Notes on many-body Bell correlations in mixed graph states

Marcin Płodzień[1]

[1] *Qilimanjaro Quantum Tech, Carrer de Veneçuela 74, 08019 Barcelona, Spain*

(Dated: December 2, 2025)

Graph states are a central resource for measurement-based quantum computation, quantum error correction, and distributed entanglement. In realistic Noisy Intermediate-Scale Quantum (NISQ) devices, these states are subject to decoherence and dissipation, which degrade the entanglement structure. In this notes we discuss three algorithmic framework for simulating and optimizing many-body Bell correlations on noisy graph states.

We start with the formalism of (i) explicit density matrix evolution in Hilbert–Schmidt space, next we discuss (ii) vectorized tensor networks in Liouville space, and finally we comment on (iii) Monte Carlo wavefunction trajectories (stochastic unravelings). We describe a concrete implementation built on JAX, exploiting just-in-time (JIT) compilation, automatic differentiation (AD), and parallelized random number generation. These notes provide an introduction to the accompanying numerical code, enabling the optimization of many-body Bell correlators in noisy graph states across multiple graph topologies.

I. INTRODUCTION

In the NISQ era, quantum devices are limited by decoherence, gate errors, and finite qubit numbers. Nevertheless, highly entangled quantum states such as *graph states* remain a key resource for quantum information processing, including measurement-based quantum computation, multi-party communication protocols, and entanglement-based sensing.

In an ideal, noiseless setting, graph states can be described exactly within the stabilizer formalism and manipulated using efficient Clifford-group techniques. However, realistic devices couple to an environment, so pure states evolve into mixed density operators ρ under the influence of decoherence and dissipation. The entanglement content must then be characterized in the presence of noise and imperfections.

To quantify the amount of quantum correlations in mixed graph state we use framework used in Refs.[1–3]. We consider minimization of the loss function parameterized by a vector of measurement angles $\boldsymbol{\theta}$ *Bell-type correlator*

$$\mathcal{L}(\boldsymbol{\theta}) = -\left|\mathrm{Tr}(\rho\,\mathcal{C}(\boldsymbol{\theta}))\right|^2, \tag{1}$$

so that large values of $|\mathrm{Tr}(\rho\mathcal{C})|$ signal strong non-classical correlations. In the code accompanying this article, the optimized correlator is expressed as

$$Q(\boldsymbol{\theta}) = \log_2\Big(2^N \left|\mathrm{Tr}(\rho\,\mathcal{C}(\boldsymbol{\theta}))\right|^2\Big), \tag{2}$$

where N is the number of qubits. The positive value of correlator $Q > 0$ implies Bell correlations, and $Q = N - 2$ corresponds to a maximally entangled GHZ-like state.

Evaluating $Q(\boldsymbol{\theta})$ many times inside a gradient-based optimizer requires repeated simulation of the underlying open-system dynamics. In this work, we describe and compare three complementary numerical architectures:

1. **Exact density matrix (DM):** evolve the full $2^N \times 2^N$ density matrix using tensor reshaping.
2. **Vectorized tensor networks (TN):** represent quantum channels as superoperator tensors in Liouville space and contract along the graph topology.
3. **Monte Carlo wavefunctions (MC):** propagate stochastic pure-state trajectories and reconstruct observables by ensemble averaging.

All three approaches are implemented in JAX using the same set of physical primitives and are embedded into a variational optimization loop using the Adam optimizer. In the following sections we present a coherent mathematical formulation of the three methods, emphasizing their similarities and differences, and we show how the numerical code realizes these ideas efficiently using JAX-specific features such as JIT compilation, `vmap` parallelization, and `lax.scan` loops.

II. PHYSICAL SETUP AND BELL CORRELATOR

A. Hilbert space and notation

We consider N qubits with Hilbert space

$$\mathcal{H} = \bigotimes_{i=1}^{N} \mathbb{C}^2. \tag{3}$$

Computational basis states are written as

$$|\mathbf{z}\rangle = |z_1 z_2 \ldots z_N\rangle, \qquad z_i \in \{0,1\}. \tag{4}$$

The Pauli operators acting on a single qubit are denoted by

$$\begin{aligned} X = \sigma_x &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \\ Y = \sigma_y &= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \\ Z = \sigma_z &= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \\ \mathbb{I} &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \end{aligned} \tag{5}$$

In the code, these operators are represented as JAX arrays of type `complex64` for performance.

B. Graph states from CZ gates

A simple and widely used family of entangled states are *graph states*. Given a graph

$$G = (V, E), \qquad |V| = N, \tag{6}$$

with vertex set V and edge set $E \subset V \times V$, the associated graph state $|G\rangle$ is prepared as follows:

1. Initialize each qubit in the $|+\rangle$ state

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \tag{7}$$

so that the initial product state is

$$|\psi_0\rangle = \bigotimes_{i=1}^{N} |+\rangle_i \,. \tag{8}$$

2. Apply a controlled-phase gate CZ_{uv} between every pair of qubits connected by an edge $(u, v) \in E$. The two-qubit gate is

$$\mathrm{CZ}_{uv} = \mathrm{diag}(1, 1, 1, -1), \tag{9}$$

acting on qubits u and v and as identity elsewhere.

The ideal pure graph state is then

$$|G\rangle = \prod_{(u,v)\in E} \mathrm{CZ}_{uv} \; |\psi_0\rangle \,. \tag{10}$$

In density matrix language, the corresponding pure-state density operator is

$$\rho_{\mathrm{ideal}} = |G\rangle \langle G| \,. \tag{11}$$

In the implementation, the initial single-qubit state $\rho_0 = |+\rangle \langle +|$ is encoded as

$$\rho_0 = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, \tag{12}$$

and tensored across sites as needed.

C. Local measurement basis and variational rotations

To probe Bell-type correlations, we introduce a parameterized family of local measurement operators. For each site i, we choose a three-component parameter vector

$$\boldsymbol{\theta}_i = (\theta_i^z, \theta_i^y, \theta_i^x) \in \mathbb{R}^3, \tag{13}$$

which defines a unit vector

$$\mathbf{n}_i = \frac{1}{\|\boldsymbol{\theta}_i\|} (\theta_i^x, \theta_i^y, \theta_i^z), \tag{14}$$

and a rotation angle

$$\phi_i = \frac{\pi}{2} \, \|\boldsymbol{\theta}_i\|. \tag{15}$$

Using these, the single-qubit rotation implemented in the code is

$$\begin{aligned} R(\boldsymbol{\theta}_i) &= \exp(-i \, \phi_i \, \mathbf{n}_i \cdot \boldsymbol{\sigma}) \\ &= \cos\phi_i \, \mathbb{I} - i \, \sin\phi_i \, (n_i^x X + n_i^y Y + n_i^z Z) \,, \end{aligned} \tag{16}$$

where $\boldsymbol{\sigma} = (X, Y, Z)$. This is an arbitrary SU(2) rotation parameterized by $\boldsymbol{\theta}_i$.

Next, we define a family of single-qubit *measurement operators* $S_\nu^{(+)}$ indexed by an integer code $\nu \in \{0, 1, 2\}$. In the implementation,

$$\begin{aligned} S_0^{(+)} &= \frac{1}{2} \, (X + iY) \,, \\ S_1^{(+)} &= \frac{1}{2} \, (Y + iZ) \,, \\ S_2^{(+)} &= \frac{1}{2} \, (Z + iX) \,, \end{aligned} \tag{17}$$

and we choose a code $\sigma_i \in \{0, 1, 2\}$ for each qubit i. These can be viewed as generalized "raising" operators along different non-orthogonal axes.

The rotated local observable at site i is

$$O_i(\boldsymbol{\theta}_i, \sigma_i) = R(\boldsymbol{\theta}_i)^\dagger \, S_{\sigma_i}^{(+)} \, R(\boldsymbol{\theta}_i). \tag{18}$$

The code implements this via the function `build_local_O_jax`.

D. Many-body correlator and Q-parameter

The global many-body observable is the tensor product

$$\mathcal{C}(\boldsymbol{\theta}, \boldsymbol{\sigma}) = \bigotimes_{i=1}^{N} O_i(\boldsymbol{\theta}_i, \sigma_i). \tag{19}$$

For a given mixed state ρ (resulting from graph state preparation plus noise), the complex correlator is

$$\mathcal{M}(\boldsymbol{\theta}) = \mathrm{Tr}(\rho \, \mathcal{C}(\boldsymbol{\theta}, \boldsymbol{\sigma})) \,. \tag{20}$$

The optimization objective used in all three numerical methods is

$$\mathcal{L}(\boldsymbol{\theta}) = -\left|\mathcal{M}(\boldsymbol{\theta})\right|^2. \tag{21}$$

The sign is chosen so that maximizing the absolute correlator is equivalent to minimizing $\mathcal{L}$.

To obtain a scale-invariant measure that is extensive in the number of qubits N, we define the Q-parameter as

$$Q(\boldsymbol{\theta}) = \log_2\left(2^N \left|\mathcal{M}(\boldsymbol{\theta})\right|^2\right). \tag{22}$$

Within the specific Bell-type construction implemented here, product (classical) states satisfy $Q \leq 0$, while certain GHZ-like states saturate $Q = N-2$. In numerical practice, the optimizer returns a near-optimal value of Q together with the corresponding angles $\boldsymbol{\theta}$.

III. NOISE CHANNELS AND OPEN QUANTUM DYNAMICS

A. CPTP maps and Kraus operators

We model environmental couplings by completely positive trace-preserving (CPTP) maps $\mathcal{E}$ acting on single qubits or, more generally, on subsets of qubits. For a single qubit, any such channel admits a Kraus representation

$$\begin{aligned} \mathcal{E}(\rho) &= \sum_k K_k \rho K_k^\dagger, \\ \sum_k K_k^\dagger K_k &= \mathbb{I}, \end{aligned} \tag{23}$$

where $\{K_k\}$ are 2×2 matrices called Kraus operators. The trace-preservation condition is encoded in the completeness relation $\sum_k K_k^\dagger K_k = \mathbb{I}$.

In the code, several single-qubit noise models are implemented: depolarizing, bit flip, phase flip / phase damping, and amplitude damping. These are applied independently to specific qubits after each two-qubit CZ gate.

B. Standard unital channels

The following channels are *unital*, meaning $\mathcal{E}(\mathbb{I}) = \mathbb{I}$.

1. Depolarizing channel

The depolarizing channel replaces the current state with the maximally mixed state with probability p, and leaves it unchanged with probability $1-p$. The Kraus operators are

$$\begin{aligned} K_0 &= \sqrt{1-p}\,\mathbb{I}, \\ K_1 &= \sqrt{\frac{p}{3}}\,X, \\ K_2 &= \sqrt{\frac{p}{3}}\,Y, \\ K_3 &= \sqrt{\frac{p}{3}}\,Z. \end{aligned} \tag{24}$$

One checks that

$$\begin{aligned} \sum_{k=0}^{3} K_k^\dagger K_k &= (1-p)\,\mathbb{I} + \frac{p}{3}(X^\dagger X + Y^\dagger Y + Z^\dagger Z) \\ &= (1-p)\,\mathbb{I} + \frac{p}{3}(3\mathbb{I}) \\ &= \mathbb{I}. \end{aligned} \tag{25}$$

2. Bit flip and phase flip

The bit-flip channel flips $|0\rangle \leftrightarrow |1\rangle$ with probability p:

$$\begin{aligned} K_0 &= \sqrt{1-p}\,\mathbb{I}, \\ K_1 &= \sqrt{p}\,X. \end{aligned} \tag{26}$$

The phase-flip channel flips the phase of $|1\rangle$ with probability p:

$$\begin{aligned} K_0 &= \sqrt{1-p}\,\mathbb{I}, \\ K_1 &= \sqrt{p}\,Z. \end{aligned} \tag{27}$$

In many physical implementations, phase damping (pure dephasing) and phase flip are equivalent at the level of single-qubit Kraus operators, differing mainly in their time dependence and microscopic origin.

C. Non-unital amplitude damping

Amplitude damping describes irreversible energy relaxation from the excited state $|1\rangle$ to the ground state $|0\rangle$ (e.g. spontaneous emission). It is a standard example of a *non-unital* channel, where $\mathcal{E}(\mathbb{I}) \neq \mathbb{I}$.

The code implements the amplitude damping channel with Kraus operators

$$\begin{aligned} K_0 &= \begin{pmatrix} 1 & 0 \\ 0 & \sqrt{1-p} \end{pmatrix}, \\ K_1 &= \begin{pmatrix} 0 & \sqrt{p} \\ 0 & 0 \end{pmatrix}. \end{aligned} \tag{28}$$

Here, p is the probability that the excited state decays during a discrete time interval Δt. One verifies trace preservation:

$$\begin{aligned} K_0^\dagger K_0 + K_1^\dagger K_1 &= \begin{pmatrix} 1 & 0 \\ 0 & 1-p \end{pmatrix} + \begin{pmatrix} p & 0 \\ 0 & 0 \end{pmatrix} \\ &= \mathbb{I}. \end{aligned} \tag{29}$$

However, the action on the maximally mixed state is

$$\begin{aligned}\mathcal{E}(\mathbb{I}) &= K_0K_0^\dagger + K_1K_1^\dagger \\ &= \begin{pmatrix}1 & 0\\ 0 & 1-p\end{pmatrix} + \begin{pmatrix}p & 0\\ 0 & 0\end{pmatrix} \\ &= \begin{pmatrix}1+p & 0\\ 0 & 1-p\end{pmatrix} \neq \mathbb{I},\end{aligned} \tag{30}$$

so the channel is manifestly non-unital.

D. Relation to Lindblad dynamics

Continuous-time Markovian evolution of an open quantum system is described by the Lindblad master equation

$$\frac{d\rho}{dt} = \mathcal{L}(\rho) = -i[H,\rho] + \sum_\alpha \gamma_\alpha \left(L_\alpha \rho L_\alpha^\dagger - \frac{1}{2}\{L_\alpha^\dagger L_\alpha, \rho\}\right). \tag{31}$$

For a small time step Δt, the formal solution can be written as

$$\rho(t+\Delta t) = \rho(t) + \Delta t\, \mathcal{L}(\rho(t)) + \mathcal{O}(\Delta t^2). \tag{32}$$

To first order in Δt, this evolution can be expressed as a Kraus map

$$\rho(t+\Delta t) = \sum_\mu K_\mu \rho(t) K_\mu^\dagger + \mathcal{O}(\Delta t^2), \tag{33}$$

with Kraus operators

$$A := iH + \frac{1}{2}\sum_\alpha \gamma_\alpha L_\alpha^\dagger L_\alpha, \tag{34a}$$

$$K_0 = \mathbb{I} - \Delta t\, A, \tag{34b}$$

$$K_\alpha = \sqrt{\gamma_\alpha \Delta t}\, L_\alpha. \tag{34c}$$

Inserting Eqs. (34) into Eq. (33) and expanding to first order in Δt gives

$$\begin{aligned}\rho(t+\Delta t) &= K_0\rho K_0^\dagger + \sum_\alpha K_\alpha \rho K_\alpha^\dagger + \mathcal{O}(\Delta t^2) \\ &= \rho - \Delta t\,(A\rho + \rho A^\dagger) + \Delta t \sum_\alpha \gamma_\alpha L_\alpha \rho L_\alpha^\dagger + \mathcal{O}(\Delta t^2).\end{aligned} \tag{35}$$

Using

$$A\rho + \rho A^\dagger = i[H,\rho] + \frac{1}{2}\sum_\alpha \gamma_\alpha \{L_\alpha^\dagger L_\alpha, \rho\}, \tag{36}$$

we obtain

$$\rho(t+\Delta t) = \rho + \Delta t\Big(-i[H,\rho] + \sum_\alpha \gamma_\alpha \big(L_\alpha \rho L_\alpha^\dagger - \tfrac{1}{2}\{L_\alpha^\dagger L_\alpha, \rho\}\big)\Big) + \mathcal{O}(\Delta t^2) = \rho + \Delta t\, \mathcal{L}(\rho) + \mathcal{O}(\Delta t^2), \tag{37}$$

which reproduces the Lindblad generator in Eq. (31) to first order in Δt.

Trace preservation is guaranteed up to $\mathcal{O}(\Delta t^2)$ by the condition

$$\sum_\mu K_\mu^\dagger K_\mu = \mathbb{I} + \mathcal{O}(\Delta t^2). \tag{38}$$

Indeed, from Eqs. (34) we find

$$\begin{aligned}K_0^\dagger K_0 &= (\mathbb{I} - \Delta t A^\dagger)(\mathbb{I} - \Delta t A) \\ &= \mathbb{I} - \Delta t\,(A + A^\dagger) + \mathcal{O}(\Delta t^2) \\ &= \mathbb{I} - \Delta t \sum_\alpha \gamma_\alpha L_\alpha^\dagger L_\alpha + \mathcal{O}(\Delta t^2),\end{aligned} \tag{39}$$

and

$$\sum_\alpha K_\alpha^\dagger K_\alpha = \sum_\alpha \gamma_\alpha \Delta t\, L_\alpha^\dagger L_\alpha. \tag{40}$$

Thus,

$$\sum_\mu K_\mu^\dagger K_\mu = K_0^\dagger K_0 + \sum_\alpha K_\alpha^\dagger K_\alpha = \mathbb{I} + \mathcal{O}(\Delta t^2), \tag{41}$$

showing that, for sufficiently small Δt, the discrete-time Kraus representation approximates the continuous Lindblad evolution while remaining approximately completely positive and trace preserving.

IV. METHOD I: EXACT DENSITY MATRIX EVOLUTION

A. Hilbert–Schmidt representation

In the exact density matrix (DM) approach, the system state is represented by the full density operator

$$\begin{aligned}&\rho \in \mathbb{C}^{2^N \times 2^N}, \qquad \rho^\dagger = \rho, \\ &\rho \geq 0, \qquad \mathrm{Tr}(\rho) = 1.\end{aligned} \tag{42}$$

Unitaries and channels act via

$$\begin{aligned}&\rho \mapsto U\rho U^\dagger, \\ &\rho \mapsto \mathcal{E}(\rho) = \sum_k K_k \rho K_k^\dagger.\end{aligned} \tag{43}$$

A straightforward implementation would construct the full $2^N \times 2^N$ matrix corresponding to each multi-qubit gate, but this quickly becomes prohibitively expensive.

B. Tensor reshaping and local updates

The code avoids constructing large Kronecker products by reshaping ρ into a $2N$-index tensor. For N qubits, we identify

$$\rho_{i_1 \ldots i_N, j_1 \ldots j_N} \equiv \langle i_1 \ldots i_N | \, \rho \, | j_1 \ldots j_N \rangle \,, \tag{44}$$

with each $i_\ell, j_\ell \in \{0, 1\}$. In array notation, this corresponds to the shape

$$\rho \in \mathbb{C}^{2 \times 2 \times \cdots \times 2 \times (2^N)}, \tag{45}$$

where the "bra" indices are sometimes grouped into a single dimension.

1. Single-qubit gate

Consider a single-qubit unitary U acting on site s. Its action on ρ is

$$\rho' = U_s \rho U_s^\dagger. \tag{46}$$

In index notation, this is

$$\rho'_{i_1 \ldots i_N, j_1 \ldots j_N} = \sum_{a_s, b_s} U_{i_s a_s} \rho_{i_1 \ldots a_s \ldots i_N, j_1 \ldots b_s \ldots j_N} U^*_{j_s b_s}. \tag{47}$$

The implementation performs this operation in two steps:

1. Left multiplication:

$$\rho^{(L)}_{i_1 \ldots i_N, j_1 \ldots j_N} = \sum_{a_s} U_{i_s a_s} \rho_{i_1 \ldots a_s \ldots i_N, j_1 \ldots j_N}. \tag{48}$$

   This is realized via `jnp.tensordot(U, rho_tens, axes=[[1],[s]])` after reshaping.

2. Right multiplication:

$$\rho'_{i_1 \ldots i_N, j_1 \ldots j_N} = \sum_{b_s} \rho^{(L)}_{i_1 \ldots i_N, j_1 \ldots b_s \ldots j_N} U^*_{j_s b_s}. \tag{49}$$

   This is implemented by transposing $\rho^{(L)}$ to exchange bra and ket indices, applying the same left-multiplication logic with $U^\dagger$, and transposing back.

2. Two-qubit gate

For a two-qubit gate G acting on sites i and j, the unitary is a 4×4 matrix (e.g. CZ) reshaped to a rank-4 tensor

$$G_{a_i a_j, b_i b_j} \equiv G_{(2a_i + a_j), (2b_i + b_j)}. \tag{50}$$

The code moves the active indices to the front, reshapes, applies the gate, and restores the original ordering. Formally,

$$\rho'_{i_1 \ldots i_N, j_1 \ldots j_N} = \sum_{a_i, a_j, b_i, b_j} G_{i_i i_j, a_i a_j} \rho_{i_1 \ldots a_i \ldots a_j \ldots i_N, j_1 \ldots b_i \ldots b_j \ldots j_N} \times G^*_{j_i j_j, b_i b_j}. \tag{51}$$

Again, the code performs a left multiplication with G, reorganizes indices, then a right multiplication with $G^\dagger$ using the transpose trick.

C. Incorporating Kraus noise

Consider a noise channel $\mathcal{E}$ with Kraus operators $\{K_\mu\}$ acting on a single qubit s. The updated state is

$$\rho' = \sum_\mu K_{\mu,s} \rho K^\dagger_{\mu,s}. \tag{52}$$

In the implementation, for each μ we compute

$$\rho_\mu = \mathrm{Apply}(K_\mu, \rho), \tag{53}$$

using the same tensor reshaping logic as for unitaries, and accumulate

$$\rho' = \sum_\mu \rho_\mu. \tag{54}$$

This pattern is applied to both endpoints of each edge after a CZ gate, realizing a sequence

$$\rho \xrightarrow{\mathrm{CZ}_{uv}} \rho' \xrightarrow{\mathcal{E}_u} \rho'' \xrightarrow{\mathcal{E}_v} \rho'''. \tag{55}$$

D. Expectation value and loss function

Once the final noisy density matrix ρ is obtained, the many-body observable matrix

$$C_{\mathrm{mat}}(\boldsymbol{\theta}) = \bigotimes_{i=1}^{N} O_i(\boldsymbol{\theta}_i, \sigma_i), \tag{56}$$

is constructed explicitly as a $2^N \times 2^N$ operator via Kronecker products. The correlator is

$$\mathcal{M}(\boldsymbol{\theta}) = \mathrm{Tr}(\rho \, C_{\mathrm{mat}}(\boldsymbol{\theta})) \,, \tag{57}$$

and the loss used by the optimizer is

$$\mathcal{L}_{\mathrm{DM}}(\boldsymbol{\theta}) = - \left| \mathcal{M}(\boldsymbol{\theta}) \right|^2 . \tag{58}$$

Computationally, this method requires $\mathcal{O}(4^N)$ memory and similar time complexity, limiting its applicability to roughly $N \lesssim 10$–12 qubits on typical hardware. It is, however, exact and serves as a high-fidelity reference for the other methods.

V. METHOD II: VECTORIZED TENSOR NETWORKS IN LIOUVILLE SPACE

A. Vectorization and Liouville space

Instead of working with ρ as a $2^N \times 2^N$ matrix, we can "flatten" it into a vector in a larger Hilbert space. Define the vectorization map

$$\begin{aligned} \mathrm{vec} : \mathcal{L}(\mathcal{H}) &\to \mathcal{H} \otimes \mathcal{H}, \\ |i\rangle\langle j| &\mapsto |i\rangle \otimes |j\rangle . \end{aligned} \tag{59}$$

For a general operator

$$X = \sum_{ij} X_{ij} |i\rangle\langle j| , \tag{60}$$

we define the double-ket

$$|\rho\rangle\rangle = \mathrm{vec}(\rho) = \sum_{ij} \rho_{ij} |i\rangle \otimes |j\rangle . \tag{61}$$

This is sometimes called Liouville space or the Hilbert–Schmidt space of operators.

A crucial identity is

$$\mathrm{vec}(AXB) = (A \otimes B^T)\,\mathrm{vec}(X), \tag{62}$$

for any matrices A, B, X of compatible size. This allows us to represent linear maps on density matrices as ordinary operators on the vectorized state. To see this directly, let $X = |m\rangle\langle n|$ be a rank-one operator. Then

$$AXB = A|m\rangle\langle n|B = |\tilde{m}\rangle\langle\tilde{n}| , \tag{63}$$

where $|\tilde{m}\rangle = A|m\rangle$ and $\langle\tilde{n}| = \langle n|B$. Now, by vectorization, we get

$$\begin{aligned} \mathrm{vec}(AXB) &= \mathrm{vec}(|\tilde{m}\rangle\langle\tilde{n}|) = |\tilde{m}\rangle \otimes |\tilde{n}\rangle \\ &= (A|m\rangle) \otimes (B^T|n\rangle) \\ &= (A \otimes B^T)\,(|m\rangle \otimes |n\rangle) \\ &= (A \otimes B^T)\,\mathrm{vec}(X), \end{aligned} \tag{64}$$

and by linearity, the identity holds for arbitrary X.

1. Choi–Jamiołkowski isomorphism and channel–state duality

The vectorization map used above is closely related to the *Choi–Jamiołkowski isomorphism*, which establishes a one-to-one correspondence between linear maps on operators and bipartite operators (often called *Choi states*).

Let $\mathcal{E} : \mathcal{B}(\mathcal{H}) \to \mathcal{B}(\mathcal{H})$ be a linear map on d-dimensional operators ($\dim \mathcal{H} = d$). Define the maximally entangled state

$$|\Phi^+\rangle = \frac{1}{\sqrt{d}} \sum_{i=0}^{d-1} |i\rangle \otimes |i\rangle , \tag{65}$$

and its projector

$$\Phi^+ = |\Phi^+\rangle\langle\Phi^+| . \tag{66}$$

The *Choi operator* (or Choi matrix) of $\mathcal{E}$ is defined as

$$J(\mathcal{E}) = (\mathcal{E} \otimes \mathbb{I})(\Phi^+) \in \mathcal{B}(\mathcal{H} \otimes \mathcal{H}), \tag{67}$$

where $\mathbb{I}$ is the identity map on $\mathcal{B}(\mathcal{H})$.

a. Choi operator in terms of Kraus operators. If $\mathcal{E}$ has a Kraus representation

$$\mathcal{E}(X) = \sum_{\mu} K_\mu X K_\mu^\dagger, \tag{68}$$

then

$$J(\mathcal{E}) = \sum_{\mu} (K_\mu \otimes \mathbb{I})\,\Phi^+\,(K_\mu^\dagger \otimes \mathbb{I}). \tag{69}$$

Using the identity

$$(K \otimes \mathbb{I})\,|\Phi^+\rangle = \frac{1}{\sqrt{d}} \sum_{i,j} K_{ij} |i\rangle \otimes |j\rangle = \frac{1}{\sqrt{d}}\,\mathrm{vec}(K), \tag{70}$$

we may write

$$J(\mathcal{E}) = \frac{1}{d} \sum_{\mu} \|K_\mu\rangle\rangle\langle\langle K_\mu\| , \tag{71}$$

where we introduced the "double-ket" notation

$$\|K_\mu\rangle\rangle = \mathrm{vec}(K_\mu). \tag{72}$$

Thus $J(\mathcal{E})$ is a positive semidefinite operator on $\mathcal{H} \otimes \mathcal{H}$ built from the stacked Kraus operators.

b. Complete positivity and trace preservation. The Choi–Jamiołkowski isomorphism has two fundamental structural properties:

- $\mathcal{E}$ is *completely positive* (CP) if and only if $J(\mathcal{E}) \geq 0$ as an operator on $\mathcal{H} \otimes \mathcal{H}$.
- $\mathcal{E}$ is *trace-preserving* (TP) if and only if

$$\mathrm{Tr}_1\big[J(\mathcal{E})\big] = \frac{1}{d}\mathbb{I}, \tag{73}$$

  where Tr_1 denotes the partial trace over the first factor of $\mathcal{H} \otimes \mathcal{H}$. For trace-*non*-increasing maps, one has $\mathrm{Tr}_1[J(\mathcal{E})] \leq \mathbb{I}/d$.

This means that, from the tensor-network point of view, encoding a channel as a rank-4 tensor that is positive and satisfies a suitable partial trace condition is equivalent to ensuring CP and TP of the underlying physical map.

c. Recovering the action of the channel from the Choi operator. Conversely, given $J(\mathcal{E})$, the action of $\mathcal{E}$ on an arbitrary operator X can be recovered via

$$\mathcal{E}(X) = \mathrm{Tr}_2\big[(\mathbb{I} \otimes X^{\mathsf{T}})\,J(\mathcal{E})\big], \tag{74}$$

where X^{T} denotes the transpose in the computational basis and Tr_2 is the partial trace over the second tensor factor. This formula is the operator analogue of

$$\mathrm{vec}\big(\mathcal{E}(X)\big) = \hat{\mathcal{S}}\,\mathrm{vec}(X), \tag{75}$$

with the superoperator $\hat{\mathcal{S}}$ introduced in previous section.

d. Relation to the superoperator tensor in the code. In the implementation we define the rank-4 tensor

$$\hat{\mathcal{S}}_{acbd} = \sum_{\mu} (K_\mu)_{ab} (K_\mu^*)_{cd}, \tag{76}$$

and use it to propagate the vectorized density matrix via

$$\mathrm{vec}\big(\mathcal{E}(\rho)\big) = \hat{\mathcal{S}}\,\mathrm{vec}(\rho). \tag{77}$$

If we group the indices as (ac) and (bd), then

$$\hat{\mathcal{S}}_{(ac),(bd)} = \sum_{\mu} (K_\mu)_{ab} (K_\mu^*)_{cd}. \tag{78}$$

Up to a simple permutation of indices and (optionally) an overall factor $1/d$, this is exactly the Choi matrix $J(\mathcal{E})$ written in components. More precisely, if we define

$$J_{(ab),(cd)}(\mathcal{E}) = \frac{1}{d} \sum_{\mu} (K_\mu)_{ac} (K_\mu^*)_{bd}, \tag{79}$$

then the difference between J and $\hat{\mathcal{S}}$ is only the choice of which pair of indices we regard as "input" and which as "output". Diagrammatically, both objects are represented by a rank-4 tensor with two incoming and two outgoing legs; in the code, we choose the particular layout that is most convenient for contracting with $\mathrm{vec}(\rho)$.

In the tensor-network diagrams used in this work, the Choi operator (or the equivalent superoperator tensor $\hat{\mathcal{S}}$) is exactly the "noise box" planted on each site. Its four indices connect:

- the incoming ket and bra legs from the previous layer, and
- the outgoing ket and bra legs to the next layer.

The CP condition $J(\mathcal{E}) \geq 0$ ensures that, regardless of what is connected to these legs, the resulting map is physically valid; the TP condition ensures that probabilities remain correctly normalized.

Thus, the TN method can be interpreted as contracting a network of Choi operators (one for each noisy location) together with unitary tensors and boundary conditions (initial states and measurement operators). The code's construction of $\hat{\mathcal{S}}$ via

$$\hat{\mathcal{S}} = \sum_{\mu} K_\mu \otimes K_\mu^* \tag{80}$$

is precisely the computational realization of this Choi–Jamiołkowski representation.

B. Superoperators for unitaries and channels

A unitary update $\rho \mapsto U\rho U^\dagger$ corresponds, under vectorization, to

$$|\rho\rangle\rangle \mapsto (U \otimes U^*)\,|\rho\rangle\rangle, \tag{81}$$

where $*$ denotes complex conjugation. Similarly, a Kraus channel

$$\mathcal{E}(\rho) = \sum_{k} K_k \rho K_k^\dagger, \tag{82}$$

acts as

$$\begin{aligned} |\rho\rangle\rangle &\mapsto \hat{\mathcal{S}}\,|\rho\rangle\rangle, \\ \hat{\mathcal{S}} &= \sum_{k} K_k \otimes K_k^*. \end{aligned} \tag{83}$$

In index notation, the corresponding rank-4 tensor has components

$$\hat{\mathcal{S}}_{ac,bd} = \sum_{k} (K_k)_{ab}\,(K_k^*)_{cd}, \tag{84}$$

which the code stores as

$$\hat{\mathcal{S}}_{acbd} = \sum_{k} (K_k)_{ab} (K_k^*)_{cd} \tag{85}$$

with index ordering (a, c, b, d).

C. Tensor network representation on the graph

In the tensor network (TN) method, the preparation of ρ is represented as a network of tensors corresponding to:

- Initial single-qubit density matrices ρ_0 (rank-2).
- Two-qubit gates CZ and their conjugates (rank-4).
- Noise superoperators $\hat{\mathcal{S}}_{\mathrm{noise}}$ (rank-4).
- Measurement operators $O_i(\boldsymbol{\theta}_i, \sigma_i)$ (rank-2).

Each tensor has indices corresponding to outgoing and incoming legs on the "ket" and "bra" rails.

The code constructs the entire network programmatically in `build_tn_contraction`. For each qubit, it creates an initial density matrix tensor. For each edge (u, v), it inserts:

- A CZ tensor on the ket legs of u and v.
- A CZ tensor on the bra legs (complex conjugate).
- A noise superoperator on site u.
- A noise superoperator on site v.

At the end, for each site i, it attaches a measurement tensor $O_i(\boldsymbol{\theta}_i, \sigma_i)$ and then contracts over all indices, yielding a scalar equal to the correlator $\mathcal{M}(\boldsymbol{\theta})$.

D. Contraction with `einsum`

The network contraction is expressed using `jnp.einsum` with an optimized contraction path. Conceptually, we collect all tensors $T^{(\ell)}$ and their index labels $\alpha^{(\ell)}$, and perform

$$\mathcal{M}(\boldsymbol{\theta}) = \sum_{\{\text{internal indices}\}} \prod_{\ell} T^{(\ell)}_{\alpha^{(\ell)}}. \tag{86}$$

JAX delegates the contraction order to an internal greedy optimizer when `optimize='greedy'` is used, greatly improving performance on sparse or low-treewidth graphs. The TN loss function is again

$$\mathcal{L}_{\mathrm{TN}}(\boldsymbol{\theta}) = -\left|\mathcal{M}(\boldsymbol{\theta})\right|^2, \tag{87}$$

with the same definition of Q as in the DM method.

The scaling of this method depends sensitively on the graph topology. For line and star graphs, the contraction width grows mildly with N, whereas for grids and complete graphs, the intermediate tensors can grow exponentially, eventually matching the complexity of the DM approach.

VI. METHOD III: MONTE CARLO WAVEFUNCTION TRAJECTORIES

A. Stochastic unraveling of Kraus maps

The Monte Carlo wavefunction (MCWF) method represents the density matrix as an ensemble average over pure states:

$$\rho = \mathbb{E}\left[|\psi\rangle\langle\psi|\right] \approx \frac{1}{M}\sum_{m=1}^{M} |\psi_m\rangle\langle\psi_m|, \tag{88}$$

where $|\psi_m\rangle$ are stochastic trajectories.

For a channel with Kraus operators $\{K_k\}$, the trajectory update rule is:

1. Given $|\psi\rangle$, compute branch weights

$$p_k = \|K_k|\psi\rangle\|^2. \tag{89}$$

   They automatically satisfy $\sum_k p_k = 1$ due to trace preservation.

2. Sample a random outcome k with probability p_k.

3. Update the state by

$$|\psi'\rangle = \frac{K_k|\psi\rangle}{\sqrt{p_k}}. \tag{90}$$

It is straightforward to verify that the ensemble-averaged density matrix obeys the Kraus map: Let $|\psi\rangle$ be the initial pure state, and define

$$\begin{aligned} p_k &= \|K_k|\psi\rangle\|^2, \\ |\psi'_k\rangle &= \frac{K_k|\psi\rangle}{\sqrt{p_k}}. \end{aligned} \tag{91}$$

The updated pure state is $|\psi'_k\rangle$ with probability p_k. Then the ensemble-averaged density matrix is

$$\begin{aligned} \mathbb{E}\left[|\psi'\rangle\langle\psi'|\right] &= \sum_k p_k |\psi'_k\rangle\langle\psi'_k| \\ &= \sum_k p_k \frac{K_k|\psi\rangle}{\sqrt{p_k}}\frac{\langle\psi|K_k^\dagger}{\sqrt{p_k}} \\ &= \sum_k K_k|\psi\rangle\langle\psi|K_k^\dagger \\ &= \mathcal{E}(|\psi\rangle\langle\psi|), \end{aligned} \tag{92}$$

as desired. By linearity, the same holds for any mixed initial state.

B. Amplitude damping: jump vs no-jump

For amplitude damping with Kraus operators K_0 and K_1 as in Eq. (28), the trajectory update can be written more explicitly. Let

$$\begin{aligned} |\phi_0\rangle &= K_0|\psi\rangle, \\ |\phi_1\rangle &= K_1|\psi\rangle. \end{aligned} \tag{93}$$

The jump (decay) probability is

$$p_{\mathrm{jump}} = \|\phi_1\|^2 = \langle\psi|K_1^\dagger K_1|\psi\rangle, \tag{94}$$

and the no-jump probability is $p_{\mathrm{no}} = 1 - p_{\mathrm{jump}} = \|\phi_0\|^2$.

The stochastic update is:

$$|\psi'\rangle = \begin{cases} \dfrac{|\phi_1\rangle}{\sqrt{p_{\mathrm{jump}}}}, & \text{with prob. } p_{\mathrm{jump}}, \\ \dfrac{|\phi_0\rangle}{\sqrt{1-p_{\mathrm{jump}}}}, & \text{with prob. } 1-p_{\mathrm{jump}}. \end{cases} \tag{95}$$

The second branch is particularly important: even if no quantum jump occurs, the state is renormalized and its excited-state population is reduced on average, correctly encoding the information "no photon was detected."

C. Trajectories for graph states

In the MC method, each trajectory proceeds as follows:

1. Start from the pure product state

$$|\psi_0\rangle = |+\rangle^{\otimes N}. \tag{96}$$

2. For each edge (u, v) of the graph,

(a) Apply the two-qubit CZ gate deterministically:

$$|\psi\rangle \mapsto \mathrm{CZ}_{uv}\,|\psi\rangle\,. \tag{97}$$

(b) Apply the chosen noise channel at qubit u via the stochastic rule described above.

(c) Apply the noise channel at qubit v.

3. At the end of the circuit, obtain a pure state $|\psi_m(\boldsymbol{\theta})\rangle$ for each trajectory m.

To evaluate the correlator for a given trajectory, we apply the measurement operator locally as a unitary (Heisenberg picture vs Schrödinger picture choice) and compute

$$\mathcal{M}_m(\boldsymbol{\theta}) = \langle\psi_m|\,\mathcal{C}(\boldsymbol{\theta},\boldsymbol{\sigma})\,|\psi_m\rangle\,. \tag{98}$$

The MC estimator of the full correlator is then

$$\widehat{\mathcal{M}}(\boldsymbol{\theta}) = \frac{1}{M}\sum_{m=1}^{M}\mathcal{M}_m(\boldsymbol{\theta}), \tag{99}$$

with standard error

$$\mathrm{SEM}(\boldsymbol{\theta}) = \frac{1}{\sqrt{M}}\,\mathrm{Std}[\mathcal{M}_m(\boldsymbol{\theta})]\,. \tag{100}$$

The loss used in the optimizer is again

$$\mathcal{L}_{\mathrm{MC}}(\boldsymbol{\theta}) = -\left|\widehat{\mathcal{M}}(\boldsymbol{\theta})\right|^2, \tag{101}$$

and the resulting Q is computed using the same logarithmic scaling as in the DM and TN methods. The code also estimates the uncertainty in Q from the SEM via simple error propagation.

D. Parallelization in JAX

The MC method is particularly well-suited to GPU or TPU acceleration because individual trajectories are independent. In the implementation, a function

$$\mathrm{run\_traj}(\boldsymbol{\theta}, k) \tag{102}$$

simulates one trajectory given a PRNG key k. A batch of M keys $(k_1,\ldots,k_M)$ is generated and `jax.vmap` is used to evaluate

$$\mathrm{vals} = \mathrm{vmap}\,(k \mapsto \mathrm{run\_traj}(\boldsymbol{\theta}, k))\,(k_1,\ldots,k_M), \tag{103}$$

returning a vector of trajectory correlators $\mathcal{M}_m$.

The entire MC loss and its gradient are then compiled by JAX into a single XLA kernel, which can be efficiently executed on accelerators.

VII. VARIATIONAL OPTIMIZATION AND AUTOMATIC DIFFERENTIATION

A. Unified loss function

All three methods compute (exactly or approximately) the same scalar correlator

$$\mathcal{M}(\boldsymbol{\theta}) = \mathrm{Tr}(\rho\,\mathcal{C}(\boldsymbol{\theta},\boldsymbol{\sigma}))\,, \tag{104}$$

where ρ is the final state representation produced by the DM, TN, or MC solver. The loss function is

$$\mathcal{L}(\boldsymbol{\theta}) = -\,|\mathcal{M}(\boldsymbol{\theta})|^2\,. \tag{105}$$

Differentiating formally, we obtain

$$\nabla_{\boldsymbol{\theta}}\mathcal{L} = -2\,\mathrm{Re}\,[\mathcal{M}^*(\boldsymbol{\theta})\nabla_{\boldsymbol{\theta}}\mathcal{M}(\boldsymbol{\theta})]\,. \tag{106}$$

The derivative $\nabla_{\boldsymbol{\theta}}\mathcal{M}$ involves gradients of the local rotations $R(\boldsymbol{\theta}_i)$ and the resulting observables $O_i(\boldsymbol{\theta}_i,\sigma_i)$. The code leverages JAX's reverse-mode automatic differentiation (`jax.grad` and `jax.value_and_grad`), which symbolically differentiates through the entire computational graph.

B. Adam optimizer and `lax.scan`

Given an initial parameter array $\boldsymbol{\theta}^{(0)}$ of shape $(N,3)$, we iterate the Adam optimizer:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \Delta\boldsymbol{\theta}^{(t)}, \tag{107}$$

where $\Delta\boldsymbol{\theta}^{(t)}$ is the Adam update based on the gradient $\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$ and the internal moment estimates. The step size and hyperparameters are chosen empirically in the code (e.g. learning rate 0.02).

The JAX code uses

$$(\boldsymbol{\theta}^{(\mathrm{final})}, \{\ell^{(t)}\}) = \mathrm{lax.scan}(\mathrm{step\_fn},\, \boldsymbol{\theta}^{(0)},\, \mathrm{range}(\mathrm{steps})), \tag{108}$$

where `step_fn` performs a single optimization step: computes the loss and gradient, updates the Adam state, and advances the PRNG key if needed (MC case). This loop is fully compiled, so the entire optimization routine runs as a single optimized kernel.

C. Randomness and differentiation in the MC case

In the MC method, the loss function depends both on $\boldsymbol{\theta}$ and on the random keys used to generate trajectories. The code defines a loss function

$$\mathcal{L}_{\mathrm{MC}}(\boldsymbol{\theta}, k) = -\left|\widehat{\mathcal{M}}(\boldsymbol{\theta}; k)\right|^2, \tag{109}$$

where k is a PRNG key that deterministically specifies a batch of trajectories. Differentiation is then performed with respect to $\boldsymbol{\theta}$ only:

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\mathrm{MC}} = \texttt{jax.grad\_}\boldsymbol{\theta}\, \mathcal{L}_{\mathrm{MC}}(\boldsymbol{\theta}, k). \tag{110}$$

This is a standard differentiable Monte Carlo estimator: the randomness is treated as fixed for each gradient evaluation, and the gradient flows through the deterministic post-processing $\widehat{\mathcal{M}}(\boldsymbol{\theta}; k)$.

VIII. GRAPH TOPOLOGIES AND NUMERICAL SCALING

A. Graph families

The code includes several graph-generation routines based on NetworkX:

- Star graphs (one central node connected to all others).
- Line / path graphs (1D chains).
- Cycle graphs (rings).
- Complete graphs (K_N).
- Two-dimensional grids (rectangular lattices).
- Turan graphs $T(N, r)$ (complete r-partite graphs with partitions as equal as possible).
- Erdős–Rényi random graphs, both $G(N, p)$ and $G(N, M)$ variants.

Each graph type induces a different pattern of entanglement and different complexity for the tensor network contraction.

B. Complexity comparison

A qualitative summary of the complexity of the three methods is given in Table I.

TABLE I. Qualitative scaling and best-use regime of the three numerical methods.

Method	Memory	Regime
DM	$\mathcal{O}(4^N)$	$N \lesssim 10$–12 (exact reference)
TN	Topology-dependent	Low-treewidth graphs (lines, stars)
MC	$\mathcal{O}(2^N)$ per trajectory	Larger N, arbitrary graphs

The DM method has the simplest conceptual structure but the steepest scaling. The TN method can exploit locality and sparse connectivity for certain graphs. The MC method has the same per-trajectory memory cost as pure-state simulation and trades additional CPU/GPU time (due to multiple trajectories) for lower memory usage.

IX. IMPLEMENTATION DETAILS IN JAX

A. Data types and numerical stability

The implementation uses complex numbers with 32-bit floating-point components:

$$\text{dtype} = \texttt{jnp.complex64}. \tag{111}$$

This choice balances numerical accuracy and device memory usage, and is well suited to GPUs/TPUs. For high-precision physics applications (e.g. when resolving differences at the 10^{-10} level), replacing `complex64` with `complex128` is straightforward but doubles memory usage.

To avoid numerical instabilities such as division by zero or denormalized vectors, the code introduces small regularizers. For example, when defining the rotation axis, it uses

$$\|\boldsymbol{\theta}_i\| \to \sqrt{(\theta_i^x)^2 + (\theta_i^y)^2 + (\theta_i^z)^2} + 10^{-12}, \tag{112}$$

and when normalizing quantum states, it divides by $\|\psi\| + 10^{-12}$.

B. Key functions and their roles

We briefly summarize some of the main functions in the code and their mathematical meaning.

a. ***`single_R_jax(theta_vec)`*** implements the SU(2) rotation $R(\boldsymbol{\theta})$ as in Sec. II.

b. ***`single_sigma_plus_jax(code)`*** returns $S_0^{(+)}$, $S_1^{(+)}$, or $S_2^{(+)}$ depending on a code in $\{0, 1, 2\}$.

c. ***`build_local_O_jax(theta_vec, sigma_code)`*** returns the local observable $O_i(\boldsymbol{\theta}_i, \sigma_i)$.

d. ***`get_dm_kraus(noise_type, p_noise)`*** returns the list of Kraus operators $\{K_\mu\}$ for one of the noise models in Sec. III.

e. ***`make_dm_loss(...)`*** constructs a JIT-compiled function that, given $\boldsymbol{\theta}$, builds the density matrix ρ via DM evolution and returns the loss and its gradient.

f. ***`get_tn_superop(noise_type, p_noise)`*** constructs the rank-4 superoperator tensor $\hat{\mathcal{S}}$ for use in the TN method.

g. ***`make_tn_loss(...)`*** builds the full tensor network for a given graph and returns a JIT-compiled loss/-gradient function using `einsum`.

h. ***`make_mc_noise(...)`*** returns a stochastic update function implementing the chosen noise channel as a trajectory-level operation, using `jax.lax.cond` to maintain JIT compatibility.

i. ***`make_mc_funcs(...)`*** builds a JIT-compiled trajectory statistics function and loss-gradient function for the MC method.

j. ***`run_opt(...)`*** runs the Adam optimizer using `lax.scan` for a fixed number of steps, returning the optimized angles and the final Q.

C. Graph visualization

For diagnostic purposes, the code plots the graph topology using NetworkX and Matplotlib, saving a PNG file with a layout chosen according to the graph type. This helps verify that the intended connectivity (e.g. star, grid, Turan graph) has been correctly instantiated before running expensive simulations.

X. SUMMARY

We have presented a unified theoretical and numerical framework for simulating noisy graph states and optimizing their many-body Bell-type correlators. Starting from a common physical model — graph states prepared from $|+\rangle^{\otimes N}$ via CZ gates and subjected to local CPTP noise channels — we discussed three complementary simulation architectures:

- Exact density matrix evolution with tensor reshaping (Hilbert–Schmidt space).
- Vectorized tensor networks with superoperator tensors (Liouville space).
- Monte Carlo wavefunction trajectories (stochastic unravelings).

All three methods compute the same optimized correlator $Q(\boldsymbol{\theta})$, but with different trade-offs in memory, runtime, and scalability. By implementing them within a single JAX-based codebase, we can cross-check results, select the most appropriate method for a given regime, and leverage automatic differentiation and hardware accelerators to perform variational optimization efficiently.

[1] M. Płodzień, J. Chwedeńczuk, and M. Lewenstein, Inherent quantum resources in stationary spin chains, Phys. Rev. A **111**, 012417 (2025).
[2] M. Płodzień, J. Chwedeńczuk, M. Lewenstein, and G. Rajchel-Mieldzioć, Entanglement classification and nonk-separability certification via greenberger-horne-zeilinger-class fidelity, Physical Review A **110**, 10.1103/physreva.110.032428 (2024).
[3] M. Płodzień, M. Lewenstein, and J. Chwedeńczuk, Many-body quantum resources of graph states, Reports on Progress in Physics **88**, 077601 (2025).