

Line-graph qubit routing: from kagome to heavy-hex and more

Joris Kattemölle¹ and Seenivasan Hariharan^{2,3}

¹*Department of Physics, University of Konstanz, Konstanz, Germany*

²*Institute of Physics, University of Amsterdam, Amsterdam, The Netherlands*

³*QuSoft, CWI, Amsterdam, The Netherlands*

Quantum computers have the potential to outperform classical computers, but are currently limited in their capabilities. One such limitation is the restricted connectivity between qubits, as captured by the hardware’s coupling graph. This limitation poses a challenge for running algorithms that require a coupling graph different from what the hardware can provide. To overcome this challenge and fully utilize the hardware, efficient qubit routing strategies are necessary. In this paper, we introduce line-graph qubit routing, a general method for routing qubits when the algorithm’s coupling graph is a line graph and the hardware coupling graph is a heavy graph. Line-graph qubit routing is fast, deterministic, and effective; it requires a classical computational cost that scales at most quadratically with the number of gates in the original circuit, while producing a circuit with a SWAP overhead of at most two times the number of two-qubit gates in the original circuit. We implement line-graph qubit routing and demonstrate its effectiveness in mapping quantum circuits on kagome, checkerboard, and shuriken lattices to hardware with heavy-hex, heavy-square, and heavy-square-octagon coupling graphs, respectively. Benchmarking shows the ability of line-graph qubit routing to outperform established general-purpose methods, both in the required classical wall-clock time and in the quality of the solution that is found. Line-graph qubit routing has direct applications in the quantum simulation of lattice-based models and aids the exploration of the capabilities of near-term quantum hardware.

I. INTRODUCTION

Quantum computing offers potential to revolutionize a wide range of domains by efficiently solving problems that are intractable for classical computers [1, 2]. To run any quantum algorithm, it must be compiled into a quantum circuit that can be executed on the quantum hardware. The hardware coupling graph of a quantum computer, which defines adjacency between qubits based on the ability to perform two-qubit gates between them, plays a crucial role in this process. The problem of ensuring that a quantum circuit is compatible with the hardware coupling graph is referred to as the qubit routing problem [3, 4]. While generalized methods exist for qubit routing [5], a standard approach to implement two-qubit gates between non-adjacent qubits is to insert SWAP gates, making the qubits effectively adjacent [3, 4, 6–9]. To obtain a practical quantum advantage on noisy intermediate-scale quantum (NISQ) [10] devices, it is imperative that overhead arising from compilation is kept to a minimum [11, 12]. However, finding the swapping strategy that requires the least number of SWAP gates is NP-hard [3, 6], making it a challenging problem to solve. Heuristic, probabilistic methods have been developed, but their classical runtime may become problematic for large circuits, and the solution they find may be far from optimal [4, 7–9]. Striking the right balance between the classical resources required for routing and minimizing the circuit depth of the routed quantum circuit is crucial in maximizing the performance of NISQ devices.

The qubit routing problem is particularly evident in the quantum simulation of lattice-based spin models.

One of the first areas in which it was realized that quantum computers could outperform classical computers was that of quantum simulation [13]. When applied specifically to lattice-based spin systems with two-body interactions, quantum simulation by Trotterization [14] approximates the overall time evolution operator of the quantum-mechanical system by a sequence of two-qubit gates, where each two-qubit gate corresponds to the time evolution according to one two-body term in the Hamiltonian (dynamic quantum simulation) [15, 16]. Additionally, by introducing variable parameters for the per-term evolution times, these circuits are transformed to circuits that prepare ansatz states for the variational quantum eigensolver (VQE) [17], designed to variationally find the ground state of the quantum-mechanical system (static quantum simulation). Before any routing, these circuits for dynamic and static quantum simulation naturally require hardware with a coupling graph that is equal to the lattice of the lattice-based spin system (the virtual graph) [17]. There will generally be a mismatch between the virtual graph and the hardware coupling graph. Efficient qubit routing plays a crucial role in overcoming this mismatch.

In this paper, we develop an efficient and deterministic qubit routing strategy, which we call line-graph qubit routing, or line-graph routing for short. It maps any circuit on a line graph $L(G)$ to hardware with coupling graph $\text{heavy}(G)$. Here, $\text{heavy}(G)$ is obtained from the graph G by placing a node on every edge of G . We call these added nodes the heavy nodes of $\text{heavy}(G)$. By definition, the nodes of the line graph $L(G)$ consist of the heavy nodes of $\text{heavy}(G)$. In $L(G)$, two nodes are adjacent if the associated edges in G are incident on the same

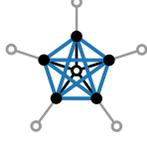
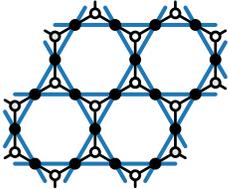
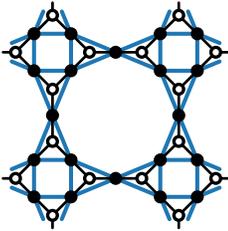
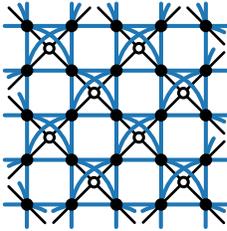
				
$L(G)$	complete	kagome	shuriken/ square-kagome	checkerboard
heavy(G)	(heavy)-star	heavy-hex	heavy-square-octagon	heavy-square/ Lieb lattice [18]
(model) material	spin glasses [19, 20]	herbertsmithite [21]	atlasovite-like [22, 23]	planar pyrochlore [24–26]

Table I. Examples of line-graph routing. Line-graph routing maps any circuit with coupling graph $L(G)$ (blue edges) to circuits with coupling graph heavy(G) (black edges). Line-graph routing finds direct application in the quantum simulation of the magnetic properties of some (model) materials with coupling graph $L(G)$ (last row) on hardware with coupling graph heavy(G).

node of G . It is instructive to verify this property for one of the pairs $(L(G), \text{heavy}(G))$ in Table I. For example, line-graph routing maps the circuits for the quantum simulation of the Heisenberg anti-ferromagnet (HAFM) on the kagome lattice to hardware with a heavy-hex coupling graph. In this example, G is the hexagonal lattice and $L(G)$ is the kagome lattice. Despite these examples, we stress that line-graph routing is applicable to any circuit on any line graph $L(G)$.

The remainder of this paper is organized as follows. We first introduce line-graph routing by example, mapping circuits for the quantum simulation of the kagome HAFM to hardware with a heavy-hex coupling graph (Sec. II). We formalize and generalize this approach to arbitrary circuits and arbitrary line graphs in Sec. III. We benchmark our software implementation of the general algorithm against existing qubit routing approaches in Sec. IV, to conclude with a discussion and outlook in Sec. V.

II. KAGOME TO HEAVY-HEX

Line-graph routing is arguably most clearly explained with an example, which we do in this section by mapping circuits for the quantum simulation of the kagome HAFM to quantum hardware with a heavy-hex coupling graph. First, we use this example due to the relevance of the kagome spin model in exploring quantum phenomena like topological states of matter and quantum spin liquids [21, 27]. The kagome lattice’s significance extends to chemistry, as it is frequently observed in transition metal compounds and metal organic frameworks [28, 29]. The ground state of the kagome HAFM is a long-standing open problem in quantum magnetism [30] that can potentially be solved on NISQ devices [16]. By classical emulation of noiseless quantum computers, it was previ-

ously demonstrated that the ground-state energy found by a VQE approaches the true ground-state energy exponentially as a function of the circuit depth [16].

Second, the heavy-hex coupling graph is the coupling graph of IBM’s current and future superconducting hardware [31, 32]. Among the emerging quantum hardware platforms, IBM’s superconducting qubits have gained significant attention due to their rapid development and scalability in the NISQ era. To optimize this superconducting qubit hardware and mitigate the occurrence of frequency collisions and crosstalk [33–35], error correcting codes are designed on low-degree graphs such as heavy-hex and heavy-square lattices, preventing errors during program execution [31, 32, 36]. This motivates the further development of hardware with these types of connectivity graphs.

The relevance of the kagome-to-heavy-hex mapping was further highlighted by the IBM Quantum’s Open Science Prize 2022, where the challenge was to prepare the ground state of the kagome HAFM using a VQE and implement it on a 16-qubit IBM Quantum Falcon device with a heavy-hex coupling graph [37]. It is important to note that, also within the context of quantum simulation, the routing problem is not unique to the quantum simulation of spin problems on the kagome lattice. Other lattice-based spin models, such the HAFM on the shuriken lattice, are also known for their geometric frustration and challenging simulation [38].

A. Line-graph routing

The first step in line-graph routing is establishing a one-to-one correspondence between the nodes of the virtual graph (in this section, the kagome lattice) and the hardware connectivity graph (in this section, the heavy-hex lattice). This correspondence is achieved by aligning

the nodes of the kagome lattice with the heavy nodes of the heavy-hex lattice, as shown in Fig. 1. Subsequently, the light (non-heavy) nodes of the heavy-hex lattice are used to mediate two-qubit gates between the spins on the nodes of the kagome lattice.

To see this in more detail, assume a kagome quantum circuit, that is, a circuit composed of single-qubit gates on qubits $\{i\}$ and two-qubit gates along the edges $\{(i, j)\}$ of a patch of the kagome lattice. To map the circuit from the kagome to the heavy-hex lattice, we label the heavy qubits on the heavy-hex lattice with the labels $\{i\}$ of the congruent qubits on the kagome lattice, as shown in Fig. 1. Under this identification, any single-qubit gate in the kagome circuit is trivially mapped to a single-qubit gate on the heavy-hex lattice.

To map the two-qubit gates, let us label the ℓ th two-qubit gate in the kagome circuit, acting on qubits (i, j) , by U_{ij}^ℓ . Any such gate can be performed on the heavy-hex lattice by mapping it to the three-qubit *mediated two-qubit gate* MU

$$U_{ij}^\ell \mapsto MU_{imj}^\ell = \text{SWAP}_{mi}U_{mj}^\ell\text{SWAP}_{im}, \quad (1)$$

where qubit $m = m_{ij}$ mediates the interaction between qubits i and j . This map provides the cornerstone of line-graph routing. The key point of line-graph routing is that for every pair of qubits (i, j) the existence and uniqueness of the mediating qubit $m = m_{ij}$ is guaranteed by the definition of $L(G)$ and $\text{heavy}(G)$ (see Sec. I).

Equation (1) introduces many SWAP gates that need not be performed physically. First, SWAP gates occurring at the beginning and end of the routed circuit can be accounted for by a relabeling of the qubits. Second, any two consecutive SWAP gates can be cancelled. These double SWAP gates are introduced by Eq. (1) if there are two consecutive two-qubit gates acting on the same two qubits (possibly with additional single-qubit gates on those qubits in between). Double SWAP gates are also introduced by Eq. (1) in the case of two consecutive two-qubit gates that have a single qubit in common and where the two resulting mediated gates have a mediating qubit in common. That is, if $m = m_{ij} = m_{ik}$, we have by Eq. (1) that

$$\begin{aligned} U_{ik}^{\ell'}U_{ij}^\ell &\mapsto (\text{SWAP}_{mi}U_{mk}^{\ell'}\text{SWAP}_{im})(\text{SWAP}_{mi}U_{mj}^\ell\text{SWAP}_{im}) \\ &= \text{SWAP}_{mi}U_{mk}^{\ell'}U_{mj}^\ell\text{SWAP}_{im}. \end{aligned} \quad (2)$$

In summary, line-graph routing first associates the nodes of $L(G)$ with the heavy nodes of $\text{heavy}(G)$, applies the map of Eq. (1) to all two-qubit gates, and finally removes superfluous SWAP gates as described above.

B. Application: quantum simulation

One immediate application of the kagome-to-heavy-hex mapping is in the quantum simulation of the kagome

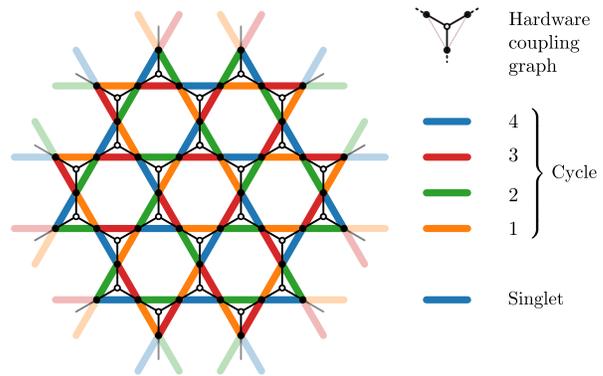


Figure 1. The kagome lattice (colored edges) is the line graph of the hexagonal lattice (black edges). As shown in the figure, the nodes of the kagome lattice can be identified with the heavy nodes of the heavy-hex lattice. The colored edges represent one possible circuit with a kagome coupling graph. Every color represents a layer in this circuit. In the first layer, singlet states are created along the blue lines. Thereafter, HEIS gates [Eq. (5)] are applied along all colored edges in sequence, defining one circuit cycle. This cycle is repeated to obtain the complete circuit. Figure adapted from Ref. [16].

HAFM on heavy-hex quantum hardware. In units where $\hbar = 1$, the HAFM has a Hamiltonian

$$H = \sum_{(i,j)} H_{ij}, \quad H_{ij} = X_i X_j + Y_i Y_j + Z_i Z_j, \quad (3)$$

where the sum is over all edges (i, j) of a given graph. In the current section, this graph could be any patch of the kagome lattice. Here, X_i denotes the Pauli- X operator acting on qubit i (similarly for Y_i, Z_i). This Hamiltonian is straightforwardly generalized to arbitrary two-spin interactions along the edges of a graph [39], which would conceptually not change the constructions that follows.

The goal of dynamic quantum simulation is to compute expectation values of observables with respect to the time-evolved state $|\psi(t)\rangle = e^{-iHt} |\psi(0)\rangle$, given some initial state $|\psi(0)\rangle$. On a quantum computer, this can be achieved by applying the unitary $\prod_{ij} e^{-iH_{ij}(t/r)}$ to $|\psi(0)\rangle$ (the latter of which is assumed to be easy to prepare) a total of r times. That is,

$$|\psi(t)\rangle \approx \left(\prod_{ij} e^{-i\frac{t}{r}H_{ij}} \right)^r |\psi(0)\rangle. \quad (4)$$

The error in this approximation is of the order t^2/r [14], assuming a perfect quantum computer. After the preparation of $|\psi(t)\rangle$, expectation values can be extracted by repeated preparation and measurements. Note that H_{ij} acts on two qubits, so that $e^{-iH_{ij}(t/r)}$ is a two-qubit unitary that can be decomposed into a few one and two-qubit gates that act on qubits i, j only. In the case of the HAFM, $e^{-iH_{ij}(t/r)}$ is called the HEIS gate [16]

$$\text{HEIS}_{ij}(\alpha) \equiv e^{-i\alpha/4} e^{-i\alpha H_{ij}/4}, \quad (5)$$

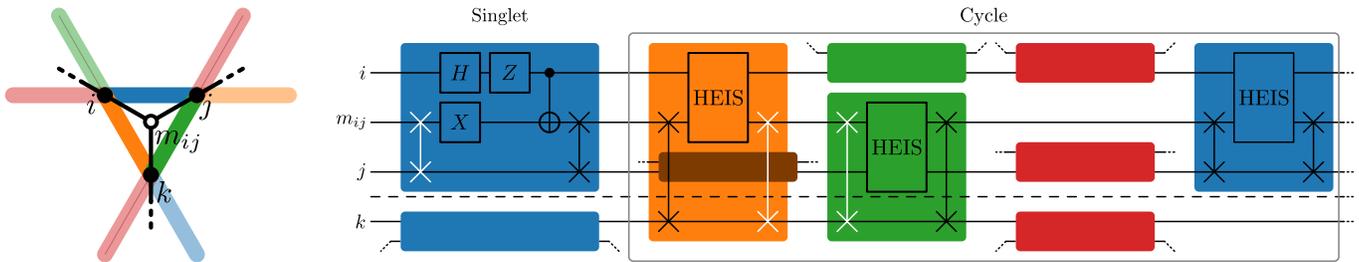


Figure 2. The line-graph routed quantum circuit for the quantum simulation of the kagome HAFM (Fig. 1), focusing on one triangle. The circuits involving the other triangles are similar. Empty colored boxes indicate subcircuits coming in from other triangles. The white SWAP gates can be omitted by a relabeling of the qubits (first SWAP gate) and the cancellation of double SWAP gates [other white SWAP gates, cf. Eq. (2)].

where, in anticipation of static simulation, we have set $\alpha = 4t/r$, and where the physically irrelevant prefactor $e^{-i\alpha/4}$ and a factor of $1/4$ in the exponent are included for consistency with Ref. [16].

We can go from the circuit for dynamical simulation [Eq. (4)] to the circuits needed for static quantum simulation by a VQE [2, 40, 41] by considering α as a free parameter in every occurrence of the HEIS gate. VQEs form a promising method for computing ground state energies of various many-body systems in the NISQ era [2, 40, 41]. In VQEs, a parameterized quantum state is prepared with a parameterized circuit. The energy of this state is measured and optimized using a classical heuristic optimization routine. By the variational principle, the lowest energy that is found in this way provides an upper bound on the ground-state energy. In principle, there are no a priori restrictions on the structure of the ansatz circuit [40]. In the subclass of VQEs using the so-called Hamiltonian variational ansatz (HVA), however, each gate in the parameterized circuit is either formed by parameterized time evolution along a term in the Hamiltonian, or by parameterized time evolution according to some reference Hamiltonian [17]. The initial state of the circuit is the (known and easy-to-prepare) ground state of the reference Hamiltonian.

One possible quantum circuit following from the above considerations is depicted in Fig. 1 (colored edges). This circuit can both be used for the dynamic quantum simulation (fixed parameters α) or for static quantum simulation using the HVA (free parameters α , to be optimized by a classical optimization routine). Here, every color depicts a different layer of the circuit. In the first layer, singlet states $(|01\rangle - |10\rangle)/\sqrt{2}$ are placed along the blue edges. This provides the initial state of either the circuit for dynamical or static quantum simulation. It is the ground state of a reference Hamiltonian $\sum_{(i,j)} H_{ij}$, where the sum is over the blue edges (i, j) . After preparation of the initial state, HEIS gates are placed along all orange, green, red, and blue edges. This combination of HEIS gates defines one *cycle* of the circuit. The cycle is repeated r times. (The color blue defines both the initial state and the last layer of the cycle.) Alternative circuits for dynamic or static quantum simulation (using

the HVA) may be achieved by changing the initial state or the order of the HEIS gates.

Line-graph routing maps the kagome circuit from Fig. 1 to a circuit on the heavy-hex lattice by first identifying nodes of the respective lattices, as in Fig. 1. Subsequently, Eq. (1) is applied to all gates, after which superfluous SWAP gates are removed [Eq. (2)]. The routed circuit thus obtained is shown in Fig. 2.

III. GENERAL CASE

Here, we formalize and generalize the routing strategy from the previous section, which leads to our main results, Theorem 1 and Algorithm 2. To introduce notation, we now define line- and heavy graphs more formally. The graph $\text{heavy}(G)$ is obtained from G by defining a node i and edges (m, i) , (i, m') for each edge (m, m') in G . The nodes i are referred to as the heavy nodes. A minor but subtle point is that this construction may lead to paths of length three as induced subgraphs of $\text{heavy}(G)$ in which two adjacent nodes are of degree one. That is, induced subgraphs of the form $P = \{(m, i), (i, m')\}$, with m and i nodes of degree one. For example, in the star graph (Table I), all paths emanating from the center node (i.e., the ‘rays’ of the star) are of the form of P . Line-graph routing [previewed in Eq. (1)] will never use mediating qubits associated with nodes locally equivalent (including neighbors and next-nearest neighbors) to node m in P and can therefore be discarded without affecting the routed circuit. In the example of the star graph (Table I), this means the qubits associated with the gray nodes may be removed from the routed circuit. To avoid frequent mention of minor but subtle point, in this paper we also refer to heavy graphs where the nodes locally equivalent (including neighbors and next-nearest neighbors) to node m in P are removed as heavy graphs. After these removals, we still refer to nodes locally equivalent (only including neighbors) to node i in P as heavy nodes.

The line graph of G , $L(G)$, is defined in terms of $\text{heavy}(G)$ as follows: construct $\text{heavy}(G)$ from G and let the node set of $L(G)$ be equal to the set of heavy nodes

$\{i\}$ of $\text{heavy}(G)$. An edge (i, j) is added to $L(G)$ if and only if there exists a node m in G such that (i, m) and (m, j) are edges in $\text{heavy}(G)$.

We say that a quantum circuit C , consisting of one- and two-qubit gates by assumption, has coupling graph $G = (V, E)$ if V is equal to the set of qubit labels in C and if (i, j) in E if and only if there exists a two-qubit gate U_{ij} in C . If a *quantum computer* has coupling graph $G = (V, E)$, then it can perform arbitrary single-qubit gates on the qubits associated with each node in V and arbitrary two-qubit gates along each edge in E . By Eq. (1), we then have the following theorem.

Theorem 1 (Line-graph routing). *Every quantum circuit C with coupling graph $L(G)$ can be performed on quantum hardware with coupling graph $\text{heavy}(G)$ with a SWAP overhead of at most two times the number of two-qubit gates in C .*

Proof. By definition, there is a one-to-one correspondence between the nodes of $L(G)$ and the heavy nodes $\{i\}$ of $\text{heavy}(G)$. Therefore, the single-qubit gates of C can be mapped directly to hardware with coupling graph $\text{heavy}(G)$. Furthermore, for every edge (i, j) in $L(G)$, there are edges (i, m) and (m, j) in $\text{heavy}(G)$, where $m = m_{ij}$ can be determined uniquely from i and j . Thus, every two-qubit gate U_{ij} in C can be mapped to the three-qubit gate $MU_{imj} := \text{SWAP}_{mi}U_{mj}\text{SWAP}_{im}$, leading to an overhead of 2λ SWAP gates, with λ the total number of two-qubit gates in C . \square

Note that the above theorem does not make any assumption about the graph G . Also note that the theorem provides a hierarchy of mappings, as $\text{heavy}(G)$ itself may be a line graph. For example, invoking the above theorem twice gives a routing from $L(L(G))$ to $\text{heavy}(L(G))$.

In practice, one may be given hardware with a coupling graph H' and asked to find the class of circuits that can be run on this hardware using the line-graph construction. (For an overview of the graphs that follow, please see Fig. 3.) This task may arise if one has specific but limited quantum hardware available and wants to explore its capabilities by looking at quantum circuits that can be routed to this hardware with low overhead. The most immediate class of such quantum circuits is obviously formed by circuits with coupling graph H' . In case H' is a heavy graph, $H' = \text{heavy}(G)$, line-graph routing extends the possibilities to the class of circuits with coupling graph $L(G)$. If H' is a heavy graph, it is trivial to find G such that $H' = \text{heavy}(G)$. It is also trivial to construct the line graph $L(G)$ from G . Therefore, given a heavy hardware coupling graph, it is trivial to find the class of circuits that can be run on it by line-graph routing. For example, given hardware with a heavy-hex coupling graph, it is trivial to find that line-graph routing yields the class of kagome circuits.

The task can also be reversed: given a circuit with a coupling graph G' , find the hardware on which it can be run with low overhead (cf. Fig. 3). Again, the first

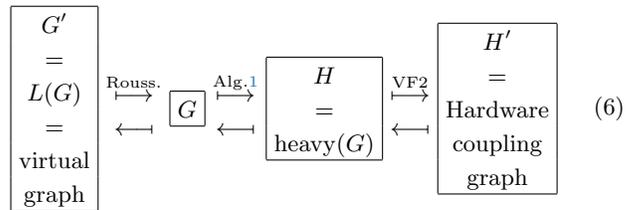


Figure 3. An overview of the graphs related to line-graph routing. All maps from the right to the left are trivial given that H' is a heavy graph.

place to look would be hardware with coupling graph G' . Line-graph routing extends the possibilities by adding hardware with a coupling graph of $\text{heavy}(G)$ with G such that $G' = L(G)$.

But how to find G from the circuit coupling graph G' ? This is less straightforward, first because it is not possible to find G from G' if G' is not a line graph. In this case, line-graph routing cannot be applied. There are numerous straightforward ways of checking whether a graph is a line graph. For example, Beineke's theorem states that a graph is a line graph if and only if it does not contain an induced subgraph out of a set of nine forbidden subgraphs [42]. One of these forbidden graphs is the claw (Υ). For example, since the heavy-hex and hexagonal lattices consist entirely of claws, they are themselves not line graphs. Second, even if G' is a line graph, it is nontrivial to find the graph G such that $G' = L(G)$. Nevertheless, Roussopoulos' algorithm [43] finds G from G' , or reports that G' is not a line graph, in time $O(\max\{n, |E_{G'}|\})$, with n the number of nodes and $|E_{G'}|$ the number of edges of G' .

We briefly introduce the concepts from Roussopoulos' algorithm that are useful to us later. If G' is a line graph, Roussopoulos' algorithm partitions the edges of G' into complete subgraphs in such a way that no node lies in more than two of the subgraphs (which is possible if and only if G' is a line graph). Then, the nodes of G correspond to the sets in the partition. Additionally, the nodes that lie in only one of the sets in the partition are added as nodes of G as sets of length one. Two nodes in G are adjacent if their corresponding sets have a nonempty overlap. For example, in Fig. 1, each triangle of colored edges forms a set in the partition of the kagome lattice because triangles are fully connected subgraphs of the kagome lattice and no node of the kagome lattice is in more than two triangles. Defining the partitions as the nodes of $L(G)$ and putting an edge between two nodes in $L(G)$ whenever the corresponding sets have a nonempty intersection results in the hexagonal lattice (open nodes in Fig. 1).

For line-graph routing [as previewed in Eq. (1)], $L(G)$ must be mapped to $\text{heavy}(G)$ in such a way that the labels of the heavy nodes of $\text{heavy}(G)$ are identical to the labels of the nodes of $L(G)$. The output of Roussopoulos'

algorithm provides a convenient way to achieve this labeling through the following algorithm. It takes a graph G , as generated by Roussopoulos' algorithm, as input and returns $H = \text{heavy}(G)$, a heavy graph where the set of heavy nodes is equal to the set of nodes (i.e., with equal labels) of $G' = L(G)$. This generalizes and automatizes the identification of nodes made in Fig. 1.

Algorithm 1 (Congruent heavy labels). Consider the edges (a, b) of G for which both a and b contain more than one node. (The nodes in a and b are from G' , the latter of which need not be provided explicitly to the current algorithm.) For all such edges (a, b) , add (a, c) and (c, b) to an empty graph H . Because a and c are distinct sets in a partition of the edges of G' into complete subgraphs and a and c have a nonempty overlap, c contains a single entry and this entry is a node from G' . Now, consider the edge cases, that is, the edges (a, b) in G where either a or b is a set of length one. For every such edge, add (a, b) to H .

Everything is now in place for the succinct presentation of line-graph routing. It maps any circuit C , with unknown coupling graph G' , to a circuit with coupling graph H whenever G' is a line graph, $G' = L(G)$, and where $H = \text{heavy}(G)$.

Algorithm 2 (Line-graph routing). Construct the coupling graph G' of the circuit C . Run Roussopoulos' algorithm on G' to obtain $G = L^{-1}(G')$. Construct $\text{heavy}(G)$ using Algorithm 1. For every two-qubit gate U_{ij} in C , let $m = m_{ij}$ be the node in $\text{heavy}(G)$ that is in between nodes i, j of $\text{heavy}(G)$ and replace U_{ij} by $MU_{imj} := \text{SWAP}_{mj}U_{im}\text{SWAP}_{jm}$.

Note the existence and uniqueness of mediating qubit m are guaranteed by the definition of line graphs. In some cases, some qubits, related to the so-called lone leaf nodes in the coupling graph of the output circuit of line-graph routing, can be removed from that circuit, reducing the SWAP and qubit count of line-graph routed circuits. This removal leads to a marginal improvement, which may nevertheless be crucial given hardware with few qubits, but arguably obfuscates the general idea of line-graph routing. The definition of lone leaf nodes and the method for their removal is presented in Appendix A.

As opposed to heuristic methods [7], line-graph routing is deterministic, allowing rigorous performance guarantees. Given the time complexity of Roussopoulos' algorithm [43], it is straightforward to show that the time complexity of line-graph routing is $O(\Lambda^2)$, with Λ the number of gates in the input circuit C . A tighter bound on the time complexity can possibly be obtained, but this requires a more sophisticated analysis. Such an analysis is unnecessary for the current purposes because a nonoptimized implementation of line-graph routing already routes circuits with thousands of qubits and hundreds of thousands of gates within a minute on a standard laptop [44].

To utilize line-graph routing in practice, one additional step may be required. Quantum hardware providers will generally use a specific labeling of the qubits on their hardware, leading to a hardware coupling graph H' . This labeling may differ from the labeling of $H = \text{heavy}(G)$ obtained through Algorithm 1. However, H can be mapped to a subgraph of H' using an algorithm that finds subgraph isomorphisms, such as the VF2 algorithm [45]. The VF2 algorithm generates a list of isomorphic subgraphs, and in practice, a selection is made based on a performance metric, such as the average two-qubit gate fidelity, to identify the subgraph with the best performance. This step is commonly implemented in quantum computing software development kits. In this paper, we also refer to H as the hardware coupling graph and in this wording leave the possible mapping to the fixed qubit labels provided by a hardware provider implicit.

IV. IMPLEMENTATION AND BENCHMARKING

In the Supplemental Material [44], we implement, showcase and benchmark line-graph routing, Algorithm 2, together with the removal of lone leaves (Appendix A). The implementation takes any Qiskit [46] quantum circuit consisting of one- and two-qubit gates, constructs its coupling graph $L(G)$ or reports that the coupling graph is not a line graph, finds G and $\text{heavy}(G)$, and outputs the routed circuit with coupling graph $\text{heavy}(G)$. The implementation does not rely heavily on Qiskit's methods and can hence straightforwardly be transformed to an implementation in other quantum software development kits.

Line-graph routing is benchmarked against all routing methods available in Qiskit by default [44, 46]. In this section, we show line-graph routing is able to confidently outperform these default methods on relevant problem instances. There are also problem instances where line-graph routing does not outperform the default methods. In the end of this section, we discuss for which types of instances we expect line-graph routing to be superior. We consider two types of circuits.

(i) *Random*. With probability $2/5$, a CNOT gate is placed along a randomly chosen edge of a given virtual graph. With a probability $3/5$ a gate from the set $\{H, S, T\}$ is chosen uniformly at random and placed at a random node.

(ii) *Quantum simulation*. As described in detail for the kagome lattice (Sec. II B), circuits for the dynamical and static quantum simulation of the HAFM on any lattice can be defined by an edge-coloring of that lattice [15, 16]. An edge coloring of a graph is an assignment of colors to the edges such that no two edges with the same color are incident on the same node. This edge coloring is called minimal if it uses the least possible number of colors. We perform edge coloring of the virtual lattices by an automatic method that generally finds an edge coloring

$L(G)$	Routing method	Opt. depth	n_{SWAP}	n_{qubit}	t_{tot} (s)	\bar{t} (s)
Complete	line-graph	1036	650	10	<1	<1
	SABRE	720	146	9	3	<1
Kagome	line-graph	226	7968	300	43	43
	SABRE	790	8486	200	886	55 ± 1
Shuriken	line-graph	209	13600	476	77	77
	SABRE	1099	17064	323	2520	157 ± 4
Checkerboard	line-graph	435	18521	393	67	67
	SABRE	2121	23060	282	1750	109 ± 1

Table II. Excerpt of the benchmarking data available in the Supplemental Material [44]. The column headers are defined in the main text.

that is not minimal. The benefit of this coloring method is that it does not require a manual assignment of edge colors. The downside is that we expect line-graph routing to work best (compared to other methods) for circuits derived from a minimal edge coloring. This does not pose a problem because, as we show in this section, line-graph routing is already able to outperform the default methods in Qiskit in routing circuits derived from a nonminimal edge coloring.

We found SABRE [7] to outperform the other methods in Qiskit and therefore we focus on a comparison between line-graph routing and SABRE in what follows. Unlike line-graph routing, SABRE is a probabilistic routing method that obtains a different qubit routing with each run. Additionally, the intensity of the optimization that is part of SABRE can be varied, leading to a tradeoff between the classical resources required and the performance characteristics of the routed circuits. We address these issues by running SABRE 16 times (at fixed optimization level) and comparing the performance against one run of line-graph routing. We do this separately for every optimization level available by default in Qiskit, which range from optimization level 0 (‘no optimization’) to optimization level 3 (‘heavy-weight optimization’) in integer steps [46].

In Table II, we show an excerpt of the benchmarking data, which includes problem instances on which line-graph routing does and does not perform well. The following performance characteristics are listed.

(i) *Opt. depth.* The optimal (lowest) depth reached by the routing method among all runs (line-graph routing is run once per virtual graph, SABRE is run 16 times per virtual graph). Routed circuits are obtained by inserting SWAP gates (as dictated by line-graph routing or SABRE) and no further gate identities are used to simplify the resulting circuits. So, in the case of random input circuits, the routed circuits consist of gates from the set $\{\text{CNOT}, H, S, T\} \cup \{\text{SWAP}\}$. The routed circuits contain, for example, double H gates if those were present in the input circuit. In the case of quantum simulation input circuits, the routed circuits consist of gates from the set $\{\text{SINGLET}, \text{HEIS}(\alpha)\} \cup \{\text{SWAP}\}$.

(ii) n_{SWAP} . The number of SWAP gates of the routed

circuit that achieved the lowest depth.

(iii) n_{qubit} . The number of active qubits in the routed circuit that achieved the lowest depth.

(iv) t_{tot} . The total wall-clock time needed to run all runs of the routing method. For line-graph routing, this includes the time needed to find $\text{heavy}(G)$ from $L(G)$. The implementation of line-graph routing repeatedly loops through all gates using (slow) Python loops and can likely be sped up considerably, if needed. We use Qiskit’s standard implementation of SABRE. SABRE is given the target graph $\text{heavy}(G)$ as input and hence finding $\text{heavy}(G)$ from $L(G)$ is not included in its wall-clock time. The benchmarks for different methods are always run on the same hardware.

(v) \bar{t} . The average wall-clock time for a single run of the routing method. The error bars are calculated by bootstrapping the individual wall-clock times and represent symmetrized 95% confidence intervals.

The first two data lines of Table II show the performance characteristics of line-graph routing and SABRE when applied to a random circuit on the complete graph with 9 nodes. SABRE was run with optimization level 1. (Passing a higher optimization level to Qiskit’s transpiler will trigger the usage of gate identities.) Already at an optimization level of 1 SABRE outperforms line-graph routing on all performance characteristics considered except the total wall-clock time.

Data lines 3 and 4 of Table II show the performance characteristics of line-graph routing and SABRE when applied to circuits for the quantum simulation of the HAFM on patches of the kagome lattice measuring 7×7 unit cells, with open boundary conditions and padded edges (see [44]). Here, SABRE was run at Qiskit’s transpiler optimization level 3, the highest optimization level available. As opposed to the transpilation of the random circuits, no gate identities are used in this process because the gates $\{\text{SWAP}, \text{SINGLET}, \text{HEIS}\}$ have unknown properties to Qiskit’s transpiler [46]. Line-graph routing outperforms SABRE in terms of the optimal depth of the routed circuits by about a factor of 3.5, while at the same time requiring less time than one run of SABRE. The line-graph routed circuit also uses less SWAP gates than the SABRE routed circuit with the lowest depth.

However, the line-graph routed circuit uses a factor of 1.5 more qubits than the SABRE routed circuit that achieved the lowest depth. Nevertheless, the space-time volume of the line-graph routed circuit is still about half of the space-time volume of the SABRE routed circuit with the lowest depth. Similar results, with an even larger performance gap, hold for the routing of circuits for the quantum simulation of the HAFM on patches of the shuriken lattice measuring 7×7 unit cells (lines 5 and 6 of Table II) and patches of the checkerboard lattice of 7.5×7.5 unit cells (last two lines of Table II).

Line-graph routing was conceived while keeping in mind its application in mapping quantum circuits to hardware with a lattice-like low-degree coupling graph. It is therefore not expected to perform well in mapping quantum circuits to hardware with a high-degree coupling graph without lattice-like structure. In fact, the benchmarking results show that line-graph routing is not well-suited for mapping circuits on the complete coupling graph to hardware with a star coupling graph. One property of line-graph routing that leads to its low effectiveness on unstructured, high-degree graphs is that in the output circuit of line-graph routing, the qubits are assigned a base location where they return to after they are acted on by one [Eq. (1)] or multiple [Eq. (2)] gates from the input circuit. This is likely advantageous for input quantum circuits with a structured, low-degree coupling graph. For example, in Fig. 2, in the third layer of the cycle (red gates), all qubits are still close to where they are needed, leading to the insertion of few SWAP gates to get them there.

However, on high-degree coupling graphs, the property of a base location need not be advantageous since any qubit can be routed to any other qubit in relatively few steps and regularly returning qubits to their base location leads to the insertion of unnecessary SWAP gates. As an extreme example, let us look at the action of line-graph routing on a circuit C with a complete coupling graph, where at one point in C a cascade of CNOT gates, $\prod_{i=n}^1 \text{CNOT}_{i,i+1}$, is prescribed. Line-graph routing maps the circuit C to a circuit on the star graph (Table I). To perform $\text{CNOT}_{i,i+1}$ on star-graph hardware, line-graph routing first swaps qubit i to the center of the star, performs a CNOT between the center qubit and qubit $i+1$, and swaps qubit i back to its original position [Eq. (1)]. Not insisting that the qubits eventually return to their original position leads to the possibility of more efficient routing strategy. To perform the $\text{CNOT}_{i,i+1}$, swap qubit i to the center, perform the CNOT between the center qubit and qubit $i+1$, and leave qubit i in the center. After repeating this procedure for the subsequent CNOT in the circuit, qubit i will end up at the initial location of qubit $i+1$. The latter approach uses half of the number of SWAP gates compared to line-graph routing.

A second situation in which line-graph routing is not expected to perform well is when there are subgraphs of the virtual graph $L(G)$ that are isomorphic to subgraphs of $\text{heavy}(G)$. In that case, unnecessary mediating qubits

may be inserted. For example, let $L(G)$ be the path graph on n nodes. When applied to circuits on this graph, line-graph routing introduces mediating qubits between all qubits of $L(G)$, leading to a circuit on a path graph with $n-1$ added mediating qubits. This happens despite the fact that no routing is needed at all. If needed, line-graph routing can be enhanced such that it detects and prevents this behavior.

V. DISCUSSION AND OUTLOOK

In this paper, we developed line-graph routing, a qubit routing strategy that efficiently and deterministically maps any quantum circuit on a line graph $L(G)$ to a circuit on the heavy graph $\text{heavy}(G)$. By software implementation and benchmarking, we showed its ability to outperform standard, general-purpose methods on input quantum circuits, circuit sizes, and hardware connectivity graphs of practical relevance.

Line-graph routing showed not to perform well in mapping circuits on the complete graph to the star graph. We attribute this to the high degree and absence of a lattice-like structure of the complete graph. Based on our benchmarking results, we expect line-graph routing to outperform general-purpose methods in routing circuits with a line-graph coupling graph to low-degree hardware coupling graphs. For superconducting qubits, these are exactly the hardware coupling graphs preferable from an engineering standpoint [31, 32, 36].

Line-graph routing is limited in its applicability because it is only defined for pairs of graphs ($L(G), \text{heavy}(G)$). General-purpose methods are able to map any circuit on any graph to a circuit on any other graph (given that the latter graph is connected and has at least the same number of nodes as the former graph). However, in general, finding the optimal routing strategy is NP-hard. It is therefore unlikely that general-purpose methods can find the optimal or close-to-optimal routing strategy. Routing strategies that are defined on a subset of possible input circuits can nevertheless be highly efficient and effective, and indeed line-graph routing is shown to have the ability to outperform standard-purpose methods on the subset of problem instances for which it is defined. The set of pairs of graphs ($L(G), \text{heavy}(G)$) is still infinitely large and contains pairs of practical relevance.

The routed circuit for the simulation of a nearest-neighbor model on $L(G)$ can be reinterpreted as a circuit for the quantum simulation of a next-nearest-neighbor model on $\text{heavy}(G)$. This is evident from Table I and Fig. 1 and follows directly from the definition of line graphs as given in Sec. III. This routed circuit for the simulation of a nearest-neighbor model on $L(G)$ can be lifted to a circuit for the quantum simulation of a model containing both nearest- and next-nearest-neighbor interactions on $\text{heavy}(G)$ by adding the gates arising from nearest-neighbor interactions on $\text{heavy}(G)$ to the routed

circuit. These gates naturally satisfy the heavy(G) connectivity graph and can thus be added to the circuit without any routing. This further enlarges the range of applicability of line-graph routing.

The effectiveness of line-graph routing may be improved further by leveraging the freedom of which qubit is swapped with the mediating qubit. In line-graph routing, for the implementation of the gate U_{ij}^ℓ , it is qubit i that is swapped with the mediating qubit [Eq. (1)]. However, in some cases, it is only when swapping qubit j with the mediating qubit that a cancellation of SWAP gates from consecutive mediated two-qubit gates MU [Eq. (1), Fig. 2] occurs. Thus, line-graph routing may be improved by letting the decision of which qubit to swap with the mediating qubit depend on the ensuing gates in the input quantum circuit.

Although we showed the ability of line-graph routing to outperform general-purpose methods, we did not prove it gives the optimal routing. In fact, it was shown to be suboptimal in cases where the input circuit is a circuit on the complete graph. A proof or refutation for the optimality of line-graph routing (possibly after including the improvement of the previous paragraph) would therefore require careful consideration of the allowed input circuits and performance characteristic for which optimality is considered. Such a proof may inspire even

stronger methods that build on or generalize line-graph routing.

A straightforward but exciting way to carry on the work in the current paper is to run our routed circuits on actual quantum hardware. The circuits available in the Supplemental Material [44] can be executed as-is on hardware with the appropriate hardware coupling graphs. Remaining challenges therein are the extraction of useful physical quantities from the generated quantum states. To obtain results that go beyond anything that can be obtained classically requires further improvements of error mitigation techniques and quantum hardware.

All code and data used to generate the results in this paper are available as Supplemental Material [44] and at Ref. [47].

ACKNOWLEDGMENTS

JK acknowledges funding from the Competence Center Quantum Computing Baden-Württemberg, under the project QORA. Benchmarking was carried out on the Scientific Compute Cluster of the University of Konstanz (SCCKN).

-
- [1] A. Montanaro, Quantum algorithms: an overview, *npj Quantum Information* **2**, 15023 (2016).
- [2] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio, *et al.*, Variational quantum algorithms, *Nature Reviews Physics* **3**, 625–644 (2021).
- [3] A. M. Childs, E. Schoute, and C. M. Unsal, Circuit transformations for quantum architectures, in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 135 (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019) pp. 3:1–3:24.
- [4] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, On the qubit routing problem, in *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 135 (Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 2019) pp. 5:1–5:32.
- [5] A. Bapat, A. M. Childs, A. V. Gorshkov, and E. Schoute, Advantages and limitations of quantum routing, *PRX Quantum* **4**, 010313 (2023).
- [6] D. Maslov, S. M. Falconer, and M. Mosca, Quantum circuit placement, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **27**, 752–763 (2008).
- [7] G. Li, Y. Ding, and Y. Xie, Tackling the qubit mapping problem for NISQ-era quantum devices, in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19 (Association for Computing Machinery, New York, NY, USA, 2019) p. 1001–1014.
- [8] M. Y. Siraichi, V. F. dos Santos, C. Collange, and F. M. Q. Pereira, Qubit allocation as a combination of subgraph isomorphism and token swapping, *Proceedings of the ACM on Programming Languages* **3**, 1–29 (2019).
- [9] R. Wille, L. Burgholzer, and A. Zulehner, Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations, in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19 (Association for Computing Machinery, New York, NY, USA, 2019).
- [10] J. Preskill, Quantum computing in the NISQ era and beyond, *Quantum* **2**, 79 (2018).
- [11] D. S. França and R. García-Patrón, Limitations of optimization algorithms on noisy quantum devices, *Nature Physics* **17**, 1221–1227 (2021).
- [12] S. Martiel and T. G. d. Brugière, Architecture aware compilation of quantum circuits via lazy synthesis, *Quantum* **6**, 729 (2022).
- [13] R. P. Feynman, Simulating physics with computers, *International Journal of Theoretical Physics* **21**, 467–488 (1982).
- [14] S. Lloyd, Universal quantum simulators, *Science* **273**, 1073–1078 (1996).
- [15] G. Burkard, Recipes for the digital quantum simulation of lattice spin systems (2022), [arXiv:2209.07918](https://arxiv.org/abs/2209.07918) [quant-ph].
- [16] J. Kattemölle and J. van Wezel, Variational quantum eigensolver for the Heisenberg antiferromagnet on the kagome lattice, *Phys. Rev. B* **106**, 214429 (2022).

- [17] D. Wecker, M. B. Hastings, and M. Troyer, Progress towards practical quantum variational algorithms, *Physical Review A* **92**, 042303 (2015).
- [18] E. H. Lieb, Two theorems on the hubbard model, *Phys. Rev. Lett.* **62**, 1201–1204 (1989).
- [19] D. Sherrington and S. Kirkpatrick, Solvable model of a spin-glass, *Phys. Rev. Lett.* **35**, 1792–1796 (1975).
- [20] G. Parisi, Infinite number of order parameters for spin-glasses, *Phys. Rev. Lett.* **43**, 1754–1756 (1979).
- [21] M. R. Norman, Colloquium: Herbertsmithite and the search for the quantum spin liquid, *Reviews of Modern Physics* **88**, 041002 (2016).
- [22] R. Siddharthan and A. Georges, Square kagome quantum antiferromagnet and the eight-vertex model, *Phys. Rev. B* **65**, 014417 (2001).
- [23] M. Fujihala, K. Morita, R. Mole, S. Mitsuda, T. Tohyama, S.-i. Yano, D. Yu, S. Sota, T. Kuwai, A. Koda, H. Okabe, H. Lee, S. Itoh, T. Hawaii, T. Masuda, H. Sagayama, A. Matsuo, K. Kindo, S. Ohira-Kawamura, and K. Nakajima, Gapless spin liquid in a square-kagome lattice antiferromagnet, *Nature Communications* **11**, 3429 (2020).
- [24] R. R. P. Singh, O. A. Starykh, and P. J. Freitas, A new paradigm for two-dimensional spin liquids, *Journal of Applied Physics* **83**, 7387–7389 (1998).
- [25] J.-B. Fouet, M. Mambrini, P. Sindzingre, and C. Lhuillier, Planar pyrochlore: A valence-bond crystal, *Phys. Rev. B* **67**, 054411 (2003).
- [26] R. F. Bishop, P. H. Y. Li, D. J. J. Farnell, J. Richter, and C. E. Campbell, Frustrated Heisenberg antiferromagnet on the checkerboard lattice: J_1 - J_2 model, *Phys. Rev. B* **85**, 205122 (2012).
- [27] L. Savary and L. Balents, Quantum spin liquids: a review, *Reports on Progress in Physics* **80**, 016502 (2016).
- [28] D. G. Nocera, B. M. Bartlett, D. Grohol, D. Papoutsakis, and M. P. Shores, Spin frustration in 2D kagome lattices: A problem for inorganic synthetic chemistry, *Chemistry - A European Journal* **10**, 3850–3859 (2004).
- [29] G. Chakraborty, I.-H. Park, R. Medishetty, and J. J. Vital, Two-dimensional metal-organic framework materials: Synthesis, structures, properties and applications, *Chemical Reviews* **121**, 3751–3891 (2021).
- [30] A. M. Läuchli, J. Sudan, and R. Moessner, $s = \frac{1}{2}$ kagome Heisenberg antiferromagnet revisited, *Phys. Rev. B* **100**, 155142 (2019).
- [31] P. Nation, H. Paik, A. Cross, and Z. Nazario, *The IBM quantum heavy hex lattice* (2022), accessed on March 16, 2023.
- [32] S. Bravyi, O. Dial, J. M. Gambetta, D. Gil, and Z. Nazario, The future of quantum computing with superconducting qubits, *Journal of Applied Physics* **132**, 160902 (2022).
- [33] M. Brink, J. M. Chow, J. Hertzberg, E. Magesan, and S. Rosenblatt, Device challenges for near term superconducting quantum processors: frequency collisions, in *2018 IEEE International Electron Devices Meeting (IEDM)* (2018) pp. 6.1.1–6.1.3.
- [34] J. B. Hertzberg, E. J. Zhang, S. Rosenblatt, E. Magesan, J. A. Smolin, J.-B. Yau, V. P. Adiga, M. Sandberg, M. Brink, J. M. Chow, and J. S. Orcutt, Laser-annealing Josephson junctions for yielding scaled-up superconducting quantum processors, *npj Quantum Information* **7**, 129 (2021).
- [35] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, Detecting crosstalk errors in quantum information processors, *Quantum* **4**, 321 (2020).
- [36] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross, Topological and subsystem codes on low-degree graphs with flag qubits, *Phys. Rev. X* **10**, 011022 (2020).
- [37] O. Lanes and A. Rasmussen, *IBM quantum’s open science prize returns* (2023).
- [38] D. Wu, R. Rossi, F. Vicentini, N. Astrakhantsev, F. Becca, X. Cao, J. Carrasquilla, F. Ferrari, A. Georges, M. Hibat-Allah, *et al.*, Variational benchmarks for quantum many-body problems (2023), [arXiv:2302.04919](https://arxiv.org/abs/2302.04919) [quant-ph].
- [39] These are described by $H = \sum_{(i,j)} P_{ij} + \sum_i P_i$, with two-spin terms $P_{ij} = \sum_{kl} \gamma_{ij}^{(kl)} \sigma_i^{(k)} \sigma_j^{(l)}$ and single-spin terms $P_i = \sum_k \gamma_i^{(k)} \sigma_i^{(k)}$, where $\{\sigma_i^{(k)}\} = \{\mathbb{1}, X_i, Y_i, Z_i\}$.
- [40] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’Brien, A variational eigenvalue solver on a photonic quantum processor, *Nature Communications* **5**, 4213 (2014).
- [41] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, The theory of variational hybrid quantum-classical algorithms, *New Journal of Physics* **18**, 023023 (2016).
- [42] L. W. Beineke, Characterizations of derived graphs, *Journal of Combinatorial Theory* **9**, 129–135 (1970).
- [43] N. D. Roussopoulos, A $\max\{m, n\}$ algorithm for determining the graph H from its line graph G, *Information Processing Letters* **2**, 108–112 (1973).
- [44] Supplemental Material available at [URL]. In the Supplemental Material, we implement, showcase and benchmark line-graph routing for various graphs. The Supplemental Material is also available at Ref. [47].
- [45] L. Cordella, P. Foggia, C. Sansone, and M. Vento, A (sub)graph isomorphism algorithm for matching large graphs, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **26**, 1367–1372 (2004).
- [46] Qiskit contributors, *Qiskit: An open-source framework for quantum computing* (2023).
- [47] J. Kattemölle and S. Hariharan, Line-graph routing (LIGRAR), <https://github.com/kattemolle/LIGRAR>.

Appendix A: Removal of lone leaf nodes

In some cases, line-graph routing produces circuits containing qubits i that are swapped with only one corresponding mediating qubit m_i during the course of the entire circuit. In these cases, the mediating qubits m_i can be removed and replaced by the corresponding qubits i , leading to a reduction in the qubit and SWAP count of the routed circuit.

We define a *lone leaf* i as a node in a coupling graph $\text{heavy}(G)$ that is of degree one and shares its only edge with a node that has no other neighbors of degree one. We do not assume the coupling graph is a tree. As an example, consider the triangle graph with an extra node i connected to one of the nodes m_i of the triangle. The node i is a lone leaf.

If a node i is a lone leaf in a coupling graph $\text{heavy}(G)$,

by the construction of line-graph routing its neighbor m must be a mediating node. For every two-qubit gate that is performed between qubit i and any other neighbor j of m , qubits i and m first need to be swapped. So, as far as the interactions with qubit i are concerned, qubit m may be fully eliminated by simply removing qubit m and reverting mediated two-qubit gates MU_{imj} to the

original gates MU_{imj} . Physically, qubit i can be put on the place of qubit m . After the removal of m and the relocation of qubit i , inspection of Eq. (1) reveals that qubit i may in fact act as a mediating node for any two-qubit gates between any two neighbors j, k of m unequal to i ; after a mediated gate MU_{jik} , qubit i returns to its starting position unaffected.

Supplemental material to: “Line-graph qubit routing: from kagome to heavy-hex and more”

Joris Kattemölle¹ and Seenivasan Hariharan²

¹Department of Physics, University of Konstanz, Konstanz, Germany

²Institute of Physics, University of Amsterdam, Amsterdam, The Netherlands

³QuSoft, CWI, Amsterdam, The Netherlands

Abstract

We showcase and benchmark line-graph qubit routing by routing random circuits (with fixed virtual graph) and circuits needed for the quantum simulation of the Heisenberg antiferromagnet on various graphs. The code implementing line-routing itself is found in `line_graph_routing.py` and maps Qiskit quantum circuits to Qiskit quantum circuits. (This can be altered to other circuit libraries with little effort.) We benchmark our results against other methods. The interactive version of this document is `line_graph_routing.ipynb`.

Contents

1	Requirements	2
2	Kagome to heavy-hex	2
2.1	Random	2
2.2	Quantum simulation	2
3	Complete graph to star graph	4
3.1	Random	4
3.2	Quantum simulation	4
4	Shuriken to heavy square-octagon	5
4.1	Random	5
4.2	Quantum simulation	5
5	Checkerboard to heavy-square	6
5.1	Quantum simulation	7
6	Random line graph to random heavy graph	7
6.1	Random	7
6.2	Quantum simulation	8
7	Benchmarking	10
7.0.1	Quantum simulation, kagome and shuriken, against SABRE	10
7.0.2	Quantum simulation, checkerboard, against SABRE	27
7.0.3	Random circuit, kagome and shuriken, against SABRE	36
7.0.4	Random circuit, complete graph, against SABRE	38
7.0.5	Against other routing methods	40
7.0.6	Wall-clock time of line-graph routing	42

1 Requirements

This notebook should typically run after installing the following packages with pip (or conda). In a terminal, run

```
pip install qiskit[visualization]
```

or

```
pip install 'qiskit[visualization]'
```

and

```
pip install netket networkx tabulate
```

Note Netket currently needs Python 3.9 (and SciPy $\geq 1.9.3$). Netket is only used to generate patches of the kagome lattice as graphs and not for line-graph routing itself. This notebook was tested with a pip environment that can be recreated with `requirements.txt` by running `pip install -r requirements.txt` (after creating a new environment).

The file `line_graph_routing.py` should be placed in the same folder as the current notebook.

```
import line_graph_routing as lgr
import networkx as nx
```

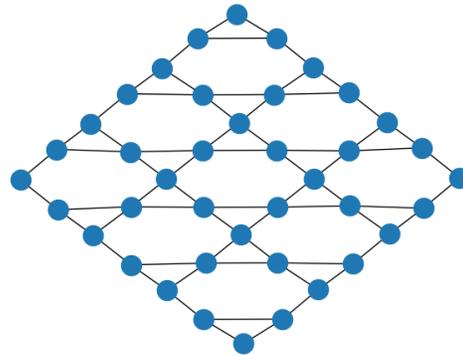
2 Kagome to heavy-hex

2.1 Random

Create a random circuit on a patch of the kagome lattice of 3×3 unit cells and show the circuit's coupling graph. With probability $2/5$, a CNOT is placed along an edge of the connectivity graph. With a probability $3/5$ a gate from $\{H,S,T\}$ is chosen uniformly at random and placed at a random node.

```
lg = lgr.kagome(3, 3)
qc = lgr.random_circuit(lg, 10**4)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())
```

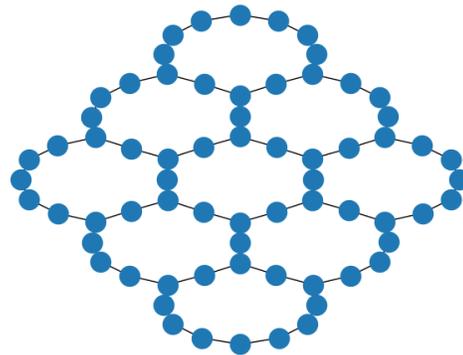
755



Route the circuit to a circuit with heavy-hex coupling graph.

```
qc = lgr.line_graph_route(qc)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())
```

1337



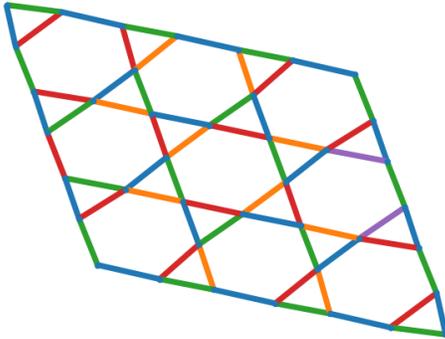
2.2 Quantum simulation

We base our circuits on edge colorings of the (kagome) lattice by identifying every color with a layer of HEIS gates. One of the colors (color '0') doubles as a color specifying the initial state by indicating along which edges singlet states are placed. The entire circuit is repeated p times, excluding initial state preparation. Every HEIS gets its own parameter. These parameters can later be bound to specific values to obtain circuits for dynamical quantum simulation or for simulated adiabatic state preparation.

First, create and show an edge coloring of the kagome lattice.

```
lg = lgr.edge_coloring(lg)
lgr.draw_edge_coloring(lg)
```

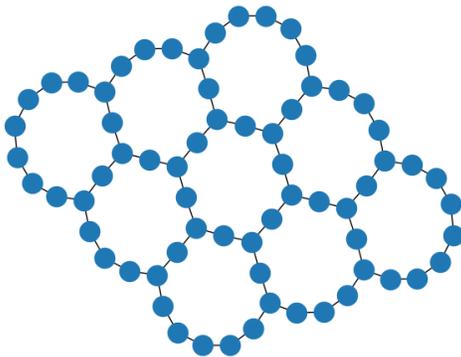
Matching is perfect
Edge coloring is not minimal



Create the associated circuit, route it to heavy-hex hardware and show the coupling graph of the routed circuit.

```
p = 1
qc = lgr.heis_circuit(lg, p)
print(qc.depth())
qc = lgr.line_graph_route(qc)
print(qc.depth())
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
```

6
15

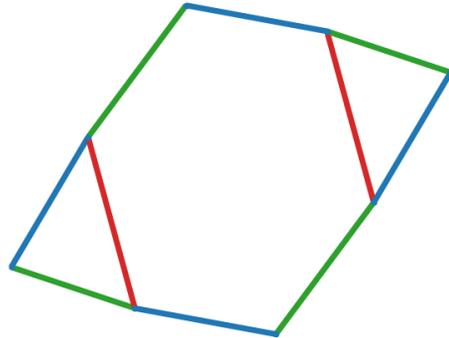


Since the above circuit has the connectivity of a 2D graph, the corresponding quantum circuit diagram will not be very insightful. However, for a single unit cell of the kagome lattice, the routed quantum circuit becomes a circuit on a circle, which allows for a clear representation as a quantum circuit diagram. We first

create an edge coloring of the unit cell patch.

```
lg = lgr.kagome(1, 1)
lg = lgr.edge_coloring(lg)
lgr.draw_edge_coloring(lg)
```

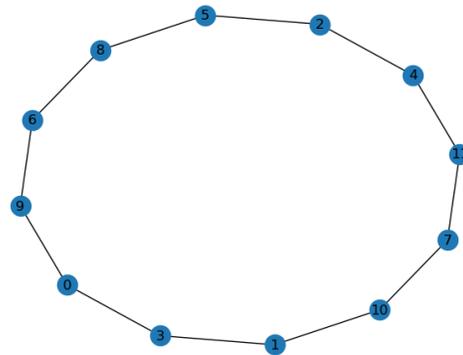
Matching is perfect
Edge coloring is not minimal



Create the HEIS circuit used on this coloring, map it to heavy-hex hardware, and show the coupling graph of the routed circuit.

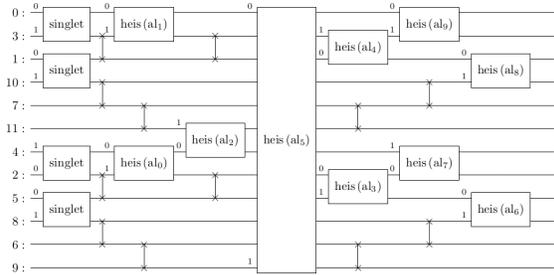
```
p = 1
qc = lgr.heis_circuit(lg, p)
print(qc.depth())
qc = lgr.line_graph_route(qc)
print(qc.depth())
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg, with_labels = True)
```

4
7



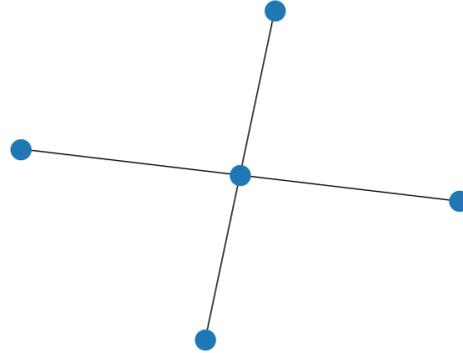
Show the circuit diagram of the routed circuit, with parameters `al_i`.

```
wo = [0, 3, 1, 10, 7, 11, 4, 2, 5, 8, 6, 9] # In the circuit diagram, place qubits in this order.
qc.draw('latex', wire_order = wo)
```



```
print(qc.depth())
```

108



The circuit depth can be reduced further by replacement of the initial and final SWAP gates between qubits (10,7) and (8,6) by a relabeling of those qubits.

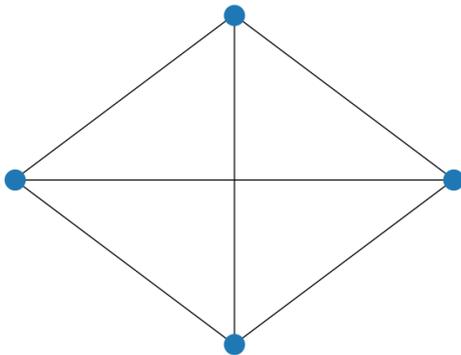
3 Complete graph to star graph

3.1 Random

Create a random circuit on the complete graph of four nodes and show the circuit's coupling graph.

```
n = 5
lg = nx.complete_graph(4)
qc = lgr.random_circuit(lg, 10**2)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())
```

56



Route the circuit to a circuit with star-graph connectivity.

```
qc = lgr.line_graph_route(qc)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
```

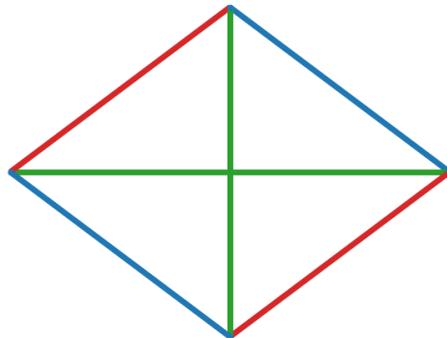
3.2 Quantum simulation

As before, circuits are defined by identifying every color with a layer of HEIS-gates. For more details, see the `kagome` to `heavy-hex` section.

Create and show an edge coloring of the complete graph.

```
lg = lgr.edge_coloring(lg)
lgr.draw_edge_coloring(lg)
```

Matching is perfect
Edge coloring is not minimal

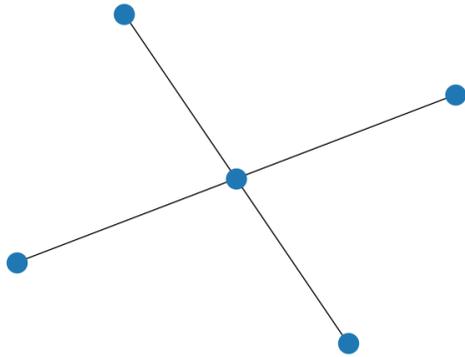


Create the associated circuit, route it to heavy-hex hardware, and show the coupling graph of the routed circuit.

```
p = 1
qc = lgr.heis_circuit(lg, p)
print(qc.depth())
qc = lgr.line_graph_route(qc)
```

```
print(qc.depth())
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
```

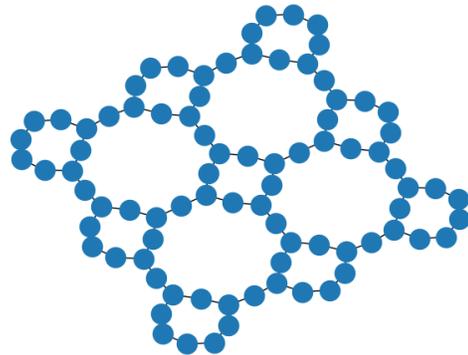
4
20



Route the circuit to a circuit with heavy-square-octagon connectivity.

```
qc = lgr.line_graph_route(qc)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())
```

877



We do not show the circuit diagram in this case because the routed circuit is not a circuit on a line.

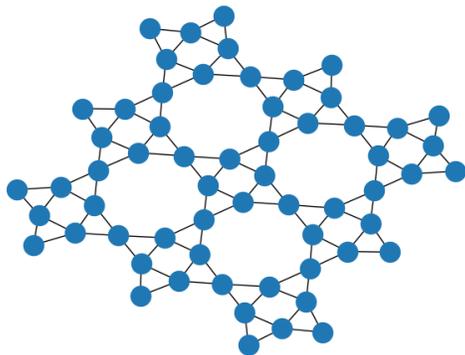
4 Shuriken to heavy square-octagon

4.1 Random

Create a random circuit on a patch of the shuriken lattice of 3×3 unit cells.

```
n = 5
lg = lgr.shuriken(3, 3)
qc = lgr.random_circuit(lg, 10**4)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())
```

517



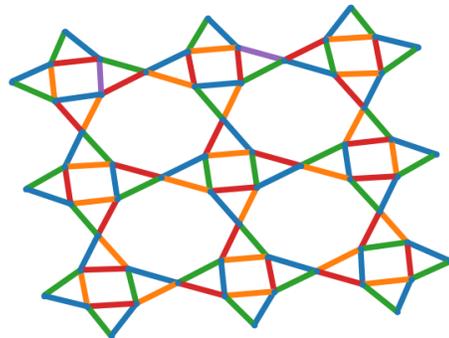
4.2 Quantum simulation

As before, circuits are defined by identifying every color with a layer of HEIS-gates. For more details, see the [kagome to heavy-hex](#) section.

Create and show an edge coloring of the shuriken lattice.

```
lg = lgr.edge_coloring(lg)
lgr.draw_edge_coloring(lg)
```

Matching is perfect
Edge coloring is not minimal



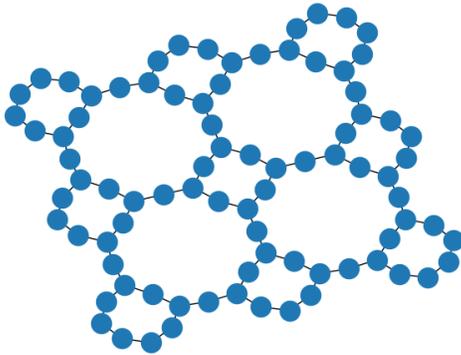
Create the associated circuit, route it to heavy-square-octagon hardware, and show the coupling graph of the routed circuit.

```

p = 1
qc = lgr.heis_circuit(lg, p)
print(qc.depth())
qc = lgr.line_graph_route(qc)
print(qc.depth())
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)

```

6
12

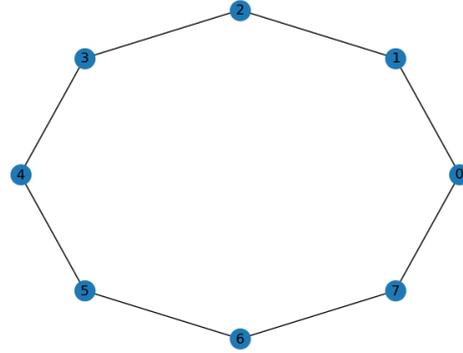


```

print(qc.depth())
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg, with_labels = True)

```

5
9



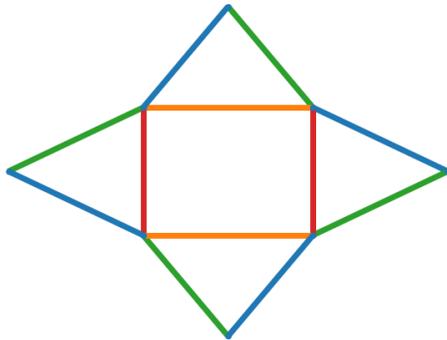
Again, the resulting circuit diagram will not be very insightful, but it will be for a single-unit cell patch of the shuriken lattice.

```

lg = lgr.shuriken(1, 1)
lg = lgr.edge_coloring(lg)
lgr.draw_edge_coloring(lg)

```

Matching is perfect
Edge coloring is minimal

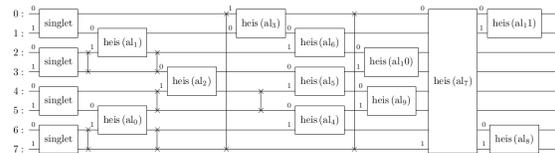


```

p = 1
qc = lgr.heis_circuit(lg, p)
print(qc.depth())
qc = lgr.line_graph_route(qc)

```

```
qc.draw('latex')
```



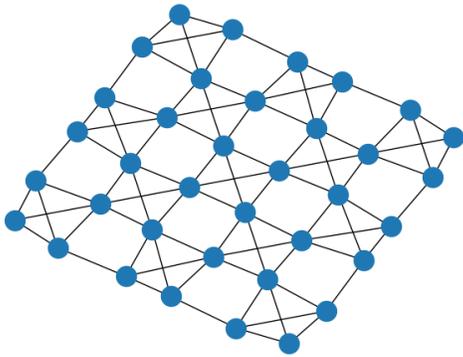
5 Checkerboard to heavy-square

```

m = 2.5 # For the checkerboard lattice, specify dimensions in nodes by nodes.
lg = lgr.checkerboard(m, m)
qc = lgr.random_circuit(lg, 10**4)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())

```

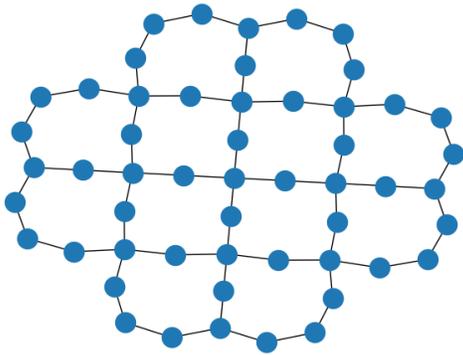
872



Route the circuit to a circuit with a heavy-square coupling graph.

```
qc = lgr.line_graph_route(qc)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())
```

1622



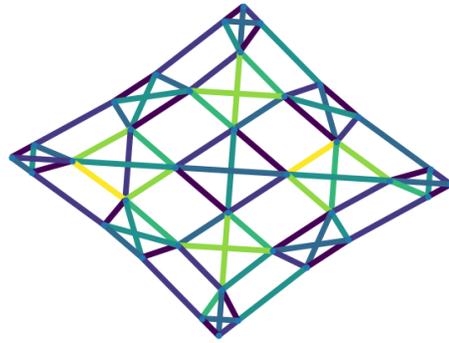
5.1 Quantum simulation

As before, circuits are defined by identifying every color with a layer of HEIS-gates. For more details, see the `kagome` to `heavy-hex` section.

Create and show an edge coloring of the checkerboard lattice

```
lg = lgr.edge_coloring(lg)
lgr.draw_edge_coloring(lg, spectral=True)
↳ # Use spectral method to find location
↳ of nodes.
```

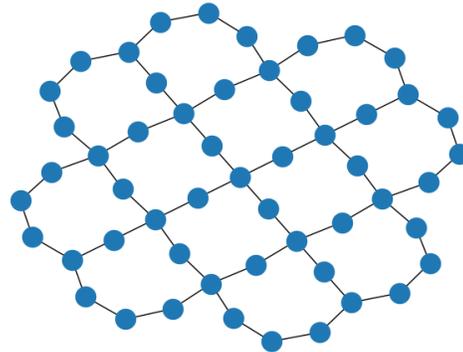
Matching is perfect
Edge coloring is not minimal



Create the associated circuit, route it to heavy-square-octagon hardware, and show the coupling graph of the routed circuit.

```
p = 1
qc = lgr.heis_circuit(lg, p)
print(qc.depth())
qc = lgr.line_graph_route(qc)
print(qc.depth())
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
```

8
33



6 Random line graph to random heavy graph

6.1 Random

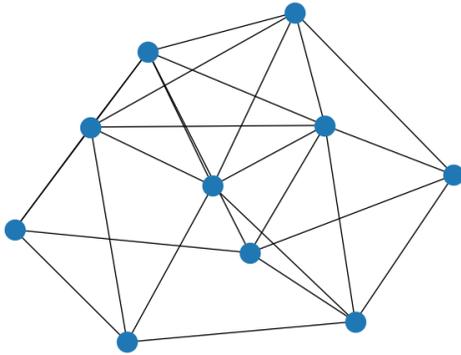
Create a random circuit on a random graph with 6 nodes and show the circuit's coupling graph. For details on `radom_line_graph` generation, see its function definition in `line_graph_routing.py`.

```

n = 6
lg = lgr.random_line_graph(6)
qc = lgr.random_circuit(lg, 10**3)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())

```

287



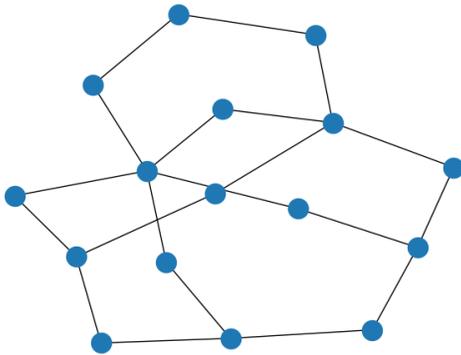
Route the circuit to a circuit with the associated heavy connectivity.

```

qc = lgr.line_graph_route(qc)
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)
print(qc.depth())

```

576



6.2 Quantum simulation

As before, circuits are defined by identifying every color with a layer of HEIS-gates. For more details, see the `kagome` to `heavy-hex` section.

Create and show an edge coloring of random graph. The method we use to find a perfect matching (needed for initial state preparation) is limited and

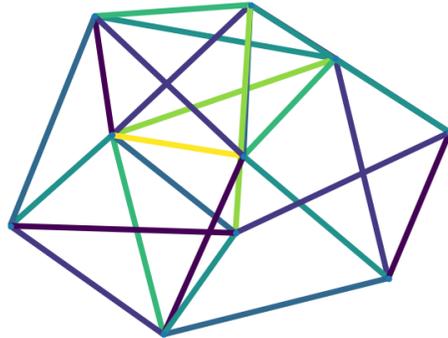
may not find a perfect matching even if it exists. If a perfect matching is not found, try to create another random line graph (i.e., evaluate the two cells above) or use more sophisticated (or brute-force) methods to find a perfect matching.

```

lg = lgr.edge_coloring(lg)
lgr.draw_edge_coloring(lg)

```

Matching is perfect
Edge coloring is not minimal



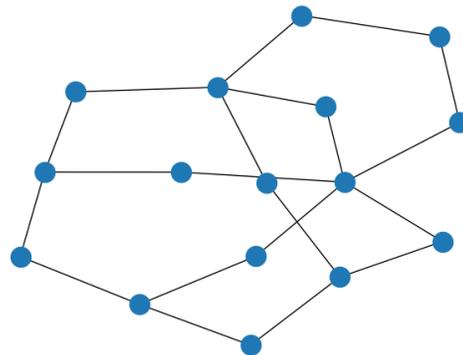
Create the associated circuit, route it to heavy-hex hardware and show the coupling graph of the routed circuit.

```

p = 1
qc = lgr.heis_circuit(lg, p)
print(qc.depth())
qc = lgr.line_graph_route(qc)
print(qc.depth())
cg = lgr.coupling_graph(qc)
nx.draw_kamada_kawai(cg)

```

8
32



We do not show the circuit diagram in this case because the routed circuit is not a circuit on a line.

7 Benchmarking

We benchmark line graph routing by performing the above routing tasks (but for larger unit cells) using both line-graph qubit routing and all methods available in qiskit. These methods are ‘basic’, ‘lookahead’, ‘stochastic’, and ‘sabre’ [1].

The benchmarking settings are specified by the following options: - **name** The name of the virtual graph, either **kagome**, **shuriken** or **complete**. - **size**. In case of **kagome** and **shuriken**: the size of the patch in unit cells by unit cells. In case of **complete**: the number of nodes of the complete graph. - **circuit type**. Either **quantum simulation** or **random**, as presented in this notebook. - **p** In case of **kagome** and **shuriken**: the number of cycles of the circuit. In case of **complete**: the number of random gates from the set **H,T,S,CNOT**. - **repetitions**. The number of runs for the methods **sabre** and **stochastic**. The methods **line-graph** and **basic** are deterministic and hence only run once. Correspondingly, the reported **total time** pertains to the time taken for this single run in case of the latter two methods. - **optimization_level**. Either 0, 1, 2, or 3. This specifies the optimization level used for the routing methods implemented in qiskit [2]. This parameter is passed directly to Qiskit’s transpiler [1]. - **methods**. The methods to benchmark line graph transpilation against. Must be a list containing elements from [**'sabre'**,**'basic'**,**'lookahead'**,**stochastic**]. These methods are passed directly to Qiskit’s transpiler [1].

The methods **sabre** and **stochastic** are probabilistic, achieving a different routing each time they are run, and hence achieve different performance characteristics with each run. We therefore run these methods *repetitions* times and report the average, confidence interval, and best performance out of these runs. Error bars on the data show the (symmetrized) 95% confidence interval and are obtained by bootstrapping the data. The error interval for **num_qubits** is sometimes given by **nan** because in those cases the number of qubits was equal for all runs. The routing methods **line-graph** and **basic** are deterministic and for these we enforce **repetitions=1**.

We consider the following performance characteristics. - **method** The routing method. - **av. n_swaps** The average number of swaps obtained among the **repetitions** runs of the routing method. - **min n_swap** The number of swaps of the run that achieved the lowest depth. - **av. depth** The average depth of the routed circuits among the **repetitions** runs of the routing method. We focus on the performance of routing so none of the gates in any of the routing methods are compiled into hardware native gates. That is, for the purposes of assessing routing performance, we assume the gate set **SWAP, HEIS, H, X, Z CNOT** for the quantum simulation circuits. For the random circuits we assume **SWAP, CNOT, H, S, T**. - **min depth** The minimum depth among the **repetitions** runs of the routing method. - **av. n_qubits** The average number obtained among the **repetitions** runs of the routing method. - **total time** The total wall clock time (in seconds) needed to perform all **repetitions** runs of the routing method. - **av. time** The average (minimum) wall clock time of the **repetitions** routing runs. - **min. time** The number wall clock run time of the run that achieved the lowest depth.

[1] Qiskit 0.43.0 documentation, <https://qiskit.org/documentation/stubs/qiskit.compiler.transpile.html>, accessed 11h May 2023.

[2] https://github.com/Qiskit/qiskit-terra/tree/main/qiskit/transpiler/preset_passmanagers

7.0.1 Quantum simulation, kagome and shuriken, against SABRE

```
import line_graph_routing as lgr # Loading these makes these cells stand-alone
import pickle

settings=[]
for name in ['kagome','shuriken']:
    for side in range(1,9,2):
        for p in [1,8,16]:
            for optimization_level in range(4):
                setting={'name':name,
                        'size': (side,side),
```

```

        'circuit_type': 'quantum_simulation',
        'p': p,
        'repetitions' : 16,
        'optimization_level' : optimization_level,
        'methods' : ['sabre']
    }
    settings.append(setting)

## Uncomment to rerun benchmarks. This takes a couple of hours.
results=[]
#for setting in settings:
#     result=lgr.benchmark(**setting)
#     results.append(result)
#     lgr.print_benchmark(result)
#
#with open('benchmark_data/kagome_shuriken.pkl','wb') as f:
#     pickle.dump(results,f)

#Load previously obtained results from disk and show them.
import pickle
with open('benchmark_data/kagome_shuriken.pkl','rb') as f:
    results=pickle.load(f)

for result in results:
    lgr.print_benchmark(result)

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 12 ± 0.0       12 7 ± 0.0    7 12 ± 0.0   12           0.08 0.08 ± 0.0     0.08
sabre     16 ± 2.06     6 12.38 ± 2.09 6 10.25 ± 1.0 12           0.7 0.04 ± 0.06    0.02
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 12 ± 0.0       12 7 ± 0.0    7 12 ± 0.0   12           0.08 0.08 ± 0.0     0.08
sabre     10 ± 0.81     6 6.44 ± 0.34 6 8.75 ± 0.75 8           0.32 0.02 ± nan     0.02
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 12 ± 0.0       12 7 ± 0.0    7 12 ± 0.0   12           0.08 0.08 ± 0.0     0.08
sabre     6 ± nan       6 6.0 ± nan  6 8.0 ± nan  8           0.33 0.02 ± 0.0     0.02
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 12 ± 0.0      12 7 ± 0.0    7 12 ± 0.0   12           0.08 0.08 ± 0.0     0.08
sabre     6 ± nan      6 6.0 ± nan   6 8.0 ± nan   8           0.75 0.05 ± 0.02    0.04
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 96 ± 0.0      96 49 ± 0.0    49 12 ± 0.0   12           0.67 0.67 ± 0.0     0.67
sabre     99 ± 7.66    48 52.94 ± 5.28 41 11.5 ± 0.75 12           1.39 0.09 ± 0.02    0.08
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 96 ± 0.0      96 49 ± 0.0    49 12 ± 0.0   12           0.64 0.64 ± 0.0     0.64
sabre     48 ± 8.62    48 42.88 ± 3.28 41 8.5 ± 0.75  8           1.67 0.1 ± 0.02    0.09
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 96 ± 0.0      96 49 ± 0.0    49 12 ± 0.0   12           0.64 0.64 ± 0.0     0.64
sabre     48 ± 7.19    48 41.56 ± 1.41 41 8.25 ± 0.62 8           2.07 0.13 ± 0.02    0.11
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 96 ± 0.0      96 49 ± 0.0    49 12 ± 0.0   12           0.64 0.64 ± 0.0     0.64
sabre     48 ± nan     48 41.0 ± nan   41 8.0 ± nan   8           5.07 0.32 ± 0.02    0.29
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 192 ± 0.0     192 97 ± 0.0    97 12 ± 0.0   12           1.29 1.29 ± 0.0     1.29
sabre     100 ± 18.19   96 99.38 ± 9.69 81 11.25 ± 0.75 12           2.78 0.17 ± 0.02    0.15
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 192 ± 0.0      192 97 ± 0.0   97 12 ± 0.0   12            1.31 1.31 ± 0.0      1.31
sabre     96 ± 17.62     96 86.69 ± 8.53 81 8.5 ± 0.75  8            3.18 0.2 ± 0.03     0.17
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 192 ± 0.0      192 97 ± 0.0   97 12 ± 0.0   12            1.3  1.3 ± 0.0       1.3
sabre     96 ± nan       96 81.0 ± nan  81 8.0 ± nan   8            3.74 0.23 ± 0.02    0.21
-----

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 192 ± 0.0      192 97 ± 0.0   97 12 ± 0.0   12            1.29 1.29 ± 0.0     1.29
sabre     96 ± nan       96 81.0 ± nan  81 8.0 ± nan   8            10.26 0.64 ± 0.02   0.67
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 112 ± 0.0      112 15 ± 0.0   15 68 ± 0.0   68            0.6  0.6 ± 0.0       0.6
sabre     307 ± 3.44     286 72.62 ± 3.5 63 55.0 ± 1.66 55            1.83 0.11 ± 0.02   0.09
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 112 ± 0.0      112 15 ± 0.0   15 68 ± 0.0   68            0.6  0.6 ± 0.0       0.6
sabre     85 ± 3.56      85 27.38 ± 1.34 22 45.25 ± 1.66 52            1.8  0.11 ± 0.02    0.1
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 112 ± 0.0      112 15 ± 0.0   15 68 ± 0.0   68            0.6  0.6 ± 0.0       0.6
sabre     96 ± 4.19      82 26.81 ± 2.41 20 46.19 ± 1.12 47            2.62 0.16 ± 0.02   0.15
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 112 ± 0.0      112 15 ± 0.0    15 68 ± 0.0    68           0.5 0.5 ± 0.0      0.5
sabre     64 ± 2.34      67 21.44 ± 1.94 14 46.0 ± 1.75 47           7.77 0.49 ± 0.02    0.55
-----

name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 896 ± 0.0      896 113 ± 0.0    113 68 ± 0.0    68           4.25 4.25 ± 0.0      4.25
sabre     1361 ± 33.51   1203 312.5 ± 9.31 287 59.5 ± 1.75 51           10.83 0.68 ± 0.03    0.69
-----

name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 896 ± 0.0      896 113 ± 0.0    113 68 ± 0.0    68           4.39 4.39 ± 0.0      4.39
sabre     625 ± 19.67    613 192.38 ± 14.0 149 45.19 ± 1.28 44           10.99 0.69 ± 0.04    0.76
-----

name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 896 ± 0.0      896 113 ± 0.0    113 68 ± 0.0    68           4.42 4.42 ± 0.0      4.42
sabre     621 ± 9.75     621 182.06 ± 9.79 152 43.56 ± 1.4 44           14.53 0.91 ± 0.04    0.95
-----

name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 896 ± 0.0      896 113 ± 0.0    113 68 ± 0.0    68           4.23 4.23 ± 0.0      4.23
sabre     601 ± 10.06    555 176.25 ± 7.62 149 43.44 ± 0.72 43           45.11 2.82 ± 0.04    2.74
-----

name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1792 ± 0.0     1792 225 ± 0.0    225 68 ± 0.0    68           8.45 8.45 ± 0.0      8.45
sabre     2627 ± 55.91   2304 553.94 ± 12.78 509 60.94 ± 2.12 63           20.65 1.29 ± 0.05    1.29
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1792 ± 0.0     1792 225 ± 0.0     225 68 ± 0.0     68           8.87 8.87 ± 0.0     8.87
sabre    1213 ± 28.0    1213 388.12 ± 18.66 318 43.75 ± 1.16 44           20.66 1.29 ± 0.05    1.35
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1792 ± 0.0     1792 225 ± 0.0     225 68 ± 0.0     68           8.4 8.4 ± 0.0     8.4
sabre    1215 ± 14.41   1142 377.69 ± 19.52 311 44.06 ± 1.19 49           28.01 1.75 ± 0.01    1.79
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1792 ± 0.0     1792 225 ± 0.0     225 68 ± 0.0     68           8.57 8.57 ± 0.0     8.57
sabre    1172 ± 15.38   1144 342.25 ± 13.38 290 43.56 ± 1.0 43           86.44 5.4 ± 0.05    5.53
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 270 ± 0.0     270 14 ± 0.0     14 164 ± 0.0     164          1.48 1.48 ± 0.0     1.48
sabre    1247 ± 9.78    1247 132.81 ± 3.09 123 128.31 ± 1.72 125          5.12 0.32 ± 0.03    0.37
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 270 ± 0.0     270 14 ± 0.0     14 164 ± 0.0     164          1.47 1.47 ± 0.0     1.47
sabre    338 ± 14.72    345 50.94 ± 3.44  38 110.69 ± 1.34 109          5.47 0.34 ± 0.02    0.3
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 270 ± 0.0     270 14 ± 0.0     14 164 ± 0.0     164          1.47 1.47 ± 0.0     1.47
sabre    364 ± 10.22    350 48.81 ± 4.88  38 110.19 ± 2.16 115          9.7 0.61 ± 0.03    0.56
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 270 ± 0.0      270 14 ± 0.0    14 164 ± 0.0    164          1.47 1.47 ± 0.0      1.47
sabre     274 ± 9.62      269 37.44 ± 3.25 30 106.25 ± 1.62 108          36.54 2.28 ± 0.03     2.33
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2160 ± 0.0     2160 98 ± 0.0     98 164 ± 0.0     164          10.96 10.96 ± 0.0     10.96
sabre     6068 ± 87.86   5579 557.62 ± 12.5 511 134.88 ± 2.03 134          27.49 1.72 ± 0.06     1.55
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2160 ± 0.0     2160 98 ± 0.0     98 164 ± 0.0     164          11.01 11.01 ± 0.0     11.01
sabre     2124 ± 45.1    2056 341.69 ± 14.66 286 106.5 ± 2.38 118          30.77 1.92 ± 0.05     1.98
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2160 ± 0.0     2160 98 ± 0.0     98 164 ± 0.0     164          10.93 10.93 ± 0.0     10.93
sabre     1894 ± 32.99   1953 324.31 ± 22.53 265 106.62 ± 2.47 118          44.89 2.81 ± 0.03     2.8
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2160 ± 0.0     2160 98 ± 0.0     98 164 ± 0.0     164          10.88 10.88 ± 0.0     10.88
sabre     1869 ± 23.79   1758 282.88 ± 13.62 236 104.12 ± 1.91 109          158.7 9.92 ± 0.08     9.89
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4320 ± 0.0     4320 194 ± 0.0    194 164 ± 0.0     164          21.85 21.85 ± 0.0     21.85
sabre     10420 ± 144.88 9561 979.94 ± 18.51 919 133.62 ± 2.94 124          56.99 3.56 ± 0.11     3.42
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4320 ± 0.0     4320 194 ± 0.0     194 164 ± 0.0     164          21.73 21.73 ± 0.0     21.73
sabre     4267 ± 109.67  4443 682.12 ± 23.85  627 107.31 ± 3.03  110          58.8  3.68 ± 0.11     3.58
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4320 ± 0.0     4320 194 ± 0.0     194 164 ± 0.0     164          21.86 21.86 ± 0.0     21.86
sabre     3972 ± 57.03   3804 628.44 ± 28.22  526 107.31 ± 2.53  109          85.09 5.32 ± 0.1      5.11
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4320 ± 0.0     4320 194 ± 0.0     194 164 ± 0.0     164          21.56 21.56 ± 0.0     21.56
sabre     3716 ± 30.9    3656 592.5 ± 13.72   562 104.94 ± 1.25  108          296.62 18.54 ± 0.15    18.2
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 498 ± 0.0      498 16 ± 0.0       16 300 ± 0.0     300          2.8  2.8 ± 0.0       2.8
sabre     3534 ± 31.01   3534 208.06 ± 8.72   175 238.75 ± 3.75  239          10.84 0.68 ± 0.03     0.71
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 498 ± 0.0      498 16 ± 0.0       16 300 ± 0.0     300          2.86 2.86 ± 0.0     2.86
sabre     1119 ± 28.5    1092 94.94 ± 7.28    70 202.19 ± 3.62  210          13.88 0.87 ± 0.03     0.8
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 498 ± 0.0      498 16 ± 0.0       16 300 ± 0.0     300          2.88 2.88 ± 0.0     2.88
sabre     994 ± 28.53    902 83.25 ± 4.84    68 200.81 ± 3.18  194          30.74 1.92 ± 0.04     1.95
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 498 ± 0.0      498 16 ± 0.0      16 300 ± 0.0      300           2.87 2.87 ± 0.0      2.87
sabre     868 ± 28.66    737 66.25 ± 3.94   53 194.75 ± 2.84   189           135.51 8.47 ± 0.05     8.39
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3984 ± 0.0     3984 114 ± 0.0      114 300 ± 0.0      300           21.51 21.51 ± 0.0     21.51
sabre     13056 ± 225.0  13056 766.81 ± 18.31 686 247.75 ± 2.97  251           61.25 3.83 ± 0.12    4.01
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3984 ± 0.0     3984 114 ± 0.0      114 300 ± 0.0      300           20.92 20.92 ± 0.0     20.92
sabre     5532 ± 133.17  4986 569.0 ± 30.89  426 199.12 ± 3.19  207           68.32 4.27 ± 0.11    4.14
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3984 ± 0.0     3984 114 ± 0.0      114 300 ± 0.0      300           20.7  20.7 ± 0.0      20.7
sabre     4758 ± 150.83  4644 511.88 ± 33.91 429 194.75 ± 3.38  199           118.43 7.4 ± 0.07     7.32
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3984 ± 0.0     3984 114 ± 0.0      114 300 ± 0.0      300           20.74 20.74 ± 0.0     20.74
sabre     4702 ± 106.77  4300 460.69 ± 27.72 375 191.88 ± 2.75  200           504.51 31.53 ± 0.21   31.37
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 7968 ± 0.0     7968 226 ± 0.0      226 300 ± 0.0      300           41.85 41.85 ± 0.0     41.85
sabre     22309 ± 375.19 21499 1238.75 ± 23.39 1182 248.75 ± 2.74  242           116.84 7.3 ± 0.19     7.39
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 7968 ± 0.0     7968 226 ± 0.0   226 300 ± 0.0   300           41.09 41.09 ± 0.0    41.09
sabre     9497 ± 345.67  9370 1049.75 ± 55.28  888 194.06 ± 3.81  198           134.06 8.38 ± 0.28    8.61
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 7968 ± 0.0     7968 226 ± 0.0   226 300 ± 0.0   300           41.59 41.59 ± 0.0    41.59
sabre     9742 ± 181.92  9337 978.25 ± 31.91  856 197.69 ± 3.94  206           227.53 14.22 ± 0.41   15.89
-----

```

```

-----
name = kagome, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 7968 ± 0.0     7968 226 ± 0.0   226 300 ± 0.0   300           42.57 42.57 ± 0.0    42.57
sabre     8452 ± 100.74  8486 899.0 ± 24.78  790 192.25 ± 2.62  200           886.06 55.38 ± 1.31   60.71
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 8 ± 0.0        8 9 ± 0.0     9 8 ± 0.0    8           0.09 0.09 ± 0.0     0.09
sabre     8 ± nan        8 8.88 ± 0.19  8 8.0 ± nan   8           0.32 0.02 ± nan     0.02
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 8 ± 0.0        8 9 ± 0.0     9 8 ± 0.0    8           0.09 0.09 ± 0.0     0.09
sabre     6 ± nan        6 7.5 ± 0.25  7 8.0 ± nan   8           0.32 0.02 ± nan     0.02
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 8 ± 0.0        8 9 ± 0.0     9 8 ± 0.0    8           0.09 0.09 ± 0.0     0.09
sabre     6 ± nan        6 7.5 ± 0.25  7 8.0 ± nan   8           0.32 0.02 ± nan     0.02
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 8 ± 0.0       8 9 ± 0.0     9 8 ± 0.0     8             0.09 0.09 ± 0.0     0.09
sabre     6 ± nan       6 7.62 ± 0.22  7 8.0 ± nan    8             0.64 0.04 ± nan     0.04
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 64 ± 0.0      64 65 ± 0.0    65 8 ± 0.0     8             0.74 0.74 ± 0.0     0.74
sabre     74 ± 1.56     64 67.94 ± 2.31 61 8.0 ± nan    8             1.66 0.1 ± 0.02     0.09
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 64 ± 0.0      64 65 ± 0.0    65 8 ± 0.0     8             0.73 0.73 ± 0.0     0.73
sabre     62 ± nan      62 60.75 ± 0.53 59 8.0 ± nan    8             1.82 0.11 ± 0.02     0.1
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 64 ± 0.0      64 65 ± 0.0    65 8 ± 0.0     8             0.77 0.77 ± 0.0     0.77
sabre     62 ± nan      62 60.19 ± 0.69 58 8.0 ± nan    8             2.04 0.13 ± 0.02     0.13
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 64 ± 0.0      64 65 ± 0.0    65 8 ± 0.0     8             0.75 0.75 ± 0.0     0.75
sabre     62 ± nan      62 60.69 ± 0.66 58 8.0 ± nan    8             5.7  0.36 ± 0.03     0.41
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 128 ± 0.0     128 129 ± 0.0   129 8 ± 0.0     8             1.5  1.5 ± 0.0       1.5
sabre     138 ± 2.94    138 140.56 ± 3.66 128 8.0 ± nan    8             3.42 0.21 ± 0.03     0.18
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 128 ± 0.0      128 129 ± 0.0    129 8 ± 0.0      8             1.53 1.53 ± 0.0      1.53
sabre     126 ± nan      126 121.12 ± 1.06 119 8.0 ± nan    8             3.57 0.22 ± 0.02    0.2
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 128 ± 0.0      128 129 ± 0.0    129 8 ± 0.0      8             1.58 1.58 ± 0.0      1.58
sabre     126 ± nan      126 121.06 ± 0.88 118 8.0 ± nan    8             4.24 0.26 ± 0.03    0.24
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 128 ± 0.0      128 129 ± 0.0    129 8 ± 0.0      8             1.5  1.5 ± 0.0        1.5
sabre     126 ± nan      126 120.81 ± 0.94 116 8.0 ± nan    8             11.06 0.69 ± 0.02    0.71
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 130 ± 0.0      130 12 ± 0.0      12 84 ± 0.0      84            1    1.0 ± 0.0        1
sabre     469 ± 8.24     460 67.94 ± 2.41 60 83.75 ± 0.22 84            2.68 0.17 ± 0.02    0.14
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 130 ± 0.0      130 12 ± 0.0      12 84 ± 0.0      84            0.89 0.89 ± 0.0      0.89
sabre     177 ± 6.53     187 34.5 ± 2.94   27 64.88 ± 1.19 67            2.79 0.17 ± 0.02    0.16
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 130 ± 0.0      130 12 ± 0.0      12 84 ± 0.0      84            0.89 0.89 ± 0.0      0.89
sabre     169 ± 7.66     123 34.5 ± 2.44   26 64.5 ± 1.0    63            4.43 0.28 ± 0.02    0.24
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 130 ± 0.0      130 12 ± 0.0    12 84 ± 0.0    84            1 1.0 ± 0.0        1
sabre    138 ± 6.41     94 29.44 ± 2.53 20 62.81 ± 0.84 63            14.29 0.89 ± 0.03    0.94
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1040 ± 0.0     1040 89 ± 0.0     89 84 ± 0.0     84            7.06 7.06 ± 0.0        7.06
sabre    2778 ± 36.62   2575 400.94 ± 8.12 362 83.81 ± 0.19 84            18.11 1.13 ± 0.03      1.16
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1040 ± 0.0     1040 89 ± 0.0     89 84 ± 0.0     84            6.92 6.92 ± 0.0        6.92
sabre    1268 ± 32.28   1039 254.25 ± 18.57 205 64.94 ± 1.41 63            16.06 1.0 ± 0.04       0.98
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1040 ± 0.0     1040 89 ± 0.0     89 84 ± 0.0     84            6.92 6.92 ± 0.0        6.92
sabre    1051 ± 26.45   926 215.75 ± 15.27 170 64.69 ± 1.41 65            22.5 1.41 ± 0.05      1.34
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1040 ± 0.0     1040 89 ± 0.0     89 84 ± 0.0     84            7.06 7.06 ± 0.0        7.06
sabre    966 ± 14.37     892 192.5 ± 13.59 145 64.44 ± 1.09 68            79.12 4.94 ± 0.07      5.05
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2080 ± 0.0     2080 177 ± 0.0    177 84 ± 0.0     84            13.7 13.7 ± 0.0        13.7
sabre    5067 ± 112.35  4490 774.94 ± 15.84 706 83.94 ± 0.16 84            31.53 1.97 ± 0.08      2.1
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2080 ± 0.0      2080 177 ± 0.0      177 84 ± 0.0      84          14.13 14.13 ± 0.0      14.13
sabre    2035 ± 50.28    1861 450.12 ± 38.78  302 65.19 ± 1.28   66          35.28 2.2 ± 0.06       2.19
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2080 ± 0.0      2080 177 ± 0.0      177 84 ± 0.0      84          13.71 13.71 ± 0.0      13.71
sabre    2080 ± 41.66    1890 398.25 ± 19.59  334 65.38 ± 0.94   63          44.87 2.8 ± 0.06       2.84
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2080 ± 0.0      2080 177 ± 0.0      177 84 ± 0.0      84          13.89 13.89 ± 0.0      13.89
sabre    1826 ± 35.68    1703 373.75 ± 26.89  303 64.44 ± 1.53   61          150.75 9.42 ± 0.09      9.59
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 418 ± 0.0      418 14 ± 0.0      14 240 ± 0.0      240          2.76 2.76 ± 0.0      2.76
sabre    1722 ± 13.92    1706 117.56 ± 4.03  100 237.69 ± 0.81   239          7.59 0.47 ± 0.03      0.43
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 418 ± 0.0      418 14 ± 0.0      14 240 ± 0.0      240          2.61 2.61 ± 0.0      2.61
sabre    993 ± 25.43     918 85.12 ± 4.62   70 179.31 ± 2.32   182          11.16 0.7 ± 0.03       0.8
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 418 ± 0.0      418 14 ± 0.0      14 240 ± 0.0      240          2.58 2.58 ± 0.0      2.58
sabre    940 ± 25.45     830 83.06 ± 7.81   54 179.06 ± 3.62   194          25.23 1.58 ± 0.02      1.55
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 418 ± 0.0      418  14 ± 0.0    14  240 ± 0.0    240           2.6  2.6 ± 0.0      2.6
sabre     702 ± 15.83    712  66.81 ± 6.6  45  176.12 ± 2.69  183          105.59  6.6 ± 0.04     6.58
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3344 ± 0.0     3344  105 ± 0.0   105  240 ± 0.0    240          19.21  19.21 ± 0.0    19.21
sabre     10085 ± 62.18  10397  654.81 ± 12.53  619  238.31 ± 0.59  238          53.61  3.35 ± 0.12    3.66
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3344 ± 0.0     3344  105 ± 0.0   105  240 ± 0.0    240          19.42  19.42 ± 0.0    19.42
sabre     4949 ± 172.47  4147  581.88 ± 36.5  461  178.88 ± 2.91  173          60.32  3.77 ± 0.09    3.72
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3344 ± 0.0     3344  105 ± 0.0   105  240 ± 0.0    240          19.65  19.65 ± 0.0    19.65
sabre     4321 ± 153.74  3792  566.06 ± 43.95  414  173.81 ± 2.09  171          109.02  6.81 ± 0.22    7.04
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 3344 ± 0.0     3344  105 ± 0.0   105  240 ± 0.0    240          19.33  19.33 ± 0.0    19.33
sabre     3405 ± 65.66   3312  408.12 ± 21.62  343  170.62 ± 2.41  165          429.48  26.84 ± 0.98   25.93
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6688 ± 0.0     6688  209 ± 0.0   209  240 ± 0.0    240          38.44  38.44 ± 0.0    38.44
sabre     18976 ± 105.01  18874  1155.94 ± 14.25  1095  238.88 ± 0.38  238          108.03  6.75 ± 0.22    6.3
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6688 ± 0.0      6688 209 ± 0.0      209 240 ± 0.0      240          38.54 38.54 ± 0.0      38.54
sabre    8889 ± 336.48   6839 1004.38 ± 82.18  832 174.88 ± 2.72  176          111.6 6.98 ± 0.17      6.8
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6688 ± 0.0      6688 209 ± 0.0      209 240 ± 0.0      240          38.42 38.42 ± 0.0      38.42
sabre    6943 ± 155.17   6914 836.81 ± 46.97  673 173.06 ± 1.69  173          183.47 11.47 ± 0.2      12.22
-----

```

```

-----
name = shuriken, size = (5, 5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6688 ± 0.0      6688 209 ± 0.0      209 240 ± 0.0      240          38.14 38.14 ± 0.0      38.14
sabre    6644 ± 94.12    6326 667.75 ± 19.72  599 170.81 ± 2.19  165          693.94 43.37 ± 1.15     43.48
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 850 ± 0.0      850 14 ± 0.0      14 476 ± 0.0      476          5.19 5.19 ± 0.0      5.19
sabre    4169 ± 24.72    4173 143.25 ± 4.62  127 471.81 ± 1.22  474          16.26 1.02 ± 0.05     1.13
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 850 ± 0.0      850 14 ± 0.0      14 476 ± 0.0      476          5.2 5.2 ± 0.0      5.2
sabre    2783 ± 64.06    2792 196.44 ± 13.5  154 343.5 ± 3.28  349          48.94 3.06 ± 0.24     2.78
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 850 ± 0.0      850 14 ± 0.0      14 476 ± 0.0      476          5.31 5.31 ± 0.0      5.31
sabre    2665 ± 65.58    2535 170.75 ± 14.04  127 350.5 ± 5.34  352          132.45 8.28 ± 0.26     7.5
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 850 ± 0.0      850 14 ± 0.0    14 476 ± 0.0    476          5.22 5.22 ± 0.0      5.22
sabre    2079 ± 57.9    1891 124.12 ± 8.31 93 342.06 ± 3.19 354          628.32 39.27 ± 0.78    37.63
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6800 ± 0.0      6800 105 ± 0.0    105 476 ± 0.0    476          38.58 38.58 ± 0.0     38.58
sabre    25487 ± 146.92 25288 875.19 ± 21.34 785 474.06 ± 1.12 475          111.3 6.96 ± 0.17     6.82
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6800 ± 0.0      6800 105 ± 0.0    105 476 ± 0.0    476          38.22 38.22 ± 0.0     38.22
sabre    14038 ± 501.85 11572 1110.0 ± 72.38 836 349.44 ± 5.69 345          167.86 10.49 ± 0.26    10.87
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6800 ± 0.0      6800 105 ± 0.0    105 476 ± 0.0    476          37.94 37.94 ± 0.0     37.94
sabre    11889 ± 326.3  12192 1092.25 ± 51.37 922 343.62 ± 5.27 347          352.13 22.01 ± 0.34    20.89
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 6800 ± 0.0      6800 105 ± 0.0    105 476 ± 0.0    476          38.25 38.25 ± 0.0     38.25
sabre    9431 ± 295.31  9200 791.69 ± 50.65 589 336.75 ± 4.19 333          1612.88 100.8 ± 1.84    99.38
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 13600 ± 0.0     13600 209 ± 0.0    209 476 ± 0.0    476          75.38 75.38 ± 0.0     75.38
sabre    45512 ± 312.53 44697 1457.12 ± 33.54 1343 473.31 ± 1.06 474          207.14 12.95 ± 0.2     13.88
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap  av. depth   min. depth  av. n_qubits  min. qubits  total time (s)  av. time (s)  min. time (s)
-----
line-graph 13600 ± 0.0    13600 209 ± 0.0    209 476 ± 0.0    476        75.82 75.82 ± 0.0    75.82
sabre     24677 ± 1008.11 19213 1943.62 ± 179.76 1252 342.62 ± 4.44 334        295.31 18.46 ± 0.33    19.18
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap  av. depth   min. depth  av. n_qubits  min. qubits  total time (s)  av. time (s)  min. time (s)
-----
line-graph 13600 ± 0.0    13600 209 ± 0.0    209 476 ± 0.0    476        75.21 75.21 ± 0.0    75.21
sabre     22096 ± 699.86 19115 1676.12 ± 96.85 1293 348.44 ± 4.94 347        573.52 35.84 ± 0.93    38.88
-----

```

```

-----
name = shuriken, size = (7, 7), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap  av. depth   min. depth  av. n_qubits  min. qubits  total time (s)  av. time (s)  min. time (s)
-----
line-graph 13600 ± 0.0    13600 209 ± 0.0    209 476 ± 0.0    476        76.74 76.74 ± 0.0    76.74
sabre     18148 ± 348.43 17064 1369.19 ± 66.21 1099 336.31 ± 4.62 323        2519.55 157.47 ± 3.75    146.54
-----

```

7.0.2 Quantum simulation, checkerboard, against SABRE

```

import line_graph_routing as lgr # Loading these makes these cells stand-alone
import pickle

settings=[]
for name in ['checkerboard']:
    for side in [i+0.5 for i in range(1,9,2)]:
        for p in [1,8,16]:
            for optimization_level in range(4):
                setting={'name':name,
                        'size': (side,side),
                        'circuit_type': 'quantum_simulation',
                        'p': p,
                        'repetitions' : 16,
                        'optimization_level' : optimization_level,
                        'methods' : ['sabre']}
                settings.append(setting)

## Uncomment to rerun benchmarks. This takes a couple of hours.
#results=[]
# for setting in settings:
#     result=lgr.benchmark(**setting)

```

```

# results.append(result)
# lgr.print_benchmark(result)
#
#with open('benchmark_data/checkerboard.pkl','wb') as f:
# pickle.dump(results,f)

#Load previously obtained results from disk and show them.
import pickle
with open('benchmark_data/checkerboard.pkl','rb') as f:
    results=pickle.load(f)

for result in results:
    lgr.print_benchmark(result)

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 42 ± 0.0      42 27 ± 0.0   27 21 ± 0.0   21            0.17 0.17 ± 0.0     0.17
sabre     45 ± 1.28    47 32.44 ± 1.97 27 19.06 ± 0.66 21            0.99 0.06 ± 0.03   0.03
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 42 ± 0.0      42 27 ± 0.0   27 21 ± 0.0   21            0.37 0.37 ± 0.0     0.37
sabre     32 ± 1.44    22 26.56 ± 2.0  19 17.69 ± 0.66 16            0.67 0.04 ± 0.03   0.03
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 42 ± 0.0      42 27 ± 0.0   27 21 ± 0.0   21            0.35 0.35 ± 0.0     0.35
sabre     28 ± 0.84    31 23.81 ± 1.81 19 17.94 ± 0.87 21            1.34 0.08 ± 0.04   0.04
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 42 ± 0.0      42 27 ± 0.0   27 21 ± 0.0   21            0.17 0.17 ± 0.0     0.17
sabre     28 ± 0.78    25 25.31 ± 2.03 18 16.81 ± 0.41 18            2.56 0.16 ± 0.04   0.11
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 322 ± 0.0      322 188 ± 0.0      188 21 ± 0.0      21           1.89 1.89 ± 0.0      1.89
sabre    362 ± 7.38      343 235.0 ± 6.03    214 21.0 ± nan     21           4.87 0.3 ± 0.06      0.18
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 322 ± 0.0      322 188 ± 0.0      188 21 ± 0.0      21           1.98 1.98 ± 0.0      1.98
sabre    239 ± 3.71      240 197.25 ± 7.75    175 17.94 ± 0.84   16           4.99 0.31 ± 0.06      0.21
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 322 ± 0.0      322 188 ± 0.0      188 21 ± 0.0      21           1.94 1.94 ± 0.0      1.94
sabre    237 ± 2.75      227 183.5 ± 7.31     160 17.12 ± 0.5    16           5.84 0.36 ± 0.05      0.28
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 322 ± 0.0      322 188 ± 0.0      188 21 ± 0.0      21           1.95 1.95 ± 0.0      1.95
sabre    217 ± 2.41      221 187.5 ± 4.34     175 16.62 ± 0.44   16          17.58 1.1 ± 0.01      1.08
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 642 ± 0.0      642 372 ± 0.0      372 21 ± 0.0      21           3.85 3.85 ± 0.0      3.85
sabre    763 ± 20.84     657 485.38 ± 12.94  417 21.0 ± nan     21           8.07 0.5 ± 0.05      0.51
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 642 ± 0.0      642 372 ± 0.0      372 21 ± 0.0      21           3.18 3.18 ± 0.0      3.18
sabre    471 ± 9.91      469 379.56 ± 17.02  311 18.62 ± 0.88   21           7.55 0.47 ± 0.04      0.4
-----

```

```

-----
name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 642 ± 0.0      642 372 ± 0.0    372 21 ± 0.0      21            3.22 3.22 ± 0.0      3.22
sabre     463 ± 5.31      466 376.0 ± 8.98  346 17.5 ± 0.81   21            10.29 0.64 ± 0.04    0.52
-----

name = checkerboard, size = (1.5, 1.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 642 ± 0.0      642 372 ± 0.0    372 21 ± 0.0      21            3.23 3.23 ± 0.0      3.23
sabre     454 ± 2.81      447 353.69 ± 12.16 321 16.88 ± 0.5    16            29.5 1.84 ± 0.03    1.96
-----

name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 271 ± 0.0      271 30 ± 0.0      30 97 ± 0.0      97            0.92 0.92 ± 0.0      0.92
sabre     550 ± 9.03      548 104.56 ± 3.78  90 88.5 ± 1.56    92            2.56 0.16 ± 0.02    0.14
-----

name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 271 ± 0.0      271 30 ± 0.0      30 97 ± 0.0      97            1.01 1.01 ± 0.0      1.01
sabre     257 ± 7.69      233 63.81 ± 4.25  48 69.0 ± 1.38    68            3.03 0.19 ± 0.02    0.17
-----

name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 271 ± 0.0      271 30 ± 0.0      30 97 ± 0.0      97            1.01 1.01 ± 0.0      1.01
sabre     228 ± 6.72      234 57.94 ± 3.75  48 70.56 ± 1.25  71            4.87 0.3 ± 0.03     0.26
-----

name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 271 ± 0.0      271 30 ± 0.0      30 97 ± 0.0      97            1.04 1.04 ± 0.0      1.04
sabre     180 ± 2.97      187 45.44 ± 2.72  37 67.38 ± 0.97   70            16.94 1.06 ± 0.03    1.1
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2077 ± 0.0     2077 198 ± 0.0     198 97 ± 0.0     97           7.69 7.69 ± 0.0     7.69
sabre     3416 ± 49.08   3159 597.62 ± 15.12 544 92.62 ± 0.75 92           18.07 1.13 ± 0.05    1.19
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2077 ± 0.0     2077 198 ± 0.0     198 97 ± 0.0     97           7.68 7.68 ± 0.0     7.68
sabre     1870 ± 26.84   1894 481.25 ± 11.75 444 70.19 ± 1.84 68           20.33 1.27 ± 0.05    1.33
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2077 ± 0.0     2077 198 ± 0.0     198 97 ± 0.0     97           7.91 7.91 ± 0.0     7.91
sabre     1785 ± 15.44   1756 456.5 ± 14.31 404 69.56 ± 1.16 73           29.02 1.81 ± 0.05    1.86
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2077 ± 0.0     2077 198 ± 0.0     198 97 ± 0.0     97           7.73 7.73 ± 0.0     7.73
sabre     1740 ± 16.03   1731 446.75 ± 14.74 387 69.12 ± 2.0 76           104.9 6.56 ± 0.05    6.48
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4141 ± 0.0     4141 390 ± 0.0     390 97 ± 0.0     97           14.99 14.99 ± 0.0    14.99
sabre     6142 ± 72.56   5875 1117.75 ± 21.0 1021 94.0 ± nan 94           36.66 2.29 ± 0.13    2.24
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4141 ± 0.0     4141 390 ± 0.0     390 97 ± 0.0     97           15.05 15.05 ± 0.0    15.05
sabre     3639 ± 29.19   3753 996.19 ± 20.97 934 70.0 ± 1.38 67           41.08 2.57 ± 0.09    2.48
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4141 ± 0.0     4141 390 ± 0.0     390 97 ± 0.0     97           15.66 15.66 ± 0.0     15.66
sabre    3684 ± 25.79   3608 949.5 ± 19.69  893 70.06 ± 2.69  71           57.58 3.6 ± 0.1       3.48
-----

```

```

-----
name = checkerboard, size = (3.5, 3.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 4141 ± 0.0     4141 390 ± 0.0     390 97 ± 0.0     97           15.64 15.64 ± 0.0     15.64
sabre    3580 ± 20.27   3621 924.56 ± 17.07  854 67.0 ± 1.47  65           207.53 12.97 ± 0.12    12.8
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 682 ± 0.0     682 34 ± 0.0     34 221 ± 0.0     221          2.48 2.48 ± 0.0     2.48
sabre    1770 ± 17.16   1791 138.56 ± 4.66  121 211.31 ± 1.47  213          6.81 0.43 ± 0.04     0.5
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 682 ± 0.0     682 34 ± 0.0     34 221 ± 0.0     221          2.45 2.45 ± 0.0     2.45
sabre    678 ± 30.68    678 107.06 ± 7.62  77 160.38 ± 1.34  160          9.63 0.6 ± 0.04     0.65
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 682 ± 0.0     682 34 ± 0.0     34 221 ± 0.0     221          2.47 2.47 ± 0.0     2.47
sabre    808 ± 21.59    773 102.81 ± 7.0   82 159.62 ± 2.76  160          18.67 1.17 ± 0.03     1.07
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 682 ± 0.0     682 34 ± 0.0     34 221 ± 0.0     221          2.41 2.41 ± 0.0     2.41
sabre    652 ± 19.28    608 83.75 ± 6.44   58 156.94 ± 1.84  158          80.01 5.0 ± 0.04     5.14
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 5148 ± 0.0      5148 223 ± 0.0      223 221 ± 0.0      221          18.16 18.16 ± 0.0      18.16
sabre    10367 ± 118.27  10404 823.12 ± 18.63   746 217.38 ± 0.62   219          46.53 2.91 ± 0.16      2.8
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 5148 ± 0.0      5148 223 ± 0.0      223 221 ± 0.0      221          18.72 18.72 ± 0.0      18.72
sabre    5644 ± 120.68   5373 800.31 ± 38.98   667 159.44 ± 2.66   166          52.7  3.29 ± 0.1       3.38
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 5148 ± 0.0      5148 223 ± 0.0      223 221 ± 0.0      221          18.19 18.19 ± 0.0      18.19
sabre    5241 ± 43.19    5091 734.25 ± 22.25   666 157.81 ± 3.06   171          89.44 5.59 ± 0.12      5.86
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 5148 ± 0.0      5148 223 ± 0.0      223 221 ± 0.0      221          18.3  18.3 ± 0.0        18.3
sabre    5054 ± 30.28    4839 723.25 ± 16.98   656 157.69 ± 2.69   159          363.83 22.74 ± 0.2      23.19
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 10252 ± 0.0     10252 439 ± 0.0      439 221 ± 0.0      221          36.56 36.56 ± 0.0      36.56
sabre    18950 ± 206.58  18281 1546.5 ± 24.0    1464 217.38 ± 0.78   218          98.11 6.13 ± 0.21      6.55
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 10252 ± 0.0     10252 439 ± 0.0      439 221 ± 0.0      221          36.48 36.48 ± 0.0      36.48
sabre    10695 ± 85.99   10470 1609.69 ± 39.03  1439 161.44 ± 2.88   158          113.03 7.06 ± 0.21      6.69
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 10252 ± 0.0     10252 439 ± 0.0     439 221 ± 0.0     221          36.69 36.69 ± 0.0     36.69
sabre     10265 ± 67.73   10411 1565.5 ± 31.69 1469 162.69 ± 4.19 163          171.93 10.75 ± 0.22    10.26
-----

```

```

-----
name = checkerboard, size = (5.5, 5.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 10252 ± 0.0     10252 439 ± 0.0     439 221 ± 0.0     221          37.43 37.43 ± 0.0     37.43
sabre     10373 ± 43.89   10384 1537.19 ± 30.88 1390 155.31 ± 2.59 158          681.17 42.57 ± 0.25    42.4
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1211 ± 0.0     1211 30 ± 0.0     30 393 ± 0.0     393          4.76 4.76 ± 0.0     4.76
sabre     4053 ± 23.25    4092 198.25 ± 3.97 187 350.88 ± 3.41 355          14.04 0.88 ± 0.04    0.91
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1211 ± 0.0     1211 30 ± 0.0     30 393 ± 0.0     393          4.49 4.49 ± 0.0     4.49
sabre     2459 ± 56.57    2107 186.38 ± 11.72 152 284.44 ± 2.91 289          25.29 1.58 ± 0.05    1.64
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1211 ± 0.0     1211 30 ± 0.0     30 393 ± 0.0     393          4.58 4.58 ± 0.0     4.58
sabre     2189 ± 48.72    1989 167.81 ± 10.59 138 286.19 ± 3.31 293          57.96 3.62 ± 0.05    3.58
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 1211 ± 0.0     1211 30 ± 0.0     30 393 ± 0.0     393          4.55 4.55 ± 0.0     4.55
sabre     1767 ± 37.66    1641 135.38 ± 6.47 105 279.06 ± 4.72 290          283.04 17.69 ± 0.12    17.6
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph  9289 ± 0.0       9289  219 ± 0.0       219  393 ± 0.0       393           34.14  34.14 ± 0.0       34.14
sabre     22164 ± 192.21   21331 1027.31 ± 17.21  951  361.69 ± 3.69   377           93.75  5.86 ± 0.21       6.22
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph  9289 ± 0.0       9289  219 ± 0.0       219  393 ± 0.0       393           34.49  34.49 ± 0.0       34.49
sabre     12118 ± 289.99   11694 1276.06 ± 66.48  1103  285.69 ± 3.28   293           118.35  7.4 ± 0.19        7.07
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph  9289 ± 0.0       9289  219 ± 0.0       219  393 ± 0.0       393           33.66  33.66 ± 0.0       33.66
sabre     12194 ± 132.53   11396 1219.62 ± 31.86  1093  287.31 ± 3.31   286           222.24  13.89 ± 0.23      13.07
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 8, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph  9289 ± 0.0       9289  219 ± 0.0       219  393 ± 0.0       393           33.34  33.34 ± 0.0       33.34
sabre     11481 ± 103.84   11276 1151.81 ± 30.17  1052  279.0 ± 3.91    278           974.77  60.92 ± 0.4       61.4
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph  18521 ± 0.0       18521  435 ± 0.0       435  393 ± 0.0       393           68.34  68.34 ± 0.0       68.34
sabre     39936 ± 352.99   38867 1884.31 ± 37.54  1805  359.75 ± 3.12   363           169.73  10.61 ± 0.29      10.02
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph  18521 ± 0.0       18521  435 ± 0.0       435  393 ± 0.0       393           67.22  67.22 ± 0.0       67.22
sabre     23853 ± 341.31   23359 2396.88 ± 96.2   2054  286.44 ± 3.78   299           222.73  13.92 ± 0.42      13.2
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 2
method   av. n_swaps   min. n_swap av. depth   min. depth av. n_qubits   min. qubits   total time (s) av. time (s)   min. time (s)
-----
line-graph 18521 ± 0.0     18521 435 ± 0.0     435 393 ± 0.0     393         68.09 68.09 ± 0.0     68.09
sabre     23286 ± 127.03 22930 2395.75 ± 51.28 2243 288.56 ± 5.34 280         389.15 24.32 ± 0.38     24.34
-----

```

```

-----
name = checkerboard, size = (7.5, 7.5), circuit_type = quantum_simulation, p = 16, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap av. depth   min. depth av. n_qubits   min. qubits   total time (s) av. time (s)   min. time (s)
-----
line-graph 18521 ± 0.0     18521 435 ± 0.0     435 393 ± 0.0     393         67.2  67.2 ± 0.0     67.2
sabre     22170 ± 140.89 23060 2352.75 ± 72.43 2121 281.62 ± 4.06 282         1750.38 109.4 ± 0.67     109.23
-----

```

7.0.3 Random circuit, kagome and shuriken, against SABRE

```

import line_graph_routing as lgr # Loading these makes these cells stand-alone
import pickle

settings=[]
for name in ['kagome','shuriken']:
    for side in range(1,7,2):
        for p in [side**2*500]:
            for optimization_level in [1]:
                setting={'name':name,
                        'size': (side,side),
                        'circuit_type': 'random',
                        'p': p,
                        'repetitions' : 16,
                        'optimization_level' : optimization_level,
                        'methods' : ['sabre']}
                settings.append(setting)

## Uncomment to rerun benchmarks.
#results=[]
#for setting in settings:
#    result=lgr.benchmark(**setting)
#    results.append(result)
#    lgr.print_benchmark(result)
#
#with open('benchmark_data/random.pkl','wb') as f:
#    pickle.dump(results,f)

#Load previously obtained results from disk and show them.
import pickle
with open('benchmark_data/random.pkl','rb') as f:

```

```

results=pickle.load(f)

for result in results:
    lgr.print_benchmark(result)

```

```

-----
name = kagome, size = (1, 1), circuit_type = random, p = 500, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 200 ± 0.0      200 249 ± 0.0     249 12 ± 0.0      12            0.37 0.37 ± 0.0      0.37
sabre     96 ± 0.62      96 207.5 ± 1.19   205 12.0 ± nan    12            1.17 0.07 ± 0.03     0.06
-----

```

```

-----
name = kagome, size = (3, 3), circuit_type = random, p = 4500, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2611 ± 0.0     2611 568 ± 0.0     568 68 ± 0.0      68            1.14 1.14 ± 0.0      1.14
sabre     1444 ± 25.94   1450 838.88 ± 35.0 720 44.31 ± 1.75 54            10.24 0.64 ± 0.03     0.66
-----

```

```

-----
name = kagome, size = (5, 5), circuit_type = random, p = 12500, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 7592 ± 0.0     7592 735 ± 0.0     735 164 ± 0.0     164           3.02 3.02 ± 0.0      3.02
sabre     5637 ± 82.31   5452 1774.44 ± 54.79 1590 108.69 ± 3.56 107           32.72 2.04 ± 0.04     2.03
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = random, p = 500, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 122 ± 0.0      122 247 ± 0.0     247 8 ± 0.0       8             0.07 0.07 ± 0.0      0.07
sabre     85 ± nan       85 231.12 ± 0.31 231 8.0 ± nan    8             0.84 0.05 ± 0.0      0.05
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = random, p = 4500, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth     min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 2262 ± 0.0     2262 415 ± 0.0     415 84 ± 0.0      84            1.05 1.05 ± 0.0      1.05
sabre     1662 ± 39.26   1447 667.12 ± 39.54 517 65.25 ± 1.38 69            10.53 0.66 ± 0.04     0.7
-----

```

```
-----
```

method	av. n_swaps	min. n_swap	av. depth	min. depth	av. n_qubits	min. qubits	total time (s)	av. time (s)	min. time (s)
line-graph	6828 ± 0.0	6828	405 ± 0.0	405	240 ± 0.0	240	2.83	2.83 ± 0.0	2.83
sabre	6811 ± 192.15	5969	1403.31 ± 94.74	1094	177.31 ± 2.94	176	38.36	2.4 ± 0.04	2.38

```
-----
```

7.0.4 Random circuit, complete graph, against SABRE

```
import line_graph_routing as lgr
import pickle

settings=[]
for name in ['complete']:
    for side in range(3,10,2):
        for p in [side*100]:
            for optimization_level in range(2):
                setting={'name':name,
                        'size': side,
                        'circuit_type': 'random',
                        'p': p,
                        'repetitions' : 16,
                        'optimization_level' : optimization_level,
                        'methods' : ['sabre']}
                settings.append(setting)

## Uncomment to rerun benchmarks
#results=[]
#for setting in settings:
#     result=lgr.benchmark(**setting)
#     results.append(result)
#     lgr.print_benchmark(result)

#with open('benchmark_data/complete.pkl','wb') as f:
#     pickle.dump(results,f)

with open('benchmark_data/complete.pkl','rb') as f:
    results=pickle.load(f)

for result in results:
    lgr.print_benchmark(result)
```

```

-----
name = complete, size = 3, circuit_type = random, p = 300, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 154 ± 0.0      154 334 ± 0.0   334 4 ± 0.0     4             0.12 0.12 ± 0.0      0.12
sabre    40 ± 0.97      37 267.88 ± 2.34 257 3.0 ± nan   3             1.01 0.06 ± 0.06     0.04
-----

```

```

-----
name = complete, size = 3, circuit_type = random, p = 300, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 140 ± 0.0      140 313 ± 0.0   313 4 ± 0.0     4             0.11 0.11 ± 0.0      0.11
sabre    34 ± nan       34 220.0 ± nan  220 3.0 ± nan   3             0.97 0.06 ± 0.0      0.07
-----

```

```

-----
name = complete, size = 5, circuit_type = random, p = 500, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 314 ± 0.0      314 548 ± 0.0   548 6 ± 0.0     6             0.19 0.19 ± 0.0      0.19
sabre    88 ± 1.25      84 458.06 ± 4.0 444 5.0 ± nan   5             0.96 0.06 ± nan      0.06
-----

```

```

-----
name = complete, size = 5, circuit_type = random, p = 500, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 286 ± 0.0      286 503 ± 0.0   503 6 ± 0.0     6             0.18 0.18 ± 0.0      0.18
sabre    72 ± nan       72 387.25 ± 0.31 386 5.0 ± nan   5             1.44 0.09 ± nan      0.09
-----

```

```

-----
name = complete, size = 7, circuit_type = random, p = 700, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 436 ± 0.0      436 723 ± 0.0   723 8 ± 0.0     8             0.25 0.25 ± 0.0      0.25
sabre    115 ± 1.59     115 609.25 ± 6.01 586 7.0 ± nan   7             1.65 0.1 ± 0.02     0.09
-----

```

```

-----
name = complete, size = 7, circuit_type = random, p = 700, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 478 ± 0.0      478 787 ± 0.0   787 8 ± 0.0     8             0.26 0.26 ± 0.0      0.26
sabre    118 ± 0.38     117 573.25 ± 1.66 567 7.0 ± nan   7             2.24 0.14 ± 0.01     0.13
-----

```

```

-----
name = complete, size = 9, circuit_type = random, p = 900, repetitions = 16, optimization_level = 0
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 593 ± 0.0     593 946 ± 0.0   946 10 ± 0.0    10            0.43 0.43 ± 0.0     0.43
sabre     151 ± 2.0     143 770.0 ± 6.84 730 9.0 ± nan   9            1.6  0.1 ± nan      0.1
-----

```

```

-----
name = complete, size = 9, circuit_type = random, p = 900, repetitions = 16, optimization_level = 1
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 650 ± 0.0     650 1036 ± 0.0   1036 10 ± 0.0    10            0.33 0.33 ± 0.0     0.33
sabre     149 ± 0.59    146 726.5 ± 1.59 720 9.0 ± nan   9            2.74 0.17 ± 0.02    0.16
-----

```

7.0.5 Against other routing methods

Above, we only ran SABRE because it outperforms the other methods available in Qiskit by default. The standard methods available are

```

from qiskit import transpiler
transpiler.preset_passmanagers.plugin.list_stage_plugins('routing')

```

```
['basic', 'lookahead', 'none', 'sabre', 'stochastic']
```

```

import line_graph_routing as lgr
import pickle

settings=[]
for name in ['kagome','shuriken']:
    for side in [1]:
        for p in [1]:
            for optimization_level in [3]:
                setting={'name':name,
                        'size': (side,side),
                        'circuit_type': 'quantum_simulation',
                        'p': p,
                        'repetitions' : 16,
                        'optimization_level' : optimization_level,
                        'methods' : ['basic', 'lookahead', 'sabre', 'stochastic']}
                settings.append(setting)

## Uncomment to rerun benchmarks
#results=[]
#for setting in settings:
#     result=lgr.benchmark(**setting)
#     results.append(result)
#     lgr.print_benchmark(result)

```

```

#with open('benchmark_data/other_methods_1x1.pkl', 'wb') as f:
#    pickle.dump(results, f)

with open('benchmark_data/other_methods_1x1.pkl', 'rb') as f:
    results=pickle.load(f)

for result in results:
    lgr.print_benchmark(result)

```

```

-----
name = kagome, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 12 ± 0.0      12 7 ± 0.0    7 12 ± 0.0   12           0.08 0.08 ± 0.0     0.08
basic     14 ± 0.0      14 17.0 ± 0.0 17 8.0 ± 0.0   8            0.05 0.05 ± 0.0     0.05
lookahead 8 ± nan       8 8.81 ± 0.19 8 8.0 ± nan   8            19.55 1.22 ± 0.04    1.26
sabre     6 ± nan       6 6.0 ± nan   6 8.0 ± nan   8            0.49 0.03 ± 0.0     0.03
stochastic 6 ± 0.62      6 6.06 ± 0.16 6 8.0 ± nan   8            0.62 0.04 ± 0.0     0.03
-----

```

```

-----
name = shuriken, size = (1, 1), circuit_type = quantum_simulation, p = 1, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap   av. depth   min. depth   av. n_qubits   min. qubits   total time (s)   av. time (s)   min. time (s)
-----
line-graph 8 ± 0.0       8 9 ± 0.0     9 8 ± 0.0    8            0.05 0.05 ± 0.0     0.05
basic     16 ± 0.0      16 19.0 ± 0.0 19 8.0 ± 0.0   8            0.04 0.04 ± 0.0     0.04
lookahead 6 ± nan       6 8.0 ± nan   8 8.0 ± nan   8            14.22 0.89 ± 0.01    0.88
sabre     6 ± nan       6 7.62 ± 0.22 7 8.0 ± nan   8            0.5 0.03 ± 0.0     0.04
stochastic 6 ± nan       6 7.0 ± nan   7 8.0 ± nan   8            0.58 0.04 ± 0.0     0.04
-----

```

The method ‘lookahead’ takes an impractical amount of time, so we exclude it when running benchmarks for larger/deeper circuits.

```

import line_graph_routing as lgr
import pickle

settings=[]
for name in ['kagome', 'shuriken']:
    for side in [3]:
        for p in [3]:
            for optimization_level in [3]:
                setting={'name':name,
                        'size': (side,side),
                        'circuit_type': 'quantum_simulation',
                        'p': p,
                        'repetitions' : 16,
                        'optimization_level' : optimization_level,

```

```

        'methods' : ['basic', 'sabre', 'stochastic']
    }
    settings.append(setting)

## Uncomment to rerun benchmarks
#results=[]
#for setting in settings:
#     result=lgr.benchmark(**setting)
#     results.append(result)
#     lgr.print_benchmark(result)
#
#with open('benchmark_data/other_methods_3x3.pkl', 'wb') as f:
#     pickle.dump(results, f)

with open('benchmark_data/other_methods_3x3.pkl', 'rb') as f:
    results=pickle.load(f)

for result in results:
    lgr.print_benchmark(result)

```

```

-----
name = kagome, size = (3, 3), circuit_type = quantum_simulation, p = 3, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap  av. depth   min. depth  av. n_qubits  min. qubits  total time (s)  av. time (s)  min. time (s)
-----
line-graph 336 ± 0.0      336 43 ± 0.0    43 68 ± 0.0    68          1.33 1.33 ± 0.0     1.33
basic     475 ± 0.0      475 287.0 ± 0.0  287 45.0 ± 0.0  45          1.21 1.21 ± 0.0     1.21
sabre     215 ± 3.56     203 63.25 ± 3.78 50 43.94 ± 0.97 45          12.37 0.77 ± 0.04  0.93
stochastic 410 ± 14.31    428 85.88 ± 4.12 71 46.44 ± 1.5  48          28.09 1.76 ± 0.07  1.46
-----

```

```

-----
name = shuriken, size = (3, 3), circuit_type = quantum_simulation, p = 3, repetitions = 16, optimization_level = 3
method   av. n_swaps   min. n_swap  av. depth   min. depth  av. n_qubits  min. qubits  total time (s)  av. time (s)  min. time (s)
-----
line-graph 390 ± 0.0      390 34 ± 0.0     34 84 ± 0.0     84          1.81 1.81 ± 0.0     1.81
basic     1195 ± 0.0     1195 491.0 ± 0.0  491 71.0 ± 0.0   71          2.14 2.14 ± 0.0     2.14
sabre     380 ± 10.03    344 78.19 ± 4.42 66 64.19 ± 0.94 67          24.94 1.56 ± 0.12   2.31
stochastic 813 ± 32.04    609 104.0 ± 7.88 76 66.56 ± 1.44 71          62.25 3.89 ± 0.09  3.76
-----

```

We see SABRE outperforms the other methods available by default in Qiskit, but not line-graph routing for the circuits considered.

7.0.6 Wall-clock time of line-graph routing

Create a random graph, construct the line graph, create a circuit on the line graph, and put this circuit into line-graph routing

```

from time import time
side=25
lg = lgr.kagome(side, side)
print('number of nodes =',lg.number_of_nodes())

```

```
La=10**5
print('number of gates =',La)
qc = lgr.random_circuit(lg, La)
begin=time()
qc = lgr.line_graph_route(qc)
end=time()
print('wall clock time =',end-begin,'(s)')
```

```
number of nodes = 1976
number of gates = 100000
wall clock time = 25.544434070587158 (s)
```