ADVANCED QUANTUM TECHNOLOGIES
www.advquantumtech.com

Exploring the Advantages of Layout Procedure with Fully-Connected Quantum Computing Technologies

Andrea Russo, Mario Simoni, Deborah Volpe, Giovanni A. Cirillo, and Mariagrazia Graziano*

Even though quantum algorithms are designed using an idealised high-level quantum circuit description, the execution on real devices must consider the backend physical properties. Contemporary noisy intermediate-scale quantum (NISQ) computers are characterized by limited inter-qubit connectivity and an exclusive set of native gates, whose execution is affected by non-negligible errors and non-idealities. Quantum compilation, consisting of logic and layout synthesis, aims to recast the original circuit description to the target hardware by solving the coupling-constraint and to minimize the execution time and the error rate. While a previous article addresses the first phase, the second one is the aim of this manuscript. Currently available layout tools mainly target superconducting technology. This article tries to match the unmet need for a layout synthesis library supporting a wider range of technologies. The most promising algorithms already present in the state of the art are adapted to extend the compatibility to all the supported technologies, and new approaches, which exploit the awareness of the coupling strength in fully-connected topologies, are proposed. The integrated procedures are benchmarked against IBM's Qiskit and Cambridge Quantum Computing's t|ket⟩ compilation toolchains, and even though these strategies can be further improved, the results are encouraging.

1. Introduction

The application-driven scenarios where quantum computing can be successfully employed are continuously increasing. Among these, optimiation,[1] machine learning,[2] and chemical simulations[3] can be mentioned. In the context of the *quantum circuit model* paradigm, quantum algorithms are usually designed by exploiting an ideal high-level quantum circuit description without considering the quantum hardware characteristics. The complexity of the translation from this abstract description to one aware of and compliant with hardware grows with circuit dimensions, thus making a manual procedure unfeasible for every practical application. Consequently, the advancement of quantum computing research demands for efficient design automation tools — analogously to what happens with digital circuits in classical computing — to produce refined and reliable quantum circuits to be adapted to the target backend.

The *execution reliability* of real devices strongly depends on their physical properties, such as quantum gate errors, relaxation, and decoherence timescales. Therefore, the aforementioned translation must also be done to optimise the performance, considering the target backend peculiarities. Accordingly, the goal of *quantum compilation toolchains*[4] is to refine the high-level quantum circuit description to execute it on a target computer, optimising some relevant figures of merit. A compiler can be subdivided into two main steps: *logic synthesis*, which consists in writing the original circuit by using only technology native gates, and *layout synthesis*, whose aims are mapping the logical qubits of the abstract description to the physical ones of the target device (*placement phase*) and solving the non-allowed interactions (*routing phase*) — associated with two-qubit gates and related to the limited connectivity of quantum hardware — possibly adding swap gates to make the final circuit compliant with the target coupling-graph.

Real quantum computers can be partially or fully-connected. However, the complete connectivity between qubits does not imply better performance. In fact, some interactions could be so weak that a direct gate between them could be not practical and reliable. Accordingly, the routing procedure can provide benefits also in this case.

This work proposes a *flexible multi-technology library for layout synthesis of quantum circuits*, having as target technologies *superconducting transmon qubits*, *quantum dots*, *nuclear magnetic resonance* (*NMR*), and *trapped ions*. It is entirely developed in

A. Russo, M. Simoni, D. Volpe, G. A. Cirillo
Department of Electronics and Telecommunications of Politecnico di Torino
Torino 10129, Italy
E-mail: mario.simoni@polito.it
M. Graziano
Department of Applied Science and Technology
Torino 10129, Italy

The ORCID identification number(s) for the author(s) of this article can be found under https://doi.org/10.1002/qute.202300128

DOI: 10.1002/qute.202300128

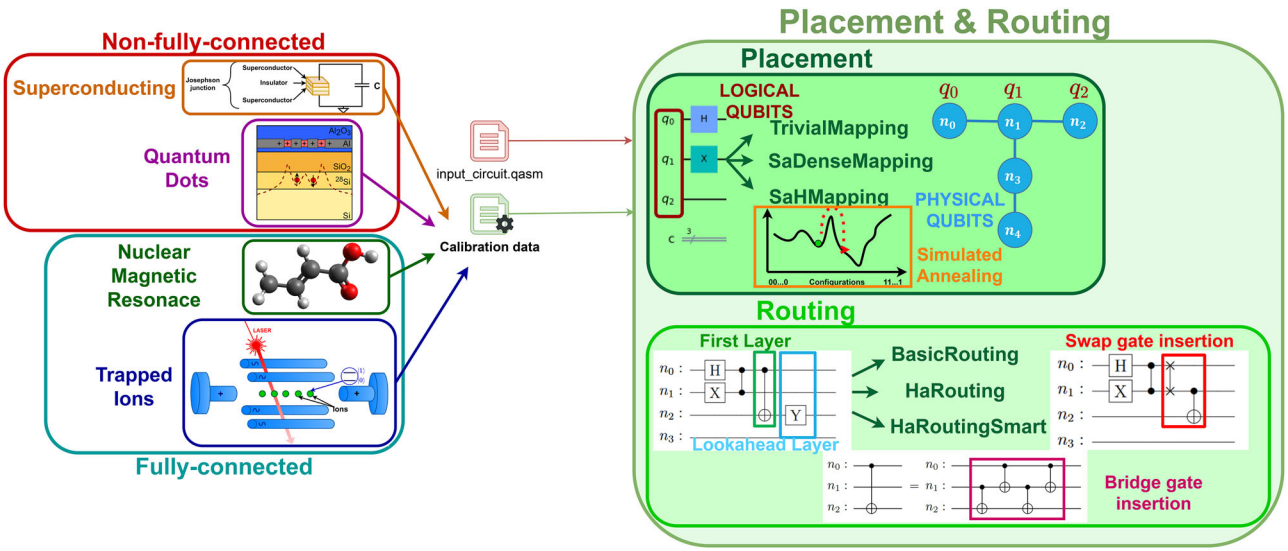

Figure 1. Layout synthesis library structure. Calibration data files, storing information on the supported quantum technologies, and an input circuit described in OpenQASM 2.0 are provided to the library for the execution of placement and routing procedures.

Python, which is chosen for its compatibility with existing quantum frameworks, and it targets circuits described with the *OpenQASM 2.0* language.[5] The most relevant and promising algorithms in the literature are *extended to all the considered technologies*, and *new strategies*, which exploit the awareness of the coupling strength in *fully-connected topologies*, are proposed.

Moreover, the library is integrated into the template-based compilation toolchain presented in[6] and benchmarked against *IBM's Qiskit*[7] and *Cambridge Quantum Computing's* t|ket⟩.[8]

The article is organised as follows. Section 2 discusses theoretical foundations in terms of the characteristics of the supported technologies and the principles of quantum circuit compilation. Section 3 introduces the algorithms supported by the library, focusing on the new proposed strategies for fully-connected technologies. In Section 4, the procedure followed for integration in the compilation toolchain is detailed whereas Section 5 reports and discusses the results. Finally, in Section 6, conclusions are drawn, and future perspectives are illustrated.

2. Background

This section briefly presents the technologies supported by the proposed layout toolchain (Section 2.1) and the motivations and characteristics of quantum circuit compilation (Section 2.2). Moreover, the distance matrix, a critical metric employed in several placement and routing algorithms, is introduced in Section 2.3.

2.1. Supported Quantum Computing Technologies

As mentioned, the proposed layout synthesis library supports superconducting transmon qubits, quantum dots, NMR,[9] and trapped ions,[10] as shown in **Figure 1**.

Superconducting qubits are consolidated devices for the fabrication of quantum processors, especially those manufactured by leading technology companies like IBM and Google. Therefore, layout tools currently available mainly target this technology.[11] A qubit is encoded on the two lowest energy states of an anharmonic LC oscillator circuit, with a non-linear inductance given by a Josephson junction. The devices are typically *non-fully-connected*. The target backend considered in the performed test is the *IBM Toronto*,[12] whose characteristics are extracted from the `FakeToronto`[13] class instance.

Quantum dots are a semiconductor technology, in which qubits are encoded onto the *spin of trapped electrons or holes*. The *exchange interaction* is exploited to implement the native two-qubit gate. It is typically a *non-fully-connected* technology, usually presenting a linear chain as a qubits coupling graph. The target backend characteristics for tests are obtained by performing simulations with the compact models available in *MATLAB QuanTO*,[14,15] a classical simulator of quantum computing technologies developed at Politecnico di Torino based on the methodology described in ref. [9].

In *NMR* quantum computers, information is encoded on the nuclear spin-$\frac{1}{2}$ energy levels of a liquid-state ensemble of molecules. It is usually a *fully-connected* technology since every couple of molecule's nuclei exhibit a J-coupling interaction that can be exploited for implementing a two-qubit gate. Nevertheless, some interactions can be so weak that exploiting them for implementing two-qubit gates is impractical. Analogously to quantum dots, the target backend characteristics for tests are obtained by performing simulations on MATLAB QuanTO.

Finally, in *trapped ions* devices, qubit is encoded onto the energy states of an ion confined in an electromagnetic Paul trap. They are typically *fully-connected*, i.e., all the ions — which are usually trapped in such a way that are linearly arranged — can interact. Unfortunately, the two-qubit gate error and execution time are proportional to the distance of the involved ions, thus making the exploitation of some interactions unfeasible. For the performed tests, the device characteristics are retrieved from refs. [16, 17, 18].

2.2. Quantum Circuit Compilation

A high-level description is usually employed for quantum algorithms. However, for the execution on a real device, it is necessary to consider the physical properties of the target. Therefore, the compiler aims to translate the abstract description into one compliant with the target backend and aware of its characteristics. This task includes two essential steps: *logic synthesis* and *layout synthesis*.

The goal of the first is to move the gates employed in the abstract description into the native ones of the target technology, optimizing some figures of merit. This step was already described in ref. [6, 19].

This work is focused on the second step, which maps the logical qubits — identified as q_i — of the abstract description to the physical ones — also called nodes and labeled as n_i — of the target device (*placement*) and solves the non-allowed interactions associated with two-qubit gates (*routing*), improving the effectiveness of a circuit execution.

The placement input is the output of logic synthesis, where all qubits employed in the description are logical ($\{q_0, q_1, q_2, q_3 \ldots\}$). The output of this step is still a high-level description that cannot be executed on hardware, but it gives a one-to-one association between logical and physical qubits ($\pi = \{q_0 \rightarrow n_1, q_1 \rightarrow n_3, q_3 \rightarrow n_0, \ldots\}$), as shown in the placement box of Figure 1. This becomes the input of the routing phase, which provides an output circuit effectively executable on the target backend. Indeed, it considers the device connectivity limits and solves all the non-allowed interactions in the input by adding *swap gates*, which permit exchanging the logical qubits (as shown in the routing box of Figure 1), or *bridge gates*, which allows the implementation of a two-qubit gate between two disconnected nodes having one common neighbor, without altering the current mapping (bottom of the routing box in Figure 1).

The layout synthesis is of interest to several companies and research groups. The majority of the solutions proposed in the literature are based on heuristics. Two successful and relevant compilers in the state of the art are chosen for comparing the obtained results: *IBM's Qiskit transpiler* available in Terra library and *Cambridge Quantum Computer's* t|ket⟩.

2.3. The Distance Matrix *D*

One main concern for the developed placement and routing strategies, which will be presented in the following sections, is the evaluation of the *cost to implement a swap gate* between any pair of physical qubits, with the aim of selecting the best one. The most promising algorithms available inside the library use the solution proposed in [20], Section 3, where this information is stored inside a *nodes distance matrix D*, containing the *approximated distance* for swapping any pair of nodes in the device, considering a weighted combination of the number of swap gates required, the total error rate and the gate time. This is computed according to:

$$D = \alpha_1 S + \alpha_2 \mathcal{E} + \alpha_3 T \tag{1}$$

where α_1, α_2, and α_3 are configurable coefficients, the matrix S contains the shortest path length between any pair of nodes in the device, the matrix $\mathcal{E}$ contains the approximated error rate for swapping any pair of physical qubits, and the matrix T contains the approximated gate time for swapping them. As explained in [20], Section 3, the matrices are obtained by exploiting the calibration data of the device and employing the Floyd–Warshall algorithm.[21] Generic entries $S[i][j]$, $\mathcal{E}[i][j]$ and $T[i][j]$ represent, respectively, the number of swap gates required, the approximated total error rate and the approximated total gate time for swapping n_i and n_j, while respecting the target coupling-constraint.

3. The Layout Synthesis Library

The proposed layout synthesis library provides a framework for designing a quantum circuit for a target NISQ device, allowing the development and application of placement and routing algorithms to satisfy the target coupling-constraint.

The library is versatile enough to model any placement and routing technique and can be combined with any logic synthesis tool to create a complete quantum circuit compiler, enabling the analysis of various strategies found in the literature.

As the computational complexity of layout synthesis depends on the size of the quantum circuit and the number of characteristics of the quantum device taken into account, the considered optimization strategies are based on *heuristics* to reduce the algorithm compilation time. Moreover, both *hardware-unaware* and *hardware-aware* solutions are considered. The former does not use any target device feature (i.e., the calibration data) except the coupling-constraint, whereas the latter use additional details, such as error rates and gate times, trying to achieve a better placement, or routing.

3.1. Generalization of Backend Modeling

The majority of the examined literature concerning the layout synthesis implemented and tested their algorithms targeting superconducting quantum devices. This is possibly a consequence of IBM offering the possibility of testing quantum circuits on those devices even for free. On the other hand, the proposed project's development prioritized flexibility to support multiple and different quantum computing technologies. Indeed, an effort was made to adapt the examined algorithms to other state-of-the-art technologies, which resulted in generalizing the representation of the target backend.

The backbone of the presented library is a *common model for quantum devices*, independent of the technology, providing a *common interface to retrieve information*, such as allowed interactions, single-qubit/two-qubit gate times, and error rates. This allowed the definition of a unified methodology for computing the information necessary for a specific algorithm, e.g., the D matrix (Equation (1)). In this way, in most cases, applying the algorithm without modification to other technologies is possible.

All the details regarding the backed device can be provided in a *configuration file* containing technology-specific information required for computing the coupling constraint and the device features, which are internally represented by the library using a Directed Acyclic Graph (DAG). Each edge of the graph represents an allowed two-qubit interaction and encodes optional attributes that are used for evaluating the device features:

- gate-specific error rates and gate times for superconducting and trapped ion technology;
- J-coupling constant of the interaction for NMR;
- Exchange-Interaction constant for Quantum Dots.

Each technology requires a different kind of data, but each of them is evaluated in terms of single and two-qubit gates characteristics, that are exploited to estimate the quality of a possible placement or routing. The computation of technology-dependent error rates in Section 5.1 can give an idea of the methodology exploited to model the different technologies supported by the library.

3.2. Placement Algorithms

The simplest placement algorithm implemented is *TrivialMapping* (TM), inspired by Qiskit's BasicSwap.[7] In this hardware-unaware methodology, the logical qubits are mapped to physical ones of the NISQ device with the same index, i.e., $\pi_{\text{trivial}} = \{q_0 \rightarrow n_0, q_1 \rightarrow n_1, q_2 \rightarrow n_2, q_3 \rightarrow n_3, \dots\}$. Due to the simplicity of this approach, no adaptations were required for its implementation on all the library-supported technologies.

Besides this simple solution, two other algorithms, exploiting the *Simulated Annealing* (SA) metaheuristic,[22] are devised. SA was considered a valid option because it is the optimisation algorithm exploited for placement in ref. [20], the article that introduced the core routing algorithm implemented in the presented work; in particular, the intent of the authors was combining a hardware-aware search space exploration to a classical SA algorithm to improve the obtained initial mapping. Their proposed strategy was used as a basis to implement two placement heuristics in the library:

- *SimulatedAnnealingHardwareAwareMapping* (HM),[20] trying to be as close as possible to the implementation proposed in the original source, but also making required interpretation, and testing it on multiple technologies;
- *SimulatedAnnealingDenseMapping* (DM), developing a hardware-unaware adaptation of the former SA algorithm, to implement the idea of Qiskit's BasicSwap[7] for finding a subset of the backend DAG with the combination of nodes that maximizes the connectivity, i.e., the total number of allowed interactions.

3.2.1. Simulated Annealing Hardware Aware Mapping

The SA algorithm requires a *neighbors' solution exploration policy* — i.e., a mechanism for obtaining the new configuration as a perturbation of the previous one — to explore the solution space during the iterative search. For the developed algorithm, as proposed in[20], Section 3B, this policy is randomly selected at each iteration between *Shuffle*, *Reset*, and *Expand*. In this adaptation, the probability of picking one of those was the same of the original article: 90%, 2%, and 8% for *Shuffle*, *Reset*, and *Expand*, respectively (as suggested in[20], Section 4A).

Shuffle returns a new map by shuffling the order of the assignment previously considered (e.g., from $\pi_{\text{old}} = \{q_0 \rightarrow n_1, q_1 \rightarrow n_3, q_3 \rightarrow n_0\}$ to $\pi_{\text{new}} = \{q_0 \rightarrow n_0, q_1 \rightarrow n_3, q_3 \rightarrow n_1\}$).

For the latter policies, in the original work, it is indicated that the device features are exploited to perturb the current solution. However, details are not provided on how they were implemented. For such reason, these two were interpreted and implemented as follows.

Expand replaces one node of the mapping with a new node of the backend that is not part of the mapping. For each node n_i in the current mapping π_{curr}, its total number of edges is computed alongside with a score:

$$\text{score}_i = \sum_{n_n \in N_i} D[n_i][n_n] \tag{2}$$

where:

- N_i is the set of the neigbhours nodes of n_i contained in π_{curr};
- D is the distance matrix computed according to Equation (1);
- score_i is the score assigned to n_i. The same formula is also used for scoring the *candidate nodes* for the replacement.

For all the $n_i \in \pi_{\text{curr}}$, a neighbor node of n_i not belonging to the current mapping is considered a *candidate node* with a corresponding score given by Equation (2). The node removed from the mapping is the one with the worst connectivity (i.e., the number of edges) and maximum score_i, replaced with the node with the maximum connectivity and minimum score.

Reset starts from a random logical qubit mapped onto a random node and expands the mapping until a complete one is generated. Being n_l the last inserted node in the new mapping π_{new}, all of its neighbors nodes are considered *candidates* to be added into π_{new}. To each candidate n_c, a score is computed as $\text{score}_c = D[n_l][n_c]$. The node with the minimum score is added to π_{new}, mapped onto a random unpaired logical qubit. If no candidate can be found, a random node not already in π_{new} is inserted.

While presenting the SA procedure, in the original source, no precise information is given on the *cost function* target of the SA optimization. Therefore, the following function was selected for the proposed work:

$$C_{\text{HM}}(\pi_{\text{curr}}) = \sum_{gate \in G} D[\pi_{\text{temp}}(gate.q1)][\pi_{\text{temp}}(gate.q2)] \tag{3}$$

where:

- π_{curr} is the current placement for which the cost shall be computed;
- G is the set of all the two-qubit gates for the considered circuit;
- $\pi_{\text{temp}}(gate.q1)$ and $\pi_{\text{temp}}(gate.q2)$ are the interacting nodes of *gate* considering the current mapping π_{curr}.

The previous cost function, depending on the D matrix, puts in evidence the importance of the *definition of a modelling methodology common to different technologies* Section 3.1. In fact, since the D matrix can also be computed for non-superconducting devices, no other twist, or modifications to the original algorithm had to be applied to test the algorithm on different quantum devices.

3.2.2. Simulated Annealing Dense Mapping

The SA procedure used for DM is a hardware-unaware adaptation of the algorithm presented in the previous section, that makes random perturbation of the current solution at each step. In other words, this mapping strategy, differently from the previous one, does not involve the *D* matrix. The core SA algorithm remained unmodified, and only the neighbor exploration policies and target cost function needed to be adapted:

- *Shuffle* is identical to the one used for HM, as it is completely hardware-blind also in its original form.
- *Expand* substitutes a random node of the current mapping with a random node of the backend that is still not part of it.
- *Reset* returns a completely new random mapping.

The cost function that the presented solution tries to maximize is the total number of nodes at distance one (connected by one edge) in the mapping.

3.3. Routing Algorithms

For what concerns the routing phase, two distinct classes of algorithms are available in the proposed library: the first is intended to solve the coupling-constraint of a *non-fully-connected* device, the latter is an attempt to improve gate time and error rate for execution on a *fully-connected* quantum computer by introducing where-needed swap gates.

3.3.1. Routing for Non-Fully-Connected Devices

The simplest *hardware-unaware* algorithm developed is the *BasicRouting* (BR), inspired by the Qiskit BasicSwap.[7] Every time a non-allowed interaction is found, the interacting nodes are connected by inserting swap gates in the shortest path between them in the coupling-graph, assuming that all arcs have the same weight. The shortest path is computed with Dijkstra's algorithm.[23] No changes from the original source were required, and the algorithm can be used on all the supported technologies, since the coupling-graph is the only requirement.

The best-performing algorithm implemented in the layout synthesis library is the Hardware-Aware Routing algorithm (HR) proposed in ref. [20], which solves the coupling-constraint and optimizes the final quantum circuit gate time and fidelity by *exploiting the calibration data* of the target device. This heuristic directly comes from the SwAp-based BidiREctional heuristic search algorithm (SABRE).[24]

The implemented algorithm iterates over the input quantum circuit, identifying two sets at each iteration: the *first layer F* — composed of the current gates having no dependencies on any other gate — and the *lookahead layer L* — a subset of the gates dependent on the ones in the first layer. The size of *L* is a configurable parameter. An example highlighting the aforementioned layers is presented in Figure 1.

At each step, the algorithm extracts single-qubit and two-qubit gates from the current *F* and inserts them in the output circuit, respecting in this way the original dependencies. A two-qubit gate requiring a two-qubit interaction that is not allowed in hardware cannot be directly added to the output circuit, and at least one swap gate is needed. The swap gate inserted is the one labeled as the *best* for the current iteration, which is identified by assigning a *score* to each *candidate swap* — i.e., any swap that can modify the mapping of qubits involved in the problematic interaction — and selecting the one that minimizes it. For the candidate swap gate evaluation, the following *cost function* is employed:

$$H = \frac{1}{|F|}\sum_{gate\in F} D[\pi_{\text{temp}}(gate.q1)][\pi_{\text{temp}}(gate.q2)] + W\cdot\frac{1}{|L|}\sum_{gate\in L} D[\pi_{\text{temp}}(gate.q1)][\pi_{\text{temp}}(gate.q2)] \quad (4)$$

where:

- $|F|$ and $|L|$ are, respectively, the numbers of quantum gates in the first and lookahead layer;
- π_{temp} is the new logical to physical qubits mapping, with the candidate swap applied;
- $\pi_{\text{temp}}(gate.q1)$ and $\pi_{\text{temp}}(gate.q2)$ are the interacting nodes of *gate* considering the new mapping π_{temp};
- $W \in [0, 1)$ is the *lookahead parameter* that *defines the importance* of the lookahead.

The *best swap gate* is by definition the one that mostly reduces the distance *D* among the qubits involved in an interaction belonging to *F* and *L* layers. However, the choice of the best swap gate could have a *negative effect on the lookahead layer* considered alone. In this case, if the two-qubit gate under analysis is a CX one, the algorithm explores a further possible solution, replacing it with a *bridge gate*. This can be performed if and only if the interacting nodes are connected by a common neighbor. When feasible, this approach is advantageous since the previously-prevented two-qubit gate can be implemented without forcing a change in the qubit mapping that proves detrimental to the lookahead layer.

An example of a single iteration of the HR algorithm (with the best swap, not with bridge) is presented in **Figure 2**. If a two-qubit gate of *F* involves qubits physically connected, this gate is always appended to the routed circuit, without considering the quality of the interaction, otherwise circuits involving swap gates are examined and the one with the lowest cost *H* is appended to the final circuit. In the same example, the expected three-qubit backend is characterized by linear connectivity (this could be a switch-based molecular quantum computer based on Cr_7Ni rings[25]), so the CX gate between physical qubits n_0 and n_2 must be replaced by a more complex circuit with a swap involving the intermediate qubit n_1. It is also interesting to observe that the application of the swap gate affects the "location" of all the following quantum gates; in this example, this occurs to the *Y* gate of the *L* layer and the following *X*.

For the proposed algorithm implementation, in order to be compliant with the original version, the formulae used for the *D* matrix computation, the heuristic cost function *H*, and the swap negative effect computation formula are the same as ref. [20].

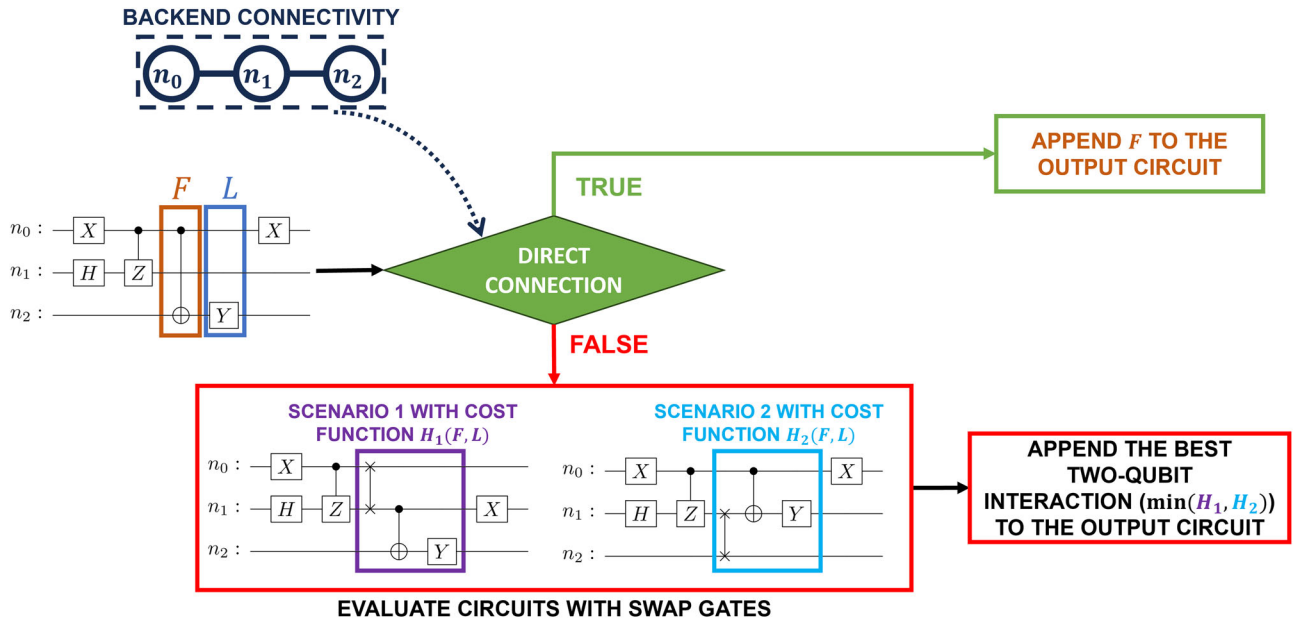

Figure 2. Example of the state-of-the-art Hardware-Aware Routing algorithm. Since the backend is not fully-connected, the two-qubit gate in *F* is evaluated by the red (false) path.

3.3.2. Reasons for Considering Routing with Fully-Connected Devices

The state-of-the-art routing algorithms do not take into account the effective quality of a two-qubit interaction; considering the HR algorithm, as highlighted in the previous section and in Figure 2, a "direct" two-qubit gate is always appended to the final routed circuit and the cost function H (Equation (4)) is evaluated only when a two-qubit gate of the first layer F involves a pair of qubits not physically connected, so only when swap or bridge operations are necessary. However, for quantum technologies having a *fully-connected topology*, such as NMR and trapped ions (single-trap devices), the information on the allowed interactions is *implicit* rather than explicit: all the interactions are *theoretically* allowed by the device, but in practice, some interactions are *hard to implement*, and their fidelity and execution time could be unacceptable[26], Section 1. Specifically, for NMR devices, the hard-to-implement interactions are the ones acting on nuclei exhibiting a weak J-coupling, while, for trapped ions devices, the problematic interactions are related to distant ions inside the trap (following the model presented in ref. [18]).

Therefore, even if routing techniques are not mandatory for fully-connected quantum computing devices, in practice, they might be *essential to have meaningful results*. For this reason, in addition to the approaches described in Section 3.3.1, the library discussed in this manuscript proposes two novel hardware-aware heuristic approaches — one based on breaking the full connectivity, the other called Hardware-Aware Smart — to perform the routing phase targeting devices having a fully-connected topology. They are implemented in the proposed library, both with the aim of avoiding the application of two-qubit gates on the weaker interactions of the target NISQ device, and each of these is described in one of the two following sections.

3.3.3. First Proposed Routing Strategy: Breaking the Full Connectivity

The first methodology consists in *breaking the fully-connected topology* by defining a threshold on the minimum interaction strength accepted. In this way, a partially-connected topology is obtained, thus allowing one to employ the algorithms discussed in Section 3.3.1. A conceptual flow of the algorithm is shown in **Figure 3**a for the crotonic acid NMR backend, characterized by two interactions ($n_0 \leftrightarrow n_2$ and $n_1 \leftrightarrow n_3$) much weaker than all the others and removed, assuming a minimum admitted coupling of 2 Hz. In the example, the HR algorithm, undoubtedly more powerful than BR, is exploited for the generation of the routed circuit. In conclusion, breaking the connectivity is the simplest way to arrange the state-of-the-art routing algorithms to fully-connected backends. This mechanism is expected to provide some advantages for backends characterized by interactions with strengths much different from each other, like the aforementioned crotonic acid.

3.3.4. Second Proposed Routing Strategy: Hardware-Aware Smart

The *Hardware-Aware Smart Routing* (HRS) is an adaptation of the original *Hardware-Aware Routing algorithm* presented in ref. [20], expanded to target fully-connected quantum devices, with the aim of *optimising the final quantum circuit execution time*. As in the original version, the algorithm iterates over the input quantum circuit to construct an output one. Every time a two-qubit gate is found in F, this approach prescribes computing the best swap following the same cost function of the Hardware-Aware Routing algorithm (cf. Equation (4)), and it evaluates if adding this auxiliary swap (also considering the overhead to perform it)

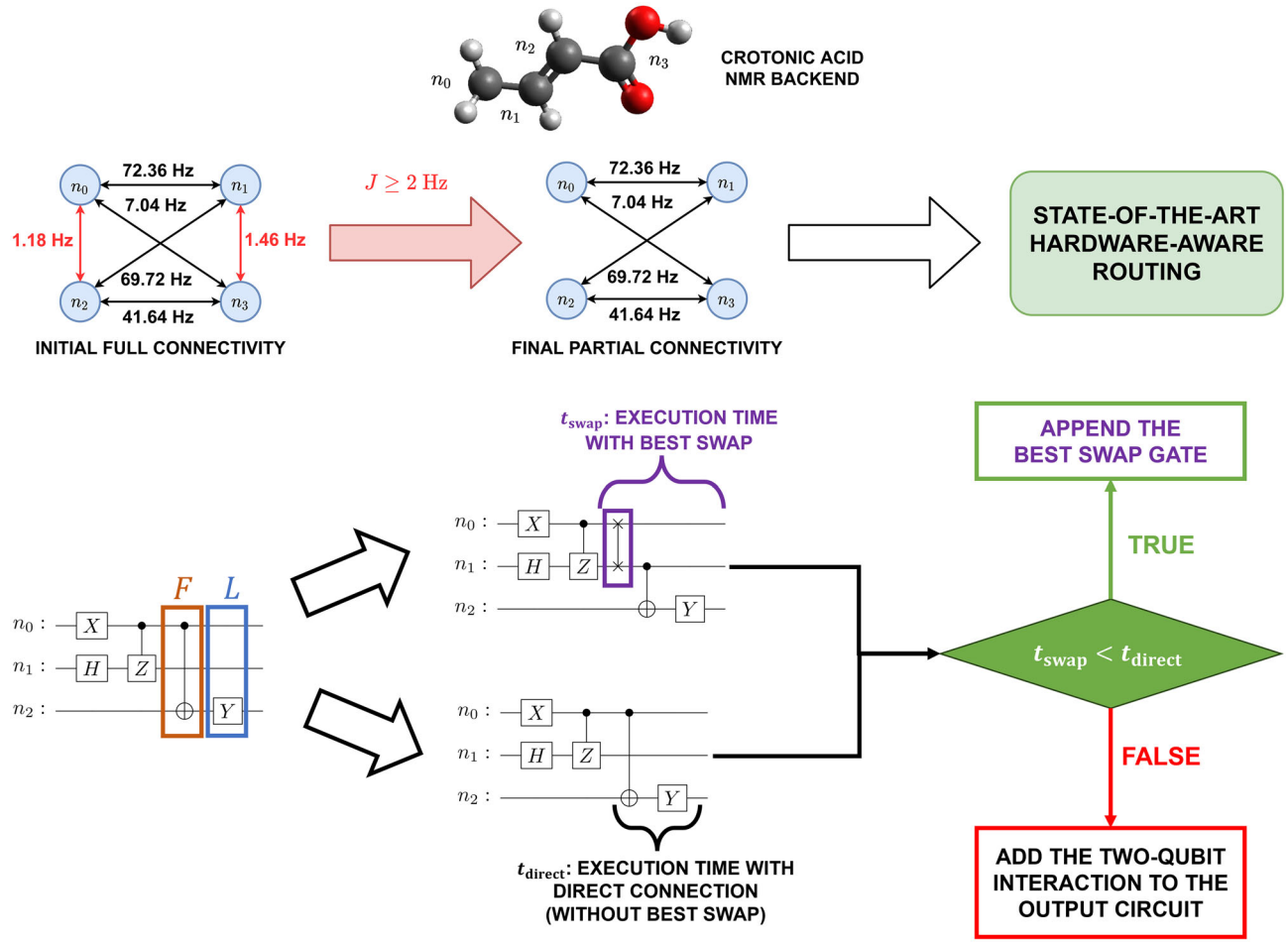

Figure 3. Overview of the proposed routing strategies for fully-connected backends.

could improve the final execution time with respect to the direct insertion of the two-qubit gate. Figure 3b shows an example of the aforementioned execution time comparison, associated with a CX gate between physical qubits n_0 and n_2. After recognizing that the best swap involves n_0 and n_1, the algorithm checks if t_{swap}, i.e., execution time with the best swap, is lower than the duration with the "direct" CX t_{direct} and chooses the sequence of gates to be appended to the output circuit list.

HRS consists of the steps reported in the following:

1. An empty quantum circuit is considered (*routedCircuit*) and the distance matrix D of the target device is computed (*initialization step*).
2. The first layer F of the input placed circuit (*placedCircuit*) is extracted — i.e., isolated and removed from the *placedCircuit*— and the lookahead layer L is considered.
3. The single-qubit gates of F are directly inserted in routedCircuit.
4. For each two-qubit gate in F, the best swap (*bestSwap*) is computed exploiting the cost function in Equation (4).
5. For each two-qubit gate in F, *the approximate first and lookahead layer execution times* (c.f. Equation (5)) are computed for two different configurations: the first one assuming that *bestSwap* has been added to the output circuit (*suppSwapTotalGateTime*), the second one supposing the circuit has been left untouched (*currTotalGateTime*).
6. If the swap insertion minimises the execution time, then *bestSwap* is added together with the two-qubit gate in the *routedCircuit*. Otherwise, the two-qubit gate is directly inserted in the *routedCircuit*.
7. Repeat from 2) until the placedCircuit is empty.

The approximate execution time is obtained by exploiting the following equation:

$$\mathcal{T}(F, L) = \sum_{gate \in F} \mathcal{T}(gate) + \sum_{gate \in L} \mathcal{T}(gate) \tag{5}$$

where $gate \in F$ and $gate \in L$ are the two-qubit gates of the first and lookahead layer respectively, and $\mathcal{T}(gate)$ is the gate execution time. The latter is computed in a different way in the two considered scenarios. In the case of swap insertion, the interacting nodes might change as a consequence of the swap addition and the time required for performing the swap is added to $\mathcal{T}(F, L)$ for taking into consideration the overhead.

Algorithm 1 Hardware-Aware Smart routing algorithm

Input: Circuit after placement placedCircuit and Backend characteristics backend
Output: Routed quantum circuit routedCircuit
Initialize:
Set routedCircuit to an empty Circuit;
Execute:
// Analyse the placed circuit layer-by-layer
while not *placedCircuit*.isEmpty() **do**
  // Extract the current layer
  $F \leftarrow$ *placedCircuit.extractLayer*1();
  // Extract the lookahead layer
  $L \leftarrow$ *placedCircuit.getLookAheadLayer*();
  // Directly insert single-qubit gates in routedCircuit
  for *singe_qubit_gate* $\in F$ **do**
    routedCircuit.append(*singe_qubit_gate*);
  end for
  // Explore two-qubit gates in the first layer
  while *two_qubit_gate* $\in F$ **do**
    curr_gate $\leftarrow$ *two_qubit_gate*;
    // Compute the best swap
    bestSwap $\leftarrow$ *getBestSwapGate*()
    // Compute the execution time (current layer + lookahaed layer) with
    // the direct insertion of the two-qubit gate
    currTime $\leftarrow 0$;
    for *gate* $\in F \cup L$ **do**
      currTime $\leftarrow$ *currTime* + *gate.GateTime*();
    end for
    // Compute the execution time (current layer + lookahaed layer) inserting
    // the best swap
    $F' \leftarrow F$ assuming the best swap gate is added.
    $L' \leftarrow L$ assuming the best swap gate is added.
    newTime $\leftarrow$ *bestSwapgate.GateTime*();
    for *gate* $\in F' \cup L'$ **do**
      newTime $\leftarrow$ *newTime* + *gate.GateTime*();
    end for
    // Choose the solution which minimises the execution time
    if *newTime* $<$ *currTime* **then**
      routedCircuit.append(*bestSwapGate*);
    else
      routedCircuit.append(*curr_gate*);
    end if
  end while
end while
Return: routedCircuit

The proposed library is able to *estimate the gate execution time* exploiting the device information contained inside the backend configuration file, with two required approximations:

1. For superconducting quantum devices, the time required for executing single-qubit gates is not taken into account when gate is a CZ gate, thus approximating its gate time with the gate time required for executing a CX between the same target nodes.
2. If gate is not a CX or CZ gate (even though this is the expected scenario since the layout synthesis is routinely performed before the CX or CZ gate decomposition):
   - For superconducting and trapped ions devices, all the non-CX and non-CZ gates are considered CX gates for the gate time computation.
   - For NMR and quantum dot devices, all the non-CX and non-CZ gates are considered CZ gates for the gate time computation.

The pseudo-code of the HRS algorithm is reported in **Algorithm 1**. Particular attention is provided to the operations required for the comparison of the approximate execution times with direct interaction and best swap, depending on F and L. The external loop is related to the fact that the final routed circuit is

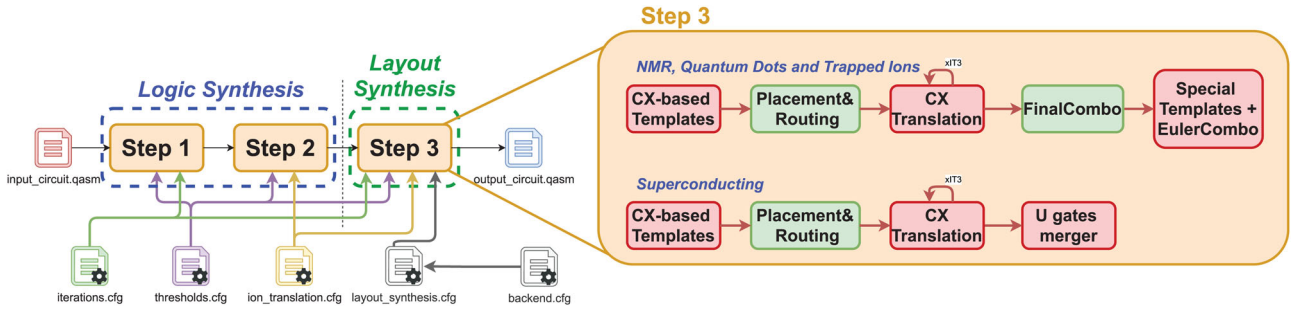

Figure 4. Integration of the presented layout library in the compilation toolchain presented in ref. [6].

obtained when all the gates of the placed circuit have been examined by the algorithm.

3.3.5. Overall Comparisons

A recap comparison between the proposed HRS, the state-of-the-art HR and the proposed method based on breaking the full connectivity is reported:

- In HRS, the direct interaction of a pair of qubits is not always assumed as the best way to implement a two-qubit gate between them, as in HR. In fact, HRS always establishes, given F and L layers, the most convenient circuit among that with the direct interaction and that employing the best swap.
- The comparison of the performance of native and best-swap-involving two-qubit gates is done, in the proposed HRS algorithm, in terms of time duration. The cost function depending on D, employed in HR, is only employed to find the best swap.
- HRS can be considered *more circumstantial* than the approach based on breaking the connectivity, since it does not exclude a priori the "weak" interactions of a given backend and chooses the best circuit according to the context, given by the lookahead layer L.

4. Integration in the Avitabile et al. Toolchain

The presented layout library was integrated into the *compilation toolchain* presented in ref. [6], which supports superconducting qubits (with IBMQ native gates), NMR and trapped ions and performs *only the logic synthesis*.

First, *quantum dots* were included in the set of target technologies, also exploiting the native gates similarity with NMR. In fact, the single-qubit gates $R_{x,y,z}(\theta)$ are supported by both technologies and the quantum dots exchange interaction and the NMR J-coupling have an analogous unitary evolution, traceable back to $R_{zz}(\theta)$.[6,9,15]

Moreover, *the layout synthesis was inserted in toolchain's Step 3* (see ref. [6]), as shown in **Figure 4**. The layout synthesis is executed before the CX and CZ gates decomposition in technology native gates to simplify the routing phase. Finally, after the two-qubit gates translation for NMR, quantum dots and trapped ions, the last compaction is inserted to optimize the newly added gates.

5. Results

This section reports the average results obtained by synthesising several benchmark circuits with the layout approaches of the *proposed libraries*, *Qiskit transpiler*[7] (Qiskit Terra library version 0.18.3[27]) and t|ket⟩ compiler[8] (pytket library version 1.3.0). The Supporting Information file shows the complete results for all the tested algorithms. The codes employed for obtaining the reported results are available in the GitHubrepository.

5.1. Setup

All tests are performed by synthesising each benchmark circuit with the placement and routing methods of the developed library, the Qiskit transpiler and the t|ket⟩ compiler. In particular, for *placement*, the following methods are considered:

- proposed library: *TrivialMapping* (TM), *SimulatedAnnealingDenseMapping* (DM) and *SimulatedAnnealingHardwareAwareMapping* (HM);
- Qiskit: *DenseLayout*(DL), *DenseLayoutUseErrors* (DLUE), *SabreLayout* (SL) and *NoiseAdaptiveLayout* (NAL);
- t|ket⟩: *LinePlacement* (LP), *GraphPlacement* (GP), *NoiseAwarePlacement* (NAP) and *NoiseAwarePlacementWithNodeErrors* (NAPWNE).

At the same time, for *routing*, the analysed approaches are the following ones:

- proposed library: *BasicRouting* (BR), *HardwareAwareRouting* (HR) — breaking the connectivity in case of fully-connected topologies (BC) by imposing a threshold on the minimum interaction strength — and *HardwareAwareRoutingSmart* (HRS) for fully-connected technologies;
- Qiskit: *StochasticSwap* (StoS) and *sabreSwap* (SaS);
- t|ket⟩: *LexiRoute* (LR).

Concerning the Simulated Annealing Dense Mapping, the Hardware Aware Routing, and the Hardware Aware Routing Smart, the benchmarks are carried out considering the coefficient α_1 always equal to 0.5, the coefficient α_2 equal to 0.0 (E 0.0) or 0.5 (E 0.5), the coefficient α_3 equal to 0.0 (T 0.0) or 0.5 (T 0.5) and a virtual (RZ) or not virtual (NRZ) R_z gates implementation — the gate time and error rate are zero in case of a

virtual implementation,[28] otherwise, they are determined assuming the R_z gate is decomposed in R_x and R_y gates.[29]

The methodology to compute the error rates and the gate times of single and two-qubit gates employed in the library algorithms and for processing the obtained results is technology dependent:

- For superconducting, the U_2 and CX error rates and gate times are extrapolated from the backend system properties of `FakeToronto` (cf. Section 2.1). When the R_z gates are implemented virtually, the U_1 (substantially corresponding to R_z) error rates and gate times are set to 0. Otherwise, they are obtained by supposing that R_z is implemented with a worst-case U_3 gate. Moreover, the U_3 error rates and gate times are set to $\mathcal{E}(U_3) = 2 \cdot \mathcal{E}(U_2)$ and $\mathcal{T}(U_3) = 2 \cdot \mathcal{T}(U_2)$. Finally, the error rate and gate time of the swap gate between i^{th} and j^{th} nodes — assuming to decompose it by exploiting three consecutive CX gates — is computed by considering a best-case scenario[20], Section 3.A, i.e., exploiting the following equations:

$$\mathcal{E}(\text{swap}) = 1 - \left(\mathcal{S}(\text{CX}_{n_i,n_j}) \cdot \mathcal{S}(\text{CX}_{n_j,n_i}) \cdot \max\left(\mathcal{S}(\text{CX}_{n_i,n_j}), \mathcal{S}(\text{CX}_{n_j,n_i})\right)\right) \quad (6)$$

$$\mathcal{T}(\text{swap}) = \mathcal{T}(\text{CX}_{n_i,n_j}) + \mathcal{T}(\text{CX}_{n_j,n_i}) + \min\left(\mathcal{T}(\text{CX}_{n_i,n_j}), \mathcal{T}(\text{CX}_{n_j,n_i})\right)$$

where $\mathcal{S}(\text{CX}_{n_a,n_b})$ ($\mathcal{T}(\text{CX}_{n_a,n_b})$) is the success rate (execution time) for implementing a CX gate between the a and b nodes and can be obtained from the backend system properties of `FakeToronto`.
- For quantum dots, the $R_x(\frac{\pi}{2})$, CZ and swap error rates and gate times are computed as the average of many simulations performed using MATLAB QuanTO[14,15] assuming the swap consists of three CX gates. The $R_x(\theta)$ error rates and gate times are computed as:

$$\mathcal{E}(R_x(\theta)) = \frac{\mathcal{E}(R_x(\frac{\pi}{2})) \cdot \theta}{\frac{\pi}{2}} \quad (7)$$

$$\mathcal{T}(R_x(\theta)) = \frac{\mathcal{T}(R_x(\frac{\pi}{2})) \cdot \theta}{\frac{\pi}{2}} \quad (8)$$

  The R_y error rates are assumed to be equal to the R_x ones, and the R_z gates — if not implemented virtually — are supposed to be decomposed by using these latter gates.
- For NMR, the same methodology of quantum dots is employed for the single-qubit, CZ and swap gates error rate estimation.
- For trapped ions, the $R_x(\frac{\pi}{2})$ and $R_y(\frac{\pi}{2})$ gate times and error rates are retrieved from the sample values reported in ref. [16]. As for the spin technologies, the error rates for $R_x(\theta)$ and $R_y(\theta)$ are computed proportionally to the $\theta = \frac{\pi}{2}$ case. The CX gate time is determined supposing a linear dependency with the interacting ions distance, following the model presented in ref. [17]. The error rate depends on the gate time, according to the approach used in ref. [18] (the greater the time of application of the two-qubits interaction, the greater the error rate). Finally, the error rate and gate time of the swap gate are computed as for superconducting technology.

The benchmark quantum circuits considered are described in *OpenQASM 2.0* and are retrieved from ref. [30] and ref. [31]. Following the approach used by ref. [24] and ref. [20], these are *divided into three sets*: *small-sized*, *medium-sized* and *large-sized*, depending on the size of the quantum register, selecting only circuits with *fewer than 35000 quantum gates*. However, for NMR and quantum dot technologies, only the small-sized circuit set is considered due to the limited size of the currently available physical devices.

In order to evaluate the performance of layout synthesis approaches independently from the logic synthesis, all circuits are *rebased* to the *technology's legal set of gates* (U_1, U_2, U_3, CX for superconducting technology, R_x, R_y, R_z, CZ for NMR and quantum dots technology and R_x, R_y, R_z, CX for trapped ions technology) by exploiting the Qiskit transpiler with *optimization level equal to 0* to avoid any further optimization.

5.2. Functional Equivalence Checking

A functional equivalence checking was performed for all the developed layout strategies to verify the correctness of the implemented placement and routing algorithms in the proposed work. In particular, the frequency of measurement for each eigenstate of the output circuit (after layout synthesis) is compared with that of the input circuit in terms of the distance (*Kullback Leibler divergence*[32]) and similarity (*Hellinger fidelity*[33] and *approximated Fidelity*[29]) between the two distributions.

5.3. Figures of Merit

Multiple figures of merit can be employed to compare quantum circuits obtained with different compilation strategies. In the following, the two metrics employed in the main article are described.

The first one is the *circuit cost*[34], which provides an overall estimate of the quality of the compilation process, by taking into account the depth of the compiled circuit and the fidelity of quantum gates:

$$C = -D \log K - \sum_i \log F_i^{1q} - \sum_j \log F_j^{2q} \quad (9)$$

In the previous formula, D is the output circuit depth, K is a constant penalization factor, F_i^{1q} is the i^{th} single-qubit gate fidelity and F_j^{2q} is the j^{th} two-qubit gate fidelity. F_i^{1q} and F_j^{2q} are both computed as $1 - \mathcal{E}_{\text{gate}}$, with $\mathcal{E}_{\text{gate}}$ gate error rate (obtained as explained in Section 5.1). Different K values are used for the four backends to ensure that $F_{avg}{}^{1q} < K < F_{avg}{}^{2q}$, as suggested in ref. [34]: $K = 0.9892$ for superconducting devices, $K = 0.9893$ for NMR, $K = 0.9994$ for quantum dots and $K = 0.9789$ for trapped ions.

The second metric is the *total execution time* of the compiled quantum circuit:

$$t_{\text{exec.}} = \sum_{gate \in G} t_{gate} \quad (10)$$

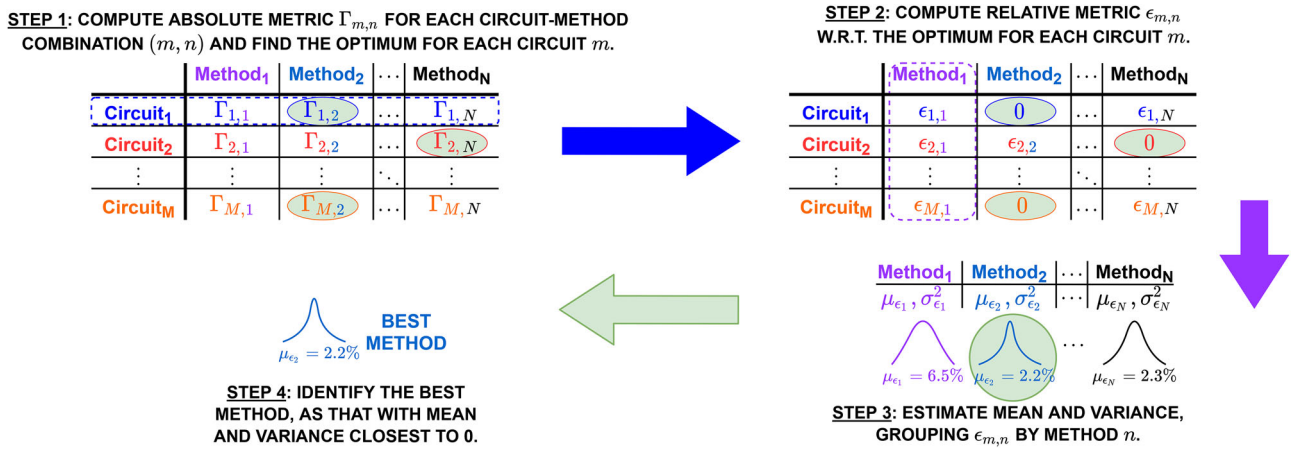

Figure 5. Overview of the flow for the identification of the average best compilation method.

where G is the set of all gates in the circuit, and t_{gate} is the duration of the single quantum gate.

In this article, average results are reported, for each technology and each compilation strategy, grouping the circuits according to their dimension (small, medium, or large). **Figure 5** shows the flow — followed in this work for the circuit cost C and the execution time $t_{exec.}$ — that allows the identification of the average best compilation method for a generic metric Γ. In the first step, the metrics $\Gamma_{m,n}$ for each circuit m and each method n are computed and grouped by circuit (by row), to find the method with the best metric for each circuit $\Gamma_{opt,m}$, highlighted by green ellipses. It is reminded that, for C and $t_{exec.}$ metrics, the lowest value corresponds to the optimal one.

Then, for each quantum circuit, the relative variation $\epsilon_{m,n}$ of each metric with respect to the optimal one is evaluated (also in this case, the processing is done by row):

$$\epsilon_{m,n}[\%] = \frac{\Gamma_{m,n} - \Gamma_{opt,m}}{\Gamma_{opt,m}} \cdot 100 \tag{11}$$

Accordingly, a better-performing method is characterized by a higher number of $\epsilon_{m,n} = 0$. At this point, as shown in the third step, the relative variations are grouped by method (by column), to identify for each of these a mean value μ_{ϵ_n} and a variance $\sigma^2_{\epsilon_n}$. In the example reported in the figure, the method $n = 2$ shows the lowest mean and variance (narrowest Gaussian with the lowest mean). Hence, it is finally identified as the best method.

As comprehensible from the previous flow, the goal is to identify a compilation method for each technology that provides *overall better performance*, considering a wide range of benchmark quantum circuits with different characteristics, such as size, length, and involved gates, employable for different application purposes. Reporting results associated with specific circuits cannot provide complete information about the effectiveness of the considered compilation algorithms, since the latter could perform better with some specific circuits (e.g., Boolean reversible) and worse with others (e.g., variational). Therefore, testing the placement and routing methods with many different quantum circuits could help to limit the impact of a specific circuit in the evaluation. Moreover, averaging the obtained results is expected to be a reasonable way to have immediate and intuitive figures of merit since a single number can provide an overall idea of the performance.

Averaging absolute values associated with the metrics could lead to a high sensitivity of the results on the characteristics of the benchmark circuits. For example, in the circuit cost case, the averaging procedure would be expected to be more affected by larger circuits than smaller ones. On the contrary, averaging relative variations with respect to the optimum could help to get more readable final results, not affected by the aforementioned issues, thus also allowing a more critical and fair performance evaluation, not significantly dependent on the circuit characteristics.

Since the considered backends involve a different number of qubits, results are grouped by quantum circuit dimension. In this way, the analyses are limited to circuits compatible, in terms of the number of qubits, with the examined quantum computers. For example, NMR and quantum dots, differently from superconducting and trapped ions, cannot handle medium and large circuits. Moreover, the circuit sets are the same for all technologies to ensure that the evaluation of the overall best method is the same for all technologies. Finally, grouping circuits by size additionally proves that averaging relative variations ensures results comparable in all cases, with a reduction of the metric sensitivity with respect to circuit dimensions.

Before concluding, it shall be noted that the Supporting Information file reports all the absolute values results, in terms of circuit cost, execution time, and swap gates count, for each quantum circuit taken individually.

5.4. Evaluation of Results

Table 1 reports the results obtained by compiling the benchmark circuits for fully-connected backends. It is possible to observe that the results associated with Qiskit and t|ket⟩ are related to placement algorithms only. The reason is that the heuristics of these compilers ignore the routing procedure for fully-connected backends. As shown on the left size of **Figure 6**, Qiskit and t|ket⟩ compilers execute routing only when the backend is partially-connected, so they do not try to associate the two-qubit gates with

Table 1. Results for fully-connected technologies. The arithmetic average (μ_ϵ and μ_{ϵ_t}) and the variance (σ^2_ϵ and $\sigma^2_{\epsilon_t}$) of cost error (ϵ_C) and the relative execution time variation (ϵ_t) are reported for each technology, algorithm, and circuit set (the lower, the better). The best values for each combination are in bold.

Method		μ_ϵ	σ^2_ϵ	μ_{ϵ_t}	$\sigma^2_{\epsilon_t}$	Technology
	DLUE	6.49e-01	9.47e-01	9.50e+00	1.14e+02	Trapped ions (L)
	DLUE	6.38e-01	1.13e+00	8.91e+00	2.03e+02	Trapped ions (M)
	DLUE	9.55e-02	9.16e-02	1.50e+00	2.25e+01	Trapped ions (S)
	DLUE	6.63e+00	2.73e+01	2.06e+01	1.15e+03	NMR (S)
	DL	6.49e-01	9.47e-01	9.50e+00	1.14e+02	Trapped ions (L)
	DL	6.38e-01	1.13e+00	8.91e+00	2.03e+02	Trapped ions (M)
	DL	9.55e-02	9.16e-02	1.50e+00	2.25e+01	Trapped ions (S)
	DL	6.63e+00	2.73e+01	2.06e+01	1.15e+03	NMR (S)
qiskit	NAL	5.91e-01	8.45e-01	8.68e+00	1.09e+02	Trapped ions (L)
	NAL	1.20e-01	5.06e-02	1.62e+00	9.19e+00	Trapped ions (M)
	NAL	8.77e-02	8.92e-02	1.38e+00	2.19e+01	Trapped ions (S)
	NAL	6.20e+00	3.26e+01	2.22e+01	7.16e+02	NMR (S)
	SL	5.88e+00	8.06e+00	7.12e+00	1.30e+03	Trapped ions (L)
	SL	4.04e+00	5.82e+00	5.44e+00	7.75e+02	Trapped ions (M)
	SL	1.99e+00	3.44e+00	3.41e+01	7.77e+02	Trapped ions (S)
	SL	2.35e+01	3.26e+02	1.18e+02	1.30e+04	NMR (S)
	GP	7.43e-01	1.56e+00	9.36e+00	8.20e+01	Trapped ions (L)
	GP	2.00e+00	8.98e+00	7.33e+00	4.62e+01	Trapped ions (M)
	GP	7.09e-01	5.11e+00	2.07e+00	2.49e+01	Trapped ions (S)
	GP	8.61e+00	4.88e+01	2.53e+01	1.32e+03	NMR (S)
	LP	4.40e+00	1.14e+01	5.10e+01	1.50e+03	Trapped ions (L)
	LP	5.07e+00	8.98e+01	5.50e+01	1.20e+03	Trapped ions (M)
	LP	3.72e+00	2.11e+01	5.30e+01	4.76e+03	Trapped ions (S)
	LP	1.01e+01	2.47e+02	6.25e+01	8.23e+03	NMR (S)
t|ket⟩	NAPWNE	3.19e+00	8.12e+00	3.66e+01	1.03e+03	Trapped ions (L)
	NAPWNE	4.36e+00	9.77e+00	4.12e+01	8.39e+02	Trapped ions (M)
	NAPWNE	1.36e+00	1.02e+01	1.22e+01	1.47e+03	Trapped ions (S)
	NAPWNE	3.33e+00	3.46e+01	2.01e+01	8.13e+02	NMR (S)
	NAP	3.19e+00	8.14e+00	3.66e+01	1.03e+03	Trapped ions (L)
	NAP	4.36e+00	9.77e+00	4.12e+01	8.39e+02	Trapped ions (M)
	NAP	1.36e+00	4.12e+01	1.22e+01	1.47e+03	Trapped ions (S)
	NAP	6.00e+00	8.39e+01	1.93e+01	8.33e+02	NMR (S)
DM		6.49e-01	9.47e-01	9.50e+00	1.14e+02	Trapped ions (L)
		6.38e-01	1.13e+00	8.91e+00	2.03e+02	Trapped ions (M)
		9.55e-02	9.16e-02	1.50e+00	2.25e+01	Trapped ions (S)
		7.59e+00	4.20e+01	2.11e+01	1.17e+03	NMR (S)
		5.14e-01	5.28e-01	7.07e+00	3.80e+01	Trapped ions (L)
HM E0.0 T0.5 HR		**1.65e-04**	**1.53e-07**	**6.77e-15**	**3.66e-28**	Trapped ions (M)
E0.0 T0.5 BC RZ		**0.00e+00**	**0.00e+00**	**6.57e-16**	**1.04e-29**	Trapped ions (S)
		4.03e+00	2.08e+01	**4.29e-01**	**3.68e+00**	NMR (S)
		1.44e+00	5.83e+00	**2.85e+00**	**1.70e+01**	Trapped ions (L)
HM E0.0 T0.5 HRS		2.20e-01	1.79e-01	1.46e+00	1.13e+01	Trapped ions (M)
E0.0 T0.5 RZ		**0.00e+00**	**0.00e+00**	**6.57e-16**	**1.04e-29**	Trapped ions (S)
		5.89e+00	2.67e+01	9.91e+00	4.78e+02	NMR (S)
		3.86e-01	2.28e-01	6.66e+00	3.83e+01	Trapped ions (L)
HM E0.0		1.19e-01	5.92e-02	1.61e+00	1.10e+01	Trapped ions (M)
T0.5 NRZ		**0.00e+00**	**0.00e+00**	**6.57e-16**	**1.04e-29**	Trapped ions (S)
		7.17e+00	3.77e+01	1.84e+01	1.17e+03	NMR (S)
		3.87e-01	2.18e-01	6.71e+00	3.76e+01	Trapped ions (L)

(*Continued*)

Table 1. (Continued)

Method	μ_ϵ	σ^2_ϵ	μ_{ϵ_t}	$\sigma^2_{\epsilon_t}$	Technology
HM E0.0	5.50e-02	1.06e-02	7.39e-01	1.94e+00	Trapped ions (M)
T0.5 RZ	**0.00e+00**	**0.00e+00**	**6.57e-16**	**1.04e-29**	Trapped ions (S)
	6.42e+00	3.70e+01	1.84e+01	1.17e+03	NMR (S)
	1.68e+00	1.03e+01	3.47e+00	2.53e+01	Trapped ions (L)
HM E0.5 T0.0 HRS	1.15e-01	3.99e-02	1.59e+00	7.27e+00	Trapped ions (M)
E0.0 T0.5 RZ	5.22e-16	6.54e-30	**6.57e-16**	**1.04e-29**	Trapped ions (S)
	2.29e+00	**7.92e+00**	1.03e+01	3.56e+02	NMR (S)
	3.89e-01	2.29e-01	6.71e+00	3.79e+01	Trapped ions (L)
HM E0.5	8.87e-02	2.75e-02	1.19e+00	5.02e+00	Trapped ions (M)
T0.0 NRZ	5.22e-16	6.54e-30	**6.57e-16**	**1.04e-29**	Trapped ions (S)
	5.97e+00	3.83e+01	2.76e+01	1.28e+03	NMR (S)
	3.46e-01	**1.97e-01**	6.28e+00	3.33e+01	Trapped ions (L)
HM E0.5	1.10e-01	4.78e-02	1.49e+00	8.83e+00	Trapped ions (M)
T0.0 RZ	5.22e-16	6.54e-30	**6.57e-16**	**1.04e-29**	Trapped ions (S)
	3.13e+00	2.90e+01	1.64e+01	8.10e+02	NMR (S)
	7.74e-01	1.12e+00	9.79e+00	1.14e+02	Trapped ions (L)
TM HR E0.0	6.38e-01	1.13e+00	8.91e+00	2.03e+02	Trapped ions (M)
T0.5 BC RZ	9.55e-02	9.16e-02	1.50e+00	2.25e+01	Trapped ions (S)
	4.97e+00	2.44e+01	1.49e+00	1.41e+01	NMR (S)
	1.94e+00	1.23e+01	4.26e+00	5.03e+01	Trapped ions (L)
TM HRS	8.06e-01	2.19e+00	7.11e+00	1.24e+02	Trapped ions (M)
E0.0 T0.5 RZ	2.03e-01	6.76e-01	1.10e+00	8.58e+00	Trapped ions (S)
	6.61e+00	2.68e+01	1.11e+01	4.92E+02	NMR (S)
	1.94e+00	1.23e+01	4.28e+00	4.99e+01	Trapped ions (L)
TM HRS	8.06e-01	2.19e+00	7.11e+00	1.24e+02	Trapped ions (M)
E0.0 T0.0 RZ	2.03e-01	6.76e-01	1.10e+00	8.58e+00	Trapped ions (S)
	6.21e+00	2.29e+01	1.04e+01	4.89e+02	NMR (S)
TM	6.49e-01	9.47e-01	9.50e+00	1.14e+00	Trapped ions (L)
	6.38e-01	1.13e+00	8.91e+00	2.03e+02	Trapped ions (M)
	9.55e-02	9.16e-02	1.50e+00	2.25e+01	Trapped ions (S)
	7.59e+00	4.20e+01	2.11e+01	1.17e+03	NMR (S)

the physical qubits characterized by a stronger interaction and connections like those eliminated in Figure 3a are still considered valid.

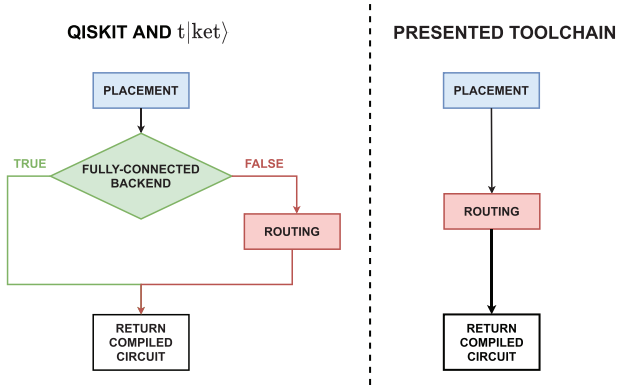

Figure 6. Comparison of place-and-route flow-charts of Qiskit/t|ket⟩ and the proposed compiler.

The main consequence of the previous observation is that the only results involving routing methods and reported in the fully-connected backends table are associated with the two algorithms proposed in this article for the first time: *HRS* and HR with BC. It is not surprising that *the new layout synthesis methods implemented in the proposed library always outperform the ones available in Qiskit and* t|ket⟩ *for fully connected topologies*. In fact, the possibility to employ both placement and routing procedures with fully-connected technologies (and not only the placement ones, like in Qiskit and t|ket⟩) allows the generation of quantum circuits more tailored to the target backend, with a higher degree of consciousness of the "effective quality" of the connections. For what concerns the specific results, it is possible to notice that applying the HRS for NMR technology ensures relevant benefits. In contrast, in the case of trapped ions technology, it is advantageous to employ HR breaking the connectivity for medium size circuits.

Table 2 reports the results obtained by compiling the benchmark circuits for non-fully-connected backends. In this case, the routing step is fundamental, and so only the results of a complete layout synthesis are reported. It is possible to observe that t|ket⟩

Table 2. Results for non-fully-connected technologies.The arithmetic average (μ_ϵ and μ_{ϵ_t}) and the variance (σ^2_ϵ and $\sigma^2_{\epsilon_t}$) of cost error (ϵ_C) and the relative execution time variation (ϵ_t) are reported for each technology, algorithm and circuit set (the lower, the better). The best values for each combination are in bold.

Method		μ_ϵ	σ^2_ϵ	μ_{ϵ_t}	$\sigma^2_{\epsilon_t}$	Technology
	DLUE-SaS	4.24e+02	1.60e+04	1.88e+03	1.90e+06	Supercond. (L)
	DLUE-SaS	1.59e+02	7.96e+03	9.04e+02	9.48e+05	Supercond. (M)
	DLUE-SaS	6.38e+01	2.14e+03	2.03e+02	2.91e+04	Supercond. (S)
	DLUE-SaS	4.39e+01	3.19e+03	1.23e+01	2.58e+02	Q-dots (S)
	DLUE-StoS	5.14e+02	4.17e+04	2.44e+03	2.77e+06	Supercond. (L)
	DLUE-StoS	2.15e+02	1.33e+04	1.27e+03	1.46e+06	Supercond. (M)
	DLUE-StoS	7.91e+01	3.53e+03	2.81e+02	5.23e+04	Supercond. (S)
	DLUE-StoS	5.62e+01	6.81e+03	1.58e+01	5.35e+02	Q-dots (S)
	DL-SaS	4.01e+02	1.74e+04	1.93e+03	1.87e+06	Supercond. (L)
	DL-SaS	1.61e+02	8.65e+03	9.44e+02	1.11e+06	Supercond. (M)
	DL-SaS	6.25e+01	2.32e+03	1.91e+02	3.32e+04	Supercond. (S)
	DL-SaS	4.39e+01	3.20e+03	1.23e+01	2.58e+02	Q-dots (S)
	DL-StoS	4.89e+02	2.40e+04	2.59e+03	3.74e+06	Supercond. (L)
	DL-StoS	2.01e+02	1.47e+04	1.20e+03	1.27e+06	Supercond. (M)
	DL-StoS	6.94e+01	2.73e+03	2.40e+02	4.35e+04	Supercond. (S)
	DL-StoS	7.10e+01	1.16e+04	1.97e+01	8.52e+02	Q-dots (S)
qiskit	NAL-SaS	4.53e+02	4.68e+04	2.06e+03	2.49e+06	Supercond. (L)
	NAL-SaS	2.47e+02	2.28e+04	8.19e+02	8.69e+05	Supercond. (M)
	NAL-SaS	6.82e+01	5.41e+03	1.48e+02	2.61e+04	Supercond. (S)
	NAL-SaS	5.68e+01	5.59e+03	1.49e+01	4.23e+02	Q-dots (S)
	NAL-StoS	6.18e+02	1.38e+05	2.64e+03	3.20e+06	Supercond. (L)
	NAL-StoS	4.83e+02	1.64e+05	1.16e+03	1.35e+06	Supercond. (M)
	NAL-StoS	8.12e+01	7.93e+03	1.84e+02	3.90e+04	Supercond. (S)
	NAL-StoS	8.51e+01	1.04e+04	2.32e+01	8.63e+02	Q-dots (S)
	SL-SaS	5.89e+02	3.46e+04	1.90e+03	1.87e+06	Supercond. (L)
	SL-SaS	3.11e+02	2.34e+04	6.55e+02	4.55e+05	Supercond. (M)
	SL-SaS	1.46e+02	1.73e+04	1.67e+02	2.29e+04	Supercond. (S)
	SL-SaS	3.54e+01	1.96e+03	**8.62e+00**	1.48e+02	Q-dots (S)
	SL-StoS	1.03e+03	1.56e+05	2.41e+03	2.31e+06	Supercond. (L)
	SL-StoS	3.21e+02	2.66e+04	8.01e+02	6.33e+05	Supercond. (M)
	SL-StoS	2.36e+02	9.47e+04	2.00e+02	4.27e+04	Supercond. (S)
	SL-StoS	3.93e+01	2.33e+03	9.87e+00	1.96e+02	Q-dots (S)
	GP	**2.44e+02**	**8.77e+03**	**1.50e+03**	1.19e+06	Supercond. (L)
	GP	**9.85e+01**	**2.27e+03**	**5.89e+02**	**3.83e+05**	Supercond. (M)
	GP	3.39e+01	**6.99e+02**	1.14e+02	1.97e+04	Supercond. (S)
	GP	3.28e+01	1.32e+03	1.09e+01	**1.46e+02**	Q-dots (S)
	LP	4.70e+02	6.03e+04	1.62e+03	1.26e+06	Supercond. (L)
	LP	2.39e+02	1.47e+04	7.39e+02	6.27e+05	Supercond. (M)
	LP	5.88e+01	2.91e+03	1.55e+02	3.70e+04	Supercond. (S)
	LP	3.40e+01	1.39e+03	1.26e+01	1.48e+02	Q-dots (S)
t|ket⟩	NAPWNE-LR	4.25e+02	5.16e+04	1.56e+03	1.25e+06	Supercond. (L)
	NAPWNE-LR	2.69e+02	1.23e+04	6.70e+02	6.47e+05	Supercond. (M)
	NAPWNE-LR	**2.87e+01**	8.78e+02	1.33e+02	3.44e+04	Supercond. (S)
	NAPWNE-LR	**3.17e+01**	1.27e+03	1.13e+01	1.46e+02	Q-dots (S)
	NAP	4.29e+02	5.26e+04	1.55e+03	**1.18e+06**	Supercond. (L)
	NAP	2.64e+02	1.09e+04	6.84e+02	6.93e+05	Supercond. (M)
	NAP	2.88e+01	8.79e+02	1.33e+02	3.45e+04	Supercond. (S)
	NAP	3.38e+01	**1.19e+03**	1.10e+01	1.49e+02	Q-dots (S)
		3.73e+02	4.69e+04	2.32e+03	2.05e+06	Supercond. (L)

(*Continued*)

Table 2. (Continued)

Method	μ_{ϵ}	σ^2_{ϵ}	μ_{ϵ_t}	$\sigma^2_{\epsilon_t}$	Technology
DM	1.89e+02	4.73e+03	1.25e+03	1.74e+06	Supercond. (M)
BR	4.74e+01	2.14e+03	1.60e+02	4.36e+04	Supercond. (S)
	5.03e+01	4.44e+03	1.39e+01	3.56e+02	Q-dots (S)
	2.88e+02	2.37e+04	1.73e+03	1.64e+06	Supercond. (L)
DM HR	1.78e+02	1.31e+04	9.41e+02	8.50e+05	Supercond. (M)
E0.0 T0.5 NRZ	4.50e+01	1.62e+03	1.40e+02	2.81e+04	Supercond. (S)
	4.63e+01	3.14e+03	1.25e+01	2.57e+02	Q-dots (S)
	2.88e+02	2.37e+04	1.73e+03	1.64e+06	Supercond. (L)
DM HR	1.78e+02	1.32e+04	9.40e+02	8.52e+05	Supercond. (M)
E0.0 T0.5 NRZ	4.50e+01	1.62e+03	1.40e+02	2.81e+04	Supercond. (S)
	4.63e+01	3.14e+03	1.25e+01	2.57e+02	Q-dots (S)
	3.45e+02	5.62e+04	1.68e+03	1.77e+06	Supercond. (L)
HM E0.0 T0.5 HR	1.95e+02	1.40e+04	9.02e+02	1.11e+06	Supercond. (M)
E0.0 T0.5 NRZ	4.05e+01	1.91e+03	**1.09e+02**	1.95e+04	Supercond. (S)
	4.43e+01	3.41e+03	1.17e+01	2.92e+02	Q-dots (S)
	3.30e+02	3.88e+04	1.72e+03	1.96e+06	Supercond. (L)
HM E0.0 T0.5 HR	2.69e+02	4.73e+04	8.55e+02	8.48e+05	Supercond. (M)
E0.0 T0.5 RZ	6.20e+01	8.56e+03	1.28e+02	2.59e+04	Supercond. (S)
	4.62e+01	4.06e+03	1.18e+01	3.10e+02	Q-dots (S)
	2.73e+02	2.66e+04	1.70e+03	1.92e+06	Supercond. (L)
HM E0.5 T0.0	1.39e+02	4.31e+03	7.75e+02	5.14e+05	Supercond. (M)
HR E0.5 T0.0 RZ	3.10e+01	1.30e+03	1.37e+02	2.21e+04	Supercond. (S)
	3.97e+01	2.80e+03	1.12e+01	2.45e+02	Q-dots (S)
	3.73e+02	4.69e+04	2.32e+03	2.05e+06	Supercond. (L)
TM	1.89e+02	4.73e+03	1.25e+03	1.74e+06	Supercond. (M)
BR	4.74e+01	2.14e+03	1.60e+02	4.36e+04	Supercond. (S)
	5.03e+01	4.44e+03	1.39e+01	3.56e+02	Q-dots (S)
	2.91e+02	2.08e+04	1.86e+03	1.80e+06	Supercond. (L)
TM HR	1.72e+02	1.05e+04	1.05e+03	1.24e+06	Supercond. (M)
E0.5 T0.5 NRZ	4.78e+01	1.88e+03	1.47e+02	3.08e+04	Supercond. (S)
	4.72e+01	3.24e+03	1.30e+01	2.73e+02	Q-dots (S)
	2.91e+02	2.08e+04	1.86e+03	1.80e+06	Supercond. (L)
TM HR E0.5	1.72e+02	1.05e+04	1.05e+03	1.24e+06	Supercond. (M)
T0.5 RZ	4.78e+01	1.88e+03	1.47e+02	3.08e+04	Supercond. (S)
	4.67e+01	3.18e+03	1.26e+01	2.61e+02	Q-dots (S)
	2.86e+02	2.52e+04	1.72e+03	1.64e+06	Supercond. (L)
TM HR	1.66e+02	1.12e+04	9.12e+02	8.11e+05	Supercond. (M)
E0.0 T0.5 NRZ	4.51e+01	1.70e+03	1.40e+02	2.73e+04	Supercond. (S)
	4.63e+01	3.14e+03	1.25e+01	2.57e+02	Q-dots (S)
	2.88e+02	2.52e+04	1.72e+03	1.63e+06	Supercond. (L)
TM HR	1.67e+02	1.11e+04	9.14e+02	8.10e+05	Supercond. (M)
E0.0 T0.5 RZ	4.51e+01	1.70e+03	1.40e+02	2.73e+04	Supercond. (S)
	4.63e+01	3.14e+03	1.25e+01	2.57e+02	Q-dots (S)
	2.67e+02	1.52e+04	1.71e+03	1.50e+06	Supercond. (L)
TM HR E0.5	1.48e+02	4.40e+03	8.95e+02	7.37e+05	Supercond. (M)
T0.0 NRZ	4.41e+01	1.58e+03	1.38e+02	2.58e+04	Supercond. (S)
	4.72e+01	3.24e+03	1.30e+01	2.73e+02	Q-dots (S)
	2.67e+02	1.52e+04	1.71e+03	1.50e+06	Supercond. (L)
TM HR	1.48e+02	4.40e+03	8.95e+02	7.37e+05	Supercond. (M)
E0.5 T0.0 RZ	4.41e+01	1.58e+03	1.38e+02	2.58e+04	Supercond. (S)
	4.77e+01	3.32e+03	1.31e+01	2.78e+02	Q-dots (S)

methods give slightly better results than ones of the proposed library and Qiskit.

6. Conclusion

This work proposes a layout synthesis library independent of any quantum compilation framework, which supports superconducting transmon qubits, quantum dots, NMR, and trapped ions and allows the effortless comparison of different placement and routing strategies. Moreover, an approach is presented for assessing whether clever routing procedures should be tried even for fully-connected technologies. The results of this study are noteworthy: when the gate features depend on the interacting nodes, even if the target device has no connectivity limitations, still swap gates addition could be beneficial for the reliability of the final quantum circuit.

Concerning future improvements, a possibility would be to move to a compiled and performant language, such as C++. Furthermore, additional quantum technologies may be supported, thanks to the modularity of the library.

The proposed library allows the identification of the best layout method for each analyzed technology. Therefore, even though the exploration of fully-connected technologies shall still be regarded as a proof of concept, the presented framework is hoped to be an encouraging first step in a structured development procedure that could aim to accomplish competitive results on the stage of the state of the art.

Supporting Information

Supporting Information is available from the Wiley Online Library or from the author.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

A.R., M.S., and D.V. contributed equally to this work.

Data availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Keywords

connectivity, layout, placement, quantum technologies, routing

Received: May 16, 2023
Revised: October 5, 2023
Published online: October 31, 2023

[1] E. Zahedinejad, A. Zaribafiyan, *arXiv:1708.05294* **2017**. https://doi.org/10.48550/arXiv.1708.05294
[2] N. Mishra, M. Kapil, H. Rakesh, A. Anand, N. Mishra, A. Warke, S. Sarkar, S. Dutta, S. Gupta, A. P. Dash, R. Gharat, Y. Chatterjee, S. Roy, S. Raj, V. Kumar Jain, S. Bagaria, S. Chaudhary, V. Singh, R. Maji, P. Dalei, B. K. Behera, S. Mukhopadhyay, P. K. Panigrahi, *in Data Management, Analytics and Innovation*, Springer, Singapore **2021**, pp. 101–145.
[3] Y. Cao, J. Romero, J. P. Olson, M. Degroote, P. D. Johnson, M. Kieferová, I. D. Kivlichan, T. Menke, B. Peropadre, N. P. D. Sawaya, S. Sim, L. Veis, A. Aspuru-Guzik, *Chem. Rev.* **2019**
[4] W. Jang, K. Terashi, M. Saito, C. W. Bauer, B. Nachman, Y. Iiyama, T. Kishimoto, R. Okubo, R. Sawada, J. Tanaka, *EPJ Web of Conf.* **2021**, *251*, 03023.
[5] A. W. Cross, L. S. Bishop, J. A. Smolin, J. M. Gambetta, arXiv:1707.03429 **2017**. https://doi.org/10.48550/ARXIV.1707.03429
[6] M. Avitabile, G. A. Cirillo, M. Simoni, G. Turvani, M. Graziano, *Quantum Information Processing* **2022**, *21*, 1.
[7] M. S. Anis, H. Abraham, AduOffei, R. Agarwal, G. Agliardi, M. Aharoni, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, S. Anagolum, E. Arbel, A. Asfaw, et al., *zenodo:2573505* **2021**. https://doi.org/10.5281/zenodo.2573505
[8] S. Sivarajah, S. Dilkes, A. Cowtan, W. Simmons, A. Edgington, R. Duncan, *arXiv:2003.10611* **2020**. https://doi.org/10.48550/arXiv.2003.10611
[9] M. Simoni, G. A. Cirillo, G. Turvani, M. Graziano, M. Zamboni, *ACM J. Emerg. Technol. Comput. Syst. (JETC)* **2021**, *18*, 1.
[10] T. S. Humble, H. Thapliyal, E. Muñoz-Coreas, F. A. Mohiyaddin, R. S. Bennink, *IEEE Design Test* **2019**, *36*, 69.
[11] A. Kole, S. Hillmich, K. Datta, R. Wille, I. Sengupta, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2020**, *39*, 2375.
[12] IBM Qiskit, *IBM Toronto device* https://quantum-computing.ibm.com/services/resources?skip=10&system=ibmq_toronto, (accessed: February 2023).
[13] IBM Qiskit, *IBM FakeToronto class*, https://qiskit.org/documentation/stubs/qiskit.providers.fake_provider.FakeToronto.html, (accessed: February 2023).
[14] D. Costa, M. Simoni, G. Piccinini, M. Graziano, *IEEE Access* **2023**, *11*, 98875.
[15] G. A. Cirillo, *Politecnico di Torino Doctoral Thesis* **2022**. https://iris.polito.it/handle/11583/2971119
[16] IonQ API Calibrations | Cirq, **2022**. https://quantumai.google/cirq/ionq/calibrations, https://quantumai.google/cirq/hardware/ionq/calibrations
[17] C. J. Trout, M. Li, M. Gutierrez, Y. Wu, S.-T. Wang, L. Duan, K. R. Brown, *New J. Phys.* **2018**, *20*, 043038.
[18] P. Murali, D. M. Debroy, K. R. Brown, M. Martonosi, arXiv:2004.04706, **2020** https://doi.org/10.48550/ARXIV.2004.04706
[19] R. Wille, S. Hillmich, L. Burgholzer, *in 2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, Piscataway, New Jersey **2020**, pp. 1–5.
[20] S. Niu, A. Suau, G. Staffelbach, A. Todri-Sanial, *IEEE Trans. Quantum Eng.* **2020**, *1*, 3101614.
[21] R. W. Floyd, *Communications of the ACM* **1962**, *5*, 345.
[22] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *Science* **1983**, *220*, 671.
[23] E. W. Dijkstra, *Numerische Mathematik* **1959**, *1*, 269.
[24] G. Li, Y. Ding, Y. Xie, *in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, Providence RI USA **2019**, pp. 1001–1014.
[25] G. A. Cirillo, G. Turvani, M. Graziano, *IEEE Transactions on Nanotechnology* **2019**, *18*, 1027.

[26] D. Maslov, S. M. Falconer, M. Mosca, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2008**, *27*, 752.
[27] Transpiler (qiskit.transpiler) – Qiskit 0.36.1 documentation, **2022**. https://qiskit.org/documentation/apidoc/transpiler.html, (accessed: February 2023)
[28] D. C. McKay, C. J. Wood, S. Sheldon, J. M. Chow, J. M. Gambetta, *Physical Review A* **2017**, *96*, 2.
[29] M. A. Nielsen, I. L. Chuang, *Quantum computation and quantum information*, 10th anniversary ed edition, Cambridge University Press, Cambridge, New York **2010**.
[30] JKU IIC circuits repository, https://github.com/iic-jku/ibm_qx_mapping/tree/master/examples, (accessed: February 2023).
[31] A. Li, QASMBench circuits repository, https://github.com/pnnl/qasmbench, (accessed: February 2023)
[32] S. Kullback, R. A. Leibler, *The Annals of Mathematical Statistics* **1951**, *22*, 79.
[33] S. Luo, Q. Zhang, *Physical Review A* **2004**, *69*, 032106.
[34] Y. Kharkov, A. Ivanova, E. Mikhantiev, A. Kotelnikov, arXiv:2202.14025, **2022**, http://arxiv.org/abs/2202.14025