# Limiting the Search Space in Optimal Quantum Circuit Mapping

Lukas Burgholzer* Sarah Schneider* Robert Wille*†
*Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
†Software Competence Center Hagenberg GmbH (SCCH), Austria
lukas.burgholzer@jku.at sarah.schneider@jku.at robert.wille@jku.at
https://iic.jku.at/eda/research/quantum/

***Abstract*—Executing quantum circuits on currently available quantum computers requires compiling them to a representation that conforms to all restrictions imposed by the targeted architecture. Due to the limited connectivity of the devices' physical qubits, an important step in the compilation process is to *map* the circuit in such a way that all its gates are executable on the hardware. Existing solutions delivering optimal solutions to this task are severely challenged by the exponential complexity of the problem. In this paper, we show that the search space of the mapping problem can be limited drastically while still preserving optimality. The proposed strategies are generic, architecture-independent, and can be adapted to various mapping methodologies. The findings are backed by both, theoretical considerations and experimental evaluations. Results confirm that, by limiting the search space, optimal solutions can be determined for instances that timeouted before or speed-ups of up to three orders of magnitude can be achieved.**

## I. Introduction

Capabilities of existing quantum computers are steadily growing, as, e.g., witnessed by IBM's hardware roadmap revealed at the end of 2020 that predicted a device with more than a thousand qubits for the year 2023 [1]. While these devices will most certainly not be able to run any useful instances of the famous Grover algorithm [2] or Shor's algorithm [3], Variational Quantum Algorithms [4] (besides many others) have been proposed as a promising way for actually making use of near-term quantum computers, e.g., by using the VQE algorithm to determine the ground state energy of a molecule [5], [6].

In order to execute a conceptual quantum algorithm on an actual device, the algorithm or circuit has to be *compiled* to a representation that adheres to all constraints imposed by the targeted device. Since quantum computers typically only support a limited set of elementary operations, the algorithm's description first has to be *decomposed* to the corresponding gate set [7]–[10]. In addition, not all physical qubits on the device might directly interact with each other. As a consequence, the resulting circuit additionally has to be *mapped* to the architecture of the device, so that any two-qubit operation is applied to physical qubits connected on the device. Typically, this is accomplished by inserting *SWAP* operations that allow to *swap* the circuit's logical qubits from one position to a connected one. Chaining those operations allows for arbitrary changes in position. However, since each added operation decreases the fidelity of the result when executed on the quantum computer, it is vital to keep the overhead of the resulting (mapped) circuit as low as possible.

Unfortunately, the search space that needs to be considered when determining the best possible mappings increases exponentially with the number of involved qubits. Thus, many existing solutions trade off accuracy/minimality for speed [11]–[19]. While these methods are constantly improving, it has been shown that there is lots of room for improvement as they frequently stray far from the optimum [20]. Accordingly, several approaches for generating *optimal* circuit mappings have been proposed in the recent past [20]–[25]. However, these methods extensively explore the immense search space of the mapping problem—significantly limiting their efficiency and applicability.

In this work, we show that this search space can be drastically limited, *while still guaranteeing optimal results*. More precisely, we present generic, architecture-independent observations that allow to substantially reduce the number of permutations to be considered in front of each gate—the origin of the huge search space and complexity. The key idea is that it suffices to permute just enough that any two qubits of the architecture may interact with each other. Those observations are additionally backed by theoretical considerations (based on group theory) showing that corresponding limitations of the search space are indeed guaranteed to preserve optimality. Based on that, strategies are proposed how these findings can be utilized in existing approaches for optimal quantum circuit mapping.

Experimental evaluations confirm the resulting benefits. By limiting the search space using the strategies proposed in this work, instances that previously suffered from timeouts can now be mapped within minutes or speed-ups of up to three orders of magnitude can be achieved—*all while preserving optimality*. The proposed strategies have been integrated on top of the quantum circuit mapping tool QMAP, which is publicly available at https://github.com/iic-jku/qmap as part of the open-source JKQ toolkit for quantum computing [26].

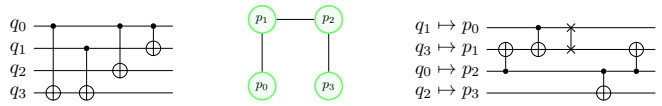
The remainder of this work is structured as follows: In Section II, we provide a review of the mapping problem and what constitutes an optimal solution to this problem. Section III describes our observations that allow to reduce the number of permutations to be considered in front of every gate. Based on that, we afterwards back those observations with a theoretical consideration in Section IV. In Section V, we then propose strategies how these findings can be utilized in existing methods. Experimental evaluations confirming the resulting benefits are summarized in Section VI, before Section VII concludes the paper.

## II. Background

In order to keep this work self-contained, this section establishes the necessary background on the quantum circuit mapping problem and, afterwards, reviews the main idea how existing optimal solutions address this problem. We refer the interested reader to the provided references for further details.

### A. Quantum Circuit Compilation and the Mapping Problem

In order to execute a quantum circuit on an actual quantum computer, it needs to be compiled to a representation that conforms to all the constraints imposed by the architecture of the device. First, the quantum circuit must be expressed using elementary operations supported by the device—a step often referred to as *decomposition* or *synthesis* [7]–[10]. In the following, we assume the elementary gate set to consist of arbitrary single-qubit gates and the controlled-NOT operation (as, e.g., provided by IBM's quantum computers), since this constitutes the de-facto standard to date. However, the findings




(a) Quantum circuit $G$ (b) 4-qubit architecture (c) Mapped circuit $G'$

Fig. 1. Quantum circuit, architecture, and potential mapping

in this work can readily be extended to alternative (future) gate-sets, e.g., including gates acting on more than two qubits.

In addition, most existing quantum computers (those based on superconducting qubits) have a rather limited connectivity between their qubits—typically described by a *coupling graph*. As a consequence, it is necessary to map a circuit's logical qubits (denoted $q_0, \ldots, q_{n-1}$ in the following) to the device's physical qubits (denoted $p_0, \ldots, p_{m-1}$ in the following) so that any elementary operation is applied to qubits connected on the device. Only the most trivial quantum circuits can be directly mapped to the physical architecture. In most cases, the mapping has to change dynamically throughout the circuit in order to conform to all the constraints. This can be accomplished by using *SWAP* gates that allow to interchange the position of two logical qubits on the architecture.

The *mapping problem* (sometimes also synonymously referred to as *qubit routing*, *qubit placement*, or *qubit allocation*), as it is considered in this work, describes the task of determining a representation of an $n$-qubit quantum circuit $G = g_0, \ldots, g_{|G|-1}$ that conforms to all constraints imposed by an $m$-qubit architecture (described by a coupling graph $(V, E)$)—all while keeping the overhead of mapping the circuit, i.e., the number of added gates, as small as possible[1]. When minimizing the number of added gates, we can restrict $G$ to only consists of two-qubit *CNOT* gates since single-qubit gates are not affected by the limited connectivity and, thus, do not require mapping.

**Example 1.** *Consider the four-qubit quantum circuit $G$ composed of four CNOT gates as shown in Fig. 1a and assume it shall be mapped to a four-qubit (linear) architecture described by the coupling graph $(V, E) = (\{p_0, p_1, p_2, p_3\}, \{e_{01}, e_{12}, e_{23}\})$, which is shown in Fig. 1b. Then, Fig. 1c shows one possible mapping of $G$ to this architecture. By assigning $q_0 \mapsto p_2$, $q_1 \mapsto p_0$, $q_2 \mapsto p_3$, and $q_3 \mapsto p_1$, only a single SWAP operation applied to $p_0$ and $p_1$ is needed in order for all gates to be executable.*

While many (heuristic) techniques have been proposed in the past that allow to determine suitable mappings, e.g., [11]–[19], determining truly optimal solutions (with as little overhead as possible) revealed to be a challenging problem. In fact, the mapping problem has been shown to be NP-complete [21], [27].

### B. Optimal Solutions for the Mapping Problem

The complexity of the mapping task mainly comes from the fact that, in principle, any possible permutation of the logical qubits (eventually realized as a series of architecture-conforming *SWAP* operations) might be applied in front each gate of the circuit in order to realize a conforming mapping. An optimal solution to the mapping problem can be determined by finding the right permutations to apply in front of every gate so that the overall resulting number of *SWAPs* is minimal. As a result, for a circuit $G$ with $|G|$ gates to be mapped to an $m$-qubit architecture the search space comprises a total of $|G| * m!$ permutations[2].

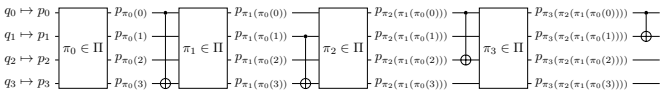



Fig. 2. Symbolic formulation for mapping the circuit shown in Fig. 1a

**Example 2.** *Assume again that the circuit shown in Fig. 1a shall be mapped to the linear architecture shown in Fig. 1b. Let $\Pi$ denote the set of all permutations of four elements. Then, Fig. 2 sketches a symbolic formulation for the mapping task. Conceptually, any permutation $\pi \in \Pi$ can be applied in front of every gate of the circuit. This amounts to $|G| * m! = 4 * 4! = 96$ permutations to be considered for determining an optimal solution.*

Several solutions have been proposed for tackling the resulting complexity. In [19], an exhaustive method is presented that are extended to heuristics for larger problems. *Siraichi et al.* used dynamic programming to determine an optimal solution [21], while *de Almeida et al.* formulated the mapping task as an integer linear programming problem. *Wille et al.* proposed a SAT formulation for the mapping problem in [20]. A systematic enumeration and pruning technique is presented in [25]. Furthermore, there are works seeking a *time-optimal* mapping using SAT [23] or guided search [22].

All these methods have in common that they eventually explore huge parts of the immense search space in order to determine an optimal solution. The question arises whether the search space can somehow be limited while preserving optimality.

## III. Limiting the Search Space

As reviewed above, methods proposed thus far to solve the quantum circuit mapping problem extensively explore the search space spanned by considering *all* possible permutations $\pi \in \Pi$ in front of every gate in order to guarantee an optimal solution. As a result, the size of the search space significantly limits the efficiency of all those approaches.

In this work, we show that the search space of the mapping problem can be significantly limited, *while still guaranteeing optimal results*. This is motivated by three observations which are described in this section. Based on that, we afterwards provide a theoretical argument confirming that these observations indeed preserve optimality (in Section IV) and propose strategies how these findings can be utilized in the methods proposed before (in Section V).

The first observation is based on the maximum length of all pairwise shortest paths between two nodes (i.e., physical qubits) in a coupling graph $(V, E)$, i.e., the longest, direct connection between two nodes. This length is $K$ in the following and can be determined in $\mathcal{O}(|V|^3)$ using, e.g., the Floyd-Warshall algorithm [28]. We observe that it is sufficient to permute just enough so that any two qubits can interact with each other, i.e., instead of all $\pi \in \Pi$ permutations, it is sufficient to only consider permutations which can be realized by at most $K - 1$ *SWAP* operations (a more formal argumentation for that is presented later in Section IV-B). The

[1]There exist other objectives besides minimizing the number of added gates, e.g., minimizing execution *time/depth* or maximizing execution *fidelity*. As will be evident later on, the findings in this work can easily be extrapolated to these objectives by adequately adjusting the considered objective function.

[2]It has been shown in [20] that grouping of gates, e.g., to capture parallel execution of gates, is not guaranteed to produce gate-optimal results. Hence, it is indeed necessary to consider an arbitrary permutation in front of every single gate in order to achieve gate-optimal results.

more connected the coupling graph (i.e., the smaller $K$ in relation to $|V|$, or the larger $|E|$), the easier it is to reach all other qubits from any single qubit. Consequently, this architecture-dependent limitation is most effective whenever the considered architecture is highly-connected.

**Example 3.** *Consider the linear 4-qubit architecture shown in Fig. 1b. Then, the longest (direct) connection involves $p_0$ and $p_3$—hence, $K = 3$. Allowing only up to $K - 1 = 2$ SWAP operations per permutation reduces the number of permutations to be considered in front of every gate from $4! = 24$ down to 9, i.e., by more than half!*

In addition to the above observation, another way to reduce the effective permutations that need to be considered when mapping a quantum circuit is already demonstrated in [20]. Assume a quantum circuit uses less qubits than the architecture provides (i.e., $n < m$). Then, instead of considering the whole architecture at once, one can consider the mapping problem on each connected subgraph of the targeted architecture composed of $n$ nodes—leading to substantially smaller problems to solve. While there are at most $\binom{m}{n}$ potential subgraphs, the sparse nature of typical coupling graphs implies that the actual number of instances is much lower—consequently reducing the overall number of permutations. By limiting the search space for all the different problem instances on various subgraphs, the number of permutations is reduced even further.

**Example 4.** *Assume the circuit shown in Fig. 1a shall be mapped to a 5-qubit linear architecture. Instead of having to consider $5! * 4 = 480$ permutations, there are only $4! * 4 = 96$ per connected sub-graph of four vertices. Since there are only two such subgraphs in the linear architecture (that look exactly like the architecture shown in Fig. 1b), this amounts to a total of 192 permutations to be considered, i.e., a reduction by 60 %.*

Finally, one can observe that the only SWAPs that are relevant before an operation, are those that change the position of either of the operation's qubits. All others can be delayed to a later point in time, as they do not serve to make the gate executable. Thus, any permutation that does not change the position of either of an operation's qubit may be ignored[3].

**Example 5.** *Assume a CNOT operation between $p_0$ and $p_3$ in a linear 4-qubit architecture (as shown in Fig. 1b) shall be applied. Then, the permutations corresponding to the identity or swapping the middle qubits $p_1$ and $p_2$ can be ignored as they cannot possibly change the executability of the gate. In general, the larger the architecture, the more permutations can be ignored in front of every gate.*

Overall, this shows substantial potential in limiting the search space of the problem and, by this, for improving the efficiency of corresponding optimal methods.

## IV. Preservation of Optimality When Limiting the Search Space

While the ideas presented above are merely based on observations, this section provides a formal argument why applying the observations summarized above still yields optimal results. To this end, we are going to use concepts from group theory [29], which are briefly revisited first.

[3]This observation assumes knowledge of the current mapping before each gate and is not feasible for blackbox implementations of the mapping problem such as SAT formulations, but rather applicable to iterative techniques such as informed search algorithms.

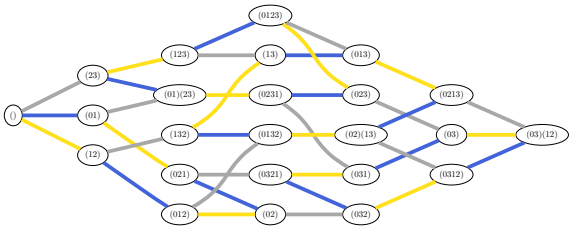



Fig. 3. Cayley graph for the permutation group over $P = \{p_0, p_1, p_2, p_3\}$ using generators (01), (12), (23).

### A. Group Theory and Permutation Groups

A group $(G, *)$ is a set of elements $G$ equipped with a binary operation $* : G \times G \to G$ such that

- $\forall a, b, c \in G$: $(a * b) * c = a * (b * c)$ (*associativity*),
- $\exists e \in G\ \forall g \in G$: $g * e = e * g = g$ (*identity element*),
- $\forall g \in G\ \exists g' \in G$: $g * g' = g' * g = e$ (*inverse element*).

If clear from the context, we will omit the $*$ when denoting group operations, i.e., we will write $g_0 g_1$ instead of $g_0 * g_1$, and we will denote the inverse of an element $g \in G$ by $g^{-1}$.

**Example 6.** *Consider the set $\Pi$ of all* permutations *of a finite set of elements $P = \{p_0, \ldots, p_{n-1}\}$. Any permutation can be written as a set of cycles where cycles of length one are typically not denoted explicitly, e.g., $(01)(23)$ describes a permutation where $p_0 \leftrightarrow p_1$ and $p_2 \leftrightarrow p_3$, while all other elements remain unchanged.*
*The composition of two permutations $\pi_0$ and $\pi_1$ is defined as*

$$(\pi_1 \circ \pi_0) \colon P \to P$$
$$p \mapsto \pi_1(\pi_0(p)).$$

*It can be shown that $(\Pi, \circ)$ forms a group—the so-called* permutation group. *Furthermore, it can easily be shown that the permutation group over a set of size $n$ consists of $n!$ elements.*

An important concept in group theory is that of *generating sets* of a group. A subset of elements $S \subseteq G$ of a group $(G, *)$ is called a generating set of the group if all elements from $G$ can be generated by (repeatedly) applying operation $*$ with the elements from $S$ to themselves or each other.

**Example 7.** *Consider the group of permutations over the four element set $P = \{p_0, p_1, p_2, p_3\}$, which contains $4! = 24$ elements. Then, the set $S = \{(01), (12), (23)\}$ consisting of three nearest-neighbour swaps already generates the whole group, e.g., $(012) = (01) \circ (12)$.*

The structure of a group $(G, *)$ with respect to a generating set $S \subseteq G$ can be represented as a directed graph—the *Cayley graph* [29]. To this end, the graphs' vertices are given by the group elements $g \in G$ and, for every $s \in S$, there is an edge $g \xrightarrow{s} g'$ with $g, g' \in G$, if $s * g = g'$.

**Example 8.** *Consider again the permutation group over the set $P = \{p_0, p_1, p_2, p_3\}$ and the set of generators $S = \{(01), (12), (23)\}$. Then, Fig. 3 shows the corresponding Cayley graph.*

### B. Theoretical Consideration

In order to understand why the total number of permutations can be reduced drastically while still preserving optimality, we look at the mapping problem from a group theoretic viewpoint. Given a coupling graph $(V, E)$, the permutation group over $|V|$

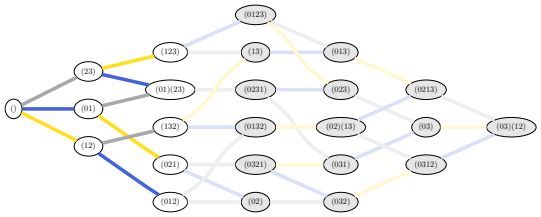



Fig. 4. Reduced Cayley graph for the linear 4-qubit architecture from Fig. 1b

elements can be generated by the set of nearest-neighbour *SWAP* operations executable on the coupling graph, i.e.,

$$S = \{(ij) \colon \forall e_{ij} \in E\}$$

generates all possible permutations of $|V|$ elements for a particular architecture.

**Example 9.** *The Cayley graph shown in Fig. 3 illustrates that* $S = \{(01), (12), (23)\}$ *generates the set of all permutations of four elements. Following any edge with the color corresponding to the generator shows the effect of applying it to the state denoted in the corresponding node. This specific set of generators corresponds to a linear* 4*-qubit architecture given by* $(V, E) = (\{0, 1, 2, 3\}, \{e_{01}, e_{12}, e_{23}\})$ *as shown in Fig. 1b.*

Now, let $K$ be the maximum length of all pairwise shortest paths between two nodes, i.e., the longest, direct connection between any two nodes in the coupling graph. Then, the reduced set of permutations (denoted $\Pi'$ in the following) is composed of all permutations that can be generated by applying at most $K-1$ generators from $S$. This is understood as constructing the Cayley graph associated to $S$, starting at the identity, and stopping after $K-1$ steps.

**Example 10.** *Consider again the linear* 4*-qubit architecture shown in Fig. 1b with* $S = \{(01), (12), (23)\}$*. Then, Fig. 4 shows the reduced Cayley graph for this architecture. As already seen in Example 3, only* 9 *permutations need to be considered here (instead of* 24*).*

In order to show that only considering the reduced permutation set $\Pi'$ preserves optimality, assume otherwise, i.e., assume that at any point in the mapping there exists a permutation $\pi \in \Pi \setminus \Pi'$ that allows for a cheaper overall mapping. More specifically, assume that in front of a $CNOT(q_c, q_t)$ (where $q_c$ is currently mapped to $p_i$ and $q_t$ to $p_j$) there exists $\pi \in \Pi \setminus \Pi'$ such that

1) the gate remains executable (i.e., satisfies the coupling constraints) and
2) the overall amount of *SWAPs* needed to map the circuit is reduced.

Since, $\pi \notin \Pi'$, realizing $\pi$ must at least require $K$ *SWAPs*. Assume that, without loss of generality, $\pi = (kl) \circ \pi'$ for some $\pi' \in \Pi'$. Since $K$ denotes the longest, direct path between any two qubits, realizing the operation on any other edge of the coupling graph could have already been done using at most $K-1$ *SWAPs*. Due to 1), any *SWAP* involving $i$ or $j$ that makes the gate non-executable is ruled out. Thus, $k \neq i, j \wedge l \neq i, j$. However, in that case

$$SWAP(p_k, p_l)CNOT(p_i, p_j) = CNOT(p_i, p_j)SWAP(p_k, p_l)$$

holds, since gates acting on distinct sets of qubits commute. Consequently, applying the *SWAP* $(kl)$ in front of the current gate cannot reduce the overall cost of the resulting circuit, since $(kl)$ will be considered once a gate involving either qubit $k$ or $l$ is encountered later on—contradicting 2). Fig. 5 illustrates this central circuit identity.

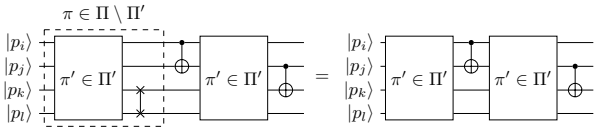



Fig. 5. Main circuit identity that allows to show optimality is preserved

This is a strong argument that shows that optimality is preserved if the first observation presented in Section III is applied. A similar argument can be made to argue that the method of ignoring permutations which do not alter the qubits involved in an operation preserves optimality. While it is not per-se clear whether distributing the problem by considering all possible connected subgraphs preserves optimality, our experimental evaluations (which are summarized in Section VI) suggest this to be the case.

## V. Resulting Strategies

Based on the considerations from above, we are now proposing two strategies for reducing the number of permutations during quantum circuit mapping. As experiments (summarized in Section VI) confirm, they allow to substantially reduce the complexity and, hence, the run-time of corresponding optimal mapping methods.

### A. Architecture Limit

In order to capitalize on the first observation from Section III, the length $K$ of the longest, direct path through the complete architecture has to be determined. It is sufficient to compute this quantity once for a given architecture, e.g., by using the Floyd-Warshall algorithm [28], and storing it for future reuse.

**Example 11.** *Computing all pairwise shortest paths for the linear* 4*-qubit architecture shown in Fig. 1b results in the following tableau*

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix},$$

*which allows to determine* $K = 3$*.*

Once $K$ is calculated, the reduced set of permutations $\Pi'$ to be considered in front of every gate needs to be determined. Since this reduction only depends on the targeted architecture and is independent of the actual gates to be executed, it can also be computed once, e.g., by constructing a representation of the Cayley graph for the given architecture and stopping after $K-1$ applications. By defining an ordering of all permutations (e.g., lexicographic ordering), a bitset of size $m!$ may be used to keep track of which permutations are enabled and which are not.

**Example 12.** *Computing the reduced Cayley graph for the linear* 4*-qubit architecture results in*

$$\Pi' = \{(), (01), (12), (23), (123), (01)(23), (132), (021), (012)\},$$

*as previously shown in Fig. 4. Assuming lexicographic ordering,* $\Pi'$ *corresponds to* 0000 0000 0001 0001 1101 1111*.*

Using $\Pi'$ instead of $\Pi$ allows for optimal mapping of circuits considering a substantially smaller search space. The strategy works best for quantum circuits having as many, or close to as many, qubits as the architecture they get mapped to. This is because this strategy capitalizes on the restrictions imposed due to the inherent structure of the given architecture and limits the number of permutations based on this.

### B. Subgraph Limit

The second strategy combines dividing the problem into smaller problems on all connected subgraphs with the idea of limiting the number of *SWAP* operations maximally considered in front of each gate[4]. In this fashion, the number of permutations to be considered is further reduced. However, the length $K$ of the longest path now has to be calculated for every possible subgraph of the targeted architecture, but not the architecture itself. Due to symmetry/regularity of many quantum architectures, many of these computations are redundant and can be skipped. Afterwards, the subset $\Pi' \subseteq \Pi$ can be determined for every possible connected subgraph the same way as in Section V-A.

**Example 13.** *Assume the circuit shown in Fig. 1a is to be mapped to a linear* 5*-qubit architecture. As shown in Example 4, there are two* 4*-qubit subsets of this architecture—both of which have precisely the structure shown in Fig. 1b. Consequently, only a single longest, direct path calculation has to be carried out, which results in the same tableau as in Example 11—and, hence,* $K = 3$*. The corresponding subsets* $\Pi'$ *are characterized by* 0000 0000 0001 0001 1101 1111 *in both cases.*

In general, the strategy's benefits are biggest whenever $n << m$, i.e., whenever a circuit is mapped to a significantly larger architecture. The architecture itself also plays a vital role in the efficiency of this technique. On the one hand, the less connected the architecture is, the fewer connected subsets there are to consider. On the other hand, the less connected the architecture, the larger $K$ (and the larger $K$, the more permutations have to be considered). Consequently, there certainly exists a "sweet spot" of architecture connectivity for employing this strategy. As our experimental evaluations (which are summarized next) show, using this strategy leads to improvements in almost all considered cases, both compared to not using subgraph-division at all, as well as just using subgraph-division.

Since we have no proof that the strategy of considering all connected subgraphs preserves optimality, there is no guarantee that results obtained from the strategy proposed in this section remain optimal. However, our experimental evaluations indicate that optimality is preserved at least for all the benchmarks we ran our experiments on. This shows great promise for applying optimal techniques for the mapping problem to larger circuits and/or larger architectures.

## VI. Experimental Results

The observations and resulting strategies proposed above can, in general, be employed on top of any optimal method for the mapping problem (such as [19]–[25]). In order to experimentally evaluate their effect, we implemented the strategies proposed in Section V on top of the quantum circuit mapping tool QMAP, which is based on the method proposed in [20] and publicly available at https://github.com/iic-jku/qmap as part of the open-source JKQ toolkit for quantum computing [26]. This led to a version which computes the necessary permutations $\Pi'$ prior to the execution of the mapping, as well as a version that considers all possible permutations $\Pi$[5].

We further distinguished the evaluations between those that determine these permutations based on the complete coupling graph (and the corresponding strategy proposed in Section V-A) and those that determine these permutations only based on the necessary *subgraphs* (and the corresponding strategy proposed in Section V-B). All evaluations have been conducted on an AMD Ryzen 9 3900X processor with 4.1 GHz and 128 GiB of main memory running Ubuntu 20.04 using a hard timeout of 1 h. All results have been verified using the method provided in [30].

Table I provides the obtained results. Here, the first four columns identify the benchmark[6], the number of qubits $n$, the number of gates $|G|$ and the optimal mapping cost $c$ (i.e., the additional number of gates needed to satisfy all coupling constraints). Note that the optimal mapping cost $c$ has been achieved by *all* approaches, independently of whether all permutations $\Pi$ or just the limited set of permutations $\Pi'$ have been considered. Afterwards, we list for all approaches the total number of considered permutations ($|\Pi|$ in case of the original approach and $|\Pi'|$ in case of the proposed approach) as well as the respectively required runtimes in CPU seconds ($t_{ref}$ in case of the reference approach considering *all* permutations $\Pi$ and $t_{prop}$ in case of the proposed approach considering the limited number of permutations $\Pi'$).

First and foremost, the results confirmed that *all* approaches yield circuits with the minimal mapping cost. This is perfectly in line with the theoretical discussion in Section IV-B and confirms that, limiting the search space as proposed in this work, still guarantees optimal results.

At the same time, limiting the search space drastically reduces the complexity of the problem. The respective columns in Table I denoted $|\Pi|$ and $|\Pi'|$ show the difference. For example, rather than 26 040 permutations, only 3 472 permutations need to be considered in case of benchmark *4_49_16* when mapping to the IBMQ London architecture. In this particular case (and some others, as reported in Table I), this makes the differences between running into a timeout of 1 h or being able to determine an optimal result in just some minutes. But also in the cases where the reference approach succeeds in obtaining a result within 1 h, limiting the search space proves beneficial. In fact, for *all* benchmarks substantial speed-ups can be observed. In the best case, speed-ups of up to three orders of magnitude are possible. And, again, these improvements are possible while, at the same time, still obtaining optimal results.

## VII. Conclusions

In this work, we proposed generic and architecture-independent strategies to limit the search space that needs to be considered when aiming for the determination of optimal results for the quantum circuit mapping problem. These strategies are motivated by observations showing that only a limited set of permutations in front of each gate needs to be considered—addressing the origin of the huge search space and complexity. Theoretical considerations (based on group theory) back these observations and experimental evaluations confirm the resulting benefits: Limiting the search space as proposed in this work allows to drastically improve the performance of corresponding approaches (allowing to complete instances within minutes that ran into a timeout before or to achieve speed-ups of up to three orders of magnitude) while, at the same time, remaining optimal.

[4]Whenever $n = m$, this naturally becomes the strategy from Section V-A.

[5]For a comparison between optimal and heuristic approaches we refer to previous work, such as [20], which shows that heuristics frequently stray far from the achievable optimum.

[6]As benchmarks we used instances that have been frequently used by related work in the past. All circuits have been mapped to the 5-qubit, T-shaped IBMQ London architecture. In addition to that, the implementation of the approach is publicly available so that the interested reader can conduct further evaluations.

TABLE I
EXPERIMENTAL EVALUATIONS

| Benchmark | | | | Without Subgraphs | | | | | With Subgraphs | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | JKQ QMAP [20] | | Architecture Limit (Section V-A) | | | JKQ QMAP [20] | | Subgraph Limit (Section V-B) | | |
| Name | $n$ | $\|G\|$ | $c$ | $\|\Pi\|$ | $t_{ref}$ [s] | $\|\Pi'\|$ | $t_{prop}$ [s] | $t_{ref}/t_{prop}$ | $\|\Pi\|$ | $t_{ref}$ [s] | $\|\Pi'\|$ | $t_{prop}$ [s] | $t_{ref}/t_{prop}$ |
| 4_49_16 | 5 | 217 | 207 | 26 040 | >1 h | 3 472 | **171.15** | - | 26 040 | >1 h | 3 472 | **171.07** | - |
| hwb4_49 | 5 | 233 | 213 | 27 960 | >1 h | 3 728 | **198.45** | - | 27 960 | >1 h | 3 728 | **199.57** | - |
| mod10_171 | 5 | 244 | 285 | 29 280 | >1 h | 3 904 | **291.04** | - | 29 280 | >1 h | 3 904 | **283.86** | - |
| mini-alu_167 | 5 | 288 | 330 | 34 560 | >1 h | 4 608 | **477.25** | - | 34 560 | >1 h | 4 608 | **475.06** | - |
| one-two-three-v0_97 | 5 | 290 | 234 | 34 800 | >1 h | 4 640 | **364.29** | - | 34 800 | >1 h | 4 640 | **363.97** | - |
| alu-v2_31 | 5 | 451 | 375 | 54 120 | >1 h | 7 216 | **1 418.29** | - | 54 120 | >1 h | 7 216 | **1 413.01** | - |
| decod24-v3_45 | 5 | 150 | 156 | 18 000 | 1 750.86 | 2 400 | **57.53** | 30.43 | 18 000 | 1 738.25 | 2 400 | **58.65** | 29.64 |
| aj-e11_165 | 5 | 151 | 129 | 18 120 | 1 691.31 | 2 416 | **47.90** | 35.31 | 18 120 | 1 689.52 | 2 416 | **47.94** | 35.25 |
| 4mod7-v1_96 | 5 | 164 | 123 | 19 680 | 1 679.76 | 2 624 | **51.87** | 32.39 | 19 680 | 1 655.91 | 2 624 | **51.49** | 32.16 |
| alu-v2_32 | 5 | 163 | 117 | 19 560 | 1 481.97 | 2 608 | **48.19** | 30.75 | 19 560 | 1 477.13 | 2 608 | **47.59** | 31.04 |
| 4gt10-v1_81 | 5 | 148 | 111 | 17 760 | 1 319.27 | 2 368 | **40.84** | 32.30 | 17 760 | 1 347.52 | 2 368 | **41.16** | 32.74 |
| one-two-three-v0_98 | 5 | 146 | 108 | 17 520 | 1 227.19 | 2 336 | **44.85** | 27.36 | 17 520 | 1 254.30 | 2 336 | **46.91** | 26.74 |
| one-two-three-v1_99 | 5 | 132 | 108 | 15 840 | 1 071.84 | 2 112 | **35.53** | 30.16 | 15 840 | 1 056.80 | 2 112 | **35.24** | 29.99 |
| 4gt5_77 | 5 | 131 | 99 | 15 720 | 914.56 | 2 096 | **31.16** | 29.35 | 15 720 | 927.59 | 2 096 | **31.53** | 29.42 |
| 4gt13_91 | 5 | 103 | 93 | 12 360 | 632.21 | 1 648 | **20.17** | 31.34 | 12 360 | 651.83 | 1 648 | **20.27** | 32.15 |
| miller_11 | 3 | 50 | 27 | 6 000 | 624.63 | 800 | **0.33** | 1 880.11 | 300 | 0.11 | 150 | **0.06** | 1.91 |
| alu-v4_36 | 5 | 115 | 87 | 13 800 | 594.61 | 1 840 | **20.36** | 29.20 | 13 800 | 586.32 | 1 840 | **20.70** | 28.33 |
| 4gt5_76 | 5 | 91 | 84 | 10 920 | 444.35 | 1 456 | **0.35** | 1 254.61 | 10 920 | 458.90 | 1 456 | **0.35** | 1 298.82 |
| decod24-v1_41 | 5 | 85 | 84 | 10 200 | 344.17 | 1 360 | **11.10** | 31.00 | 10 200 | 341.90 | 1 360 | **11.46** | 29.84 |
| decod24-v2_43 | 4 | 52 | 27 | 6 240 | 254.36 | 832 | **0.24** | 1 047.78 | 1 248 | 26.65 | 468 | **0.32** | 83.75 |
| 4mod5-v1_23 | 5 | 69 | 66 | 8 280 | 198.41 | 1 104 | **6.55** | 30.31 | 8 280 | 210.68 | 1 104 | **6.56** | 32.11 |
| 4gt13_92 | 5 | 66 | 78 | 7 920 | 193.88 | 1 056 | **5.79** | 33.49 | 7 920 | 196.83 | 1 056 | **5.75** | 34.23 |
| rd32_270 | 5 | 84 | 54 | 10 080 | 155.96 | 1 344 | **5.59** | 27.89 | 10 080 | 155.42 | 1 344 | **5.64** | 27.58 |
| 4mod5-v0_18 | 5 | 69 | 48 | 8 280 | 135.65 | 1 104 | **3.52** | 38.57 | 8 280 | 144.92 | 1 104 | **3.44** | 42.15 |
| one-two-three-v2_100 | 5 | 69 | 48 | 8 280 | 133.85 | 1 104 | **0.23** | 577.65 | 8 280 | 131.56 | 1 104 | **0.23** | 580.92 |
| mod5d2_64 | 5 | 53 | 42 | 6 360 | 84.36 | 848 | **0.22** | 385.96 | 6 360 | 83.82 | 848 | **0.22** | 381.36 |
| alu-v1_28 | 5 | 37 | 30 | 4 440 | 65.02 | 592 | **0.13** | 506.02 | 4 440 | 65.82 | 592 | **0.13** | 508.11 |
| 3_17_13 | 3 | 36 | 18 | 4 320 | 35.31 | 576 | **0.26** | 137.53 | 216 | 0.08 | 108 | **0.04** | 1.92 |
| rd32-v1_68 | 4 | 36 | 18 | 4 320 | 12.85 | 576 | **0.13** | 102.67 | 864 | 0.51 | 324 | **0.17** | 2.96 |
| rd32-v0_66 | 4 | 34 | 18 | 4 080 | 12.83 | 544 | **0.12** | 103.10 | 816 | 0.51 | 306 | **0.17** | 2.97 |
| 4gt13_90 | 5 | 107 | 114 | 12 840 | 10.73 | 1 712 | **0.30** | 35.36 | 12 840 | 10.90 | 1 712 | **0.30** | 36.02 |
| mod5mils_65 | 5 | 35 | 24 | 4 200 | 2.53 | 560 | **0.12** | 21.51 | 4 200 | 2.49 | 560 | **0.12** | 21.13 |
| alu-v4_37 | 5 | 37 | 30 | 4 440 | 2.26 | 592 | **0.15** | 14.71 | 4 440 | 2.25 | 592 | **0.16** | 14.47 |
| one-two-three-v3_101 | 5 | 70 | 66 | 8 400 | 1.69 | 1 120 | **0.26** | 6.45 | 8 400 | 1.65 | 1 120 | **0.27** | 6.20 |
| alu-v3_34 | 5 | 52 | 51 | 6 240 | 1.57 | 832 | **0.19** | 8.44 | 6 240 | 1.60 | 832 | **0.19** | 8.60 |
| decod24-v0_38 | 4 | 51 | 27 | 6 120 | 1.33 | 816 | **0.31** | 4.26 | 1 224 | 35.77 | 459 | **0.82** | 43.49 |
| qe_qft_5 | 5 | 107 | 9 | 12 840 | 0.99 | 1 712 | **0.22** | 4.54 | 12 840 | 0.97 | 1 712 | **0.22** | 4.41 |
| 4mod5-v0_19 | 5 | 35 | 30 | 4 200 | 0.96 | 560 | **0.83** | 1.16 | 4 200 | 1.43 | 560 | **0.82** | 1.74 |
| 4gt11_82 | 5 | 27 | 45 | 3 240 | 0.93 | 432 | **0.24** | 3.92 | 3 240 | 1.37 | 432 | **0.24** | 5.79 |
| alu-v3_35 | 5 | 37 | 30 | 4 440 | 0.87 | 592 | **0.15** | 5.63 | 4 440 | 1.07 | 592 | **0.15** | 6.93 |

$n$: Number of qubits $|G|$: Gate count of $G$ $c$: Resulting cost (i.e., added gates to satisfy all coupling constraints)
$|\Pi|$: Original number of considered permutations $|\Pi'|$: Maximum number of reduced permutations
$t_{ref}$: Runtime of JKQ QMAP [20] (with and without subgraphs) $t_{prop}$: Runtime of proposed scheme (with and without subgraphs)
The optimal cost $c$ has been achieved by *all* approaches, independently of whether all permutations $\Pi$ or just the limited set $\Pi'$ have been considered.


## ACKNOWLEDGMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 101001318). It has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria as well as by the BMK, BMDW, and the State of Upper Austria in the frame of the COMET program (managed by the FFG).


## REFERENCES


[1] J. Gambetta, "IBM's Roadmap For Scaling Quantum Technology," *IBM Research Blog*, 2020. [Online]. Available: https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," *Proc. of the ACM*, pp. 212–219, 1996.

[3] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[4] M. Cerezo *et al.* "Variational Quantum Algorithms." arXiv: 2012.09265. (2020).

[5] A. Peruzzo *et al.*, "A variational eigenvalue solver on a photonic quantum processor," *Nat Commun*, vol. 5, no. 1, p. 4213, 2014.

[6] A. Kandala *et al.*, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[7] A. Barenco *et al.*, "Elementary gates for quantum computation," *Phys. Rev. A*, vol. 52, no. 5, pp. 3457–3467, 1995.

[8] D. Maslov, "On the advantages of using relative phase Toffolis with an application to multiple control Toffoli optimization," *Phys. Rev. A*, vol. 93, no. 2, p. 022 311, 2016.

[9] R. Wille *et al.*, "Improving the mapping of reversible circuits to quantum circuits using multiple target lines," in *Asia and South Pacific Design Automation Conf.*, 2013.

[10] A. M.-v. de Griend and R. Duncan. "Architecture-aware synthesis of phase polynomials for NISQ devices." arXiv: 2004.06052 [quant-ph]. (2020).

[11] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the IBM QX architectures," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 38, no. 7, pp. 1226–1236, 2019.

[12] K. N. Smith and M. A. Thornton, "A quantum computational compiler and design tool for technology-specific targets," in *Int'l Symp. on Computer Architecture*, 2019, pp. 579–588.

[13] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019.

[14] A. Matsuo, W. Hattori, and S. Yamashita, "Reducing the overhead of mapping quantum circuits to IBM Q system," in *IEEE International Symposium on Circuits and Systems*, 2019.

[15] P. Murali *et al.*, "Noise-adaptive compiler mappings for Noisy Intermediate-Scale Quantum computers," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1015–1029.

[16] M. Amy and V. Gheorghiu. "Staq – A full-stack quantum processing toolkit." arXiv: 1912.06070. (2019).

[17] S. Sivarajah *et al.*, "Tlket⟩: A Retargetable Compiler for NISQ Devices," *Quantum Sci. Technol.*, vol. 6, no. 1, p. 014 003, 2020.

[18] A. Zulehner and R. Wille, "Compiling SU(4) quantum circuits to IBM QX architectures," in *Asia and South Pacific Design Automation Conf.*, Tokyo, Japan, 2019, pp. 185–190.

[19] Y. Hirata *et al.*, "An efficient conversion of quantum circuits to a linear nearest neighbor architecture," *Quantum Inf. Comput.*, vol. 11, pp. 142–166, 2011.

[20] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Design Automation Conf.*, 2019.

[21] M. Y. Siraichi *et al.*, "Qubit allocation," in *Int'l Symp. on Code Generation and Optimization*, 2018.

[22] C. Zhang *et al.*, "Time-optimal qubit mapping," in *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, 2021.

[23] B. Tan and J. Cong, "Optimal layout synthesis for quantum computing," in *Int'l Conf. on CAD*, 2020.

[24] A. A. A. de Almeida, G. W. Dueck, and A. C. R. da Silva, "Finding optimal qubit permutations for IBM's quantum computer architectures," in *Symp. on Integrated Circuits and Systems Design*, 2019.

[25] P. Zhu, X. Cheng, and Z. Guan, "An exact qubit allocation approach for NISQ architectures," *Quantum Inf Process*, vol. 19, no. 11, p. 391, 2020.

[26] R. Wille, S. Hillmich, and L. Burgholzer, "JKQ: JKU tools for quantum computing," in *Int'l Conf. on CAD*, 2020.

[27] A. Botea, A. Kishimoto, and R. Marinescu, "On the complexity of quantum circuit compilation," in *Int'l Symp. on Combinatorial Search*, 2018.

[28] T. H. Cormen *et al.*, *Introduction to Algorithms, Third Edition*, 3rd. The MIT Press, 2009.

[29] N. C. Carter, *Visual group theory*. Mathematical Association of America, 2009.

[30] L. Burgholzer and R. Wille, "Advanced equivalence checking for quantum circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, 2021.