Ruh, Jannis; Devitt, Simon

Quantum Circuit Optimization and MBQC Scheduling With a Pauli Tracking Library

Published in:
IEEE Transactions on Quantum Engineering

DOI:
10.1109/TQE.2025.3610112

Published: 01/01/2025

Document Version
Publisher's PDF, also known as Version of record

Please cite the original version:
Ruh, J., & Devitt, S. (2025). Quantum Circuit Optimization and MBQC Scheduling With a Pauli Tracking Library. *IEEE Transactions on Quantum Engineering*, *6*, Article 2500612. https://doi.org/10.1109/TQE.2025.3610112

IEEE Transactions on Quantum Engineering

Received 7 May 2025; revised 5 August 2025; accepted 10 September 2025; date of publication 15 September 2025; date of current version 13 October 2025.

Digital Object Identifier 10.1109/TQE.2025.3610112

Quantum Circuit Optimization and MBQC Scheduling With a Pauli Tracking Library

JANNIS RUH[1] AND SIMON DEVITT[1,2]

[1]Centre for Quantum Software and Information, School of Computer Science, Faculty of Engineering and Information Technology, University of Technology Sydney, Ultimo, NSW 2007, Australia

[2]InstituteQ, Aalto University, 02150 Espoo, Finland

Corresponding author: Jannis Ruh (e-mail: jannis.ruh1@uts.edu.au)

The work of Jannis Ruh was supported by Sydney Quantum Academy, Sydney, NSW, Australia. This work was supported by Defense Advanced Research Projects Agency through the Quantum Benchmarking Program under Grant HR00112230007 and Grant HR001121S0026. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

ABSTRACT In this article, we present a generic framework for the scheduling of qubits in measurement-based quantum computing and error-corrected circuits that are implemented through Clifford circuits. The framework is based on the commutation of Pauli operators through quantum circuits, which is called Pauli tracking. We complement the framework by providing an independent software library to perform the Pauli tracking, specifically in the context of measurement-based quantum computing. Tracking Pauli operators allows one to reduce the number of Pauli gates that must be executed on quantum hardware and to capture the constraints on the order of measurements. The scheduling problem is numerically investigated on small scale.

INDEX TERMS Taxonomy: optimal scheduling, quantum computing, quantum circuit, qubit.

I. INTRODUCTION

Realizing fault-tolerant quantum gates and qubits is costly in terms of space, time, and energy. It is crucial to reduce the number of gates as much as possible. This optimization can be tackled on multiple levels, e.g., optimizing the quantum algorithm on the logical circuit level, or trying to develop more efficient error correcting codes.

Another problem that appears in the context of measurement-based quantum computation (MBQC) [2], [3], but partially also in quantum error correction (QEC), is dynamic corrections that are induced into the computation because of the nondeterminism of the measurements. For an efficient computation, these corrections have to be actively accounted for and cannot be ignored through postselection. They are also an essential piece of information for qubit scheduling, i.e., reusing qubits after they have been measured, as it defines an order on the measurements, which we shall explain in more detail.

In this article, we shall focus on the classical tracking of Pauli operators through a quantum circuit, called Pauli tracking [4], [5], [6], [7], which deals with the second problem introduced above and (partially) with the first problem. Pauli tracking, we shall explain it below, is an optimization done on the software level that directly reduces the number of Pauli gates that must be executed on the quantum hardware. Furthermore, it can capture the information of dynamic Pauli correction induced by measurements. Pauli tracking is applicable whenever the quantum circuit is comprised of Clifford gates (and arbitrary measurements), as, for example, in MBQC, but also in the context of QEC, e.g., the surface code [8], [9]. The QEC codes themselves are often Clifford circuits on the level of the (uncorrected) physical qubit; but also on the logical fault-tolerant level, the non-Clifford gates are usually realized via injection of certain "magic" states or specific measurements with the help of additional ancilla qubits that are entangled using only Clifford gates (cf., Fig. 1) [9], [10], [11]. Therefore, Pauli tracking can be applied on both levels in QEC, the physical level and the logical level. This gives it, together with MBQC, protocols a wide range of applications.

Let us now sketch out what we mean with Pauli tracking and how it works; it is based on the mathematical foundations discussed in [12], [13], and [14]. The Clifford group is the normalizer of the Pauli group, meaning that conjugating the Pauli group with Clifford operators preserves the Pauli group. This implies that Pauli operators can be classically efficiently, i.e., without exponential costs, commuted with Clifford operators. In the context MBQC or QEC where the circuits only consist of Clifford gates and measurements, it is therefore possible to traverse the Pauli operators through

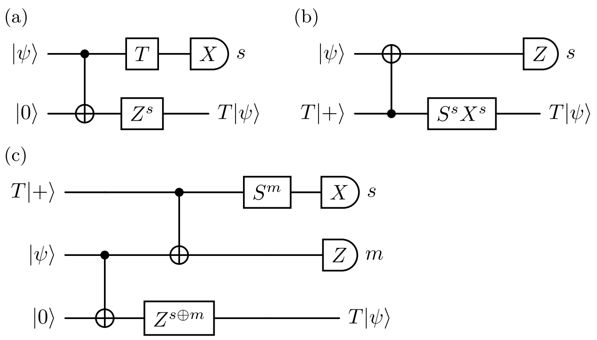

FIGURE 1. Example of T gate teleportation: these are typical protocols in measurement-based quantum computation (MBQC) but also especially in quantum error correction (QEC) where the ancilla states, $T|+\rangle$, are prepared in a distillation process [1]. (a) T teleportation is achieved with a "magic" measurement $TXT^\dagger$. This introduces a Z correction on the output qubit depending on the measurement result. The correction can be tracked as a Pauli frame through a subsequent Clifford+measurement circuit, unblocking the execution until a measurement is reached for which the Pauli frame defines a nontrivial correction. (b) Implementation of the T teleportation through injection of a "magic" state $T|+\rangle$. This protocol introduces a potential S (phase gate) correction on the output qubit. Since this correction cannot be tracked through the subsequent circuit, it is blocking. (c) Implementing the T teleportation with a magic state injection, however, without blocking the execution. This circuit is constructed by (a) taking circuit and then (b) implementing the T gate using circuit.

the circuit by commuting them with the Clifford gates until a measurement is reached and then to account for the Pauli operators through postprocessing or adaptation of the measurement [cf., Fig. 1(c)]. This effectively reduces the number of Pauli gates that have to be executed on the quantum hardware to the number of qubits or less.

In Section III, we present a library that can be used to perform the Pauli tracking [15]. It is a low-level library, natively written in Rust, with a Python wrapper and a partial C interface. The library is designed to be dynamically used when compiling or executing quantum circuits and supports various generic data structures for different use cases. A simple example application that is supported by our library is the generic calculation of the $Z^{s\oplus m}$ correction on the output qubit in circuit (c) in Fig. 1: commute the X correction in circuit (b) past the S correction, turning it into a $ZX \propto Y$ correction; the X part does not matter for the following X measurement and can be completely removed; the Z part, however, flips the measurement result s, which can be accounted for by moving the Z correction onto the output qubit as a Z correction (since s induces a Z correction). All of these three operations, *commute*, *remove*, and *move*, are supported in our Pauli tracking library. In Appendix B, we briefly compare the library to other software tools that provide similar functionality, e.g., Stim [16]. See Section VI for how to access the library.

Now, when teleporting non-Clifford gates in the context of QEC, or in general any gate as in MBQC, the nondeterminism of the corresponding measurements usually introduces Pauli corrections (or in general Clifford corrections) conditioned on the measurement outcomes, as for example, depicted in Fig. 1. These corrections effectively define a strict partial time order for the execution of the circuit because of their nondeterminism in general. The tracking of the Pauli corrections through the circuit directly captures this order [except for the order induced by possible Clifford corrections, however, they can also be deferred with an additional teleportation; cf., Fig. 1(c)] and reduces it to the measurements. Analyzing this information, in connection with entanglement structure of the circuit or graph state, gives knowledge about how the computation can be optimized in space and in time, i.e., what is the minimal number of required qubits (fully space optimized), what is the minimal number of steps of parallel measurements (fully time optimized), or in general, given a fixed number of qubits, how many steps of parallel measurements are required? Furthermore, capturing this time order and the corrections is necessary for certain MBQC compilation strategies and frameworks, e.g., [17], [18], and [19], where a logical circuit is transformed into a logical graph state (using teleportation techniques and stabilizer simulations) and the captured time order defines how the graph state is consumed.

In Section IV, we shall focus on this problem. More specifically, we define the scheduling problem for MBQC, which may also be applied in the context of QEC circuits locally at the parts where non-Clifford gates are teleported, and tackle it detached from the underlying quantum circuit on the logical level (i.e., we do not consider the quantum hardware). The problem will be defined in a way that is applicable to MBQC protocols, such as the Raussendorf cluster [2], but also other protocols that involve more complex graph states, e.g., [17], [18], and [19], or in general, whenever the entanglement structure of the circuit can be described with a graph, and teleportation techniques are used for non-Clifford gates. Finding a time-optimal solution can be done trivially in polynomial time based on the Pauli tracking of the measurement corrections. Moreover, this solution is generally independent of the underlying graph, i.e., it is not restricted to pure MBQC protocols. However, finding a space-optimal solution is NP-hard, (the problem is related to finding the pathwidth of a graph [20]). This can be resolved if the input can be split into smaller problems, e.g., one might be able to serially split the circuit [18], [21]. We provide limited numerical results on these solutions.

As an example application of our library and scheduling framework, it has been used in [22] to estimate quantum resources in a superconducting quantum computer. Furthermore, it is part of a full-stack quantum compiler and resource estimator [18].

II. MEASUREMENT-BASED QUANTUM COMPUTING

Measurement-based quantum computation is a compilation strategy of quantum algorithms based on gate teleportations through measurements. We provide a brief exemplary introduction to MBQC to explain the idea of Pauli tracking and scheduling. A more extensive discussion can, for example, be found in [3] and [5].

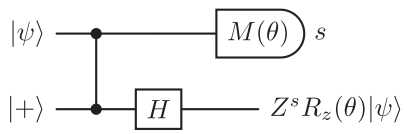

FIGURE 2. Teleportation protocol for the $R_z(\theta)$ gate. The measurement operator is defined as $M(\theta) = R_z(\theta) X R_z(\theta)^\dagger$; the vertical line represents the CZ gate.

Let us first introduce some basic gates and states. The Pauli operators, $X, Y,$ and Z, are defined as

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} .$$

The Hadamard gate H and the controlled-Z gate CZ, are defined as

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \mathrm{CZ} := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$$

respectively. We denote the eigenstates of X as $|+\rangle := H|0\rangle$ and $|-\rangle := H|1\rangle$. Lastly, the z-rotation gate is defined as $R_z(\theta) := \mathrm{e}^{-i\theta Z/2}$, $\theta \in \mathbb{R}$. Note that the gate set $\{\mathrm{CZ}, H, R_z(\theta)\}$ forms a universal gate set for quantum computation, i.e., any quantum circuit can be expressed with these gates [23].

Consider the state $R_z(\theta)|\psi\rangle$ for an arbitrary initial qubit state $|\psi\rangle$ and angle θ. In the standard quantum circuit model, this state is realized by preparing the qubit in the state $|\psi\rangle$ and then applying the unitary $R_z(\theta)$ to that state. MBQC provides an alternative approach to that. The action of the rotation gate is realized by a teleportation protocol with an appropriate measurement, as depicted in Fig. 2.

The idea of pure MBQC is to implement all gates by such teleportation protocols, i.e., entangle the qubits and then perform local measurements on some qubits such that the desired output state is on the remaining qubits.

There are three crucial observations we can make about the MBQC protocol. First, we can split the circuit in Fig. 2 into two parts. Assuming $|\psi\rangle = |+\rangle$, the first part is an entangling operation on $|+\rangle$ states via the controlled-Z gate, and the second part is only local measurements up to Hadamard and Pauli gates. This is still true if we extend the circuit by additional teleportation protocols. Therefore, the quantum state before the measurements (up to Hadamards and Paulis) is effectively a graph state; a graph state is defined as follows: let $G = (V, E)$ be a simple undirected graph with vertex set V and edge set E. Then, the graph state $|G\rangle$ is defined as

$$|G\rangle = \prod_{(u,v)\in E} \mathrm{CZ}_{uv} \bigotimes_{v\in V} |+\rangle_v .$$

It is the unique state stabilized by the stabilizers $g_u = X_u \prod_{v\in\mathcal{N}}(u) Z_v$, where $\mathcal{N}(u)$ is the set of neighbors of u in G. In [2], it was shown that a 2-D lattice graph state is sufficient for universal quantum computation by gate teleportation.

The second observation is that on the output state in Fig. 2, we have a Z correction depending on the measurement result s. In general, the nondeterminism of the measurements can introduce such corrections, which impose a time order, as discussed in Section I. For example, in Fig. 2, the output state can in general not be used until the measurement result s is known.

The last observation is that except for the measurement, the circuit in Fig. 2 is a pure Clifford circuit. Clifford circuits are characterized by the fact that they map Pauli operators to Pauli operators under the conjugation operation, i.e., if p is a Pauli operator and c is a Clifford operator, then cpc^{-1} is again a Pauli operator. This implies that Pauli operators can be traversed through a Clifford circuit, i.e., we have $cp = \tilde{p}c$ for some appropriate Pauli operator $\tilde{p}$. This process—calculating $\tilde{p}$—is called Pauli tracking and can be done classically efficiently. A specific application of Pauli tracking is the simulation of pure Clifford circuits, i.e., Clifford circuits where all measurements are Pauli measurements. In that case, the initial state can be efficiently represented by Pauli strings, called stabilizers, and the action of the circuit can be efficiently calculated by updating the stabilizers via Pauli tracking. The measurements are then simulated by coin flips according to the anticommutation relations of the measurements with the updated stabilizers (cf., [23] and [24]). This is called a stabilizer simulation; Gidney [16], for example, provided an efficient implementation of such a simulator. In the context of general MBQC, however, this is not applicable, since the measurements are not necessarily Pauli measurements. However, we can still use Pauli tracking to keep track of the induced nondeterministic Pauli corrections and imposed time order.

In Section III-A, we shall provide a more rigorous description of Pauli tracking.

III. PAULI TRACKING LIBRARY

In Section I, we already sketched out the basic idea. Here, we shall focus on a mathematical description which directly represents how the library is implemented. Afterward, we discuss some features of the library.

A. MATHEMATICAL FORMULATION

Given the Pauli operators, $X, Y, Z \in \mathrm{U}(\mathbb{C}^2)$, where U denotes the unitary group, we define the Pauli group as follows.

Definition 1 (Pauli Group): Let $n \in \mathbb{N}$. The Pauli group $\mathcal{P}_n \leq \mathrm{U}(\mathbb{C}^{2^n})$ is defined by its generators via

$$\mathcal{P}_n := \langle i, X_1, Z_1, \ldots, X_n, Z_n \rangle .$$

For later reference, we also define the (Heisenberg–Weyl) group $\mathcal{P}^{\mathrm{H}}_n := \langle X_1, Z_1, \ldots, X_n, Z_n \rangle$.

The projective groups are $\overline{\mathcal{P}}_n = \mathcal{P}_n / \langle i \rangle$ and $\overline{\mathcal{P}^{\mathrm{H}}}_n = \mathcal{P}^{\mathrm{H}}_n / \langle -1 \rangle$, respectively.

One can also define the Pauli group differently, e.g., by including phases or arbitrary complex prefactors, but note that the respective projective groups, which are the groups we are interested in, are all isomorphic, e.g., it is $\overline{\mathcal{P}}_n \cong \overline{\mathcal{P}^{\mathrm{H}}[n]}$.

The Clifford group is now defined as the normalizer of the Pauli group.

Definition 2 (Clifford Group): Let $n \in \mathbb{N}$. The (unitary) Clifford group is defined by

$$\mathcal{C}_n := \{U \in \mathrm{U}(\mathbb{C}^{2^n}) | U\mathcal{P}_n U^{-1} = \mathcal{P}_n\} .$$

The projective group is given by $\overline{\mathcal{C}}_n = \mathcal{C}_n/\mathrm{U}(1)$.

Alternatively, the Clifford group can be defined through its generators, e.g., [25]

$$\mathcal{C}_n = \langle u, H_i, S_i, \mathrm{CZ}_{ij} | u \in \mathrm{U}(1);\; i, j \in \mathbb{N}_{\le n};\; i \neq j\rangle$$

where H is the Hadamard operator $H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1\end{pmatrix}$, S is the phase operator $S = \mathrm{diag}(1, i)$, and CZ is the controlled-Z operator $\mathrm{CZ} = \mathrm{diag}(1, 1, 1, -1)$.

Again, we could have chosen slightly different definitions, but the resulting groups would be either the same or at least the projective groups would be isomorphic.[1]

Now, let us assume that we have two Pauli operators $p_1, p_2 \in \mathcal{P}_n$ and a Clifford operator $c \in \mathcal{C}_n$ that form the circuit $p_1 c p_2$. Under Pauli tracking, we understand the process of transforming this circuit into pc for some $p \in \mathcal{P}_n$, that is, we traverse all Pauli operators to the end of the circuit and collapse them into one Pauli operator. It is clear that this is achieved by setting $p = p_1 c p_2 c^{-1}$, i.e., conjugating the second Pauli operator with the Clifford operator. However, we can reduce the problem to a simpler form. First, since we are dealing with quantum circuits, scalar factors do not matter, and we can reduce the Pauli group to its projective group, i.e., set $p_1, p_2, p \in \overline{\mathcal{P}}_n$. Second, we also only have to consider the projective Clifford group, i.e., set $c \in \overline{\mathcal{C}}_n$, since scalar factors are canceled when conjugating. Moreover, since conjugating Pauli operators with Pauli operators only introduces a phase (more specifically, the centralizer of $\overline{\mathcal{P}}_n$ in $\overline{\mathcal{C}}_n$ is $\overline{\mathcal{P}}_n$ [13]), we can reduce the Clifford group to the quotient group $\overline{\mathcal{C}}_n/\overline{\mathcal{P}}_n$. This leads us to the following definition of the Pauli tracking problem.

Definition 3 (Pauli Tracking): Let $m, n \in \mathbb{N}$. Given a sequence $(g_i)_{1\le i\le m} \subseteq \overline{\mathcal{P}}_n \cup \overline{\mathcal{C}}_n/\overline{\mathcal{P}}_n$, calculate $(p_i)_{1\le i\le m} \subseteq \overline{\mathcal{P}}_n$, which is recursively defined by

$$p_i = \begin{cases} g_i p_{i-1}, & \mathrm{if} g_i \in \overline{\mathcal{P}}_n \\ g_i p_{i-1} g_i^{-1}, & \mathrm{if} g_i \in \overline{\mathcal{C}}_n/\overline{\mathcal{P}}_n \end{cases} \tag{1}$$

for $i \in \{1, \ldots, m\}$, where $p_0 = \mathbb{1}$.

We refer to $(p_i)_{0\le i\le m} \subseteq \overline{\mathcal{P}}_n$ as the Pauli frame sequence that is tracked through the circuit sequence $(g_i)_{1\le i\le m}$.

[1] For example, one could replace the unitary group U with the linear group GL, which would result in a Clifford group $\mathrm{C}_n \le \mathbb{C}^\times \mathrm{U}(\mathbb{C}^{2^n})$, i.e., more specifically, the same group as before, but with arbitrary prefactors $\mathbb{C}^\times = \mathbb{C}\backslash\{0\}$, cf., [12] and [13]. Using $\mathbb{C}^\times\mathcal{P}_n$ as a normalized group leads to the same Clifford group; however, using $\mathcal{P}_n^{\mathrm{H}}$ as a normalized group would lead to a different group, since it is not invariant under Clifford conjugations.

This is the computational task we want to solve with our library, which is essentially achieved by implementing the following two isomorphisms.

Proposition 4: Let $n \in \mathbb{N}$. $\overline{\mathcal{P}}_n$ is isomorphic to the abelian group $\overline{\mathrm{H}(\mathbb{Z}_2^n)} = (\mathbb{Z}_2^n \times \mathbb{Z}_2^n, +) \cong \mathbb{Z}_2^{2n}$ with the standard addition via

$$\overline{\tau} : (\mathbb{Z}_2^n \times \mathbb{Z}_2^n) \to \overline{\mathcal{P}}_n, \quad (z, x) \mapsto \bigotimes_{j=1}^{n} \overline{Z}_j^{z_j} \overline{X}_j^{x_j} . \tag{2}$$

Proposition 5 (See [12] and [13]): Let $n \in \mathbb{N}$. The projective Clifford group, modulo Pauli operators, is isomorphic to the symplectic group, i.e.,

$$\kappa : \overline{\mathcal{C}}_n/\overline{\mathcal{P}}_n \to \mathrm{Sp}_{2n}(\mathbb{Z}_2) \quad c\overline{\mathcal{P}}_n \mapsto S_c = \overline{\tau}^{-1} \circ \mathrm{inn}_c \circ \overline{\tau}$$

where $\mathrm{Sp}_{2n}(\mathbb{Z}_2)$ is the symplectic group of the $\mathbb{Z}_2^{2n}$ vector space with respect to the standard symplectic form $\begin{pmatrix} 0 & \mathbb{1} \\ \mathbb{1} & 0\end{pmatrix}$, and inn_c is the inner automorphism induced by c, i.e., conjugation with $c \in \overline{\mathcal{C}}_n$.

The first isomorphism $\overline{\tau}$ in Proposition 4 describes how Pauli operators are represented in our library, i.e., they are simply represented by a pair of Boolean values or bits, and multiplication is done via the XOR operation. The second isomorphism κ in Proposition 5 then says that the according operation of a conjugation with a Clifford element $c\overline{\mathcal{P}}_n$ in the binary Pauli representation is given by the symplectic operator $\kappa(c\overline{\mathcal{P}}_n)$. Gross [26] explicitly listed the symplectic operators for some Clifford elements that are implemented in the library.

Note that the process of Pauli tracking is very similar to stabilizer simulations. The group $\mathrm{H}(\mathbb{Z}_2^n)$ is the Heisenberg group and $\mathcal{P}_n^{\mathrm{H}}$ is its isomorphic Weyl representation [12], [13], [14], [27], [28], [29]. In stabilizer simulations, these isomorphisms are adjusted to nontrivial stabilizer subgroups of $\mathcal{P}_n$, and the updating of the stabilizers works similar to the Pauli tracking, with the differences that signs have to be accounted for.

The matrix representation of the Clifford conjugations gives us an upper bound for the computation cost of the Pauli tracking per gate, that is, the cost of one conjugation is bounded by $\mathcal{O}(n^2)$. However, in reality, the cost per gate is usually much lower, since the standard Clifford gates used in quantum circuits are usually local to a lower number of qubits $m \in \mathbb{N}$, $m \le n$, and they can often be implemented in a more efficient way than simple matrix multiplication. For example, a Hadamard gate is often just one memory swap, or a controlled-Z gate can be implemented with two XOR operations. If we bound m by a constant, for example, $m = 2$, and the circuit consists of $l \in \mathbb{N}$ gates, and we track $k \in \mathbb{N}$ Pauli frames simultaneously, then the Pauli tracking can be performed in $\mathcal{O}(lk)$ time with $\mathcal{O}(nk)$ memory. In the case of MBQC, under the assumption that each teleportation induces a Pauli correction, that is, a Pauli frame (cf., Fig. 1), it is roughly $l \propto k \propto n$, i.e., both costs, time and space, are of the order $\mathcal{O}(n^2)$.

B. LIBRARY FEATURES

In Section III-A, we focused on how the Pauli tracking logic is implemented. In this section, we shall give a brief overview of how the library can be used. However, we shall not give examples or explicitly discuss the API; for that, refer to the documentation of the software packages (cf., Section VI).

There are essentially two modes in which the library can be used. The first one is for Pauli tracking when all gates are known, for example, dynamically during execution of a quantum circuit. In this mode, there is one Pauli frame (cf., Definition 3), that is, one Pauli operator for each qubit, and this frame is updated according to the circuit instructions. The required memory for this is linear in the number of qubits.

The second mode is to perform the Pauli tracking when defining the quantum circuits, or compiling them, which contain gate teleportations, e.g., as in MBQC. In this mode, for each potentially induced Pauli correction (it is usually nondeterministic since it depends on a measurement result), one Pauli frame can be captured and then tracked through the circuit. These frames can then be used to determine the strict partial time order of the measurements that we discussed in Section I. During execution, the Pauli corrections before the measurements can then be obtained from the frames. In this mode, the required memory is linear in the number of qubits and frames, i.e., induced corrections. For pure MBQC circuits, this means that the memory scales approximately quadratically with the number of vertices in the graph state. The frames are stored in major-qubit-minor-frame order. This way, the conjugations can be performed through vectorized operations and simple memory swaps.

The way the Pauli tracking is performed from the user side is to initialize a tracking object, and then update it by calling the according methods that correspond to the circuit instructions (cf., library documentation examples). This works, for example, similar to how quantum circuits are constructed in some of the quantum computing libraries, e.g., Circ [30] or Qiskit [31], where one initializes a circuit object and then adds gates to it.

When using the Rust native library, the user can choose among different data structures for the representation of the Pauli frames. This is achieved by designing the library generically (through static dispatch, i.e., monomorphization) in its core data structures. This way, the user can choose the most appropriate data structure for the specific use case. For example, while single instruction, multiple data bit-vectors may allow faster executions of the Clifford conjugations, normal bit-vectors can be more efficient if the user often has to access the Pauli frames. The library directly supports some standard data structures, but the user can also support their own data structure by providing the required methods. For example, to support a data structure as top-level tracking object, the data structure only has to provide methods that implement the H, S, and CZ gates since every other Clifford gate can be constructed through default methods according to Definition 2.

When using the Python wrapper, this generic flexibility is not completely given, but provided to a restricted set of data structures.

IV. MBQC SCHEDULING

In MBQC [2], [3], the quantum gates are realized through gate teleportation protocols, i.e., entanglement of the qubits with additional qubits and then performing specific measurements, which effectively realize the gate, as described in Section II. The entangled resource state is usually describe by a graph where the vertices represent the qubits and the edges represent the entanglement. The graph usually contains a large number of vertices, i.e., qubits, and one critical aspect of MBQC is to reuse qubits after they have been measured. More specifically, it is important to schedule initialization, entanglement, and measurement of the qubits in a way that reduces the quantum memory requirement (space cost) and the execution time (time cost). This, of course, depends heavily on the underlying hardware and its architecture, but even without considering that, it is a hard problem.

The scheduling is also restricted by certain constraints. One of the constraints is induced by the nondeterminism of the measurements. Since the measurement outcome is not known prior to execution, they introduce Pauli corrections depending on the measurement outcome. If these corrections commute or anticommute with the subsequent measurement, then they can be accounted for by postprocessing; however, this is not the case in general. Therefore, the corrections define a time order for the measurements. For certain MBQC protocols, such as the Raussendorf 2-D cluster [2], a correct scheduling of the measurements is directly included in the protocol and graph state; however, this scheduling may not be time optimal. Moreover, for other protocols, that, for example, involve certain graph transformations, finding a time order that is not too restrictive may be decoupled from the graph state. This is where the Pauli tracking can help. For example, consider the protocol proposed in [17], [18], and [19]. In this protocol, a quantum circuit is first transformed into a larger Clifford circuit that includes gate teleportations, and then a graph state is computed from the Clifford circuit. The graph state is independent of the Pauli corrections; however, they are captured via Pauli tracking during the transformation to the Clifford circuit. From the captured Pauli frames, one can then calculate the measurement time order for the graph simply by checking for each frame on which qubits it induces potential corrections.

The other constraint on the scheduling is that a qubit can only be measured when all its neighbors have been initialized and entangled with the qubit. This is because the entanglement does not commute with the measurement.

In the following, we formulate a framework that describes the scheduling problem. We shall then describe how to solve this problem and present some numerical results that give indications about how much can be gained with this optimization [32].

A. FRAMEWORK

We start by defining a valid measurement schedule that accounts for the constraints described above. The graphs are undirected and simple, i.e., no self-loops or multiple edges between the same two vertices are allowed.

Definition 6 (Measurement Schedule): Let $(V, E, \prec)$ be a triple, where $G = (V, E)$ is a graph (with vertices V and edges E) and $\prec$ is a strict partial order on V. A measurement schedule is a sequence $S = (M_i, I_i)_{1\le i\le n}$, where $M_i, I_i \subseteq V$ and $n \in \mathbb{N}$, such that for all $1 \le i \le n$ (negative indexed sets are empty)

a) $M_i \cup \mathcal{N}(M_i) \subseteq I_i$;
b) $\mathrm{PRE}(M_i) \subseteq \bigcup_{1\le j<i} M_j$;
c) $\dot{\bigcup}_{1\le i\le n} M_i = V$;
d) $I_{i-1}\backslash M_{i-1} \subseteq I_i \subseteq V\backslash \bigcup_{1\le j<i} M_j$;

where $\mathcal{N}(M_i) = \{\mathcal{N}(x) | x \in M_i\}$ is the set of all neighbors of the vertices in M_i and $\mathrm{PRE}(M_i) = \{x \in V | \exists y \in M_i : x \prec y\}$ is the set of all vertices that are smaller than a vertex in M_i.

The first two conditions (a) and (b) describe the constraints under which a vertex can be measured, that is, the vertex itself and all its neighbors have to be initialized (and entangled), and all smaller vertices that induce possible corrections on the to-be-measured vertices have to be measured before. The last two conditions (c) and (d) are just for soundness, i.e., all vertices have to be measured, and each vertex is only initialized once and kept initialized (and entangled) until it is measured.

We can now define the space and time cost of a schedule, i.e., how much quantum memory is required and how many steps of parallel measurements are executed.

Definition 7 (Schedule Cost): Let $(V, E, \prec)$ be a graph with a strict partial order and $S = (M_i, I_i)_{1\le i\le n}$ be a corresponding measurement schedule, $n \in \mathbb{N}$. The space cost, sc, and time cost, tc, of the schedule are defined by

$$\mathrm{sc}(S) = \max_{1\le i\le n} |I_i|$$

$$\mathrm{tc}(S) = n\ .$$

The optimization problem, one is now faced with, is to find a schedule S with small space and time cost. For the special case that there is effectively no time order, i.e., there are no $\prec$ relations, it has been shown that finding a space-optimal schedule is equivalent to finding a path decomposition of the graph that realizes the pathwidth [20], implying that the pathwidth is a lower bound for the space cost $\mathrm{sc}(S)$. Just approximating the pathwidth is already NP-hard [20].

Given a graph and a time order, there are many possible schedules, e.g., given a sequence $(M_i)_i$, there are many possible $(I_i)_i$, such that $S = (M_i, I_i)_i$ is a valid measurement schedule. However, we are only interested in minimal $(I_i)_i$ w.r.t. a measurement pattern $(M_i)_i$.

Definition 8 (Measurement Pattern): Let $(V, \prec)$ be a tuple of a set of vertices V with a strict partial order $\prec$. A measurement pattern is a sequence $P = (M_i)_{1\le i\le n}$, where $M_i \subseteq V$ and $n \in \mathbb{N}$, such that for all $1 \le i \le n$

1) $\mathrm{PRE}(M_i) \subseteq \bigcup_{1\le j<i} M_j$;
2) $\dot{\bigcup}_{1\le i\le n} M_i = V$.

In contrast to a measurement schedule, a measurement pattern depends only on the time order but not on the spatial structure of the graph. However, we can construct a unique minimal schedule with respect to the pattern.

Proposition 9: Let $(V, E, \prec)$ be a graph with a strict partial order and $P = (M_i)_{1\le i\le n}$ be a measurement pattern of $(V, \prec)$, $n \in \mathbb{N}$. Then, the schedule $S = (M_i, I_i)_{1\le i\le n}$ with

$$I_i = M_i \cup \mathcal{N}(M_i) \cup (I_{i-1}\backslash M_{i-1}) \tag{3}$$

for all $1 \le i \le n$ (negative indexed sets are empty) is space optimal w.r.t. P in the sense that the measurement sets M_i, $1 \le i \le n$, are the same for P and S, and the space cost is minimal.

Proof: First, we need to show that S is a valid schedule. The only nontrivial condition is $I_i \subseteq V\backslash \bigcup_{1\le j<i} M_j$, for all $1 \le i \le n$, but this follows easily by induction.

The relative space optimality now simply follows from the fact that I_i is obviously the minimum set that satisfies conditions (a) and (d) for all $1 \le i \le n$. □

The proposed framework takes as input a graph with a strict partial order on the vertices. Therefore, it is applicable to a wide range of protocols, e.g., the one discussed in [17], [18], and [19], but also to potentially optimize protocols, such as the Raussendorf cluster, by greedily tracking Pauli corrections through the corresponding circuit. The greedy tracking of Pauli corrections may be compared to the signal shifting process discussed and used in [3], [33], and [34], respectively. There, given an MBQC pattern, the pattern is optimized by classically preprocessing Pauli measurements and corrections if possible. In contrast, our framework creates a measurement schedule after Pauli corrections have been classically preprocessed via Pauli tracking (the preprocessing of Pauli measurements may be compared to the compilation strategy in [17], [18], and [19]).

B. ALGORITHM

In the following two sections, we investigate the usefulness of scheduling optimizations on smaller scales, i.e., how much can the time and space costs be potentially reduced depending on the graph structure and the time order. Finding the time-optimal schedule can be achieved in $\mathcal{O}(|V|^3)$ time, where $|V|$ is the number of vertices in the graph, as we describe at the end of this section. Finding the space-optimal scheduling, however, is a computationally hard problem, and real applications will usually not reach the lower bound. The presented results give insights on how much improvement in the space costs by scheduling optimizations may be expected. The goal is not to provide an algorithm that can be used for large-scale quantum computing.

Our algorithm chooses different measurement patterns P, i.e., a sequence of parallel measurement steps that are allowed by the time order, and then creates a measurement schedule S according to Proposition 9. For this schedule, it calculates then the space and time costs. The hardness of the problem lies in the sheer number of possible choices for P. Again, considering the extreme case that there is no time order, the possible choices for P are the ordered partitions of the set V, and the number of those partitions is given by the ordered Bell number, which is approximately $|V|!/(2(\log 2)^{|V|+1})$, asymptotically in $|V|$ [35].

However, not all patterns have to be considered necessarily. In our algorithm, the measurement patterns are dynamically created, sketched out in the following—Algorithm 1 in Appendix A provides a simplified pseudocode description—(for more details, refer to the source code; it is essentially a depth-first-search). Let us assume we already calculated the space and time costs for some schedules. Now, when choosing a new pattern, we first choose a first measurement set M_1. For this set, we construct I_1 and calculate the space and time costs so far (i.e., $|I_1|$ and 1, respectively). If these costs are worse than the costs for the best schedules we found so far, then we directly discard the pattern and try a new set M_1'. If the costs are better, then we continue and choose a second set M_2. Again, we calculate the costs so far and either discard the second set or continue. We do this until we finally have a complete pattern and then repeat the process for a new pattern (in reality, we are not actually always starting with a first set M_1, but rather going a few steps back and then forward again with different measurement sets). This technique allows us to potentially skip many possible patterns; however, it highly depends on the structure of the graph and the time order. Although the scaling is in general still expensive. We tried to calculate the complexity of this implementation, but failed to derive a simple expression. Fig. 5 shows the run-times for some calculations; however, note that this is not a benchmark but only qualitative information.

The algorithm described so far covers only an exact optimization, i.e., finding the minimum costs; however, often approximations are sufficient. We implement an approximative version of the algorithm by putting a probabilistic condition on the acceptance of a next measurement set for a pattern, that is, if we choose some set M_i and the costs are better than the best costs so far, then the set is still only accepted with a certain probability. The probability function depends on appropriate parameters and can be specified by the user. This allows us to reduce the run-time; however, the final results are not guaranteed to be optimal, but only approximate the minimum costs, since some optimal patterns might be discarded (and depending on how aggressively measurement sets are discarded it might still not be scalable).

To get a time-optimal measurement schedule, however, is trivial, given a time order. For the next measurement set, one simply chooses all vertices that are allowed to be measured. Given some Pauli frames, the Pauli tracking library provides a method to calculate the according time order in $\mathcal{O}(|V|^3)$ time, representing it as a reduced directed acyclic graph. In this graph, the qubits are sorted into layers $(M_i)_i$ that describe when they can be measured the earliest, i.e., the layers define the measurement pattern $P = (M_i)_i$ for the trivial time-optimal schedule (cf., Definition 8). Obtaining the Pauli frames via Pauli tracking from the MBQC circuit is bounded by $\mathcal{O}(|V|^2)$, i.e., the schedule is obtained in $\mathcal{O}(|V|^3)$ time during compilation.

To make the space-optimal schedule search scalable, the search can be cutoff after a polynomial time at the cost of losing accuracy.

In practice, it may be possible to upper bound the number of vertices, i.e., $|V| \leq C$, for some constant $C \in \mathbb{R}$. For example, if it is possible to sequentially split the quantum algorithm into smaller parts with a bounded number of qubits and operations during compilation, then the space–time optimization becomes efficient w.r.t. C.

C. NUMERICAL RESULTS

In Figs. 3 and 4, we show some numerical results of the space and time costs for random graphs and time orders. The results are not directly based on quantum circuits but rather on random graphs and random time orders. The graphs are randomly generated by uniformly creating edges with a certain density p_e. Instead of directly drawing a random time order, we draw random Pauli frames and then calculate the time order from them using the Pauli tracking library. We do this to create a closer connection to the underlying MBQC scheduling problem. First, we randomly pick one vertex, which represents a vertex/qubit in a teleportation protocol that is going to be "measured" (cf., Fig. 1). Then, we randomly induce Pauli corrections on the other vertices (which have not been "measured" yet), dependent on the picked vertex, with a certain probability p_c. These corrections form a random Pauli frame that represents the correction induced by the picked vertex, which would have been tracked through a hypothetical circuit. This way, one might argue that the correction density p_c resembles the spreading of the induced Pauli corrections because of entanglement.

Alternatively, to have a closer relation to quantum algorithms, we could have directly drawn random circuits, or maybe specific circuits, and then used different MBQC protocols to transform them into graphs and time orders. However, this compilation is not trivial to implement and not part of this project.

Keeping this in mind, the shown results should be viewed as qualitative indications whether it might be worth to invest in this optimisation technique. In a specific application, a more structured search may be appropriate. Furthermore, it will also heavily depend on the specific quantum algorithm, the MBQC protocol, and the underlying hardware, which puts additional constraints on the scheduling. However, our results give insights on how much can be potentially gained by such a search on average without additional knowledge of the circuit. These insights are important for generic quantum

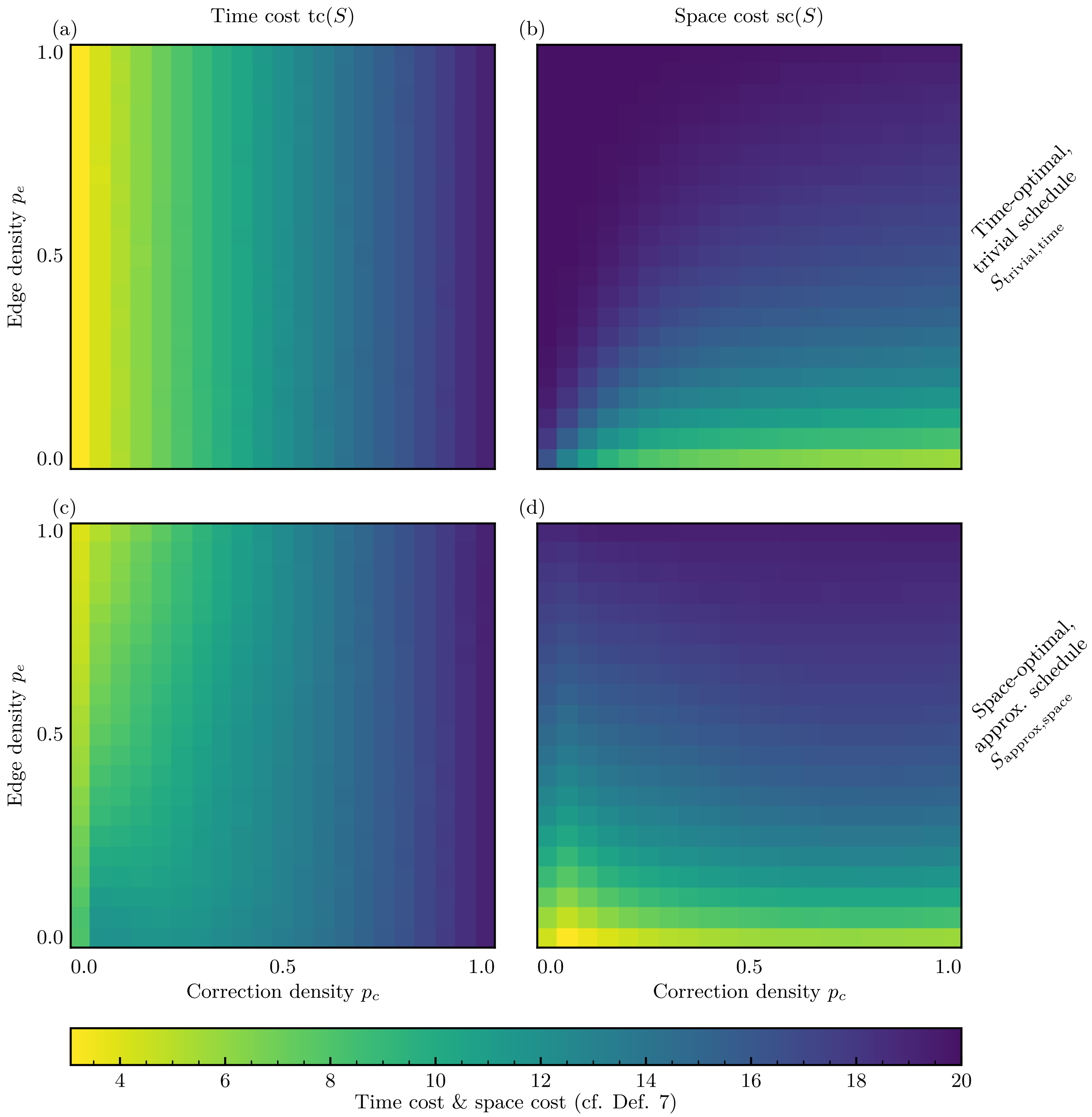

FIGURE 3. **Time and space costs (cf., Definition 7) for the trivial time-optimal schedule and the approximated space-optimal schedule, for $|V| = 20$ vertices. (a) Time cost tc for the time-optimal schedule $S_{\text{trivial,time}}$ directly obtained from the time order induced by the Pauli frames. Up to numerical errors, the time cost is independent of the edge density p_e and increases with the correction density p_c. (b) Space cost sc for the same schedule $S_{\text{trivial,time}}$. (c) and (d) According costs for the approximated space-optimal schedules $S_{\text{approx,space}}$, respectively. For details about the probabilistic approximated search, look at Appendix A.**

compilers where a full-algorithm analysis is generally not possible. For example, the results show that for high edge densities, not much can be gained by searching for a space-optimal schedule.

It also should be noted that the results are not strict benchmarks for larger $|V|$, since sampling enough data in a stable environment would require extremely long calculations due to the hardness of the problem. The approximated schedule searches were generally cutoff after $\mathcal{O}(|V|^2)$ time.

Fig. 3 shows the time and space costs for the trivial time-optimal schedule and an approximated space-optimal schedule. The costs are calculated for a fixed number of vertices, $|V| = 20$, but different edge densities p_e and correction densities p_c. The time cost for the time-optimal schedule [see Fig. 3(a)] is completely independent of the edge density p_e; this is clear, since it is directly calculated from the time order, without any reference to the graph. The lower the correction density p_c is, the lower time cost is. This is because the time order has less order relations, implying less layers in the directed time order graph, on average. Regarding the space cost for the approximated space-optimal schedule, in [see Fig. 3(d)], we see that it increases rather fast with the

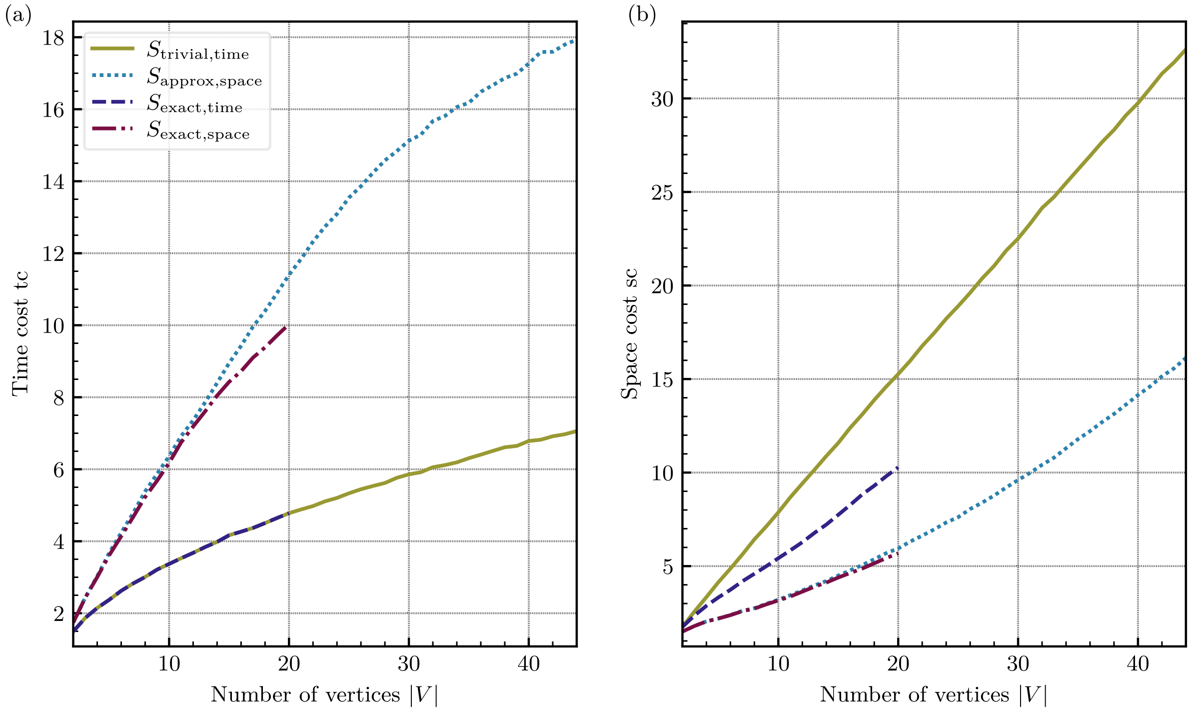

FIGURE 4. **Time and space costs depending on the number of vertices $|V|$. The edge and correction densities scale with $|V|$: $p_e = p_c = 0.5/\sqrt{|V|-1}$. (a) Time cost tc and (b) space cost sc for different schedules. $S_{\text{trivial,time}}$ and $S_{\text{approx,space}}$ are the trivial time-optimal schedule and approximated space-optimal schedule, as in Fig. 3. $S_{\text{exact,time}}$ and $S_{\text{exact,space}}$ are the according data of a full search for exact optimality, that is, $S_{\text{exact,time}}$ is a time-optimal schedule with the lowest possible space cost and $S_{\text{exact,space}}$ is a space-optimal schedule with the lowest possible time cost. These exact data points are only calculated for $|V| \leq 20$ since the run-time becomes very long. It is $\text{tc}(S_{\text{exact,time}}) = \text{tc}(S_{\text{trivial,time}})$ and the costs for $S_{\text{approx,space}}$ are close to the costs of $S_{\text{exact,space}}$; however, $\text{sc}(S_{\text{exact,time}})$ is fairly lower than $\text{sc}(S_{\text{trivial,time}})$.**

edge density p_e. The reason for that is that more vertices have to be initialized when we want to measure a vertex, according to Definition 6(a), since the vertex has more neighbors. With increasing correction density p_c, there are fewer choices to measure first a vertex with less neighbors [see Definition 6(b)], and therefore, the cost increases. The other two plots, as shown in Fig. 3(b) and (c), show mainly artifacts of how the algorithm is implemented. For low correction densities, the space cost of the trivial time-optimal schedule is high because this schedule greedily measures as many vertices as possible. The time cost for the approximated space-optimal schedule decreases with the correction density because the algorithm always first tries the more time-optimal schedules.

While these plots are not based on real quantum circuits, they confirm the expected behavior of the costs and maybe provide some qualitative intuition for when it is worth it to search for a more space-optimal schedule, if this is wanted. For example, if the edge density is relatively high, e.g., $p_e > 0.8$, and the correction density is not too low, e.g., $p_c > 0.6$, then it might not be worth searching for a space-optimal schedule because the space cost probably cannot be reduced significantly. On the other hand, if the edge density is low, the space cost can probably be significantly reduced.

In Fig. 4, we can see analogous costs, but now for a varying number of vertices $|V|$. The edge and correction densities scale via $p_e(|V|) = p_c(|V|) = 0.5/\sqrt{|V|-1}$. This is equivalent to the vertex degree scaling with $\sqrt{|V|-1}$. Importantly, in the asymptotic limit $|V| \to \infty$, it is $p_e(|V|) > \ln(|V|)/|V|$, which ensures that the graph is connected almost surely (cf., e.g., [36][2]). Notably, we see that the space costs for the approximated space-optimal schedule are close to the exact space-optimal schedule, at least up to the number of vertices for which we performed the exact optimization (with much faster run-times, cf., Fig. 5). For larger graphs, however, this deviation would of course increase. Furthermore, the space cost of the time-optimal schedule obtained from the exact optimization is lower than the space cost of the trivial time-optimal schedule. This is because the trivial time-optimal schedule does not try to measure first vertices with less neighbors.

For additional information about the numerical data, e.g., which probabilistic acceptance function is chosen or for recorded run-times, see Appendix A.

In general, the results show that appropriate scheduling of qubit initialization and measurement can potentially greatly reduce the space and time cost, depending on the system parameters (especially compared to just naively initializing the whole graph).

[2]Let $n \in \mathbb{N}$ be the number of vertices, i.e., $n = |V|$. In [36], it is proven that if $p_e(n) = (c + \ln n)/n$, then the graph G is connected with probability $\mathrm{e}^{-\mathrm{e}^{-c}}$ in the asymptotic limit $n \to \infty$ (a result by Erdös and Rényi).

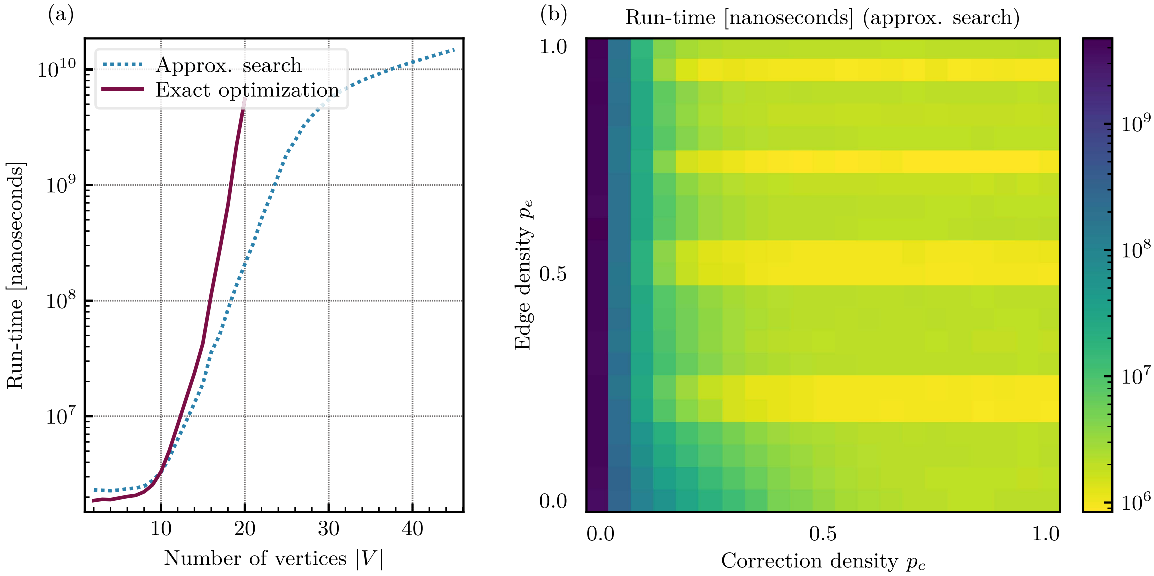

FIGURE 5. Recorded run-times of the calculations. This is not a benchmark. In both plots, the run-time is the average time it took to perform the optimization, given a graph and a time order. (a) Shows how the approximate algorithm performs compared to the exact algorithm, w.r.t. the number of vertices $|V|$. Considering the logarithmic scale, it is magnitudes faster. Starting at approximately 30 vertices, the timeout that scales quadratically with the number of vertices is reached, and the search returns early. (b) Shows how the run-time depends on the edge density p_e and the correction density p_c. It drastically increases for low correction densities. Vertically, for different edge densities, there are discontinuities. This is because for each selected edge density, the according data points, w.r.t. the correction density, were calculated on a different cluster node, which were under different load (running the simulations on smaller grids locally on a single laptop does not show these discontinuities).

Finding optimal schedules, however, is a hard problem, and will require optimized algorithms for larger scales. For general quantum circuits, it cannot be expected to find the optimal scheduling efficiently. The trivial time-optimal schedule, however, can be obtained in polynomial time from appropriately tracked Pauli frames, by greedily measuring the qubits as allowed by the induced time order.

V. CONCLUSION

In this article, we presented a software library that provides the functionality to track Pauli gates through a (Clifford) quantum circuit. The library is designed to be low level and generic, allowing easy integration into other projects. The library is based on the isomorphism between the Clifford group and the symplectic group. Tracking Pauli gates allows to reduce the number of Pauli gates to the number of qubits or less. Furthermore, when tracking Pauli corrections in MBQC, the information can be used to calculate the strict partial (time) order of the measurements. This allows us to perform scheduling optimizations. We presented a framework that covers the scheduling problem on an abstract level reduced to the underlying graph and the time order induced by the tracked Pauli corrections. The numerical results we showed give indications on how much can be gained with this optimization technique.

ACKNOWLEDGMENT

The authors like to thank Samuel Elman, Thinh Le, and Ryan Mann for helpful discussions. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

SOFTWARE AVAILABILITY

The source code of the Pauli tracking library can be found online in the taeruh/pauli_tracker repository[3] on GitHub [15]. The Rust library is published online,[4] and the Python wrapper is available online,[5] where you can also find links to the documentation. For the source code of the MBQC scheduling project, see online[6] [32].

APPENDIX A
MORE ON THE NUMERICAL RESULTS

Algorithm 1 provides a simplified pseudocode description of the exact search algorithm. The actual implementation takes some shortcuts and is multithreaded; additionally, it also allows a probabilistic acceptance function for the next measurement set.

The code for the data generation and the numerical data can be found at the commits 6aaa4d8[7] and 040e289[8] in taeruh/mbqc_scheduling/results [32],[9] respectively. The data

[3]https://github.com/taeruh/pauli_tracker
[4]https://crates.io/crates/pauli_tracker
[5]https://pypi.org/project/pauli-tracker/pypi.org
[6]https://github.com/taeruh/mbqc_scheduling
[7]https://github.com/taeruh/mbqc_scheduling/tree/6aaa4d8bf867075c3d90c91df87962e9e6c29377
[8]https://github.com/taeruh/mbqc_scheduling/tree/040e28959cb87bba7b0ab64e97d2ad5b4fe1bebb
[9]https://github.com/taeruh/mbqc_scheduling/tree/main/results

Algorithm 1: Scheduling Search.

procedure ExhaustiveSearch$V$, E, $\prec$
  schedules ← [] //index=tc(S), value=S
  space_costs ← [∞; $|V|$] : //index=tc(S), value = sc(S); initialise with worst case
  iter ← MeasurementPatternIterator(V, $\prec$)
  schedule ← []
  time_cost ← 0
  while measurement ← Next(iter) **do**
    time_cost ← time_cost +1
    AppendToSchedule(schedule, measurement, V, E)
    space_cost ← SpaceCost(schedule, V, E)
    if space_cost $\leq$ space_costs[time_cost] **then**
      if everything is measured **then**
        schedules[time_cost] ← schedule
        space_costs[time_cost] ← space_cost
        schedule ← []
        time_cost ← 0
      end if
    else
      SkipNode(iter)
      schedule.pop() //remove the last measurement
      time_cost ← time_cost −1
    end if
  end while
  returnschedules, space_costs
end procedure
function MeasurementPatternIterator$V$, $\prec$
    return... //depth-first-search iterator drawing measurement sets M_i
end Function
function Nextiter
    return... //next measurement set M_i in the depth-first-search
end Function
function SkipNodeiter
    return... //skip to next horizontal node in the depth-first-search
end Function
function AppendToSchedule$S$, M, V, E
    $S \leftarrow \ldots$ //update S according to Proposition 9
end Function
function SpaceCost$S$, V, E
    return... //calculate the space cost sc(S)
end Function

capture all the information that is needed to reproduce the data. However, an exact reproduction for the approximated results is impossible. This is because the algorithm is multithreaded and the tasks that are sent to the threads communicate intermediate results between each other, which change how the tasks continue their execution. While this is in theory deterministic, in practice it depends on how the operating system schedules the threads and what the CPU is doing.

The probabilistic acceptance function that we used for the approximated space-optimal schedules is given by

$$p_{\text{accept}}(\Delta, V, M) = \Theta(|\Delta|)|V|^2 \mathrm{e}^{-\frac{|V||V\setminus M|}{|\Delta|^3(|M|)}} \tag{4}$$

where V are the vertices in the graph, M are the vertices that have been measured so far (including the ones in the currently picked measurement set), Δ is the difference between the best memory and the required memory so far, and Θ is the Heaviside step function (being 1 for positive elements). The step function ensures that we only focus on space-optimal schedules. For finding schedules somewhere between space optimal and time optimal, i.e., a full space–time optimization, one can choose a smoother function. The exponential decay ensures that the run-time is sufficiently small for our experiments, focusing on schedules with a rather large space improvement. In a real application, e.g., a real quantum compiler toolchain, one may choose a more optimized function, fitting to the required needs, and optimize its parameters. The new potential measurement sets M_i are greedily chosen, i.e., the larger sets are tried first.

The probabilistic searches have a timeout, which increases quadratically with the number of qubits. The effect of that can be seen in Fig. 5(b), where the run-time drops down onto a quadratic curve for larger number of qubits. Note that the behavior of the according curves in Fig. 4 indicates that the implied penalty on the costs is not too high.

The standard deviations, which we do not show for simplicity, on the plotted results are relatively high. This is an artifact of combinatorics. For a small number of vertices, the standard deviation is just relatively high, and for a larger number of vertices it becomes a sampling issue. Because of that, we emphasize again that the results are rather qualitative indications than quantitative benchmarks.

In Fig. 5, we see the average run-time for the calculations. This is not a benchmark but only shows the recorded run-times for our calculations. The calculations were performed on a cluster where the vertices were under different loads; therefore, these plots are only qualitative indications. We see that the approximate optimization runs much faster than the exact search for larger graphs. In general, the run-time heavily depends on the chosen acceptance function (4). For small correction densities, the run-time drastically increases, because the number of possible measurement steps, per step, scales according to the ordered Bell numbers.

APPENDIX B
RELATION TO OTHER SOFTWARE TOOLS

Pauli tracking is not a novel idea, and there are already existing software tools that provide similar functionality, the most notable one probably being Stim [16]. Stim is a stabilizer simulator that internally tracks Pauli operators to simulate error models. A subproject of Stim also provides a graphical application to visualize how Pauli errors propagate through a Clifford circuit. Our library does not try to directly compete with these existing projects w.r.t. their Pauli tracking functionality in terms of performance. Instead we want to provide flexible, low-level functionality, which focuses solely on the Pauli tracking, which can be integrated easily into multiple other projects. The goal is to provide an API that allows for simple integration with minimal overhead.

In terms of feature comparison, our library provides certain key functionalities that are, to the best of the authors' knowledge, lacking in other libraries. Particularly, our library allows one to dynamically add and remove frames and qubits. Furthermore, it allows specialized functionality for MBQC, such as relocating Pauli errors from one qubit to another. This functionality is crucial for allowing optimization techniques in MBQC scheduling. Furthermore, our library allows the choosing of data structures specialized for the task at hand. For example, the user can choose whether the underlying data structure for the tracking mechanism is based on a hash map or a vector; while the former is more efficient when many measurements are involved, the latter is more efficient on gate actions.

REFERENCES

[1] T. E. Campbell and M. Howard, "Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost," *Phys. Rev. A*, vol. 95, 2017, Art. no. 022316, doi: 10.1103/PhysRevA.95.022316.

[2] R. Raussendorf and H. J. Briegel, "A one-way quantum computer," *Phys. Rev. Lett.*, vol. 86, pp. 5188–5191, 2001, doi: 10.1103/PhysRevLett.86.5188.

[3] V. Danos, E. Kashefi, and P. Panangaden, "The measurement calculus," *J. ACM*, vol. 54, pp. 8–es, 2007, doi: 10.1145/1219092.1219096.

[4] E. Knill, "Quantum computing with realistically noisy devices," *Nature*, vol. 434, pp. 39–44, 2005, doi: 10.1038/nature03350.

[5] A. Paler, S. Devitt, K. Nemoto, and I. Polian, "Software-based Pauli tracking in fault-tolerant quantum circuits," in *Proc. Des., Autom. Test Europe Conf. Exhib.*, 2014, pp. 1–4, doi: 10.7873/DATE.2014.137.

[6] L. Riesebos, X. Fu, S. Varsamopoulos, C. G. Almudever, and K. Bertels, "Pauli frames for quantum computer architectures," in *Proc. 54th Annu. Des. Autom. Conf.*, 2017, pp. 1–6, doi: 10.1145/3061639.3062300.

[7] J.-H. On, C.-Y. Kim, S.-C. Oh, S.-M. Lee, and G.-I. Cha, "A multilayered Pauli tracking architecture for lattice surgery-based logical qubits," *ETRI J.*, vol. 45, pp. 462–478, 2023, doi: 10.4218/etrij.2022-0037.

[8] S. B. Bravyi and A. Yu Kitaev, "Quantum codes on a lattice with boundary," 1998, *arXiv:quant-ph/9811052*, doi: 10.48550/arXiv.quant-ph/9811052.

[9] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," *Phys. Rev. A*, vol. 86, 2012, Art. no. 032324, doi: 10.1103/PhysRevA.86.032324.

[10] D. Gottesman and I. L. Chuang, "Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations," *Nature*, vol. 402, pp. 390–393, 1999, doi: 10.1038/46503.

[11] X. Zhou, D. W. Leung, and I. L. Chuang, "Methodology for quantum logic gate construction," *Phys. Rev. A*, vol. 62, 2000, Art. no. 052316, doi: 10.1103/PhysRevA.62.052316.

[12] B. Bolt, T. G. Room, and G. E. Wall, "On the Clifford collineation, transform and similarity groups. I," *J. Australian Math. Soc.*, vol. 2, pp. 60–79, 1961, doi: 10.1017/S1446788700026379.

[13] B. Bolt, T. G. Room, and G. E. Wall, "On the Clifford collineation, transform and similarity groups. II," *J. Australian Math. Soc.*, vol. 2, pp. 80–96, 1961, doi: 10.1017/S1446788700026380.

[14] B. Bolt, "On the Clifford collineation, transform and similarity groups. (III) generators and involutions," *J. Australian Math. Soc.*, vol. 2, pp. 334–344, 1962, doi: 10.1017/S1446788700026926.

[15] Pauli-Tracker contributors, "Pauli tracker: A library to track Pauli gates through Clifford circuits," 2024. [Online]. Available: https://github.com/taeruh/pauli_tracker

[16] C. Gidney, "Stim: A fast stabilizer circuit simulator," *Quantum*, vol. 5, 2021, Art. no. 497, doi: 10.22331/q-2021-07-06-497.

[17] M. K. Vijayan et al., "Compilation of algorithm-specific graph states for quantum circuits," *Quantum Sci. Technol.*, vol. 9, 2024, Art. no. 025005, doi: 10.1088/2058-9565/ad1f39.

[18] A. Robertson et al., "Pandora-Rottnest: A fault-tolerant quantum compiler and performance analytics platform," 2025. [Online]. Available: https://github.com/QSI-BAQS/rottnest

[19] Cabaliser contributors, "Cabaliser: A graph state compiler written in C," 2024. [Online]. Available: https://github.com/Alan-Robertson/cabaliser

[20] J. Samuel, J. Elman Gavriel, and R. L. Mann, "Optimal scheduling of graph states via path decompositions," *Phys. Rev. A*, 2025, vol. 111, no. 1, Art. no. 012627, doi: 10.1103/PhysRevA.111.012627.

[21] Pandora contributors, "Pandora: Ultra-large-scale quantum circuit design automation," 2025. [Online]. Available: https://github.com/ioanamoflic/pandora

[22] S. N. Saadatmand et al., "Superconducting qubits at the utility scale: The potential and limitations of modularity," 2025, *arXiv:2406.06015*, doi: 10.48550/arXiv.2406.06015.

[23] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge, U.K.: Cambridge Univ. Press, 2010, doi: 10.1017/CBO9780511976667.

[24] D. Gottesman, "The Heisenberg representation of quantum computers," 1998, *arXiv:quant-ph/9807006*, doi: 10.48550/arXiv.quant-ph/9807006.

[25] P. Selinger, "Generators and relations for n-qubit Clifford operators," *Log. Methods Comput. Sci.*, vol. 11, no. 2, 2015, doi: 10.2168/lmcs-11(2:10)2015.

[26] Pauli-Tracker contributors, "Supported Clifford gates in the library at the time of writing," 2024. [Online]. Available: https://github.com/taeruh/pauli_tracker/blob/372c536d2dab68e2a884e1e1a8a5cfc597245018/docs/conjugation_rules.pdf

[27] D. Gross, "Hudson's theorem for finite-dimensional quantum systems," *J. Math. Phys.*, vol. 47, 2006, Art. no. 122107, doi: 10.1063/1.2393152.

[28] M. R. Kibler, "Variations on a theme of Heisenberg, Pauli and Weyl," *J. Phys. A: Math. Theor.*, vol. 41, 2008, Art. no. 375302, doi: 10.1088/1751-8113/41/37/375302.

[29] J. Tolar, "On Clifford groups in quantum computing," *J. Phys.: Conf. Ser.*, vol. 1071, 2018, Art. no. 012022, doi: 10.1088/1742-6596/1071/1/012022.

[30] Circ contributors, "Circ: An open source framework for programming quantum computers," 2023.

[31] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.

[32] MBQC-Scheduling contributors, "MBQC scheduling: Scheduling in measurement-based quantum computing," 2024. [Online]. Available: https://github.com/taeruh/mbqc_scheduling

[33] S. Sunami and M. Fukushima, "Graphix: Optimizing and simulating measurement-based quantum computation on local-Clifford decorated graph," 2022, *arXiv:2212.11975*, doi: 10.48550/arXiv.2212.11975.

[34] Graphix contributors, "Graphix," 2023, doi: 10.5281/zenodo.7861382.

[35] R. W. Bailey, "The number of weak orderings of a finite set," *Social Choice Welfare*, vol. 15, pp. 559–562, 1998, doi: 10.1007/s003550050123.

[36] J. Spencer, *Ten Lectures on the Probabilistic Method*, 2nd ed. Philadelphia, PA, USA: Soc. Ind. Appl. Math., 1994, doi: 10.1137/1.9781611970074.