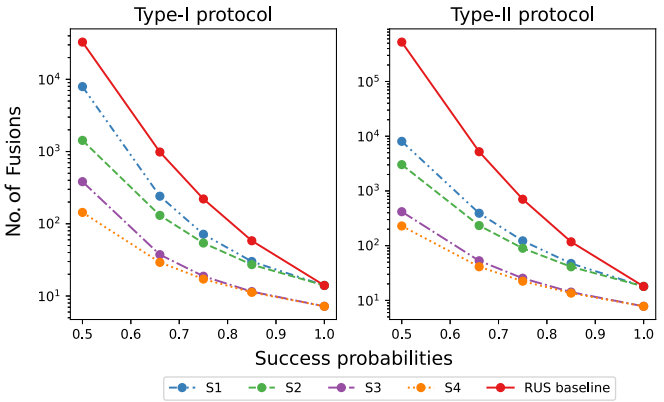

FIG. 11. Averaged mean first passage time for creating random graph states of 6 vertices and 10 edges with the strategies presented in Sec. VII A. Additionally, we compare against a repeat-until-success baseline, that is, the mean first passage time, if we would restart the entire graph-state building procedure anew after any fusion failure.

succeeds. All $pM_{n\leftarrow k}$ terms are already fixed by fusion network and protocol, thus the only term we can optimize with different orderings is $qM_{n\leftarrow j}$. Assuming that $M_{n\leftarrow j} \geqslant M_{n\leftarrow i}$, i.e., failed fusions always lead to states with higher mean first passage times than the current one, the best case for $qM_{n\leftarrow j}$ would be if $j = i$, that is, if on fusion failure we end up in the same state as before.

This is the case if we apply fusions on disjoint resource states, because a failed fusion does not undo previous fusions and only destroys the involved resource states instead of qubits in a larger graph state. Since we assume that new resource states can be injected at any time into the creation process, we end up in the same state as before the fusion. The core idea of our algorithm thus is to choose an ordering where we apply fusions on disjoint connected sets of the graph as long as possible. For this we proceed as follows:

(1) We start with the initial fusion network and an empty tree T.

(2) We contract the fusion network (H, F) on all nonfusion edges, that is, all edges $E' \in E \setminus F$. This yields a graph $H' = (V, F)$ where each edge represents a fusion and each node a connected set of entangled qubits.

(3) If H' consists of only a single node, we go to step 4, otherwise we apply the blossom algorithm [39] on H' to find a maximal set of disjoint edges F'. In our context, we get a maximal set of fusions acting on disjoint connected sets of the graph. For each fusion we add a vertex to T: in the first iteration as leaves, in the subsequent iterations as internal vertices adjacent to all leaves of the corresponding connected sets. We then go back to step 2 with fusion network $(H', F \setminus F')$.

(4) We return the fusion order as inverse dfs order on T starting from its root vertex.

An illustrative example of this algorithm is shown in Fig. 10. The standard implementation of the blossom algorithm runs in time $O(|E||V|^2)$, thus the entire fusion-order finding algorithm is upper bounded by $O(|F|^2|V|^2)$ for the contracted fusion network $H' = (V, F)$.

VII. EVALUATION

To analyze the improvement on mean first passage times with the developed algorithms, we implemented a pipeline with the following steps:

(1) We optionally optimize the number of edges in the graph using either the tree decomposition method from Bouchet [36], or the greedy heuristic in [38].

(2) We build a fusion network for the graph using the algorithms in Sec. III C and optionally optimize the fusion ordering.

(3) We generate all possible transitions by running the corresponding protocol on the initial fusion network and iterate through all succeeding states in a depth-first way. Optionally after each fusion failure we again optimize the fusion order.

(4) We build the transition matrix for a fixed fusion success probability and calculate the MFPT.

We run this pipeline on both protocols for type-I fusion with two-qubit linear clusters and type-II fusion with three-qubit linear clusters as resource states. For each configuration, we average over ten graphs $G(m, n)$ that are randomly generated connected graphs with a fixed number of m vertices and n edges. The code used for the evaluations is available at Zenodo [40] allowing reproducibility and possible integration into other research projects.

A. Comparison against repeat until success scheme

Figure 11 shows the averaged mean first passage times for random $G(6, 10)$ graphs for varying fusion success probabilities 50%, 66%, 75%, and 85%.

We compare four different optimization strategies:

S1: The standard type-I or type-II fusion protocol without prior edge minimization via local complementation and no optimization of the fusion order.

S2: No edge minimization but with optimization of the fusion order.

S3: No optimization of the fusion order but with edge minimization.

S4: Both edge minimization and optimization of the fusion order.

We further compare against a repeat until success baseline where we restart the entire graph state buildup if any fusion fails.

Our strategies clearly outperform the baseline for all fusion success probabilities, which indicates that just using a protocol to recycle graph states on fusion failure improves the number of required fusions significantly. Both optimization of fusion orders and edge minimization via local complementation are then able to reduce the amount of required fusions by several orders of magnitude, especially for lower success probabilities. In general, mean first passage times for the type-II protocol are higher when compared to type-I. One possible reason could be that in the type-II protocol we discard more vertices from the graph state in the failure case.

B. Comparison to the Lee and Jeong approach

In [17] authors propose a similar framework to ours for graph state generation from type-II fusion measurements and three-qubit linear clusters where they also first optimize edges in a graph via graph-theoretic rewrites, then extract fusion networks and last optimize the fusion order with the goal of minimizing the average number of fusions. Their protocol can be considered as an improved repeat-until-success scheme where fusions are arranged in a tree structure and the graph state is built in a bottom-up manner similar to Fig. 10. Each time a fusion fails, the graph state corresponding to the subtree below the failed tree node needs to be rebuilt entirely. In particular if the very last fusion fails, the buildup is reset to the beginning. With failed fusions on disjoint subtrees not affecting each other their protocol poses a clear improvement over the naive repeat until success protocol from the previous section, yet it is still not adaptive in the sense of recycling leftover graph states.

In Fig. 12, we compare strategy S4, i.e., our protocol with optimization of fusion order and prior edge minimization against the approach from [17] using the corresponding software package opt-graph-state [41]. To better highlight the improvements achieved through adaptivity, we also include a strategy where edge minimization and fusion network extraction are done with the opt-graph-state package instead of our framework.

Results show that both strategies outperform the approach from [17]: The combined strategy reduces the average number of fusions by up to 26% when compared to [17], which clearly shows that adaptive protocols can reduce the expected number of fusions significantly. With strategy S4 we obtain even higher reductions by up to 43%, indicating that edge minimization based on finding local equivalent graphs can reduce more edges than the edge minimization scheme in [17].

C. Effect of fusion network optimizations

We now analyze more in depth how different fusion network optimizations improve the mean first passage time. Figure 13 shows the relative improvement on the number of required fusions for graphs with and without prior edge

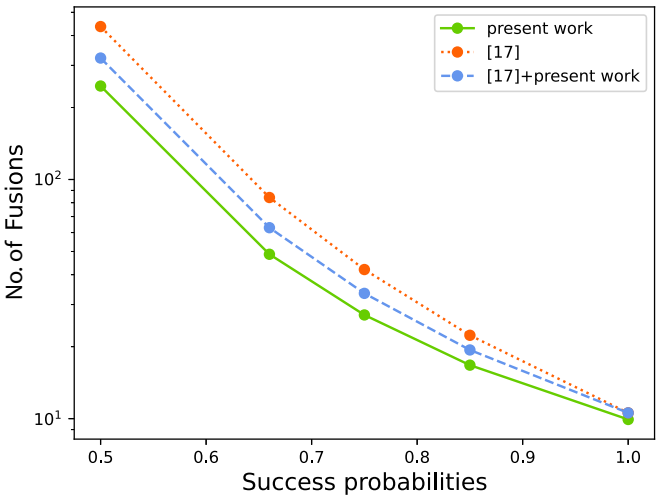

FIG. 12. Average expected number of fusions for 100 connected random graphs with $V = |7|$ and $E = |12|$. *Present work* corresponds to strategy S4, i.e., our compilation pipeline including both edge minimization and fusion order optimization, *[17]* denotes the approach in [17], and in *[17]+present work* edge minimization and fusion network extraction are done with the approach in [17] and only the graph-state buildup process is replaced by our adaptive protocol.

minimization using local complementation on random graphs with six nodes and increasing number of edges.

While the average improvement is comparably small between 20% and 40% for graphs with seven edges, it increases continuously towards 70−100% in the case of 11 edges. The smaller improvement for graphs with seven to eight edges can be explained by the fact that in a connected graph with $m = 6$ vertices there are not many edges to optimize since the number of edges is lower bounded by $m - 1$, i.e., the graph is a tree. In accordance with Fig. 11, the highest optimization potential shows for protocols with lower success probabilities.

In a similar way, Fig. 13 shows the relative improvement on the number of required fusions when we optimize the fusion order during the protocol. Here the improvement is a bit more modest with an average improvement up to 20% for higher success probabilities and up to 70% for lower success probabilities. Still, results indicate that the more edges a graph has, the more improvement on mean first passage times is possible using the fusion order optimization.

VIII. CONCLUSION AND OUTLOOK

In this work, we proposed a framework to generate graph states with adaptive fusion protocols for both type-I and type-II fusion schemes on linear cluster resource states. In contrast to other work, we make use of adaptive fusion protocols allowing us to reprocess parts of leftover graph states after fusion failure has occurred. We utilized mean first passage times as a tool to estimate the number of required fusions in a sequential buildup process, and we investigated several algorithms to optimize fusion networks towards fewer fusion steps. Our results show substantial improvements in reducing the number of required fusions which can be up to several orders of magnitude when compared to simple repeat-until-success schemes.

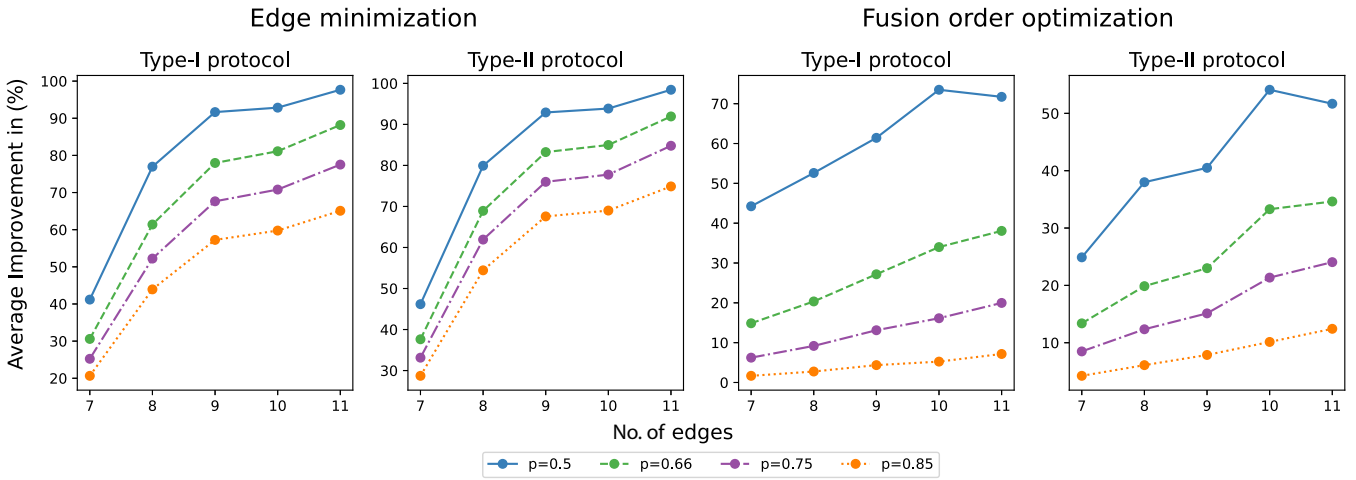

FIG. 13. Averaged relative improvement of mean first passage times using the strategies described in Sec. VI for graphs of six nodes and varying number of edges. The two plots on the left show the improvement when minimizing edges in advance, and the two plots on the right show the improvement when optimizing the fusion order.

Our analysis can be extended in the direction of finding more efficient type-II fusion networks by exploiting the pivot behavior of the $\{XZ, ZX\}$ basis measurement on graph states. Another line of exploration is to interleave different fusion types in the generation process.

Regarding speedups of the graph-state building process, we point out that an extension of our framework to incorporate parallel fusions is straightforward. It can be taken into account by redefining the states such that they correspond to concurrent manipulations of different parts of the graph.

While in this work we focused only on all-photonic schemes, the Markov process formalism that we developed may also prove useful when analyzing other nondeterministic graph-state generation protocols. Potential playgrounds are nonphotonic cluster states as they have been created for error-correction with robustness against single-qubit errors [4], graph states produced in trapped-ion systems [5] and superconducting platforms with 16 qubits [9] and more recently 20-qubit systems on the IBM Q Poughkeepsie device [10]. The past few years have also seen a large number of developments regarding Rydberg atom systems, where graph states have also played a crucial role [11]. All of these systems are currently subject to high levels of noise such that graph states have to be constructed in a stochastic fashion. Thus, formulating their construction as Markovian processes may turn out to be a fruitful perspective.

ACKNOWLEDGMENTS

The authors thank Francesco Debortoli for the early implementation of the translation to photonic circuits, Patrik Isene Sund for the valuable discussion on the design of optical fusions and quantum emitters, as well as Francesco Giorgino and Lorenzo Carosini for a discussion on the experimental capabilities of such a scheme. This research was funded in whole, or in part, from the Horizon Europe research and innovation programme under Grant Agreement No. 101135288 (EPIQUE).

DATA AVAILABILITY

The data that support the findings of this article are openly available [40].

APPENDIX A: DERIVATION OF THE FIRST PASSAGE TIME

In the main text we introduced the MFPT matrix $M_{ij} = M_{i\leftarrow j}$. This is the average number of steps to start at state j and arrive at state i for the first time. As we explained, the MFPT satisfies a simple recursion relation:

$$M_{i\leftarrow j} = p_{i\leftarrow j} + \sum_{k\neq i} p_{k\leftarrow j}(M_{i\leftarrow k} + 1). \tag{A1}$$

Thus, all we have to do in order to find our answer is to solve this equation for M. We do this in several steps. First, we can simplify Eq. (A1):

$$M_{i\leftarrow j} = p_{i\leftarrow j} + \sum_{k\neq i} p_{k\leftarrow j} + \sum_{k\neq i} p_{k\leftarrow j} M_{i\leftarrow k}$$

and thus

$$M_{i\leftarrow j} = 1 + \sum_{k\neq i} p_{k\leftarrow j} M_{i\leftarrow k}. \tag{A2}$$

Secondly, we derive a useful fact. Multiply Eq. (A2) by the stationary vector component π_j and sum over j:

$$\sum_j M_{i\leftarrow j}\pi_j = \sum_j \pi_j + \sum_j \sum_{k\neq i} p_{k\leftarrow j}\pi_j M_{i\leftarrow k}.$$

The definition of the stationary vector is that $\sum_j p_{k\leftarrow j}\pi_j = \pi_k$. Thus we have

$$\sum_j M_{i\leftarrow j}\pi_j = 1 + \sum_{k\neq i} M_{i\leftarrow k}\pi_k,$$

where we also used $\sum_j \pi_j = 1$, since π is also a probability vector. Finally, moving the second term on the right to the left,

we find that almost all terms cancel, except for one:

$$M_{i\leftarrow i}\pi_i = 1.$$

Thus we learn the diagonal entries of the MFPT matrix M:

$$M_{i\leftarrow i} = \frac{1}{\pi_i}.$$

The next task is to identify the remaining entries of M. Starting from Eq. (A2), we rewrite this as

$$M_{ij} = 1 + (M\cdot P)_{ij} - p_{i\leftarrow j}M_{i\leftarrow i} \tag{A3}$$

$$= 1 + (M\cdot P)_{ij} - \frac{1}{\pi_i}p_{i\leftarrow j}$$

$$\rightarrow M = E + M\cdot P - D\cdot P, \tag{A4}$$

where we defined the matrix with all entries equal to 1:

$$E_{ij} \equiv 1$$

and the diagonal matrix

$$D_{ij} \equiv \frac{\delta_{ij}}{\pi_i}.$$

At this point, one may believe that the solution of Eq. (A4) is straightforward, since one just has to solve for M by bringing the second term on the right to the left and multiply by $(1-P)^{-1}$. But this is not correct, since $1-P$ is singular and thus not invertible. It is singular because P has an eigenvalue 1. Thus $1-P$ has an eigenvalue 0 and therefore $\det(1-P)=0$. However, it turns out that $E - D\cdot P$ is also singular. Nevertheless, it is possible to extract the matrix M.

Equation (A4) can be solved differently. We begin by showing that the solution is unique. Assume Eq. (A4) had another solution M'. Then

$$M = E + M\cdot P - D\cdot P,$$

$$M' = E + M'\cdot P - D\cdot P.$$

Subtracting the equations, we obtain

$$M - M' = (M - M')\cdot P.$$

Interpreting this equation column by column, this equation states that each column of $M-M'$ is equal to the left eigenvector of P corresponding to eigenvalue 1. We already know that this eigenvector takes the form $c\cdot(1,1,1,\ldots,1)$ for some real c. Moreover, we already proved that all M matrices have diagonal elements $M_{ii}=1/\pi_i$. Thus $M-M'$ has zeros on the diagonal. But this implies $c=0$. Therefore,

$$M - M' = 0,$$

in other words the solution of Eq. (A4) is unique.

Then it remains to find this unique solution. We first state the solution and then prove that it solves Eq. (A4). The solution is

$$\boxed{M = D(\mathbf{1} - Z + Z_0 E)}. \tag{A5}$$

Here, Z is the fundamental matrix defined as

$$Z = (\mathbf{1} - P + \Pi)^{-1},$$

with Π being the *limiting matrix* of P. This is a matrix with the stationary π vector repeated on $\dim M$ columns, i.e., $\Pi = (\pi, \pi, \ldots, \pi)$. From this we can deduce the relation

$$Z - Z\cdot P + \Pi = \mathbf{1}, \tag{A6}$$

which will be needed below. The matrix Z_0 is the diagonal part of Z, i.e., $[Z_0]_{ii} = Z_{ii}$ and $[Z_0]_{ij}=0$ for all $i\neq j$. We can quickly check that M satisfies Eq. (A4). First off, we can check that the diagonal parts of M and D match, as we found before:

$$M_{ii} = \sum_j D_{ij}(\mathbf{1} - Z + Z_0 E)_{ji} = D_{ii}(\mathbf{1} - Z + Z_0 E)_{ii}$$

and now notice that $-Z + Z_0 E$ is zero on the diagonal. Thus the relation

$$M_{ii} = D_{ii}$$

holds. Next, we multiply M in Eq. (A5) from the right by P:

$$M\cdot P = D(P - Z\cdot P + Z_0\cdot E).$$

Here we used that $E\cdot P = E$ because the columns of P sum to 1. Now we use Eq. (A6) and find

$$M\cdot P = D\cdot(P - \mathbf{1} - Z - \Pi + Z_0\cdot E)$$

$$= M + D\cdot P - D\cdot\Pi$$

$$= M + D\cdot P - E,$$

where in the penultimate step we used Eq. (A5) again. Thus we find that this M satisfies Eq. (A4) and has diagonal entries $M_{ii}=1/\pi_i$ and we have therefore found the unique solution.

APPENDIX B: ALGORITHMS FOR FUSION NETWORKS

In this section we give pseudocode algorithms for graph state generation using Type-I fusion and two-qubit linear clusters following Secs. IV and III C: Algorithm 1 describes the protocol for building graph states from fusion networks, whereas Algorithm 2 describes a procedure to generate fusion networks for arbitrary target graph states.

1. Graph-state generation protocol

The `BuildGraphState` procedure sequentially iterates through all fusions of a given fusion network (H,F) and at each step fuses the indicated qubits v_1 and v_2. Depending on the fusion outcome, the network is updated accordingly: In the success case, we process according to Definition 1, whereas in the failure case we call a separate procedure `UpdateNetworkOnFailure`. In here, we first create a graph G containing v_1 and v_2 together with another graph T representing the leftover graph state. We then reconstruct each edge between a vertex in T and v_1 or v_2 since these edges got destroyed by fusion failure. To this end, we iterate through all the connected components C in T and collect, inside a new set D, all the destroyed edges that connect vertices in C to either v_1 or v_2. In the special case where C consists only of a single vertex v, we remove it from the existing graph state T and add the first edge in D to G where we label one end point with v. We then iterate over all remaining destroyed edges and add them to G such that in the end G contains all edges destroyed by fusion failure. Also during iteration we add fusions to connect the end points of the destroyed edges to the leftover graph state. We then build a fusion network for G

ALGORITHM 1. Graph-state generation from type-I fusion network.

procedure BUILDGRAPHSTATE(H, F)
  while $F \neq \emptyset$ **do**
    $v_1, v_2 \leftarrow F$.pop()
    success $\leftarrow$ FUSEv_1, v_2 ▷ *Type-I fusion on photonic hardware*
    if success **then** ▷ *Fusion success*
      for all $n \in N(v_2)$ **do** ADDEDGE($H, (v_1, n)$)
      DELETEVERTEX(H, v_2)
    else ▷ *Fusion failure*
      $H, F \leftarrow$ UPDATEFUSIONNETWORKONFAILURE(H, F, v_1, v_2)
procedure UPDATENETWORKONFAILURE(H, F, v_1, v_2)
  $G \leftarrow (\{v_1, v_2\}, \emptyset)$ ▷ *Target graph for fusion sub-network*
  $A \leftarrow H.E \setminus F$ ▷ *Nonfusion edges*
  $T \leftarrow (H.V \setminus \{v_1, v_2\}, \{(v, w) | (v, w) \in A \text{ and } (v, w \notin \{v_1, v_2\})\})$
  for C $\in$ CONNECTEDCOMPONENTST **do**
    $D \leftarrow \{(v, w) \in A | (v \in C) \text{ and } (w \in \{v_1, v_2\})\}$ ▷ *Collect destroyed edges*
    if $|C| = 1$ **then** ▷ *Replace component if it is too small*
      $v, w \leftarrow D$.pop()
      $G \leftarrow (G.V \cup \{v\}, G.E \cup \{(v, w)\})$
      DELETEVERTEX(T, v)
    while $D \neq \emptyset$ **do** ▷ *Rebuild all destroyed edges*
      $v, w \leftarrow D$.pop()
      $x \leftarrow$ADDNEWVERTEX(G)
      $G \leftarrow (G.V \cup \{x\}, G.E \cup \{(w, x)\})$
      $F = F \cup \{(v, x)\}$
  $S, F' \leftarrow$ BUILDFUSIONNETWORK(G) ▷ *Generate new fusion network*
  $F \leftarrow F \cup (v_1, v_2) \cup F'$ ▷ *Update fusions*
  $H \leftarrow (T.V \cup S.V, T.E \cup S.E \cup F)$ ▷ *Update fusion graph*
  return H, F

and merge it with the existing network where we additionally add the fusion $\{v_1, v_2\}$ which failed originally.

2. Fusion network generation

The `BuildFusionNetwork` procedure first initializes an empty fusion network (H, F) and a position map to store which vertex in the original graph G corresponds to which vertex in the fusion network. For each edge $\{v, w\}$ in G we add a new two-qubit linear cluster to H. Depending on the three cases outlined in Fig. 5, we add fusions to F connecting the new cluster with the existing fusion network. Further, the position map is updated where the new cluster now represents the edge $\{v, w\}$. At the end, we add the edges in F to H and return the fusion network.

ALGORITHM 2. Type-I random traversal fusion network generation.

procedure BUILDFUSIONNETWORK(G)
  $H \leftarrow (V = \emptyset, E = \emptyset)$ ▷ *Init fusion network*
  $F \leftarrow \emptyset$ ▷ *Additional fusions*
  $P \leftarrow \emptyset$ ▷ *Position map*
  $E \leftarrow$ randomShuffle($G.E$) ▷ *Traverse target graph in random order*
  while $E \neq \emptyset$ **do**
    $v, w \leftarrow E$.pop()
    $x_1, x_2 \leftarrow$ADDVERTICES(H) ▷ *Add resource state to network*
    $H.E \leftarrow H.E \cup \{x_1, x_2\}$
    if $v \in P$ and $w \in P$ **then** ▷ *Fuse resource state with existing network*
      $F \leftarrow F \cup \{(x_1, P[v]), (x_2, P[w])\}$
    else if $v \in P$ and $w \notin P$ **then**
      $F \leftarrow F \cup \{(x_1, P[v])\}$
    else if $v \notin P$ and $w \in P$ **then**
      $F \leftarrow F \cup \{(x_2, P[w])\}$
    $P[v] \leftarrow x_1$ ▷ *Update position map*
    $P[w] \leftarrow x_2$
  $H.E \leftarrow H.E \cup F$
  return H, F

[1] R. Raussendorf and H. J. Briegel, A one-way quantum computer, Phys. Rev. Lett. **86**, 5188 (2001).

[2] S. Bartolucci, P. Birchall, H. Bombin, H. Cable, C. Dawson, M. Gimeno-Segovia, E. Johnston, K. Kieling, N. Nickerson, M. Pant, *et al.*, Fusion-based quantum computation, Nat. Commun. **14**, 912 (2023).

[3] K. Azuma, S. E. Economou, D. Elkouss, P. Hilaire, L. Jiang, H.-K. Lo, and I. Tzitrin, Quantum repeaters: From quantum networks to the quantum internet, Rev. Mod. Phys. **95**, 045006 (2023).

[4] X.-C. Yao, T.-X. Wang, H.-Z. Chen, W.-B. Gao, A. G. Fowler, R. Raussendorf, Z.-B. Chen, N.-L. Liu, C.-Y. Lu, Y.-J. Deng, *et al.*, Experimental demonstration of topological error correction, Nature (London) **482**, 489 (2012).

[5] B. P. Lanyon, P. Jurcevic, M. Zwerger, C. Hempel, E. A. Martinez, W. Dür, H. J. Briegel, R. Blatt, and C. F Roos, Measurement-based quantum computation with trapped ions, Phys. Rev. Lett. **111**, 210501 (2013).

[6] 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, Nat. Commun. **5**, 3658 (2014).

[7] P. Walther, K. J. Resch, T. Rudolph, E. Schenck, H. Weinfurter, V. Vedral, M. Aspelmeyer, and A. Zeilinger, Experimental one-way quantum computing, Nature (London) **434**, 169 (2005).

[8] N. Kiesel, C. Schmid, U. Weber, G. Tóth, O. Gühne, R. Ursin, and H. Weinfurter, Experimental analysis of a four-qubit photon cluster state, Phys. Rev. Lett. **95**, 210502 (2005).

[9] Y. Wang, Y. Li, Z.-Q. Yin, and B. Zeng, 16-qubit IBM universal quantum computer can be fully entangled, npj Quantum Inf. **4**, 46 (2018).

[10] G. J. Mooney, C. D. Hill, and L. C. L. Hollenberg, Entanglement in a 20-qubit superconducting quantum computer, Sci. Rep. **9**, 13465 (2019).

[11] D. Bluvstein, H. Levine, G. Semeghini, T. T. Wang, S. Ebadi, M. Kalinowski, A. Keesling, N. Maskara, H. Pichler, M. Greiner, *et al.*, A quantum processor based on coherent transport of entangled atom arrays, Nature (London) **604**, 451 (2022).

[12] E. Knill, R. Laflamme, and G. J. Milburn, A scheme for efficient quantum computation with linear optics, Nature (London) **409**, 46 (2001).

[13] N. H. Lindner and T. Rudolph, Proposal for pulsed on-demand sources of photonic cluster state strings, Phys. Rev. Lett. **103**, 113602 (2009).

[14] B. Li, S. E. Economou, and E. Barnes, Photonic resource state generation from a minimal number of quantum emitters, npj Quantum Inf. **8**, 11 (2022).

[15] D. E. Browne and T. Rudolph, Resource-efficient linear optical quantum computation, Phys. Rev. Lett. **95**, 010501 (2005).

[16] P. Hilaire, L. Vidro, H. S. Eisenberg, and S. E. Economou, Near-deterministic hybrid generation of arbitrary photonic graph states using a single quantum emitter and linear optics, Quantum **7**, 992 (2023).

[17] S.-H. Lee and H. Jeong, Graph-theoretical optimization of fusion-based graph state generation, Quantum **7**, 1212 (2023).

[18] H. Zhang, A. Wu, Y. Wang, G. Li, H. Shapourian, A. Shabani, and Y. Ding, ONEQ: A compilation framework for photonic one-way quantum computation, in *Proceedings of the 50th Annual International Symposium on Computer Architecture, ISCA '23* (ACM Press, New York, NY, 2023), pp. 1–14.

[19] F. Zilk, K. Staudacher, T. Guggemos, K. Fürlinger, D. Kranzlmüller, and P. Walther, A compiler for universal photonic quantum computers, in *2022 IEEE/ACM Third International Workshop on Quantum Computing Software (QCS)* (IEEE, New York, 2022), pp. 57–67.

[20] S. C. Wein, T. G. de Brugière, L. Music, P. Senellart, B. Bourdoncle, and S. Mansfield, Minimizing resource overhead in fusion-based quantum computation using hybrid spin-photon devices, PRX Quantum **6**, 040362 (2025).

[21] E. Takou, E. Barnes, and S. E. Economou, Optimization complexity and resource minimization of emitter-based photonic graph state generation protocols, npj Quantum Inf. **11**, 108 (2025).

[22] M. Hein, J. Eisert, and H. J. Briegel, Multi-party entanglement in graph states, Phys. Rev. A **69**, 062311 (2004).

[23] A. Kotzig, Eulerian lines in finete 4valent graphs and their transfomations, Theor. Graphs 219 (1968).

[24] M. Van den Nest, J. Dehaene, and B. De Moor, An efficient algorithm to recognize local Clifford equivalence of graph states, Phys. Rev. A **70**, 034302 (2004).

[25] N. Claudet and S. Perdrix, Local equivalence of stabilizer states: A graphical characterisation, in *42nd International Symposium on Theoretical Aspects of Computer Science (STACS 2025)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 327 (Schloss Dagstuhl Leibniz-Zentrum fr Informatik, 2025), pp. 27:1–27:18.

[26] M. Mhalla and S. Perdrix, Graph states, pivot minor, and universality of (X, Z)-measurements, Int. J. Unconv. Comput. **9**, 153 (2013).

[27] F. Ewert and P. van Loock, 3/4-efficient Bell measurement with passive linear optics and unentangled ancillae, Phys. Rev. Lett. **113**, 140403 (2014).

[28] S. Bartolucci, P. M. Birchall, M. Gimeno-Segovia, E. Johnston, K. Kieling, M. Pant, T. Rudolph, J. Smith, C. Sparrow, and M. D. Vidrighin, Creation of entangled photonic states using linear optics, arXiv:2106.13825.

[29] G. de Felice, B. Poór, L. Yeh, and W. Cashman, Fusion and flow: Formal protocols to reliably build photonic graph states, arXiv:2409.13541.

[30] This definition differs from the definitions given in [2,29] where fusions are not represented as separate edges in the graph. We choose this alternate definition since it allows for an easier description of fusion operations as shown in the following sections.

[31] R. Diestel, *Graph Theory* (Springer Nature, 2025).

[32] M. C. Löbl, L. A. Pettersson, S. Paesani, and A. S. Sørensen, Transforming graph states via Bell state measurements, Quantum **9**, 1795 (2025).

[33] That convention can also be adapted to hardware by configuring the linear optical elements in a type-II fusion (cf. [32]).

[34] J. G. Kemeny *et al.*, *Finite Markov Chains* (van Nostrand, Princeton, NJ, 1969), Vol. 26.

[35] S. Redner, *A Guide to First-Passage Processes* (Cambridge University Press, Cambridge, 2001).

[36] A. Bouchet, Transforming trees by successive local complementations, J. Graph Theor. **12**, 195 (1988).

[37] H. Sharma, K. Goodenough, J. Borregaard, F. Rozpedek, and J. Helsen, Minimizing the number of edges in LC-equivalent graph states, Quantum **10**, 2001 (2025).

[38] K. Staudacher, T. Guggemos, S. Grundner-Culemann, and W. Gehrke, Reducing two-qubit gate count for ZX-calculus based quantum circuit optimization, Electron. Proc. Theor. Comput. Sci. **394**, 29 (2023).

[39] J. Edmonds, Paths, trees, and flowers, Can. J. Math. **17**, 449 (1965).

[40] K. Staudacher, Adaptive framework for failure-aware protocols in fusion-based graph-state generation [Computer software], Zenodo, 2026, https://doi.org/10.5281/zenodo.18316338.

[41] The corresponding software package is available at https://github.com/seokhyung-lee/OptGraphState.