Quantum Information & Computation
ISSN 1533-7146 Volume 25 (2025) pp.509–541
©Cerebration Science Publishing

Article

Implementation of Quantum Fourier Transform and Quantum Hashing for Quantum Devices with Arbitrary Qubit Connectivity Graphs

Kamil Khadiev [1,2,*], Aliya Khadieva [1,2], Zeyu Chen [3] and Junde Wu [3]

[1] Institute of Computational Mathematics and Information Technologies, Kazan Federal University, Kremlyovskaya, 35, Kazan 420008 , Russia
[2] Zavoisky Physical-Technical Institute, FRC Kazan Scientific Center of RAS, Sibirsky tract, 10/7, Kazan 420029, Russia
[3] School of Mathematics Sciences, Zhejiang University, Hangzhou 310058, China
* Corresponding author: kamilhadi@gmail.com

Received date: 28 May 2025; Accepted date: 2 September 2025; Published online: 15 October 2025

Abstract: In the paper, we consider quantum circuits for quantum fingerprinting (quantum hashing) and quantum Fourier transform (QFT) algorithms. Quantum fingerprinting (quantum hashing) is a well-known technique for comparing large objects using small images. The QFT algorithm is a very popular technique used in many algorithms. We present a generic method for constructing quantum circuits for these algorithms for quantum devices with restrictions. Many quantum devices (e.g., based on superconductors) have restrictions on applying two-qubit gates. The restrictions are presented by a qubit connectivity graph. Typically, researchers consider only the linear nearest neighbor (LNN) architecture, but current devices have more complex graphs. We present a method for arbitrary connected graphs that minimizes the number of CNOT gates in the circuit. The heuristic version of the method is fast enough and works with $O(n^6)$ time complexity, where n is the number of qubits. The certain version of the algorithm has an exponential time complexity that is $O(n^2 2^n)$. We compare quantum circuits built by our algorithm with quantum circuits optimized for specific graphs that are Linear-nearest-neighbor (LNN) architecture, "sun" (a cycle with tails, presented by the 16-qubit IBMQ device) and "two joint suns" (two joint cycles with tails, presented by the 27-qubit IBMQ device). Our generic method gives similar results with a few more CNOT gates. At the same time, our method allows us to construct a circuit for arbitrary connected graphs.

Keywords: quantum circuits; quantum Fourier transform; QFT; quantum hashing; quantum fingerprinting; qubit connectivity graph

1. Introduction

The quantum Fourier transform (QFT) [1] is a well-known computational technique used in many quantum algorithms. The technique is used in quantum addition [2], quantum phase estimation (QPE) [1], quantum amplitude estimation (QAE) [3], the algorithm for solving linear systems of equations [4], Shor's factoring algorithm [5], 1D wave equation [6], Quantum walks [7–9] with applications to quantum backtracking [10,11] and others.

In this paper, we are interested in the implementation of this technique on quantum devices as a quantum circuit with basic gates. There are different ways to optimize the circuits, such as invoking it in a parallel way [12] or approximate QFT [13]. We are focusing on the minimization of two-qubit quantum gates in such a circuit because they are the most "expensive" to implement. Many types of quantum computers (e.g., quantum devices based on superconductors) do not allow us to apply two-qubit gates to an arbitrary pair of qubits but have a graph that represents such a restriction. Vertices of the graph correspond to qubits, and two-qubit gates can be applied only to qubits corresponding to vertices connected by an edge. We say that such a graph represents the qubit topology for a device. In this paper, we focus on the number of CNOT gates in a quantum circuit for the QFT algorithm with respect to such a graph. Let a CNOT cost of a circuit be the number of CNOT gates in the circuit. The CNOT cost for a linear nearest-neighbor (LNN) architecture (where the graph is just a chain) was explored by Park and Ahn in [14]. They presented

DOI: 10.2478/qic-2025-0028

a circuit that has $n^2 + n - 4$ CNOT cost, where n is the number of qubits. It improved the previous results of [15–22]. At the same time, as the authors mentioned, their technique cannot be generalized to more complex graphs. Khadieva in [23] suggested a quantum circuit for a more complex architecture that is a cycle with tails (like a "sun" or "two joint suns"). At the same time, the CNOT cost of this circuit is $1.5n^2$.

Here, we present a general method that allows us to develop a quantum circuit of the QFT algorithm for an arbitrary connected graph that represents a qubit connectivity graph. Our solution is based on a solution for the Travelling salesman problem (TSP) for a modification of the original graph. Here we present a method and an algorithm for constructing a circuit with $O(n^2 2^n)$ time complexity, where n is the number of vertices of the original graph and the approximate solution with polynomial time complexity $O(n^6)$ and the differential approximation ratio $\frac{1}{2}$.

The constructed circuit has the CNOT cost from $n^2 - n - 4$ to $3n^2 - 3n - 10$ depending on the complexity of the graph. If the graph has a Hamiltonian path, then it has the minimum possible CNOT cost at most $1.5n^2 - 1.5n - 1$. In addition, we compare our results with circuits for specific graphs. In the case of LNN, the CNOT cost is $1.5n^2 - 1.5n - 1$, that is, 1.5 times larger than the result of [14]. The circuits built by our approach are larger by n CNOT gates than the circuit of [23] with the CNOT cost $1.5n^2 - 2.5n - 1$. For more complex graphs such as 16-qubit and 27-qubit Eagle r3 architectures of IBMQ that is a cycle with tails (like a "sun") or its modifications, our generic technique gives a CNOT cost a bit larger than the circuit [23] that was especially constructed for these architectures. Our circuits have CNOT cost 342 for the 16-qubit architecture and 1009 for the 27-qubit one; and [23] has CNOT cost 324 and 957, respectively. The difference is about 5%.

Another technique discussed in this paper is quantum fingerprinting, or quantum hashing. This technique is well known, and it allows us to compute a short hash or fingerprint that identifies an original large data with high probability. The nature of the technique and its implementation is similar to the QFT algorithm [23,24].

The probabilistic (randomized) technique was developed by Frievalds [25]. Then, Ambainis and Frievalds [26] developed its quantum counterpart for automata that was improved by Ambainis and Nahimovs in [27,28]. Later, Buhrman et al. in [29] provided an explicit definition of the quantum fingerprinting technique to construct an efficient quantum communication protocol for equality testing. The technique was applied for branching programs by Ablayev, Gainutdinova, Vasiliev, and co-authors [30–34]. Later, they developed the concept of cryptographic quantum hashing [35–46]. Then, different versions of hashing functions were applied by Ablayev, Vasiliev, Ziiatdinov, Zinnatullin, and other researchers [24,47–50]. The technique was also extended for qudits [51]. This approach was widely used in different areas like stream processing algorithms [52,53], query model algorithms [54,55], online algorithms [56–62], branching programs [63–65], developing quantum devices [66], automata (discussed earlier, [67–71]), and others.

Turaykhanov, Akat'ev, and co-authors implemented the technique in a real quantum device based on photons [72,73]. The alternative implementation for photon-based real quantum devices was developed by Plachta et al. [74] and Zhao et al. [75]. At the same time, the algorithm was embedded in this device. When we discuss the implementation of the algorithm for "universal" quantum devices such as IBMQ quantum computers or similar, it is important to minimize the number of quantum gates with respect to the architecture restrictions of a quantum device, as we discussed for QFT above. The basic implementation part of quantum hashing circuits is a uniformly controlled rotation operator. The CNOT cost of this operator with respect to the qubit connectivity graph was previously discussed in [76,77] for the linear nearest-neighbor (LNN) architecture and in [78] for more complex graphs. At the same time, the presented approaches have issues with the rotation angle precision, which cannot be too small. This issue is discussed in [79,80].

Kālis in his master's thesis [81] suggested a shallow circuit for quantum fingerprinting for 3 qubits. Later, the approach was developed in detail and presented in a general way for quantum hashing by Ziatdinov, Khadieva, and Yakaryılmaz [82,83]. The approach allows us to reduce the number of CNOT gates exponentially in computational experiments. At the same time, the theoretical exponential superiority of the shallow circuit is not shown [82,83]. By the way, the method is very perspective for current and near-future quantum devices that definitely cannot support a huge number of CNOT gates. The efficient implementation of the shallow circuit for an automaton that recognizes the $MOD_p = \{a^k : k \bmod p = 0\}$ language based on the quantum hashing algorithm was proposed by Khadieva, Salehi, and Yakaryılmaz [79] for devices based on the LNN architecture. Later, a similar technique was applied by Khadieva [23] for a more complex architecture that is a cycle with tails (like a "sun" and "two joint suns") represented by 16-qubit and 27-qubit Eagle r3 IBMQ architectures. Vasiliev [84] discussed a similar method, but for Rz gates instead of Ry gates.

In this paper, we present a general method that allows us to develop a quantum circuit of the quantum hashing algorithm for an arbitrary qubit connectivity graph. It uses the same tools and approaches as the algorithm for con-

structing the circuit of QFT. Note that the shallow circuit for quantum fingerprinting and the circuit for QFT have similar structures as was discussed by Khadieva [23]. That is why we consider these two topics together and use common methods for both algorithms.

The CNOT cost of the constructed circuit is from $2n-2$ to $6n-11$ depending on the complexity of the graph. The presented cost value is for one implementation step of the quantum hashing algorithm. If we present the complexity of ℓ steps of the algorithm, then the constructed circuit has the CNOT cost from $(2n-4)\ell+2$ to $(6n-13)\ell+2$, which depends on the complexity of the graph. When we apply our approach to the LNN architecture, we obtain a circuit with the same CNOT cost as the circuit specially built for the LNN architecture [79]. If we consider 16- and 27-qubit Eagle r3 IBMQ architectures, then we have a similar situation [23].

The structure of this paper is as follows: Section 2 describes the required notations and preliminaries. Graph theory tools are presented in Section 3. Section 4 provides an algorithm for generating a quantum circuit for quantum hashing. The circuit for the quantum Fourier transform is discussed in Section 5. The final Section 6 concludes the paper and contains some open questions.

2. Preliminaries

2.1. Graph Theory

Consider an undirected unweighted graph $G=(V,E)$, where V is the set of vertices and E is the set of undirected edges. Let $n=|V|$ be the number of vertices, and $m=|E|$ be the number of edges.

A non-simple path P is a sequence of vertices $(v_{i_1},\dots,v_{i_h})$ that are connected by edges, i.e., $(v_{i_j},v_{i_{j+1}})\in E$ for all $j\in\{1,\dots,h-1\}$. A path is called a non-simple cycle if $v_{i_1}=v_{i_h}$. Note that a non-simple path and a non-simple cycle can contain duplicates. Let the length of the path be the number of edges in the path, $len(P)=h-1$.

A path $P=(v_{i_1},\dots,v_{i_h})$ is called simple if there are no duplicates among $v_{i_1},\dots,v_{i_h}$. A simple cycle $C=(v_{i_1},\dots,v_{i_h})$ is a non-simple path with only one duplicate $v_{i_1}=v_{i_h}$. The distance $dist(v,u)$ is the length of the shortest path between vertices v and u.

Let $\textsc{Neighbors}(v)$ be a list of neighbors for a vertex v, i.e., $\textsc{Neighbors}(v)=(u_{i_1},\dots,u_{i_k})$ such that $(v,u_{i_j})\in E$.

Let us consider an undirected weighted graph $S=(V',E')$, where V' is the set of vertices, and E' is the set of undirected weighted edges. Let $w:V'\times V'\to\mathbb{R}$ be a weight function. We assume that $w(v,u)=\infty$ if $(v,u)\notin E$.

For a path $P=(v_{i_1},\dots,v_{i_h})$, the length $w(P)$ is the sum of edge weights in the path, i.e., $w(P)=\sum_{j}^{h-1} w(v_{i_j},v_{i_{j+1}})$.

2.2. Quantum Circuits

Quantum circuits consist of qubits and gates. A state of a qubit is a column vector from the $\mathcal{H}^2$ Hilbert space. It can be represented by $a_0|0\rangle+a_1|1\rangle$, where a_0,a_1 are complex numbers such that $|a_0|^2+|a_1|^2=1$, and $|0\rangle$ and $|1\rangle$ are unit vectors. Here we use the Dirac notation. A state of n qubits is represented by a column vector from the $\mathcal{H}^{2^n}$ Hilbert space. It can be represented by $\sum_{i=0}^{2^n-1}a_i|i\rangle$, where a_i is a complex number such that $\sum_{i=0}^{2^n-1}|a_i|^2=1$, and $|0\rangle,\dots|2^n-1\rangle$ are unit vectors. Graphically, on a circuit, qubits are presented as lines.

As basic gates, we consider the following ones:

$$H=\frac{1}{\sqrt{2}}\begin{pmatrix}1&1\\1&-1\end{pmatrix},\ X=\begin{pmatrix}0&1\\1&0\end{pmatrix},\ R_y(\xi)=\begin{pmatrix}cos(\xi/2)&-sin(\xi/2)\\sin(\xi/2)&cos(\xi/2)\end{pmatrix},$$

$$R_z(\xi)=\begin{pmatrix}e^{\frac{i\xi}{2}}&0\\0&e^{-\frac{i\xi}{2}}\end{pmatrix},\ CNOT=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{pmatrix}.$$

Additionally, we consider five non-basic gates

$$R_d=\begin{pmatrix}1&0\\0&e^{\frac{i\pi}{2^{d-1}}}\end{pmatrix},\ CR_d=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&e^{\frac{i\pi}{2^{d-1}}}\end{pmatrix},\ CR_z(\xi)=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&e^{\frac{i\xi}{2}}&0\\0&0&0&e^{-\frac{i\xi}{2}}\end{pmatrix},$$

$$SWAP=\begin{pmatrix}1&0&0&0\\0&0&1&0\\0&1&0&0\\0&0&0&1\end{pmatrix},\ CR_y(\xi)=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&cos(\xi/2)&-sin(\xi/2)\\0&0&sin(\xi/2)&cos(\xi/2)\end{pmatrix}.$$

The reader can find more information about quantum circuits in [15,85,86]

3. Tools

Let us consider an undirected unweighted connected graph $G = (V, E)$ such that $n = |V|$ is the number of vertices and $m = |E|$ is the number of edges.

In this section, we consider the problem of searching for the shortest non-simple path in the graph G that visits all vertices at least once. Formally, we want to construct a non-simple path $P = (v_{i_1}, \dots, v_{i_k})$ such that

- the path visits all vertices, i.e., $\{v_{i_1}, \dots, v_{i_k}\} = V$; and
- the length of the path P is as small as possible.

The problem is close to the Hamiltonian path problem [87], but here we are allowed to visit a vertex several times. That is why any connected graph has the required path.

To solve this problem, we construct a new complete weighted graph $S = (V', E')$ such that

- The set of vertices V' is the same as the set of vertices V.
- Each pair of vertices $u', v' \in V'$ is connected, and the weight $w(u', v')$ is the length of the shortest path between the corresponding vertices u and v from V, i.e., $w(u', v') = dist(u, v)$.

We call it the "supergraph." Let a TSP path (or a Travelling salesman problem path) P' be the shortest simple path in S that visits all vertices from V' exactly once [87]. In a similar way, we can define a TSP cycle C'.

For a TSP path $P' = (v_{i_1}, \dots, v_{i_n})$, let $\alpha(P')$ be the path in G such that

$$\alpha(P') = P_{v_{i_1}, v_{i_2}} \circ \tilde{P}_{v_{i_2}, v_{i_3}} \circ \tilde{P}_{v_{i_3}, v_{i_4}} \circ \dots \circ \tilde{P}_{v_{i_{n-1}}, v_{i_n}}.$$

Here $P_{v_{i_1}, v_{i_2}}$ is the shortest path in G between v_{i_1} and v_{i_2}; the path $\tilde{P}_{v_{i_j}, v_{i_{j+1}}}$ is the shortest path in G between v_{i_j} and $v_{i_{j+1}}$ excluding the first vertex in the path that is v_{i_j}; $\circ$ is the concatenation operation.

A TSP path for S and the shortest non-simple path in G that visits all vertices at least once has the following connection.

Lemma 1. *The shortest non-simple path P in G that visits all vertices can be obtained by a TSP path P' for S as $P = \alpha(P')$.*

Proof. Let $P = \alpha(P')$. Assume that there is another non-simple path $\hat{P}$ that is shorter than P and visits all vertices of G at least once, i.e., $len(\hat{P}) < len(P)$. Let $(i_1, \dots, i_n)$ be the permutation of $(1, \dots, n)$ that presents the order of visiting vertices by $\hat{P}$ for the first time. The length of the simple path $\hat{P}' = (v'_{i_1}, \dots, v'_{i_n})$ in S is less than or equal to the length of the non-simple path $\hat{P}$ in G, i.e., $len(\hat{P}) \geq w(\hat{P}')$. This happens because the weight of an edge in S is the length of the shortest path between the corresponding vertices of G. At the same time, we have no guarantee that $\hat{P}$ goes by this shortest path between vertices. Formally, for any index j, the weight of the edge in S between the vertices v'_{i_j} and $v'_{i_{j+1}}$ is $w(v'_{i_j}, v'_{i_{j+1}}) \leq i_{j+1} - i_j$. At the same time, $\hat{P}'$ visits all vertices of S. So, $w(P') \leq w(\hat{P}')$ because P' is a TSP path. Summarizing, $len(P) = w(P') \leq w(\hat{P}') \leq len(\hat{P})$. It contradicts our assumption $len(P) > len(\hat{P})$. □

There is also a similar connection for cycles.

Lemma 2. *The shortest non-simple cycle C in G that visits all vertices can be obtained by a TSP cycle C' for S as $C = \alpha(C')$.*

Proof. The proof is the same as for Lemma 1. □

We can note the following property:

Lemma 3. *The length of the shortest non-simple path P that visits all vertices in G at least once is such that $n - 1 \leq len(P) \leq 2n - 2$. A similar result is for a cycle C, i.e., $n \leq len(C) \leq 2n - 2$*

Proof. Let us consider a spanning tree of the graph $G = (V, E)$. It is a tree $T = (V, E')$, where $E' \subset E$. The path should visit all vertices. So, we can construct a non-simple path P that is the Euler tour [87] of the tree T. The path covers all the vertices of the graph G, but may not be the shortest. Each edge in the tour is visited at most twice (in the up- and down-direction). There are $n - 1$ edges in the tree. Therefore, the length of the path is at most $2n - 2$. For a cycle, we have a similar situation. □

Based on this connection, we can suggest an algorithm for searching the shortest non-simple path in the graph G that visits all vertices at least once.

Assume that we have a procedure that finds a TSP path for S that is $\text{TSPPATH}(S)$. As an algorithm, we can use the Bellman-Held-Karp algorithm [88,89]. The algorithm has $O(n^2 2^n)$ time complexity.

Additionally, we have a procedure $\text{SHORTESTPATHS}(G)$ that constructs two $n \times n$-matrices W and A by a graph G.

Elements of the matrix W are the lengths of the shortest paths between each pair of vertices in G, i.e., $W[v,u] = dist(v,u)$. We can use W as a weight matrix for S. In other words, $w(v',u') = W[v,u]$, where $v',u' \in V'$, and v and u are the corresponding vertices from V.

The matrix A represents the shortest paths between the vertices of G. The element $A[v,u]$ is the last vertex before u in the shortest path from v to u. In other words, if $t = A[v,u]$, then $P_{v,u} = P_{v,t} \circ u$. Based on this fact, we can present a procedure $\text{GETPATH}(v,u)$ that computes $P_{v,u}$ using the matrix A. The implementation of the procedure is presented in Algorithm 1.

Algorithm 1 Implementation of $\text{GETPATH}(v,u)$

```
t ← A[v,u]
P_{v,u} ← (u)
while t ≠ v do
    P_{v,u} ← (t) ∘ P_{v,u}
    t ← A[v,t]
end while
P_{v,u} ← (v) ∘ P_{v,u}
return P_{v,u}
```

Let the procedure $\text{GETPATHNOFIRST}(v,u)$ return $\tilde{P}_{v,u}$, that is, the path $P_{v,u}$ without the first element. The implementation is the same, but without the $P_{v,u} \leftarrow (v) \circ P_{v,u}$ line.

We can construct these two matrices using n invocations of the Breadth First Search (BFS) algorithm [87]. The total time complexity for the construction of the matrices is $O(n^3)$. The algorithm for constructing A and W is presented in Appendix D for completeness of presentation.

Due to $V' = V$, and the matrix W representing the weights of the supergraph's edges, we can say that $\text{SHORTESTPATHS}(G)$ constructs the graph S.

Finally, we can present Algorithm 2 to solve the problem.

Algorithm 2 Implementation Algorithm for searching a shortest non-simple path in the graph G that visits all vertices at least once

```
W, A ← SHORTESTPATHS(G)                          ▷ W and V define S
(v_{i_1}, ..., v_{i_k}) = P' ← TSPPATH(S)
α(P') ← GETPATH(v_{i_1}, v_{i_2})
for j ∈ {2, ..., k − 1} do
    α(P') ← α(P') ∘ GETPATHNOFIRST(v_{i_j}, v_{i_{j+1}})
end for
```

Note that if we replace the procedure $\text{TSPPATH}(S)$ by the procedure $\text{TSPCYCLE}(S)$ that finds a TSP cycle, then we find the shortest non-simple cycle in G that visits all vertices at least once. The time complexity of $\text{TSPCYCLE}(S)$ is the same.We can also modify $\text{TSPPATH}(S)$ such that it finds a path from the fixed starting vertex v_{start} that is $\text{TSPPATH}(S, v_{start})$.

Theorem 1. *The time complexity of the presented solution for searching the shortest non-simple path, a path with a fixed starting vertex or a cycle that visits all vertices at least once in the graph $G = (V,E)$ is $O(n^2 2^n)$, where $n = |V|$.*

Remark 1. *We can solve the same problem using quantum algorithms. For $\text{TSPPATH}(S)$, $\text{TSPPATH}(S, v_{start})$ and $\text{TSPCYCLE}(S)$, we can use the quantum analogue of Bellman-Held-Karp algorithm presented by Ambainis et al. [90]. For $\text{SHORTESTPATHS}(S)$, we can use the quantum version of the BFS algorithm [91–93] several times.*

The query complexity of the algorithm [90] presented by Ambainis et al. is $O^*(1.728^n)$, where O^* hides log-factors. The algorithm is based on the Grover search algorithm, whose time complexity is more than the query complexity for an additional log factor [94,95]. So, the time complexity is also $O^*(1.728^n)$. If we want to discuss it more precisely, then we can check [90,96] which claims the query complexity $O(n^3 1.728^n)$.

Finally, the complexity of the quantum algorithm for the problem is $O^*(1.728^n)$. At the same time, for current applications, we need only a classical solution.

3.1. Heuristic Solution for Searching a Shortest Non-simple Path That Visits All Vertices Problem

In the practical case, when we have more than 20–30 vertices in a graph, the exact algorithm presented before is too slow. In this section, we present a heuristic solution with polynomial-time complexity, and the outcome is close, under certain criteria, to the optimal solution.

Using a path $P = (v_{i_1}, \ldots, v_{i_h})$, we can construct a sequence of edges $((v_{i_1}, v_{i_2}), (v_{i_2}, v_{i_3}), \ldots, (v_{i_{h-1}}, v_{i_h}))$. We can say that these are two equivalent representations of the path. We use both in this section.

Note that the supergraph S defined in Section 3 satisfies the following conditions:

- S is complete and weighted.
- The weight of an edge from S is at most n because it is the distance between two vertices in the original graph. Formally, $w(v', u') = dist(v, u) \leq n$, where $v', u' \in V'$, and $v, u \in V$.
- S is metric. This means that the edges satisfy the triangle inequality: $w(v'_i, v'_j) \leq w(v'_i, v'_k) + w(v'_k, v'_j)$ for any $v'_i, v'_j, v'_k \in V'$.

Now we present a heuristic procedure 2-Opt(S) to find a good solution to the TSP problem. It finds a TSP cycle C. The algorithm for cycles in the case of general weighted metric graphs was presented in [97–99]. Here we present modifications for our graph and for searching a path.

We assume that we have two operations for modification of the list of edges that represents a cycle C:

- Replace$(C, (v_{i_j}, v_{i_{j+1}}), (u, v))$ that replaces the edge $(v_{i_j}, v_{i_{j+1}})$ in C by the edge (u, v)
- Reverse$(C, (v_{i_j}, v_{i_{j+1}}), (v_{i_k}, v_{i_{k+1}}))$ that reverses all elements of the list between $(v_{i_j}, v_{i_{j+1}})$ and $(v_{i_k}, v_{i_{k+1}})$. Both elements $(v_{i_j}, v_{i_{j+1}})$ and $(v_{i_k}, v_{i_{k+1}})$ remain in place.

Both operations can be implemented with $O(1)$ time complexity if C is implemented by the two-directional Linked List data structure [87].

Firstly, we initialize the cycle by $((v_1, v_2), \ldots, (v_{n-1}, v_n), (v_n, v_1))$. Then we repeatedly replace two edges of the cycle as long as this yields a shorter cycle. We check all possible pairs. The implementation is presented in Algorithm 3.

In the case of searching for a path, we have two modifications. That is, if a starting vertex v_s and a finishing vertex v_f are fixed, then we initialize the path by

$$C \leftarrow ((v_s, v_1), \ldots, (v_{s-2}, v_{s-1}), (v_{s-1}, v_{s+1}), \ldots, (v_{f-2}, v_{f-1}), \\ (v_{f-1}, v_{f+1})(v_{f+1}, v_{f+2}), \ldots, (v_{n-1}, v_n)), (v_n, v_f)).$$

Without loss of generality, we can assume that $s < f$. If the starting and finishing vertices are not fixed, then we check all possible vertices from V' as v_s and v_f.

Experiments on real-world instances have shown that 2-Opt(S) achieves much better results than Christofide's algorithm [100]. Generally, the worst case of 2-Opt(S) may lead to exponential complexity. However, direct analysis shows that the complexity of 2-Opt(S) in the case of supergraphs is $O(n^4)$ for cycles and for paths with a fixed starting vertex, where n is the number of vertices.

Algorithm 3 Implementation of 2-Opt(S)

$C \leftarrow ((v_1, v_2), \ldots, (v_{n-1}, v_n), (v_n, v_1))$
$stopFlag \leftarrow False$
while $stopFlag = False$ **do**
  $stopFlag \leftarrow True$
  for $k \in \{1, \ldots, n\}$ **do**
    for $j \in \{k+1, \ldots, n\}$ **do**
      if $w(v_{i_k}, v_{i_{k+1}}) + w(v_{i_j}, v_{i_{j+1}}) > w(v_{i_k}, v_{i_j}) + w(v_{i_{k+1}}, v_{i_{j+1}})$ **then**
        Reverse$(C, (v_{i_k}, v_{i_{k+1}}), (v_{i_j}, v_{i_{j+1}}))$
        Replace$(C, (v_{i_k}, v_{i_{k+1}}), (v_{i_k}, v_{i_j}))$
        Replace$(C, (v_{i_j}, v_{i_{j+1}}), (v_{i_{k+1}}, v_{i_{j+1}}))$
        $stopFlag \leftarrow False$
        break both For-loops
      end if
    end for
  end for
end while
return C

Theorem 2. *The time complexity of 2-Opt(S) for the supergraph S constructed by G is $O(n^4)$ in the case of a cycle; in the case of a path with a fixed starting vertex, it is $O(n^5)$; and in the case of a path, it is $O(n^6)$.*

Proof. Initialization of C takes $O(n)$ time. Since the weights of the edges are bounded above by n, the weight of any Hamiltonian cycle C is bounded above by n^2. In each step of "replacing two edges," the weight of C is reduced by at least 1, which implies that there are $O(n^2)$ steps of "replacing two edges." In each step, there are $O(n^2)$ pairs of edges to compare, and the comparison and replacement of edges takes constant time. In summary, the total complexity is $O(n^4)$.

In the case of a path with a fixed starting vertex, we also check all finishing vertices. The number of finishing vertices is n. So, the complexity is $O(n^5)$ for searching the path.

In the case of a path, we also check all starting vertices. The number of starting vertices is n. So, the complexity is $O(n^6)$ for searching the path. □

The heuristic algorithm 2-Opt(S) is designed to compute feasible solutions that are as close as possible to the optimal ones with respect to some criterion. We present two paradigms dealing with polynomial approximation. More details can be found in [98,101].

Definition 1. *Let I be an input and $\mathcal{A}$ be an approximation algorithm. Let $len_{opt}(I)$ be the length of a path that returns an optimal solution for the given instance I. Let $len_w(I)$ be the worst possible length that a feasible solution returns for I. Let $len_{\mathcal{A}}(I)$ be the length of the approximation algorithm's solution. Then, for an instance I*

- *the standard approximation ratio is*
  $\rho_{\mathcal{A}}(I) = \frac{len_{\mathcal{A}}(I)}{len_{opt}(I)}$;
- *the differential approximation ratio is*
  $\delta_{\mathcal{A}}(I) = \frac{|len_w(I) - len_{\mathcal{A}}(I)|}{|len_w(I) - len_{opt}(I)|}$.

Note that if the ratios are close to 1, then the algorithm has better quality.

For the heuristic algorithms for the traveling salesman problem, we have the following results.

Lemma 4 ([97–99])**.** *The differential approximation ratio for the 2-Opt(S) solution of the Traveling salesman problem achieves $\frac{1}{2}$, and if the graph is metric, the standard approximation ratio achieves $\sqrt{\frac{n}{2}}$. Both of the ratios are tight.*

4. Method for Constructing a Circuit for Quantum Hashing

In this section, we present a method that allows us to construct a circuit for the quantum fingerprinting or quantum hashing algorithm for a connected graph $G = (V, E)$ that is the qubit connectivity graph for a device. More infor-

mation on the quantum fingerprinting (quantum hashing) algorithm can be found in Appendix B. Here we consider a shallow circuit [81,82] for n control qubits. If we do not have restrictions for applying two-qubit gates (when G is a complete graph or a "star" graph), then the circuit is such that we presented in Figure 1. Here we assume that we have qubits $q_1, \ldots, q_n$ such that $q_1, \ldots, q_{n-1}$ are control ones and q_n is the target one.

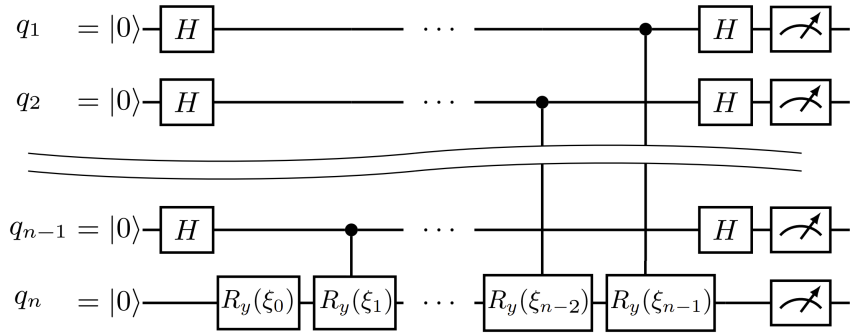

Figure 1. Shallow circuit for quantum fingerprinting or quantum hashing algorithm.

Let q_i be the logical qubits of the original circuit; the qubits of a physical device are associated with vertices of the graph G, and we call them v_i.

The algorithm for constructing a circuit is the following. We also present it in a flowchart in Figure 2.

Step 1. We find the shortest non-simple path in the graph G that visits all vertices at least once using the algorithm from Section 3. Assume that the path is $P = (v_{i_1}, \ldots, v_{i_k})$.

Step 2. The target qubit q_n corresponds to the vertex v_{i_2}. The control qubits $q_1, \ldots, q_{n-1}$ correspond to other vertices. We assume that we have a set U of control qubits that are already used. Initially, it is empty $U \leftarrow \emptyset$. Let j be the index of the element in the path P that corresponds to the target qubit. Initially, $j \leftarrow 2$.

Step 3. We apply controlled rotation with the control v_{i_1} and the target v_{i_2} qubits. Then, we add v_{i_1} to the set U, i.e., $U \leftarrow U \cup \{v_{i_1}\}$.

In the next steps the target qubit travels along the path P.

Step 4. If $v_{i_{j+1}} \notin U$, then we apply a controlled rotation to the control $v_{i_{j+1}}$ and the target v_{i_j} qubits. After that, we add $v_{i_{j+1}}$ to the set U, i.e., $U \leftarrow U \cup \{v_{i_{j+1}}\}$. If $j = k-1$, then we terminate the algorithm. Otherwise, we go to Step 5.

Step 5. If $v_{i_{j+2}} \neq v_{i_j}$, then we continue; otherwise, we go to Step 6. We apply the SWAP gate to v_{i_j} and $v_{i_{j+1}}$. After that, we update $j \leftarrow j+1$ because the value of the target qubit moves to $v_{i_{j+1}}$. Then, we go to Step 4.

Step 6. If $v_{i_{j+2}} = v_{i_j}$, then we update $j \leftarrow j+2$. Note that here we stay on the same vertex of the graph G. Then, we go to Step 4.

Let us present a procedure ConstructForPath(P) in Algorithm 4 that implements the above idea for a given path $P = (v_{i_1}, \ldots, v_{i_k})$. We assume that we have cR(u,v) procedure that applies the controlled rotation operator CR_y to u as a control qubit and v as a target one. The angle ξ_r corresponds to the qubit q_r associated with the vertex v. Additionally, we have swap(u,v) procedure that applies the swap gate to u and v qubits.

We assume that ShortestNSPath(G) returns a shortest non-simple path in the graph G that visits all vertices at least once. The implementation of this procedure can be found in Algorithm 2 or Algorithm 3. Finally, we can present Algorithm 5 as an implementation of the quantum fingerprinting (quantum hashing) algorithm.

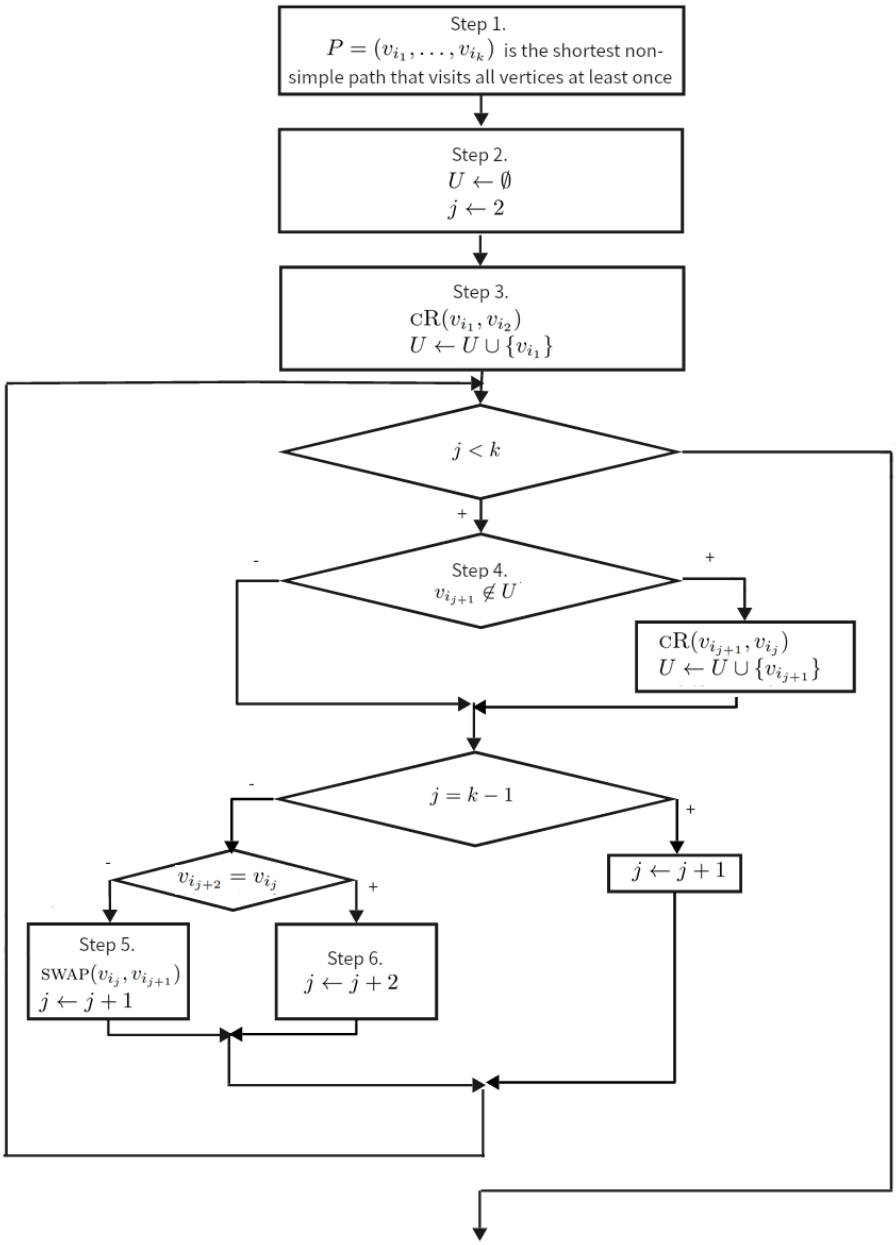

Figure 2. The flowchart for algorithm constructing a circuit for quantum hashing.

Proof. Let us look to the shortest path $P = (v_{i_1}, \ldots, v_k)$ that visits all vertices at least once.

The shallow circuit (Figure 1) is such that we should apply controlled rotation operators with each qubit as the control qubit and a specific qubit as the target.

The algorithm places the target qubit at v_{i_2}. Then, we apply the controlled rotation operator with v_{i_1} as the control qubit. After that, we apply the controlled rotation operator with v_{i_3} as the control qubit. Then, we move the target qubit to v_{i_3} using the swap operator and apply the controlled rotation operator with v_{i_4} as the control qubit.

So, if the target qubit is in v_{i_j}, then we apply the controlled rotation operator with $v_{i_{j+1}}$ as the control qubit and move the target qubit to $v_{i_{j+1}}$ using the swap gate.

We apply the controlled rotation operator with $v_{i_{j+1}}$ as the control qubit only if it is not in the set U and then add it to the set U. This action guarantees us that we apply the controlled rotation with each qubit only once. At the same time, the path visits each vertex at least once. This means that each qubit (vertex) occurs as $v_{i_{j+1}}$ for some j. So, we apply the controlled rotation operator with each qubit exactly once, and there are no other modifications of the target qubit. Therefore, the circuit implements the shallow circuit of the quantum fingerprinting (quantum hashing) algorithm presented in Figure 1. At the same time, we apply the swap or controlled rotation operators only to neighboring qubits. □

Algorithm 4 Implementation of ConstructForPath(P) procedure. Algorithm of constructing circuit for quantum hashing or quantum fingerprinting for a path $P = (v_{i_1}, \ldots, v_{i_k})$

$U \leftarrow \emptyset$ ▷ Step 2
$j \leftarrow 2$
cR(v_{i_1}, v_{i_2}) ▷ Step 3
$U \leftarrow U \cup \{v_{i_1}\}$
while $j < k$ **do**
  if $v_{i_{j+1}} \notin U$ **then** ▷ Step 4
    cR$(v_{i_{j+1}}, v_{i_j})$
    $U \leftarrow U \cup \{v_{i_{j+1}}\}$
  end if
  if $j = k-1$ **then**
    $j \leftarrow j+1$
  else
    if $v_{i_{j+2}} = v_{i_j}$ **then**
      $j \leftarrow j+2$ ▷ Step 6
    else
      SWAP$(v_{i_j}, v_{i_{j+1}})$ ▷ Step 5
      $j \leftarrow j+1$
    end if
  end if
end while

Algorithm 5 Algorithm of constructing circuit for quantum hashing or quantum fingerprinting for a path $P = (v_{i_1}, \ldots, v_{i_k})$

$P = (v_{i_1}, \ldots, v_{i_k}) \leftarrow$ ShortestNSPath(G) ▷ Step 1
ConstructForPath(P)

Let us show the correctness of the algorithm.

Theorem 3. *The quantum circuit produced by Algorithm 5 implements the shallow circuit of quantum hashing from Figure 1.*

We can represent the controlled rotation operator cR(u, v) and the corresponding angle ξ as a sequence of R$(v, \xi/2)$, CNOT(u, v), R$(v, -\xi/2)$, and CNOT(u, v), (see Figure 3). Here R$(v, \xi/2)$ is the $R_y(\xi/2)$ gate applied to the qubit v; CNOT(u, v) is the $CNOT$ for u as a control and v as a target.

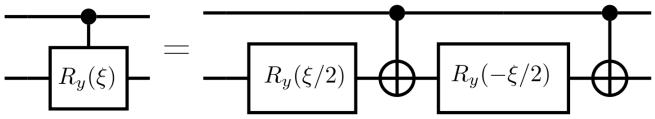

Figure 3. Representation of CR_y gate using only basic gates

Additionally, the SWAP(u, v) gate can be represented as a sequence CNOT(u, v), CNOT(v, u), and CNOT(u, v) (see Figure 4).

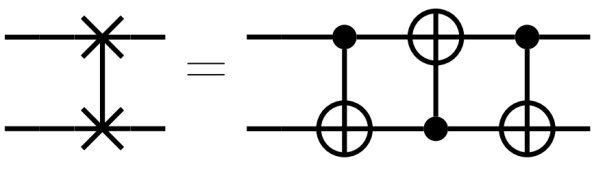

Figure 4. Representation of $SWAP$ gate using only basic gates

Two sequential operators CR_y and $SWAP$ that are $\text{cR}(u,v)$ and $\text{SWAP}(u,v)$ procedures can be represented by a circuit in Figure 5.

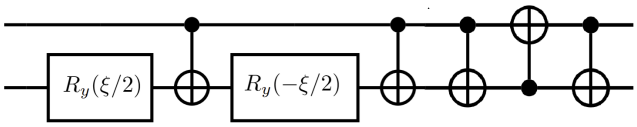

Figure 5. Representation of a pair CR_y and $SWAP$ gates using only basic gates

Note that two sequential $\text{CNOT}(u,v)$ gates are annihilated, as presented in [23,79]. So, we obtain the circuit in Figure 6.

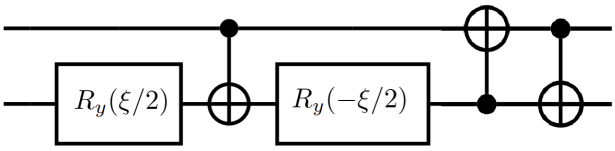

Figure 6. Reduced representation of a pair CR_y and $SWAP$ gates using only basic gates

Let us look at the CNOT cost of these operators, where the CNOT cost is the number of CNOT operators in the representation of a circuit using only basic gates.

We can say that the CNOT cost of $\text{cR}(u,v)$ is 2; CNOT cost of $\text{SWAP}(u,v)$ is 3, CNOT cost of two sequential operators $\text{cR}(u,v)$ and $\text{CNOT}(u,v)$ is 3.

Finally, we can discuss the CNOT cost of the constructed circuit.

Theorem 4. *CNOT cost of the circuit generated by Algorithm 5 is* $3k-4b-5 \leq 3k-5$*, where* k *is the length of a shortest non-simple path* $P=(v_{i_1},\ldots,v_{i_k})$ *in the graph* G *that visits all vertices at least once, and* b *is the number of indices* j *such that* $v_{i_j}=v_{i_{j+2}}$.

Proof. If we look at Algorithm 5, then we can see that it is a sequence of one of three options:

- a pair of CR_y and $SWAP$ gates if $v_j \neq v_{j+2}$ and $v_{j+1} \notin U$.
- $SWAP$ gate if $v_j \neq v_{j+2}$ and $v_{j+1} \in U$.
- CR_y gate if $v_j = v_{j+2}$ and $v_{j+1} \notin U$.

Note that the option $v_j = v_{j+2}$ and $v_{j+1} \in U$ is not possible because $v_{j+1} \in U$ means that the corresponding qubit was already visited. In that case, we can just exclude it from the path because $v_j = v_{j+2}$. At the same time, it contradicts the condition that P is the shortest path.

The CNOT cost for pairs of CR_y and $SWAP$ gates is 3; for the $SWAP$ gate, it is 3; for the CR_y gate, it is 2.

Let b be the number of steps where we apply pairs of CR_y and $SWAP$ gates or the $SWAP$ gate, and let b be the number of steps where we apply CR_y (here we do not count the first and the last vertex from the path because they are processed by the different logic). Here $a+2b=k-3$ because (i) if $v_j = v_{j+2}$, then we use only CR_y to v_{j+1} and skip v_{j+2} (ii) we do not count the target qubit. So, the total number of CNOT gates is $3a+2b+4 = 3(a+2b+3)-4b-5 = 3k-4b-5$. □

Using Lemma 3 we can estimate the CNOT cost of the circuit and present it in the next corollary.

Corollary 1. *CNOT cost of the circuit that is generated using Algorithm 5 is between* $2n-2$ *and* $6n-11$.

Proof. The circuit has the minimal CNOT cost if each operation required only two CNOT gates, which means no SWAP gates. This happens if we have a "star" graph, where one vertex (a middle vertex) is connected to all other $n-1$ vertices. In that case $k=2(n-1)-1=2n-3$, we start from a non-middle vertex and come to the middle vertex after each other, except the last one. The value $b=n-3$ because each non-middle vertex, except the first and last in the path satisfies the required property. So, the CNOT cost according to Theorem 4 is $3(2n-3)-4(n-3)-5 = 6n-9-4n+12-5 = 2n-2$.

The maximum value is for $k=2n-1$, and the minimum value for $b=0$. So, the maximum value for the CNOT cost is $3(2n-1)-5 = 6n-6-5 = 6n-11$. □

Typically, the quantum fingerprinting (quantum hashing) algorithm is used several times. For example, for hashing a string of ℓ symbols, we should apply the algorithm ℓ times (see Appendix B for examples). For this reason, we suggest two strategies.

Strategy with a Path.

Let a step be a single application of the quantum fingerprinting algorithm. After each step, we reverse the path P. Then, the target qubit travels back. Additionally, we can see that the operator $\textsc{cR}(v_{i_{k-1}}, v_{i_k})$ is applied at the end of the odd step and at the beginning of the even step. We can apply it once with a double angle. The same situation with $\textsc{cR}(v_{i_1}, v_{i_2})$ on the meeting of even and odd steps. Therefore, we have the following CNOT cost for ℓ applications of the quantum hashing algorithm.

Theorem 5. *CNOT cost of the circuit to apply the quantum fingerprinting (quantum hashing) algorithm ℓ times is at most $(3k-4b-7)\ell+2 \le (3k-7)\ell+2$, where k is the length of the shortest non-simple path $P=(v_{i_1},\dots,v_{i_k})$ in the graph G that visits all vertices at least once, and b is the number of indices j such that $v_{i_j}=v_{i_{j+2}}$.*

Proof. The first step costs $3k-5$ due to Theorem 4. Each next step skips the first cR gate because it is "merged" with the last cR from the previous step. So, its CNOT cost is $3k-7$ because cR costs 2. The total cost is $3k-5+(3k-7)(\ell-1)=(3k-7)\ell+2$. □

Similarly to Corollary 1 we can show the next result.

Corollary 2. *The CNOT cost of the circuit for application of the quantum fingerprinting (quantum hashing) algorithm ℓ times is between $(2n-4)\ell+2$ and $(6n-13)\ell+2$.*

Strategy with a Cycle.

At the beginning of each step, starting from the second one, we add $\textsc{swap}(v_{i_k}, v_{i_1})$ and $\textsc{swap}(v_{i_1}, v_{i_2})$ gates. We need the first swap because we should start from the beginning of the cycle next time. We add the second one because we should start from the second element of the cycle. These two gates increase the cost by 2 because they are jointed with corresponding cR gates. At the same time, all the logical qubits are moved to position 1 of the cycle. That is why we need only $k-1$ travels by the cycle for doing k steps.

Therefore, we have the following CNOT cost for ℓ applications of the quantum hashing algorithm.

Theorem 6. *CNOT cost of the circuit for application of the quantum fingerprinting (quantum hashing) algorithm ℓ times is at most $3(k-1)(\ell+1)-(4b+3)(\ell-\frac{\ell}{k})+5 \le 3(k-1)(\ell+1)-3(\ell-\frac{\ell}{k})+5$, where $k+1$ is the length of the shortest non-simple cycle $C=(v_{i_1},\dots,v_{i_k},v_{i_1})$ in the graph G that visits all vertices at least once, and b is the number of indices j such that $v_{i_j}=v_{i_{j+2}}$.*

Proof. The first step costs $3k-4b-5$ due to Theorem 4. Each next step costs $3k-4b-3$ because of two additional swap gates. At the same time, we should apply them only $\ell-\left\lfloor\frac{\ell}{k}\right\rfloor$ due to the movement of the logic qubits by the cycle.

The total cost is $3k-4b-5+(3k-4b-3)(\ell-\left\lfloor\frac{\ell}{k}\right\rfloor-1)=(3k-4b-3)(\ell-\left\lfloor\frac{\ell}{k}\right\rfloor)+2=$

$3k\ell-4b\ell-3\ell-3k\left\lfloor\frac{\ell}{k}\right\rfloor+(4b+3)\left\lfloor\frac{\ell}{k}\right\rfloor+2 \le 3k\ell-4b\ell-3\ell-3k(\frac{\ell}{k}-1)+(4b+3)\frac{\ell}{k}+2=3k\ell-4b\ell-3\ell-3\ell+3k+4b\frac{\ell}{k}+3\frac{\ell}{k}+2=3k\ell-3\ell+3k-4b(\ell-\frac{\ell}{k})-3(\ell-\frac{\ell}{k})+2=3k\ell-3\ell+3k-(4b+3)(\ell-\frac{\ell}{k})+2=3k(\ell+1)-3(\ell+1)+3-(4b+3)(\ell-\frac{\ell}{k})+2=3(k-1)(\ell+1)-(4b+3)(\ell-\frac{\ell}{k})+5.$

Due to $b\ge 0$, we have $3(k-1)(\ell+1)-(4b+3)(\ell-\frac{\ell}{k})+5 \le 3(k-1)(\ell+1)-3(\ell-\frac{\ell}{k})+5$. □

Similarly to Corollary 1 we can show the next result.

Corollary 3. *The CNOT cost of the circuit for application of the quantum hashing (quantum fingerprinting) algorithm ℓ times is between $(2n-4)\ell+2\ell-\frac{2\ell}{n-1}-2$ and $(6n-13)\ell+\ell+6n+\frac{6\ell}{2n-1}-11$.*

Proof. Let us estimate the minimal cost similarly to Corollary 1. The first step costs $2n-2$. Each next step costs $2n-2+2=2n$. The total complexity is $2n-2+2n(\ell-\lfloor\frac{\ell}{k}\rfloor-1)=2n(\ell-\lfloor\frac{\ell}{k}\rfloor)-2 \ge 2n(\ell-\frac{\ell}{k})-2 \ge$

$2n(\ell - \frac{\ell}{n-1}) - 2 = 2n\ell - \frac{(2n-2+2)\ell}{n-1} - 2 = 2n\ell - \frac{2(n-1)\ell}{n-1} - \frac{2\ell}{n-1} - 2 = 2n\ell - 2\ell - \frac{2\ell}{n-1} - 2 = (2n-4)\ell + 2\ell - \frac{2\ell}{n-1} - 2$.

Let us estimate the maximal cost similarly to Corollary 1. The first step costs $6n-11$. Each next step costs $6n-11+2 = 6n-9$. The total complexity is $6n-11+(6n-9)(\ell - \lfloor\frac{\ell}{k}\rfloor - 1) = (6n-9)(\ell - \lfloor\frac{\ell}{k}\rfloor) - 2 \leq (6n-9)(\ell - \lfloor\frac{\ell}{2n-1}\rfloor) - 2 \leq (6n-9)(\ell - \frac{\ell}{2n-1} + 1) - 2 = (6n-9)\ell + (6n-9) - 3\frac{(2n-1-2)\ell}{2n-1} - 2 = (6n-9)\ell + (6n-9) - 3\ell + 6\frac{\ell}{2n-1} - 2 = (6n-13)\ell + \ell + 6n + \frac{6\ell}{2n-1} - 11$. □

We can see that the complexity presented in Corollary 1 is less than the result from Corollary 3 because each cycle that visits all the vertices at least once can be converted to a path by removing the last edge. At the same time, the repeated application of the same pattern of gates for each step can be useful in some hardware implementations.

4.1. Comparing with Existing Results

Let us compare our generic approach with existing circuits for specific graphs. The LNN architecture is a common layout graph for many quantum devices. The graph is a chain where v_1 is connected with v_2; v_i is connected with v_{i-1} and v_{i+1} for $i \in \{2, \ldots, n-1\}$; v_n is connected with v_{n-1}.

The quantum fingerprinting (quantum hashing) algorithm for the LNN architecture was discussed in [79]. The path that visits all vertices in the graph is simply $P = (v_1, \ldots, v_n)$ (see Figure 7). So, our method gives us the same circuit as in [79]. The circuit for the quantum fingerprinting algorithm on 5 qubits is presented in Figure 8.

Figure 7. The graph for the 5-qubit LNN architecture. The path that visits all vertices at least once is green.

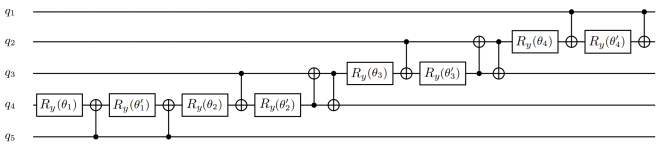

Figure 8. A quantum circuit for the Quantum hashing (quantum fingerprinting) algorithm for 5 qubits LNN architecture device.

The CNOT cost of the circuit is presented in the next lemma.

Lemma 5. *The CNOT cost of the produced circuit of the quantum fingerprinting (quantum hashing) algorithm with n qubits for the LNN architecture is $3n-5$ for one application and $3n\ell - 7\ell + 2$ for ℓ applications.*

Let us consider more complex graphs like a cycle with tails (like a "sun" or "two joint suns"). The 16- and 27-qubit IBM machines of this architecture were considered in [23]. The graphs and paths are presented in Figures 9 and 10.

Note that the red part of the path is such that $v_{j+2} = v_j$. Therefore, we do not apply a SWAP gate on this part of the path, but only a CR_y gate. So, our generic method gives the same circuits as the circuits specially constructed for these devices [23]. The CNOT cost is 40 for the 16-qubit architecture, and 69 for the 27-qubit one.

We collect the results for the discussed architectures in Table 1 for LNN and Table 2 for others. Additionally, we added results in Table 3 for connectivity graphs of IBMQ Melbourne, Rigetti Aspen-4, and a 5×5-grid as a connectivity graph that can be found, for example, in [102]. They are presented in Figure 11.

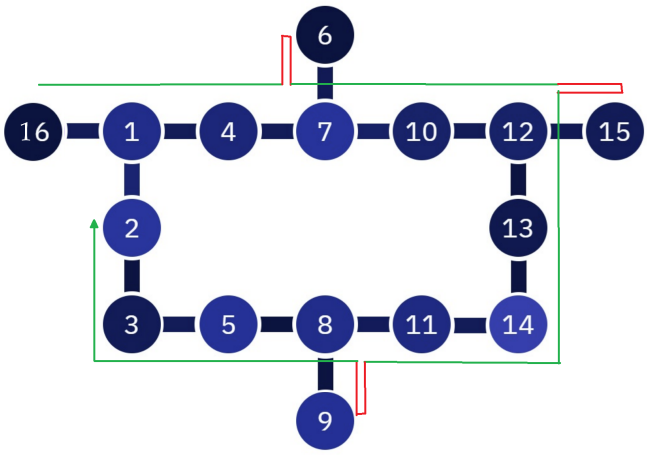

Figure 9. The graph for 16-qubit "sun" architecture. The path that visits all vertices at least once is green. Red parts are such that $v_{i_j} = v_{i_{j+2}}$.

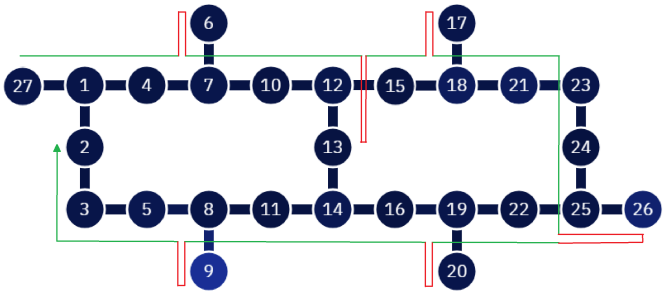

Figure 10. The graph for 27-qubit "two joint suns" architecture. The path that visits all vertices at least once is green. Red parts are such that $v_{i_j} = v_{i_{j+2}}$.

Moreover, we compare our algorithm with Qiskit transpiler [103] optimized circuit for the mentioned architectures. The Qiskit's algorithm is randomized heuristic algorithm for circuit optimization. That is why we invoke it 100 times and choose the minimum of the produced circuits' CNOT costs. We added the results to the tables.

The results were obtained using the algorithm implemented in C++. The implementation can be found in GitLab repository [104].

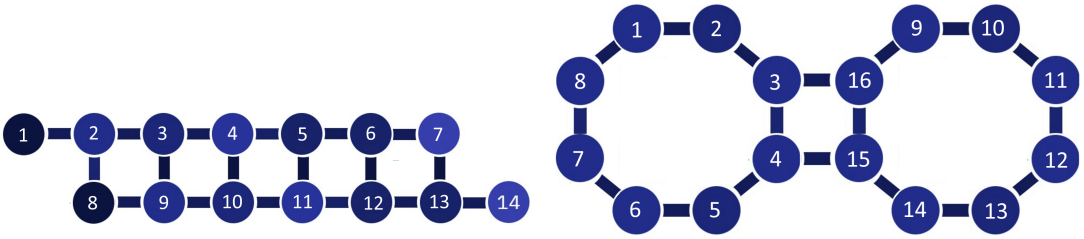

Figure 11. Connectivity graphs of IBMQ Melbourne, Regetti Aspen-4, and 5×5-grid.

Table 1. The CNOT cost for the LNN architecture. The CNOT cost of circuits produced by algorithms from [79] and this paper is $3n - 5$

The number of qubits n	[79] and this paper	Qiskit transpiler
5	10	14
10	25	39

Table 2. The CNOT cost for "sun" and "two joint suns" architectures

Connectivity graph's type	[23] and this paper	Qiskit transpiler
16-qubit "sun" architecture	40	57
27-qubit "two joint suns" architecture	69	115

Table 3. The CNOT cost for IBMQ Melbourne, Regetti Aspen-4, and 5×5-grid

Connectivity graph's type	This paper	Qiskit transpiler
14-qubit IBMQ Melbourne	36	44
16-qubit Regetti Aspen-4	43	66
25-qubit 5×5-grid	70	84

Remark 2. *In this paper, we focus on the CNOT cost (the number of CNOT gates) because two-qubit gates are harder to implement on real devices and give larger computational error compared to one-qubit gates. At the same time, there is another common metric of circuit quality that is depth (the number of steps, where each step is a layer of gates that can be done in parallel on different qubits). Often, the depth of a circuit is associated with time complexity. We do not optimize this metric in this paper. However, if we compute the depth of the circuit, then it can be obtained from the CNOT cost by adding* $2(n-1)$ *because we do* $n-1$ *controlled rotations in total and each of them requires two rotation gates as shown in Figure 3.*

5. Method for Constructing a Circuit for Quantum Fourier Transform

Here we present a method that allows us to construct a circuit for the Quantum Fourier Transform (QFT) for a connected graph $G = (V, E)$ that is a qubit connectivity graph for a device. More information about the QFT algorithm can be found in Appendix C. If we do not have restrictions for applying two-qubit gates (when G is a complete graph for instance), then the circuit is as presented in Figure 12.

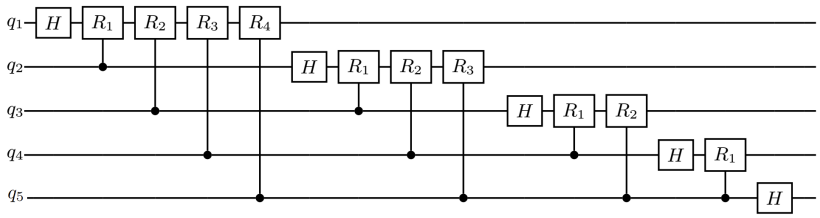

Figure 12. A quantum circuit for the Quantum Fourier Transform algorithm for fully connected 5 qubits.

The author of [23] shows that the circuit for QFT is a series of CNOT cascades. Each of the cascades has a gate structure similar to the shallow circuit for the quantum hashing algorithm. Assume that we have a ConstructQFTForPath(P', k', r) procedure that constructs the r-th cascade of the circuit for QFT for a path P' of length k'. Note that for QFT, the size of each cascade is different. Here P' is a path that visits only the vertices corresponding to the qubits used in the current cascade. Firstly, we present the main algorithm in Section 5.1. Then we present an algorithm for the ConstructQFTForPath(P', k', r) procedure in Section 5.2. After that we discuss the complexity of the circuit in Section 5.3. Finally, we compare the circuit with existing results in Section 5.4.

5.1. The Main Algorithm

Let us present the whole algorithm for constructing the quantum circuit for the QFT algorithm.

Firstly, we find a non-simple path $P = (v_{i_1}, \ldots, v_{i_k})$ that visits all vertices at least once. Then we assign logical qubits to the vertices according to this path. Let indices $j_1, \ldots, j_n$ be the indices of the first occurrence of each vertex. In other words, j_r is such that $i_{j_r} = r$, and $i_h \neq r$ for each $1 \leq h < j_r$.

Then, we assign $Q_r \leftarrow j_r$ the position of r-th logical qubit in the graph, and $T_{j_r} \leftarrow r$ the index of the logical qubit associated with a vertex.

After that, we start to construct the circuit for cascades. For the first cascade, we invoke ConstructQFTForPath(P,k). Similarly to the quantum hashing (quantum fingerprinting) algorithm, the target qubit (i.e., 1st logical qubit) moves to the end of the path, i.e., $Q_1 = i_k$.

For the second cascade, we exclude the vertex with index Q_1 from the graph and find the path P' that visits all vertices at least once in the updated graph such that it starts from the vertex v_{Q_2}. The vertex v_{Q_2} contains the 2nd logical qubit that should be the target in the second cascade. Then, we invoke ConstructQFTForPath(P',k'), where k' is the length of P'.

For the next cascades, we act in a similar way. For the r-th cascade, we exclude $v_{Q_{r-1}}$, find the path P' that visits all vertices at least once in the updated graph such that it starts from the vertex v_{Q_r}, and invoke ConstructQFTForPath(P',k').

Note that a path in the updated graph always exists because it is connected. We have at least one path that visits all vertices at least once, and we exclude only one vertex, that is, the last one on this path.

During the implementation, we do not remove the vertex from the graph; we just store a set of "deleted" vertices and we prohibit visiting vertices from this set during the algorithm for the path search.

Let us present the algorithm.

Step 1. We find a non-simple path $P = (v_{i_1}, \ldots, v_{i_k})$ that visits all vertices at least once.

Step 2. We find indices $j_1, \ldots, j_n$, where j_r is such that $i_h \neq i_{j_r}$ for each $1 \leq h < j_r$. Then, we assign $Q_r \leftarrow j_r$ and $T_{i_{j_r}} \leftarrow r$ for $1 \leq r \leq n$.

Step 3. We assign $P' \leftarrow P$, and k' is the length of P'. We assign $r \leftarrow 1$, which is an index of the cascade.

The next steps are repeated while the graph contains at least one vertex.

Step 4. We find a non-simple path P' of length k' that visits all non-deleted vertices of the graph and starts from v_{Q_r}. If $r = 1$, then we do nothing because P' was already found in Steps 1-3.

Step 5. We invoke ConstructQFTForPath(P',k',r), which is the r-th cascade. Assume that the procedure keeps the Q and T indices actual.

Step 6. We exclude v_{Q_r} from the graph.

Step 7. If $r = n$, then we terminate the algorithm. Otherwise, we increase the index of the cascade $r \leftarrow r+1$ and go to Step 4.

The implementation of the algorithm is presented in Algorithm 6. Assume that we have the ShortestNSPath(G) procedure that returns the shortest non-simple path visiting all vertices, and the ShortestNSPath(G,v) procedure that returns a similar path that starts from the vertex v. Let getFirstIndices(P) be a procedure that returns $j_1, \ldots, j_n$. The implementation of this procedure is simple, has $O(k \log n)$ time complexity, and is presented in Appendix D.

Algorithm 6 Implementation of the algorithm for constructing the whole circuit for QFT for a path $P = (v_{i_1}, \ldots, v_{i_k})$

```
P ← ShortestNSPath(G), k ← len(P)
(j_1, ..., j_n) ← getFirstIndices(P)
for r ∈ {1, ..., n} do
    Q_r ← j_r
    T_{i_{j_r}} ← r
end for
P' ← P, k' ← k, r ← 1
while r ≤ n do
    ConstructQFTForPath(P', k', r)
    V ← V\{v_{Q_r}}
    r ← r + 1
    if r ≤ n then
        P' ← ShortestNSPath(G, v_{Q_r}), k' ← len(P')
    end if
end while
```

Let us discuss the time complexity of the algorithm. The time complexity of ShortestNSPath(G) and ShortestNSPath(G,v) is $O(n^2 2^n)$ due to Theorem 1. In each step, the number of vertices is reduced by 1. So,

the total complexity is $O\left(\sum_{i=1}^{n}\left(i^2 2^i\right)\right) = O(n^2 2^n)$. In the case of the heuristic solution, we have a similar situation. The complexity of the presented solution from Section 3.1 is $O\left(\sum_{i=1}^{n} i^5\right) = O(n^6)$.

5.2. Quantum Circuit for One Cascade

The procedure ConstructQFTForPath(P', k', r) is very similar to the ConstructForPath(P) procedure that constructs the circuit for the quantum hashing algorithm. At the same time, there are many small, specific modifications. That is why we present the whole algorithm here.

Step 1. We start with the first qubit in the path $j \leftarrow 1$. We apply the Hadamard transformation to the qubit corresponding to the vertex v_{i_1}. We denote this action by $\mathrm{H}(v_{i_1})$. If $k' = 1$, then we terminate our algorithm; otherwise, go to Step 2.

Step 2. If $j = k'$, then we terminate the algorithm; otherwise, we continue. If $v_{i_{j+1}} \notin U$, then we apply the controlled phase gate CR_d with the control $v_{i_{j+1}}$ and the target v_{i_j} qubits, where $d = T_{i_{j+1}} - r$. Then, we add $v_{i_{j+1}}$ to the set U, i.e., $U \leftarrow U \cup \{v_{i_{j+1}}\}$. Then, we go to Step 3.

Step 3. If $j = k' - 1$ or $v_{i_{j+2}} \neq v_{i_j}$, then we continue; otherwise, we go to Step 4. We apply the SWAP gate to v_{i_j} and $v_{i_{j+1}}$, and swap the indices of qubits for these vertices. In other words, if $w_1 = T_{i_j}$ and $w_2 = T_{i_{j+1}}$ are indices of the corresponding logical qubits, then we swap Q_{w_1} and Q_{w_2} values, and T_{i_j} and $T_{i_{j+1}}$ values. Then, we update $j \leftarrow j + 1$ because the value of the target qubit moves to $v_{i_{j+1}}$. Then, we go to Step 2.

Step 4. If $v_{i_{j+2}} = v_{i_j}$, then we update $j \leftarrow j + 2$. Note that here we stay on the same vertex of the graph G. Then, we go to Step 2.

Finally, we obtain ConstructQFTForPath(P', r) procedure whose implementation is presented in Algorithm 7. This procedure constructs the r-th part (cascade) of the circuit for QFT for the path P'.

Algorithm 7 Implementation of ConstructQFTForPath(P', k', r) procedure. Algorithm of constructing the circuit for the r-th cascade for the path $P' = (v_{i_1}, \ldots, v_{i_{k'}})$

```
j ← 1                                                        ▷ Step 1
H(v_{i_j})
U ← ∅
while j ≤ k' do                                              ▷ Step 2
    if v_{i_{j+1}} ∉ U then
        d ← T_{i_{j+1}} − r
        CR_d(v_{i_{j+1}}, v_{i_j})
        U ← U ∪ {v_{i_{j+1}}}
    end if
    if j = k' or v_{i_{j+2}} ≠ v_{i_j} then                  ▷ Step 3
        if k' ≠ 2 then
            SWAP(v_{i_j}, v_{i_{j+1}})
            w_1 ← T_{i_j}, w_2 ← T_{i_{j+1}}
            Q_{w_1} ← i_{j+1}, Q_{w_2} ← i_j
            T_{i_j} ← w_2, T_{i_{j+1}} ← w_1
        end if
        j ← j + 1
    else                                                     ▷ Step 4
        j ← j + 2
    end if
end while
```

5.3. The CNOT Cost of the Circuit and Correctness of the Algorithm

Let us start with discussion correctness of the algorithm.

Theorem 7. *The quantum circuit produced by Algorithm 6 implements the circuit of the quantum Fourier transform algorithm from Figure 12.*

Proof. The r-th cascade of the QFT circuit is such that we should apply controlled phase operators with each logical qubit with indexes larger than r as the control qubit and r-th qubit as the target.

For the cascade, the algorithm computes the path that starts from the vertex corresponding to the r-th logical qubit and visits all vertices corresponding to the logical qubit with indexes larger than r. Let the path be $P' = (v_{i_1}, \ldots, v_{i'_k})$. Here we use the qubit from v_{i_1} as a target qubit. It corresponds to the r-th logical qubit. Then, we apply the controlled phase operator with v_{i_2} as the control qubit. After that, we move the target qubit to v_{i_2} using the swap operator and apply the controlled phase operator with v_{i_3} as the control qubit.

So, if the target qubit is in v_{i_j}, then we apply the controlled phase operator with $v_{i_{j+1}}$ as the control qubit and move the target qubit to $v_{i_{j+1}}$ using the swap gate.

We apply the controlled rotation phase with $v_{i_{j+1}}$ as the control qubit only if it is not in the set U, and then add it to the set U. This action guarantees us that we apply the controlled phase operator with each required qubit only once. At the same time, the path visits each vertex corresponding to logical qubits with indexes larger than r at least once. This means that each qubit (vertex) occurs as $v_{i_{j+1}}$ for some j. So, we apply the controlled phase operator with each qubit exactly once, and there are no other modifications of the target qubit. Therefore, the circuit implements the r-th cascade.

After that we exclude the vertex corresponding to r-th logical qubit from the graph. Note that the rest of the graph is connected because the r-th logical qubit moved to the end of the path P'. So, we can be sure that all vertices corresponding to logical qubits with indexes larger than r are connected with path P' without the last vertex. Therefore, we can repeat the algorithm for the next cascades.

So, the produced circuit implements all cascades of the QFT circuit presented in Figure 1. At the same time, we apply the swap or controlled phase operators only to neighboring qubits. □

Let us continue with CNOT cost of the circuit.

Note that the CR_d gate can be represented using only two CNOT gates and three R_z gates similar to CR_y [105] (see Figure 13).

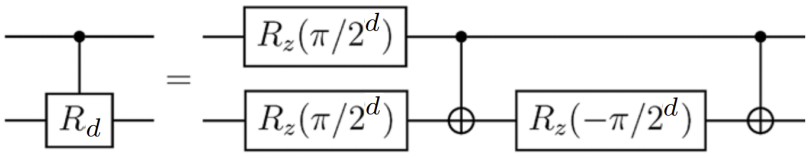

Figure 13. Representation of CR_d gate using only basic gates.

A pair CR_d and $SWAP$ can be represented using three CNOT gates. (See Figure 14).

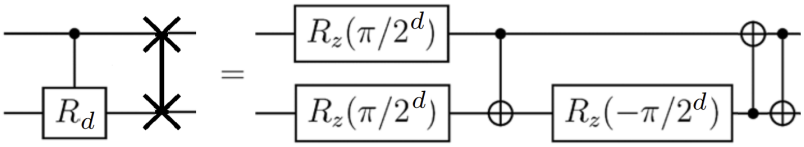

Figure 14. Reduced representation of a pair R_d and $SWAP$ gates using only basic gates.

Theorem 8. *If the graph G with n vertices has a Hamiltonian path, then the CNOT cost of the produced circuit for the QFT algorithm is* $1.5n^2 - 1.5n - 1$.

Proof. Because the graph has a Hamiltonian path, the path $P = (v_{i_1}, \ldots, v_{i_n})$ is exactly this path. So, in each step, we move the target qubit. We can say that in constructing the r-th cascade, we use the path $P' = (v_{i_1}, \ldots, v_{i_{n-r+1}})$. The CNOT cost of the r-th cascade for $1 \leq r \leq n-2$ is $3(n-r)$ because we apply SWAP and the controlled phase gates for each edge of the path. The CNOT cost of the $(n-1)$-th cascade is 2 because we do not apply the SWAP gate. The CNOT cost of the n-th cascade is 0. Therefore, the total cost is $\sum_{r=1}^{n-1}(3(n-r)) - 1 = 1.5n^2 - 1.5n - 1$. □

Let us discuss the CNOT cost of the algorithm in the next theorem.

Theorem 9. *The CNOT cost of a circuit that is generated using Algorithm 5 is between* $n^2 + n - 4$ *and* $3n^2 - 3n - 10$.

Proof. Similarly to Corollary 1, the minimal possible cost is when we move the target qubit the minimum possible times. It is the situation with the "star" graph where we have a vertex connected with all others. The cost of the r-th cascade in that case is $3+3+2\cdot(n-2-r)=2(n-r)+2$, because we always remove one "non-middle" vertex. The path starts from "non-middle" and finishes in "non-middle." Therefore, we should make two moves of the target qubit. For the $(n-1)$-th cascade, the cost is only 2. So, the total complexity is at least $\sum_{r=1}^{n-2}(2(n-r)+2)+2 = n^2+n-4$.

Similarly to the proof of Theorem 4, we can show that the CNOT cost of one cascade is at most $3(k-1)$. After each cascade, the number of vertices is decreased by 1. The maximum possible value for k is $2n-1$. In the r-cascade, it is $2(n-r+1)-1=2(n-r)+1$. The cost of the $(n-1)$-th cascade is 2, not 3, because we do not move the target qubit. The cost of the $(n-2)$-th cascade is 6, because the length of the path in the graph with 3 vertices is always 3. So, the total complexity is at most $\sum_{r=1}^{n-3}(3\cdot 2(n-r))+6+2 = 3n^2-3n-10$. □

5.4. Comparing with Other Results

Similarly to Section 4.1 we discuss the LNN architecture and the more complex architecture. Firstly, let us consider the LNN architecture. As discussed earlier (see Figure 7), the path visits all vertices from v_1 to v_n one by one. The paths constructed for each cascade are presented in Appendix E. The circuit is similar to the circuit developed in [23]. The presented path is the Hamiltonian path in the graph. Due to Theorem 8, we get the following CNOT cost for the LNN architecture.

Lemma 6. *The CNOT cost of the produced circuit of the QFT algorithm using n qubits for the LNN architecture is* $1.5n^2-1.5n-1$.

At the same time, [14] gives the circuit with n^2+n-4 CNOT cost, and [23] gives the circuit with $1.5n^2-2.5n+1$ CNOT cost. Our circuit is better than [14] only if $n\leq 3$. The result from [23] is better than our result. However, it is not known how to apply the results of [14] for more complex architectures, and the results of [23] are presented only for a specific architecture.

Secondly, let us consider more complex architectures like 16-qubit "sun" (Figure 9), and 27-qubit "two joint suns" (Figure 10). The paths are presented in figures. Note that the red part of the path is such that $v_{j+2}=v_j$. Therefore, we do not apply a SWAP gate on this step, but only the controlled phase operator. The CNOT cost for the 16-qubit machine is 342, and for the 27-qubit machine is 1009.

So, our generic method gives circuits that are a bit worse than the circuits constructed especially for these devices [23], which CNOT costs are 324 and 957 for the 16-qubit and the 27-qubit architectures, respectively. The difference is about 5%.

We have collected the results of discussed architectures in Table 4 for LNN and Table 5 for others. Additionally, we added results in Table 6 for connectivity graphs of IBMQ Melbourne, Rigetti Aspen-4, and a 5×5-grid as a connectivity graph that can be found, for example, in [102]. They are presented in Figure 11.

Moreover, we compare our algorithm with the Qiskit transpiler [103] optimized circuit for the mentioned architectures. The Qiskit's algorithm is a randomized heuristic algorithm for circuit optimization. That is why we invoke it 100 times and choose the minimum of the produced circuits' CNOT costs. We added the results to the tables.

The results were obtained using the algorithm implemented in C++. The implementation can be found in the GitLab repository [104].

Table 4. The CNOT cost of quantum circuit for the QFT algorithm for LNN architecture. The CNOT cost of circuits produced by the algorithm from [14] is n^2+n-4; by the algorithm from [79] is $1.5n^2-2.5n+1$; the algorithm from this paper is $1.5n^2-1.5n-1$.

The number of qubits n	This paper	[14]	[79]	Qiskit transpiler
3	8	8	7	9
4	17	16	15	21
5	29	26	26	38
10	134	106	126	207

Table 5. The CNOT cost of quantum circuit for the QFT algorithm for "sun" and "two joint suns" architectures

Connectivity graph's type	This paper	[23]	Qiskit transpiler
16-qubit "sun" architecture	342	324	549
27-qubit "two joint suns" architecture	1009	957	1839

Table 6. The CNOT cost of quantum circuit for the QFT algorithm for IBMQ Melbourne, Regetti Aspen-4, and 5×5-grid

Connectivity graph's type	This paper	Qiskit transpiler
14-qubit IBMQ Melbourne	269	335
16-qubit Regetti Aspen-4	359	585
25-qubit 5×5-grid	899	1158

Remark 3. *As we discussed in Remark 2, here, we focus on the CNOT cost but not the depth of the circuit. If we compute the depth of the circuit, then it can be obtained from the CNOT cost by adding n^2 because for r-th cascade we do $n-r$ controlled phase gates in total and each of them required two additional layers for rotation gates as shown in Figure 13. Moreover, we add one H gate. So, the depth of the r-th cascade is $2(n-r)+1$. The total depth is $c+\sum_{r=1}^{n}(2(n-r)+1) = c+n^2$, where c is the CNOT cost of the produced circuit.*

6. Conclusion

We present a generic method for constructing quantum circuits for quantum fingerprinting (quantum hashing) and quantum Fourier transform algorithms for an arbitrary connected qubit connectivity graph for hardware. The heuristic version of the method is fast enough and works with $O(n^6)$ time complexity. The certain version has an exponential time complexity that is $O(n^2 2^n)$. The algorithm works for an arbitrary qubit connectivity graph.

At the same time, if we consider samples of graphs like LNN, "sun" (16-qubit IBMQ Eagle r3 architecture), "two joint suns" (27-qubit IBMQ Eagle r3 architecture), then our generic algorithm gives us the same circuit as the techniques provided especially for these graphs [23,79] in the case of quantum hashing. In the case of QFT, our algorithm gives a bit worse circuit compared to the techniques optimized for these graphs [14,23]. At the same time, our approach works for arbitrary connected graphs, but the existing results work only for some specific graphs. Additionally, we consider other different architectures like IBMQ Melbourne, Rigetti Aspen-4, and a 5×5-grid as a connectivity graph. We show that for all of the considered connectivity graphs, including LNN, "sun," and "two joint suns," our algorithm produces circuit better than standard Qiskit transpiler optimization.

An open question is to develop a technique for QFT for an arbitrary connected graph that gives us the same or better results than the existing ones for specific architectures like LNN, "sun" and "two joint suns."

Author Contributions

K.K., A.K., Z.C. wrote the paper, designed the algorithms, main idea. J.W. supervision, editing, notes on the algorithms. K.K. wrote implementation of algorithms. All authors have read and agreed to the published version of the manuscript.

Funding

This work is supported by the National Natural Science Foundation of China under Grant No. 61877054, 12031004,12271474, and the Foreign Experts in Culture and Education Foundation under Grant No. DL2022147005L. Kamil Khadiev thanks these projects for supporting his visit. The research by K.K. and A.K. in Section 5 is supported by Russian Science Foundation Grant 25-11-00366, https://rscf.ru/en/project/25-11-00366/. The study by K.K. in Section 4 was performed under the development program of the Volga Region Mathematical Center (agreement no. 075-02-2025-1725/1).

Conflicts of Interest Statement

The authors declare no conflicts of interest.

Data Availability Statement
No new data created.

Acknowledgments
We thank Ramis Yamilov for helpful discussions.

Appendix A. Implementation of the Procedure ShortestPathes for Shortest Paths Searching

Here we discuss how to construct matrices W and A such that $W[v,u]$ is the length of the shortest path between vertices v and u, and $A[v,u]$ is the last vertex in the shortest path between v and u. The procedures are simple, but we present them for the completeness of the results representation.

Firstly, we present a procedure SingleSrcShortestPath(v) that finds the shortest paths for a single source vertex v that is based on the BFS algorithm [87]. The algorithm calculates the v-th rows of W and A. The implementation is presented in Algorithm A1. Here we assume that we have a queue data structure [87] that allows us to do the next actions in constant time:

- Add$(queue, v)$ adds an element to the queue;
- Remove$(queue)$ removes an element from the queue and returns the element;
- Init$()$ returns an empty queue;
- isEmpty$(queue)$ returns *True* if the queue is empty and *False* otherwise.

Algorithm A1 Implementation of SingleSrcShortestPath(v)

```
queue ← Init()
Add(queue, v)
for u ∈ V do
    W[v,u] ← ∞
    A[v,u] ← NULL
end for
W[v,v] ← 0
while isEmpty(queue) = False do
    t ← Remove(queue)
    for r ∈ Neighbors(t) do
        if W[v,r] = ∞ then
            A[v,r] ← t
            W[v,r] = W[v,t] + 1
            Add(queue, r)
        end if
    end for
end while
```

As an implementation of the ShortestPaths procedure, we invoke SingleSrcShortestPath(v) for each vertex $v \in V$.

Algorithm A2 Implementation of ShortestPaths(G) for a $G = (V, E)$ graph

```
for v ∈ V do
    SingleSrcShortestPath(v)
end for
return (W, A)
```

Lemma A1. *The time complexity of the ShortestPathes procedure is* $O(n^3)$.

Proof. Time complexity of BFS is $O(n+m) = O(n^2)$ due to [87]. Invocation of n BFS algorithms for each $v \in V$ is $O(n^3)$. □

Appendix B. Quantum Fingerprinting or Quantum Hashing

Let us present some basic concepts of the quantum fingerprinting technique from [26–29,106]. This technique is well known and allows us to compute a short hash or fingerprint that identifies the original large data with high probability.

For the problem to be solved, we choose a positive integer m, an error probability bound $\varepsilon > 0$, fix $t = \lceil (2/\varepsilon) \ln 2m \rceil$, and construct a mapping $g : \{0,1\}^n \to \mathbb{Z}$. Then for arbitrary binary string $\sigma = (\sigma_1 \dots \sigma_n)$ we create it's fingerprint $|h_\sigma\rangle$ composing t single qubit fingerprints $|h_\sigma^i\rangle$:

$$|h_\sigma^i\rangle = cos\frac{2\pi k_i g(\sigma)}{m}|0\rangle + sin\frac{2\pi k_i g(\sigma)}{m}|0\rangle,$$

$$|h_\sigma\rangle = \frac{1}{\sqrt{t}}\sum_{i=1}^{t}|i\rangle|h_\sigma^i\rangle$$

Here, the last qubit is rotated by t different angles about the $\hat{y}$ axis of the Bloch sphere. The chosen parameters $k_i \in \{1\dots,m-1\}$, for $i \in \{1\dots t\}$ are "good" in the following sense. A set of parameters $K = \{k_1,\dots,k_t\}$ is called "good" for $g \neq 0 \mod m$ if

$$\frac{1}{t^2}\left(\sum_{i=1}^{t} cos\frac{2\pi k_i g}{m}\right)^2 < \varepsilon$$

The left side of the inequality is the squared amplitude of the basis state $|0\rangle^{\otimes \log_2 t}|0\rangle$ if the operator $H^{\otimes \log_2 t} \otimes I$ has been applied to the fingerprint $|h_\sigma\rangle$. Informally, this kind of set guarantees that the probability of error will be bounded by a constant below 1.

The following lemma from [27,28,106] proves the existence of a "good" set.

Lemma A2 ([106]). *There is a set K with $|K| = t = \lceil (2/\varepsilon) \ln 2m \rceil$ that is "good" for all $g \neq 0 \mod m$.*

We use this result for the fingerprinting technique [106] choosing the set $K = \{k_1,\dots,k_t\}$ that is "good" for all $g = g(\sigma) \neq 0$. It allows us to distinguish those inputs whose image is 0 modulo m from the others.

That hints at how this technique may be applied:

1. We construct $g(x)$, which maps all acceptable inputs to 0 modulo m and others to arbitrary non-zero (modulo m) integers.
2. After the necessary manipulations with the fingerprint, the $H^{\otimes \log_2 t}$ operator is applied to the first $\log_2 t$ qubits. This operation "collects" all cosine amplitudes at the all-zero state. That is, we obtain the state of the type
$$|h'_\sigma\rangle = \frac{1}{t}\sum_{i=1}^{t} cos\left(\frac{2\pi k_i g(\sigma)}{m}\right)|00\dots0\rangle|0\rangle + \sum_{i=2}^{2t}\alpha_i|i\rangle$$
3. This state is measured on the standard computational basis. Then we accept the input if the outcome is the all-zero state. This happens with the probability
$$Pr_{accept}(\sigma) = \frac{1}{t^2}\left(\sum_{i=1}^{t} cos\frac{2\pi k_i g(\sigma)}{m}\right)^2,$$
that is 1 for the inputs, whose image is $0 \mod m$ and is bounded by ε for the others.

Due to [81,82], the algorithm can be implemented using the shallow circuit presented in Figure 1. In that case, the angles $\frac{2\pi k_i}{m}$ should be linear combinations of ξ_j. Due to [82], it is not known whether we can keep the same number of qubits or should we increase the number of qubits exponentially. At the same time, computational experiments show [82] that we can find enough good parameters ξ_i such that t qubits are enough for ε error probability.

Appendix C. Quantum Fourier Transform

QFT is a quantum version of the discrete Fourier transform. The definition of n-qubit QFT and its inverse are as follows:

$$QFT|j\rangle = \sum_{k=0}^{2^n-1} e^{\frac{2\pi i j k}{2^n}}|k\rangle,$$

$$QFT^{-1}|j\rangle = \sum_{k=0}^{2^n-1} e^{-\frac{-2\pi ijk}{2^n}}|k\rangle,$$

The n-qubit QFT circuit requires $0.5n^2 - 0.5n$ controlled phase (CR_d) gates and n Hadamard (H) gates if we have no restriction on the application of two-qubit gates (See Figure 12). The CR_d gate is represented by basic gates that require two CNOT and three R_z gates [105]. Therefore, $n^2 - n$ CNOT gates are required to construct an n-qubit QFT circuit. At the same time, if a quantum device has the LNN architecture, then for implementing the QFT, the number of CNOT gates is much larger than $n^2 - n$ [14,16–19,107]. If we consider a general graph, then the situation is much worse [23].

Appendix D. Constructing the List of First Indices

For a path $P = (v_{i_1}, \ldots, v_{i_k})$ we construct a list $(j_1, \ldots, j_k)$ such that $j_r \notin \{i_1, \ldots i_{j_r-1}\}$. We use a set U' that is implemented using the Red-Black tree [87]. Let J be the results list.

Initially, $U' \leftarrow \varnothing$ is an empty set, and $J \leftarrow ()$ is an empty list. For each h from 1 to k we check whether $i_h \in U'$. If $i_h \notin U'$, then we add i_h to J, otherwise, we skip it.

Algorithm A3 Constructing the list $J = (j_1, \ldots, j_n)$ by the path P

$U' \leftarrow \varnothing$
$J \leftarrow ()$
for $h \in \{1, \ldots, k\}$ **do**
  if $i_h \notin U'$ **then**
    $U' \leftarrow U' \cup \{i_h\}$
    $J \leftarrow J \circ i_h$
  end if
end for
return J

Appendix E. Graphs for Each Cascade for LNN Architecture

The path for the 5-qubit LNN architecture is presented in Figure A1. It is used for the first cascade, the circuit of which is presented in Figure A2.

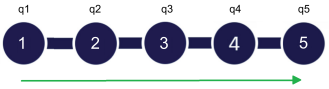

Figure A1. The graph for the 5-qubit LNN architecture. The path that visits all vertices at least once is green. $q1, \ldots, q5$ are assigned logical qubits for the vertices.

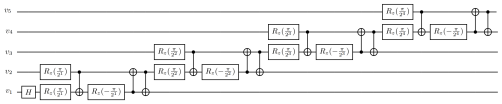

Figure A2. The circuit for the first cascade.

The path for the 2nd cascade is shown in Figure A3, and the corresponding circuit is shown in Figure A4.

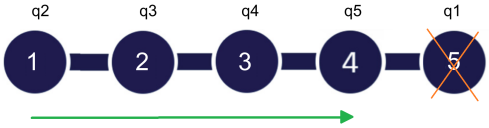

Figure A3. The graph for the 5-qubit LNN architecture. The second cascade excludes the vertex that corresponds to 1-st logical qubit. The path that visits all vertices at least once is green.

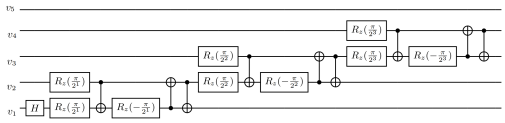

Figure A4. The circuit for the second cascade.

The paths for the 3rd and 4th cascades are presented in Figures A5 and A6. The corresponding circuit including the last 5th cascade is shown in Figure A7.

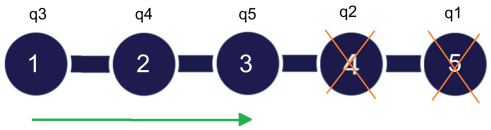

Figure A5. The graph for the 5-qubit LNN architecture. The 3rd cascade excludes the vertices that correspond to 1st and 2nd logical qubits. The path that visits all vertices at least once is green.

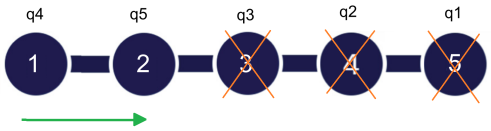

Figure A6. The graph for the 5-qubit LNN architecture. The 4th cascade excludes the vertices that correspond to 1st, 2nd, and 3rd logical qubits. The path that visits all vertices at least once is green.

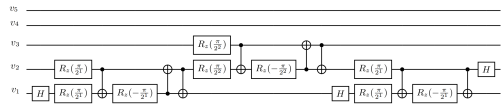

Figure A7. The circuit for the 3rd, 4th, and 5th cascades.

Appendix F. Graphs for Each Cascade for the 16-qubit "sun" Architecture

The paths for each of the first 15 cascades are in the following Figures A8–A23.

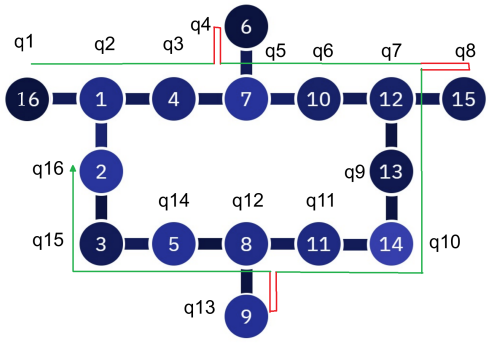

Figure A8. The graph for the 16-qubit "sun" architecture. The path that visits all vertices at least once is green. Red parts are such that $v_{i_j} = v_{i_{j+2}}$. The labels $q1, \ldots, q16$ are names of assigned logical qubits for the vertices.

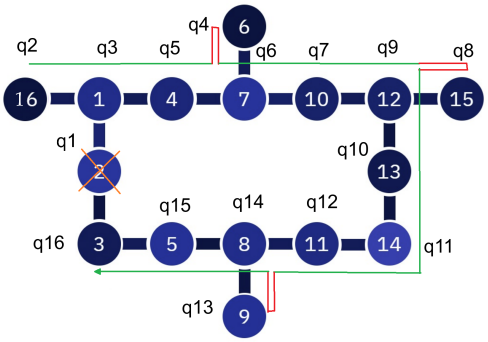

Figure A9. The graph for the 16-qubit "sun" architecture. The second cascade excludes the vertex that corresponds to the 1-st logical qubit. The path that visits all vertices at least once is green. Red parts are such that $v_{i_j} = v_{i_{j+2}}$.

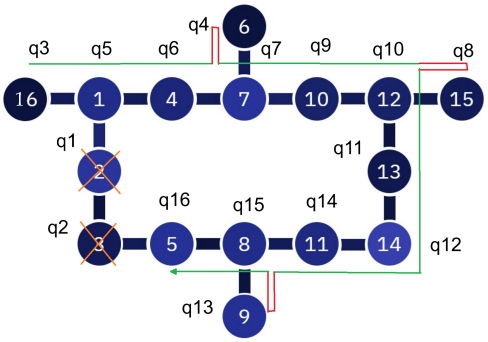

Figure A10. The graph for the 16-qubit "sun" architecture. The 3rd cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1, 2\}$ set.

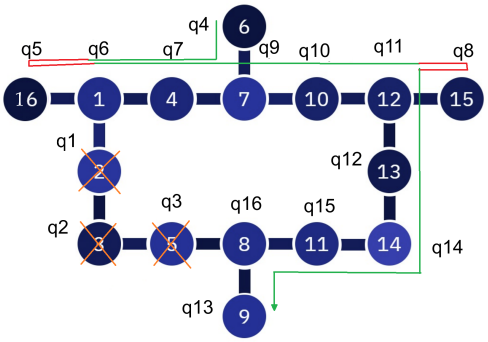

Figure A11. The graph for the 16-qubit "sun" architecture. The 4th cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1, 2, 3\}$ set.

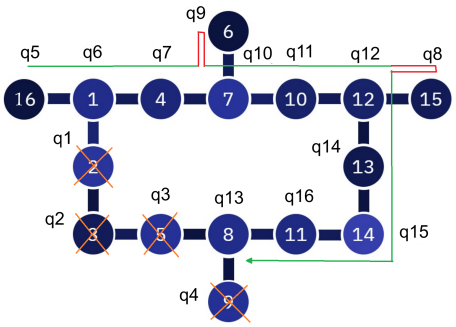

Figure A12. The graph for the 16-qubit "sun" architecture. The 5th cascade that excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 4\}$ set.

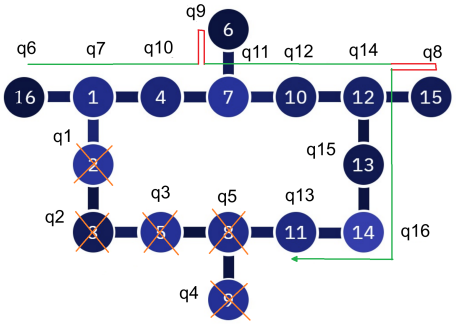

Figure A13. The graph for the 16-qubit "sun" architecture. The 6th cascade that excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 5\}$ set.

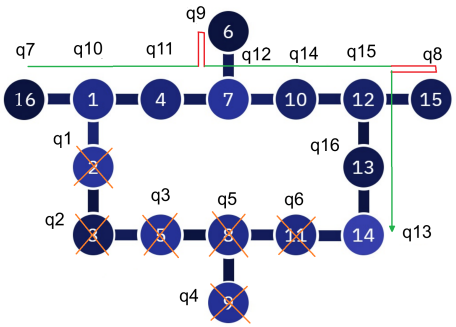

Figure A14. The graph for the 16-qubit "sun" architecture. The 7th cascade that excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 6\}$ set.

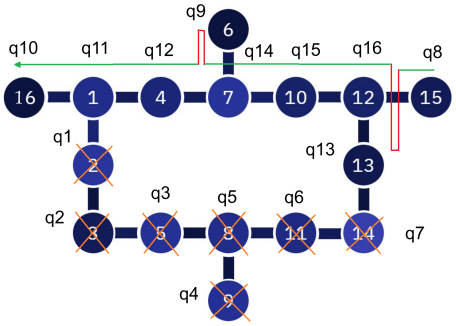

Figure A15. The graph for the 16-qubit "sun" architecture. The 8th cascade that excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 7\}$ set.

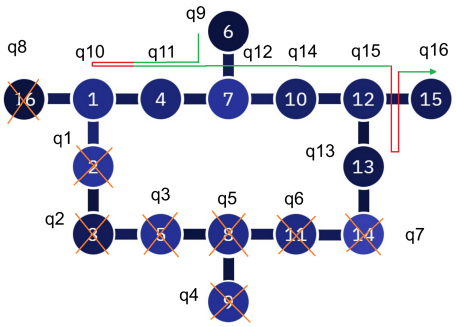

Figure A16. The graph for the 16-qubit "sun" architecture. The 9th cascade that excludes the vertex that corresponds to the logical qubits with indices from the $\{1,\ldots,8\}$ set.

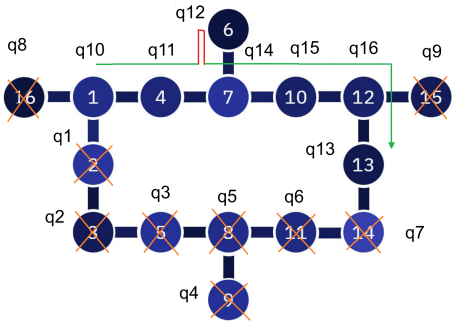

Figure A17. The graph for the 16-qubit "sun" architecture. The 10th cascade that excludes the vertex that corresponds to the logical qubits with indices from the $\{1,\ldots,9\}$ set.

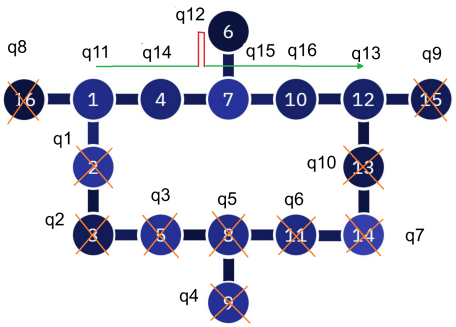

Figure A18. The graph for the 16-qubit "sun" architecture. The 11th cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1,\ldots,10\}$ set.

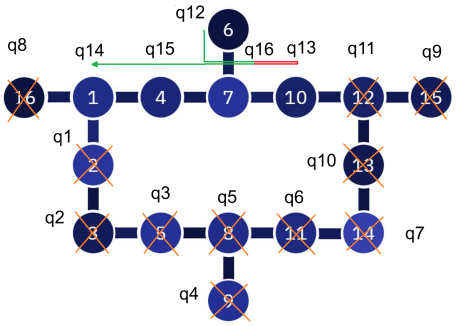

Figure A19. The graph for the 16-qubit "sun" architecture. The 12th cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1,\ldots,11\}$ set.

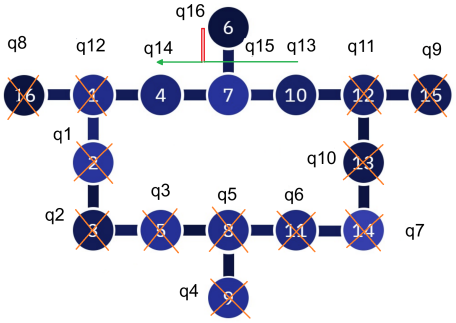

Figure A20. The graph for the 16-qubit "sun" architecture. The 13th cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 12\}$ set.

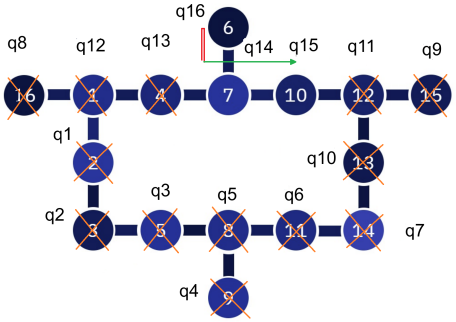

Figure A21. The graph for the 16-qubit "sun" architecture. The 14th cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 13\}$ set.

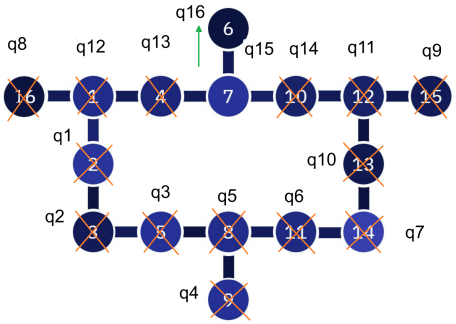

Figure A22. The graph for the 16-qubit "sun" architecture. The 15th cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 14\}$ set.

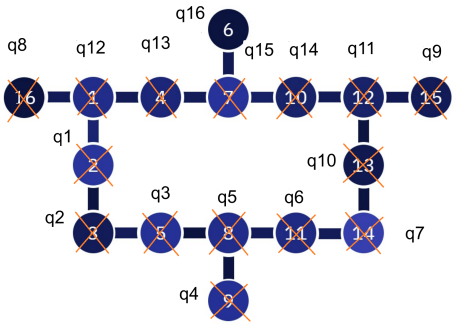

Figure A23. The graph for the 16-qubit "sun" architecture. The 16th cascade excludes the vertex that corresponds to the logical qubits with indices from the $\{1, \ldots, 15\}$ set.

References

1. A Yu Kitaev (1995). "Quantum measurements and the abelian stabilizer problem". *arXiv preprint quant-ph/9511026*.
2. T. G. Draper (2000). "Addition on a quantum computer." *arXiv preprint quant-ph/0008033*.
3. G. Brassard, P. Høyer, M. Mosca, and A. Tapp (2022). "Quantum amplitude amplification and estimation." *Contemporary Mathematics*, 305, 53–74.
4. A. W. Harrow, Avinatan Hassidim, and Seth Lloyd (2009). "Quantum algorithm for linear systems of equations." *Physical Review Letters*, 103: 15, 150502.
5. P. W Shor (1999). "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer." *SIAM Review*, 41: 2, 303–332.
6. L. Wright, C. Mc Keever, J. T. First, R. Johnston, J. Tillay, S. Chaney, M. Rosenkranz, and M. Lubasch (2024). "Noisy intermediate-scale quantum simulation of the one-dimensional wave equation." *Physical Review Research*, 6, 043169.
7. T. Lee, R. Mittal, B. W. Reichardt, R. Špalek, and M. Szegedy (2011). "Quantum query complexity of state conversion." In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pp. 344–353. IEEE.
8. A. Belovs (2013). "Quantum walks and electric networks." *arXiv preprint arXiv:1302.3143*.
9. A. Belovs, A. M. Childs, S. Jeffery, R. Kothari, and F. Magniez (2013). "Time-efficient quantum walks for 3-distinctness". In *International Colloquium on Automata, Languages, and Programming*, pp. 105–122. Springer.
10. A. Montanaro (2020). "Quantum speedup of branch-and-bound algorithms." *Physical Review Research*, 2: 1, 013056.
11. A. Montanaro (2018). "Quantum-walk speedup of backtracking algorithms." *Theory of Computing*, 14: 15, 1–24.
12. R. Cleve and J. Watrous (2000). "Fast parallel circuits for the quantum Fourier transform." In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 526–536. IEEE.
13. Y. Nam, Y. Su, and D. Maslov (2020). "Approximate quantum Fourier transform with o (n log (n)) t gates." *NPJ Quantum Information*, 6: 1, 26.
14. B. Park and D. Ahn (2023). "Reducing cnot count in quantum Fourier transform for the linear nearestneighbor architecture." *Scientific Reports*, 13: 1, 8638.
15. M. A Nielsen and I. L Chuang (2010). *Quantum computation and quantum information*. Cambridge univ. Press.
16. A.G. Fowler, S.J. Devitt, and L.C.L. Hollenberg (2004). „Implementation of Shor's algorithm on a linear nearest neighbour qubit array." *Quantum Information & Computation*, 4: 4, 237–251.
17. M. Saeedi, R. Wille, and R. Drechsler (2011). "Synthesis of quantum circuits for linear nearest neighbor architectures." *Quantum Information Processing*, 10, 355–377.
18. A. Kole, K. Datta, and I. Sengupta (2017). "A new heuristic for *n*-dimensional nearest neighbor realization of a quantum circuit." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37: 1, 182–192.
19. A. Bhattacharjee, C. Bandyopadhyay, R. Wille, R. Drechsler, and H. Rahaman (2019). "Improved look-ahead approaches for nearest neighbor synthesis of 1d quantum circuits." In *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, pp. 203–208. IEEE.
20. A. Barenco, A. Ekert, K.-A. Suominen, and P. Törmä (1996). "Approximate quantum Fourier transform and decoherence." *Physical Review A*, 54: 1, 139.
21. Y. Takahashi, N. Kunihiro, and K. Ohta (2007). "The quantum Fourier transform on a linear nearest neighbor architecture." *Quantum Information & Computation*, 7: 4, 383–391.
22. B. Park and D. Ahn (2022). "T-count optimization of approximate quantum Fourier transform." *arXiv preprint arXiv:2203.07739*.
23. A. Khadieva (2024). "Quantum hashing algorithm implementation.: *arXiv preprint*. arXiv:quant-ph/2024.
24. F. Ablayev and A. Vasiliev (2020). "Quantum hashing and Fourier transform." In *Journal of Physics: Conference Series*, volume 1680, p. 012001. IOP Publishing.
25. R. Freivalds (1979). "Fast probabilistic algorithms." In *Mathematical Foundations of Computer Science 1979*, vol. 74 of *LNCS*, pp. 57–69.
26. A. Ambainis and R. Freivalds (1998). "1-way quantum finite automata: strengths, weaknesses and generalizations." In *FOCS'98*, pp. 332–341. IEEE.

27. A. Ambainis and N. Nahimovs (2008). “Improved constructions of quantum automata.” In *TQC*, volume 5106 of *LNCS*, pp. 47–56. Springer.
28. A. Ambainis and N. Nahimovs (2009). ”Improved constructions of quantum automata.” *Theoretical Computer Science*, 410: 20, 1916–1922.
29. H. Buhrman, R. Cleve, J. Watrous, and R. De Wolf (2001). “Quantum fingerprinting.” *Physical Review Letters*, 87: 16, 167902.
30. F. Ablayev, A. Gainutdinova, M. Karpinski, C. Moore, and C. Pollett (2005). “On the computational power of probabilistic and quantum branching program.” *Information and Computation*, 203: 2, 145–162.
31. F. Ablayev and A. Gainutdinova (2005). “Complexity of quantum uniform and nonuniform automata.” In *Developments in Language Theory*, vol. 3572 of *LNCS*, pp. 78–87. Springer.
32. F. Ablayev and A. Vasiliev (2009). “On quantum realisation of boolean functions by the fingerprinting technique.” *Discrete Mathematics and Applications*, 19: 6, 555–572.
33. F. Ablayev and A. Vasiliev (2011). “Classical and quantum parallelism in the quantum fingerprinting method.” In *International Conference on Parallel Computing Technologies*, pp. 1–12. Springer.
34. F. M. Ablayev and A. Vasiliev (2013). “Algorithms for quantum branching programs based on fingerprinting.” *International Journal of Software and Informatics*, 7: 4, 485–500.
35. F.M. Ablayev and A.V. Vasiliev (2013). “Cryptographic quantum hashing.“ *Laser Physics Letters*, 11: 2, 025202.
36. F. Ablayev and A. Vasiliev (2014). “Computing Boolean functions via quantum hashing.” In *Computing with New Resources: Essays Dedicated to Jozef Gruska on the Occasion of His 80th Birthday*, pp. 149–160.
37. F.M. Ablayev, M.F. Ablayev, and A.V. Vasilev (2014). “Universal quantum hashing.” *Uchenye Zapiski Kazanskogo Universiteta. Seriya Fiziko-Matematicheskie Nauki*, 156, 7–18.
38. F. Ablayev and M. Ablayev (2015). “On the concept of cryptographic quantum hashing.” *Laser Physics Letters*, 12: 12, 125204.
39. F. Ablayev, M. Ablayev, and A. Vasiliev (2016). “On the balanced quantum hashing.” *Journal of Physics: Conference Series*, 681, 012019.
40. F. Ablayev, M. Ablayev, A. Vasiliev, and M. Ziatdinov (2016). “Quantum fingerprinting and quantum hashing. computational and cryptographical aspects.” *Baltic Journal of Modern Computing*, 4: 4, 860.
41. A. Vasiliev (2016). “Binary quantum hashing.” *Russian Mathematics*, 60: 9, 61–65.
42. A Vasiliev, M Latypov, and M Ziatdinov (2017). “Minimizing collisions for quantum hashing.” *Journal of Engineering and Applied Sciences*, 12: 4, 877–880.
43. F. Ablayev, M. Ablayev, K. Khadiev, and A. Vasiliev (2018). “Classical and quantum computations with restricted memory.” *LNCS*, 11011, 129–155.
44. F. Ablayev, M. Ablayev, and A. Vasiliev (2018). “Computing quantum hashing in the model of quantum branching programs.” *AIP Conference Proceedings*, 1936, 020020.
45. A.V. Vasiliev, A.R. Vasilov, and M.A. Latypov (2019). “Analysis of properties of quantum hashing.” *Journal of Mathematical Sciences*, 241: 2, 117–124.
46. F. Ablayev, M. Ablayev, and A. Vasiliev. (2020). “Quantum hashing and fingerprinting for quantum cryptography and computations.” *International Computer Science Symposium in Russia*, pp. 1–15. Springer.
47. A. Vasiliev (2016). “Quantum hashing for finite Abelian groups.” *Lobachevskii Journal of Mathematics*, 37: 6, 753–757.
48. M. Ziatdinov (2016). “From graphs to keyed quantum hash functions.” *Lobachevskii Journal of Mathematics*, 37: 6, 705–712.
49. M Ziatdinov (2016). “Quantum hashing group approach.” *Lobachevskii Journal of Mathematics*, 37: 2, 222–226.
50. I. Zinnatullin (2023). “Cryptographic properties of the quantum hashing based on expander graphs.” *Lobachevskii Journal of Mathematics*, 44: 2, 776–787.
51. F. Ablayev and A. Vasiliev (2022). “Quantum hashing on the high-dimensional states.” In *International Conference on Micro-and Nano-Electronics 2021*, vol. 12157, pp. 565–570. SPIE.
52. F. Le Gall (2009). “Exponential separation of quantum and classical online space complexity.” *Theory of Computing Systems*, 45: 2, 188–202.
53. F. Le Gall (2006). “Exponential separation of quantum and classical online space complexity.” In *SPAA ’06*, pp. 67–73. ACM.

54. F. Ablayev, M. Ablayev, K. Khadiev, N. Salihova, and A. Vasiliev (2022). “Quantum algorithms for string processing.” In *Mesh Methods for Boundary-Value Problems and Applications*, vol.141 of *LNCSE*.
55. F. Ablayev, N. Salikhova, and M. Ablayev (2024). “Hybrid classical–quantum text search based on hashing.” *Mathematics*, 12: 12, 1858.
56. K. Khadiev and A. Khadieva (2021). “Quantum online streaming algorithms with logarithmic memory.” *International Journal of Theoretical Physics*, 60, 608–616.
57. Kamil Khadiev and Aliya Khadieva (2022). “Quantum and classical log-bounded automata for the online disjointness problem.” *Mathematics*, 10, 1.
58. K. Khadiev, A. Khadieva, and I. Mannapov (2018). “Quantum online algorithms with respect to space and advice complexity.” *Lobachevskii Journal of Mathematics*, 39: 9, 1210–1220.
59. K. Khadiev, A. Khadieva, M. Ziatdinov, I. Mannapov, D. Kravchenko, A. Rivosh, and R. Yamilov (2022). “Two-way and one-way quantum and classical automata with advice for online minimization problems.” *Theoretical Computer Science*, 920, 76–94.
60. A. Khadiev, K.and Khadieva, D. Kravchenko, I. Mannapov, A. Rivosh, and R. Yamilov (2023). “Quantum versus classical online streaming algorithms with logarithmic size of memory.” *Lobachevskii Journal of Mathematics*, 44: 2, 687–698.
61. K. Khadiev and A. Khadieva (2020). “Two-way quantum and classical automata with advice for online minimization problems.” In *Formal Methods. FM 2019 International Workshops*, pp. 428–442.
62. K. Khadiev and A. Khadieva (2019). “Two-way quantum and classical machines with small memory for online minimization problems.” In *International Conference on Micro- and Nano-Electronics 2018*, vol. 11022 of *Proc. SPIE*, p. 110222T.
63. K. Khadiev and A. Khadieva (2017). “Reordering method and hierarchies for quantum and classical ordered binary decision diagrams.” In *CSR 2017*, vol. 10304 of *LNCS*, pp. 162–175. Springer.
64. K. Khadiev, A. Khadieva, and A. Knop. “Exponential separation between quantum and classical ordered binary decision diagrams, reordering method and hierarchies.” *Natural Computing*, 22: 4, 723–736.
65. F. Ablayev, A. Gainutdinova, K. Khadiev, and A. Yakaryılmaz (2016). “Very narrow quantum OBDDs and width hierarchies for classical OBDDs.” *Lobachevskii Journal of Mathematics*, 37: 6, 670–682.
66. A Vasiliev (2016). “A model of quantum communication device for quantum hashing.” In *Journal of Physics: Conference Series*, vol. 681, p. 012020. IOP Publishing.
67. A. Gainutdinova and A. Yakaryılmaz (2017). “Nondeterministic unitary obdds.” In P. Weil (ed.), *Computer Science - Theory and Applications - 12th International Computer Science Symposium in Russia, CSR 2017, Kazan, Russia, June 8-12, 2017, Proceedings*, vol. 10304 of *Lecture Notes in Computer Science*, pp. 126–140. Springer.
68. A. Yakaryılmaz and A. C. Cem Say (2010). “Languages recognized by nondeterministic quantum finite automata.” *Quantum Information and Computation*, 10: 9&10, 747–770.
69. A. Gainutdinova and A. Yakaryılmaz (2018). “Unary probabilistic and quantum automata on promise problems.” *Quantum Information Processing*, 17: 2, 28.
70. A. Gainutdinova and A. Yakaryılmaz (2015). “Unary probabilistic and quantum automata on promise problems.” In *Developments in Language Theory*, pp. 252–263. Springer.
71. A. Khadieva and M. Ziatdinov (2023). “Deterministic construction of QFAS based on the quantum fingerprinting technique.” *Lobachevskii Journal of Mathematics*, 44: 2, 713–723.
72. D.O. Akatev, A.V. Vasiliev, N.M. Shafeev, F.M. Ablayev, and A.A. Kalachev (2022). “Multiqudit quantum hashing and its implementation based on orbital angular momentum encoding.” *Laser Physics Letters*, 19: 12, 125205.
73. D.A. Turaykhanov, D.O. Akatev, A.V. Vasiliev, F.M. Ablayev, and A.A. Kalachev (2021). “Quantum hashing via single-photon states with orbital angular momentum.” *Physical Review A*, 104: 5, 052606.
74. S. Z. D. Plachta, M. Hiekkamäki, A. Yakaryılmaz, and R. Fickler (2022). “Quantum advantage using high-dimensional twisted photons as quantum finite automata.” *Quantum*, 6, 752.
75. Y.-Y. Zhao, K. Li, C. Li, S. Zheng, and Z. He (2023). “Experimental demonstration advantage of photonic finite automata.” In *2023 Asia Communications and Photonics Conference/2023 International Photonics and Optoelectronics Meetings (ACP/POEM)*, pp. 1–3. IEEE.
76. M. Möttönen and J. J. Vartiainen (2006). „Decompositions of general quantum gates.” *Trends in Quantum Computing Research*, pp. 149. arXiv preprint quant-ph/0504100.

77. V. Bergholm, J. J. Vartiainen, M. Möttönen, and M. M. Salomaa (2005). “Quantum circuits with uniformly controlled one-qubit gates.” *Physical Review A—Atomic, Molecular, and Optical Physics*, 71: 5, 052330.
78. I. Zinnatullin, K. Khadiev, and A. Khadieva (2023). “Efficient implementation of amplitude form of quantum hashing using state-of-the-art quantum processors.” *Russian Microelectronics*, 52: Suppl 1, S390–S394.
79. A. Khadieva, O Salehi, and A. Yakaryı lmaz (2024). “A representative framework for implementing quantum finite automata on real devices.” In *Proceedings of UCNC 2024*, vol. 14776 of *LNCS*, pp. 163–177.
80. I. Zinnatullin and K. Khadiev (2025). “Efficient algorithms for quantum hashing.” In *Proceedings of UCNC 2025*, LNCS. arXiv preprint arXiv:2507.07002.
81. M. Kālis (2018). “Kvantu algoritmu realizâcija fiziskâ kvantu datorâ.” Master’s thesis, University of Latvia.
82. M. Ziiatdinov and A. Khadieva and A. Yakaryılmaz (2023). “Gaps for shallow implementation of quantum finite automata.” In *Proceedings of AFL 2023*, vol. 386 of *Electronic Proceedings in Theoretical Computer Science*, pp. 269–280. Open Publishing Association.
83. M. Ziiatdinov, A. Khadieva, and K. Khadiev(2025). “Shallow implementation of quantum fingerprinting with application to quantum finite automata.” *Frontiers in Computer Science*, 7, 2025.
84. A Vasiliev (2023). “Constant-depth algorithm for quantum hashing.” *Russian Microelectronics*, 52: Suppl 1, S399–S402.
85. F. Ablayev, M. Ablayev, J. Z. Huang, K. Khadiev, N. Salikhova, and D. Wu (2019). “On quantum methods for machine learning problems Part I: Quantum tools.” *Big Data Mining and Analytics*, 3: 1, 41–55.
86. K. Khadiev (2022). “Lecture notes on quantum algorithms.” *arXiv preprint arXiv:2212.14205*.
87. T. H Cormen, C. E Leiserson, R. L Rivest, and C. Stein (2001). *Introduction to Algorithms*. McGraw-Hill.
88. R. Bellman (1962). “Dynamic programming treatment of the travelling salesman problem.” *Journal of the ACM (JACM)*, 9: 1, 61–63.
89. M. Held and R. M. Karp (1962). “A dynamic programming approach to sequencing problems.” *Journal of the Society for Industrial and Applied Mathematics*, 10: 1, 196–210.
90. A. Ambainis, K. Balodis, J. Iraids, M. Kokainis, K. Prûsis, and J. Vihrovs (2019). “Quantum speedups for exponential-time dynamic programming algorithms.” In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 1783–1793. SIAM.
91. C. Y.-Y. Lin and H.-H. Lin (2015). “Upper bounds on quantum query complexity inspired by the Elitzur-Vaidman bomb tester.” In *30th Conference on Computational Complexity (CCC 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
92. C. Y.-Y. Lin and H.-H. Lin (2016). „Upper bounds on quantum query complexity inspired by the Elitzur–Vaidman bomb tester.” *Theory of Computing*, 12: 18, 1–35.
93. C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla (2006). “Quantum query complexity of some graph problems.” *SIAM Journal on Computing*, 35: 6, 1310–1328.
94. S. Arunachalam and R. de Wolf (2017). “Optimizing the number of gates in quantum search.” *Quantum Information and Computation*, 17: 3&4, 251–261.
95. L. K. Grover (2002). “Trade-offs in the quantum search algorithm.” *Physical Review A*, 66: 5, 052314.
96. K. Khadiev, C. M. Bosch-Machado, Z. Chen, and J. Wu (2024). “Quantum algorithms for the shortest common superstring and text assembling problems.” *Quantum Information and Computation*, 24: 3–4, 267–294.
97. J. Monnot, V. Th Paschos, and S. Toulouse (2003). “Approximation algorithms for the traveling salesman problem.” *Mathematical Methods of Operations Research*, 56, 387–405.
98. V. Th Paschos (2016). “An overview on polynomial approximation of np-hard problems.” *Yugoslav Journal of Operations Research*, 19: 1.
99. S. Hougardy, F. Zaiser, and X. Zhong (2020). “The approximation ratio of the 2-opt heuristic for the metric traveling salesman problem.” *Operations Research Letters*, 48: 4, 401–404.
100. J. J. Bentley (1992). “Fast algorithms for geometric traveling salesman problems.” *ORSA Journal on Computing*, 4: 4, 387–411.
101. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi (2012). *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer Science & Business Media.
102. W. Sun, X. Li, L. Yu, Z. Wang, G. Chen, and G. Yang (2025). “Mlqm: Machine learning approach for accelerating optimal qubit mapping.” *Future Generation Computer Systems*, 107906.
103. IBMQ. Transpiler.

104. K. Khadiev (2025). “Git lab reposetory of the algorithms”. https://gitlab.com/kamilbek/qhqftoptimizer.
105. A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter (1995). “Elementary gates for quantum computation.” *Physical Review A*, 52: 5, 3457.
106. F. Ablayev, A. Khasianov, and A. Vasiliev (2010). “On complexity of quantum branching programs computing equality like Boolean functions.” *ECCC*.
107. R. Wille, A. Lye, and R. Drechsler (2014). “Exact reordering of circuit lines for nearest neighbor quantum architectures.” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33: 12, 1818–1831.