# Qubit Routing via SWAP Network: a Minimum Dominating Path Approach*

Riccardo Romanello[1 [0000−0002−2855−1221]],
Daniele Lizzio Bosco[1,2 [0009−0002−7372−6518]],
Giuseppe Serra[1 [0000−0002−4269−4501]], and
Carla Piazza[1 [0000−0002−2072−1628]]

[1] Department of Mathematics, Computer Science and Physics, University of Udine, Udine, Italy
[2] Department of Biology, University of Naples Federico II, Naples, Italy
{riccardo.romanello, giuseppe.serra, carla.piazza}@uniud.it,
lizziobosco.daniele@spes.uniud.it

**Abstract.** Efficient execution of quantum circuits on noisy intermediate-scale quantum (NISQ) devices requires adapting logical circuits to hardware with limited qubit connectivity. SWAP networks provide a circuit-agnostic solution by prescribing SWAP schedules that guarantee adjacency between every pair of qubits, thereby decoupling routing from circuit-level structure. However, constructing short and scalable SWAP networks for arbitrary hardware topologies remains a challenging combinatorial problem. This work introduces a fast, deterministic pipeline for generating high-quality SWAP networks on general connectivity graphs. The method is built around minimum Dominating Paths, which serve as compact structural backbones from which the full SWAP schedule is derived. The proposed four-stage procedure—path extraction, internal routing, external qubit integration, and final refinement—operates in the optimal number of SWAP gates at each step and avoids stochastic search entirely. We evaluate the approach on IBM quantum processor topologies ranging from 5 to 133 qubits. Across all architectures beyond 8 qubits, the method consistently outperforms the state-of-the-art simulated-annealing-based baseline, achieving up to 25% reductions in network length while providing strong interpretability and significantly lower runtime. These results demonstrate that minimum-path-based constructions offer an effective and scalable foundation for circuit-agnostic routing in the NISQ era.

**Keywords:** Swap network · Qubit Routing · Quantum Circuit Compilation · Graph Theory

* This work has been partially supported by INdAM-GNCS project *Algebra lineare quantistica, state preparation e compilazione di circuiti quantistici* (CUP E53C25002010001) and by the regional project QUASAR-FVG *Calcolo e simulazione quantistica: sviluppo, applicaizoni e ricerca in Friuli Venezia Giulia* (CUP G23C25001510002).

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

## 1 Introduction

Quantum computing has emerged as a promising paradigm for addressing computational tasks that are intractable for classical devices [1, 2]. As quantum devices continue to grow in scale and reliability, their practical use increasingly depends on the ability to execute quantum circuits efficiently on hardware with constrained qubit connectivity [3–5]. By *constrained connectivity*, we refer to the limitations imposed by the underlying physical architecture of superconducting quantum computers, in which physical qubits are arranged according to a specific topology, formally described as a *graph*, where the edges denote the set of permissible two-qubit interactions.

Achieving this compatibility requires quantum circuit compilation [6, 7], a process that adapts a circuit to the topology of a target quantum processor.

A central component of compilation is routing, whereby qubits that must interact are brought into proximity through sequences of SWAP gates, i.e. a quantum operation that exchange the value of two qubits. Traditional routing strategies operate in a circuit-dependent manner and typically rely on heuristic search or optimization procedures [4, 8, 9].

While effective, these approaches depend on the task under considerations: each circuit may require a different routing.

Recently, SWAP networks have been proposed as circuit-agnostic constructs that predefine a schedule of SWAP operations ensuring that every pair of qubits becomes adjacent at least once [10, 11]. By doing so, SWAP networks allow to consider any possible circuit connectivity, decoupling circuit-level optimization and topology constraints. In addition, SWAP networks are *re-usable*: once a SWAP network has been computed for a given hardware topology, it can be applied to any possible circuits with a compatible qubit count.

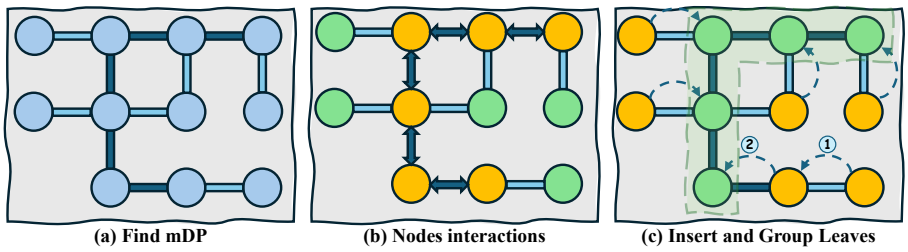


Fig. 1: Scheme of the steps of our proposed method. In (a), we compute a minimum Dominating Path (mDP), highlighted in dark blue). In (b), we first label qubits as nodes (in orange) and leaves (in green). We then guarantee nodes interaction with the Swap-and-Position procedure. In (c), we insert and group leaves into the mDP, and apply the SWAP procedure to conclude.

In current NISQ era [3], reducing the number of two-qubits gates is essential due to their high noise rate. Therefore, constructing short and efficient SWAP

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

networks is a key challenge for enabling high-quality quantum computation on contemporary hardware.

In this work, we introduce a fast and deterministic pipeline for generating efficient SWAP networks on arbitrary hardware topologies. The pipeline is built around the concept of a minimum Dominating Path [12] (mDP), which serves as the structural backbone for decomposing a given connectivity graph. The method proceeds in four stages: identifying a mDP, make all qubits/nodes in the mDP interact, integrating the external qubits in the mDP, and refining the resulting structure to include interactions between external qubits (see Figure 1). Each stage is addressed efficiently, deterministically, and in the optimal number of SWAP gates, ensuring a final SWAP network with a limited gate count.

We evaluate our approach on all hardware topologies provided by IBM [13] ranging from 5 to 133 qubits. Across all architectures with more than 8 qubits, our pipeline consistently generates better SWAP networks than those obtained with current state-of-the-art Simulated Annealing-based strategy [11], achieving more than 25% reduction in network length for larger systems compared to the state-of-the-art, heuristic baseline [11]. Moreover, our pipeline is faster, and provides a principled and interpretable construction mechanism.

Our results demonstrate that exploiting the structural regularities captured by minimum Dominating Paths yields SWAP schedules that are both compact and naturally aligned with current hardware constraints. Beyond performance gains, the deterministic nature of our construction eliminates the variability inherent to stochastic optimization, providing predictable compilation behaviour and enabling reliable integration into larger toolchains. These properties make the proposed framework well suited for contemporary compilation workflows, especially in settings where circuit-agnostic routing is required at scale.

This work is organized as follows. In Section 2 we introduce circuit routing, with a focus on SWAP networks. In Section 3 we provide a theoretical background on minimum Dominating Paths. In Section 4 we describe each component of the proposed algorithm. In Section 5 evaluate our method on IBM architectures. Finally, in Section 6, some conclusions are drawn.

## 2 Circuit Routing and SWAP Networks

Routing is a fundamental step of quantum circuit transpilation [4, 8, 14], whose purpose is to adapt an abstract circuit to the connectivity constraints of a target quantum architecture. This process consists of dynamically relocating physical qubits across the device coupling graph through the insertion of SWAP gates, enabling the execution of two-qubit operations that would otherwise violate hardware connectivity.

The set of SWAP operations required to render a circuit executable is typically determined at compilation or execution time and depends explicitly on both the circuit structure and the underlying hardware topology. Consequently, any modification to the circuit or any change in the target architecture necessitates recomputation of the routing strategy.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

Unlike routing procedures, which depend jointly on the specific quantum circuit and the target architecture, a *swap network* is determined solely by the connectivity graph of the device. Its objective is to schedule a sequence of SWAP operations such that every pair of physical qubits becomes adjacent at least once over time. Under this condition, any two-qubit gate can be executed at some time step without requiring circuit-dependent routing decisions.

A key advantage of SWAP networks is their circuit agnosticism: once constructed for a given architecture, the same network can be reused across different input circuits, making this approach especially appealing for workloads involving repeated execution of different circuits requiring the same amount of qubits.

The notion of SWAP networks was first introduced as a quantum circuit compilation step in [10]. In that work, the authors introduce a framework for executing arbitrary sets of disjoint $k$-qubit gates on architectures with limited connectivity by modelling such gates as hyperedges. The resulting construction enables depth-efficient scheduling—achieving $O(n^{k-1})$ depth on a linear array—with optimal scaling for dense layers of multi-qubit interactions. This provides a compilation strategy for commuting layers like QAOA [15] and for Trotterized Hamiltonian simulation [16] on near-term quantum hardware with linear connectivity.

Subsequent work extended this idea to more general settings. In [11], the authors focus on architectures with generic connectivity graphs and investigate the use of simulated annealing to construct efficient SWAP networks tailored to two-qubit gates. Their results demonstrate that precomputed SWAP networks can be effectively integrated into highly repetitive circuit families, such as variational circuits used in QAOA, yielding competitive performance while avoiding circuit-specific routing overhead.

## 3 Minimum Dominating Paths

This section outlines the theoretical background and surveys related work concerning minimum Dominating Paths (mDPs). Furthermore, we present novel results regarding the computational complexity of the mDP problem.

Let $G = (V, E)$ be a connected, undirected graph. A (simple) *path* in $G$ is a sequence of distinct vertices $P = (v_1, \ldots, v_k)$ such that each consecutive pair $(v_i, v_{i+1})$ is an edge in $E$. The *length* of $P$ is $k - 1$.

We focus on the following notion.

**Definition 1 (Dominating Path).** *A path $P = (v_1, \ldots, v_k)$ in $G$ is* dominating *if for every vertex $v \in V$ either $v \in P$ or $\exists\, u \in P$ such that $\{u, v\} \in E$.*

To the best of our knowledge, the first appearance of Dominating Paths has to be attributed to [17]. In such work, authors introduced the notions of $\Delta_\lambda$-cycles and $\Delta_\lambda$-paths, as a *restriction* of $\mathsf{D}_\lambda$-cycles and $\mathsf{D}_\lambda$-paths introduced in [12].

While the latter are out of the scope of this work, we state the definition of $\Delta_\lambda$-paths. Before doing so, we recall that the distance between two vertices $u, v \in V$ in a graph $G = (V, E)$ is defined as the length of a shortest path connecting $u$ and $v$ in $G$.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

**Definition 2 ($\Delta_\lambda$-paths).** *Let $G = (V, E)$ be a graph and let $\lambda \geq 1$ be a natural non-zero number. A path $P = (v_1, v_2, \ldots, v_k)$ in $G$ is a* $\Delta_\lambda$-path *if every vertex $v \in V$ is at a distance less than $\lambda$ from at least one vertex of $P$.*

It is straightforward to observe that a $\Delta_1, \Delta_2$-paths are Hamiltonian and dominating paths, respectively. In our setting, the objective is to construct a SWAP network of minimal length. Since dominating paths constitute the first stage of our pipeline, the overall length of the SWAP network is largely determined by the length of the initial dominating path. This consideration leads naturally to the following computational problem.

**Definition 3 (Minimum Dominating Path).** *Let $G = (V, E)$ be a graph. The* minimum Dominating Path *problem asks for finding a shortest dominating path in $G$.*

We now state the first novel result of this paper: we prove that finding a dominating path of a given length is NP-complete (decision version), and, as a consequence, the optimization version is NP-hard.

Before doing so, for sake of completeness we recall that in [18] the authors show that finding a dominating path with minimal weight is a weighted graph is NP-hard. More in detail, they define the weight of a path as the sum of the weights of its edges, and they prove that finding a dominating path with minimal weight is NP-hard.

It is important to note that the minimum Dominating Path problem can be considered as a specific instance of the minimal weight dominating path problem given in [18], where the weights correspond to the constant function $w(e) = 1$ for all $e \in E$, i.e. the weight of the path corresponds to its length. However, the NP-hardness of the minimal weight dominating path problem does not imply the hardness of the minimum Dominating Path problem, which is a specific instance of the former. On the other hand, our result imply that the minimal weight version is at least NP-Hard.

We now prove that finding a dominating path of length at most $k$ is NP-complete by applying a reduction from the Set-Cover problem [19]. This reduction also shows that the problem 3 is NP-hard.

**Theorem 1.** *Given a graph $G = (V, E)$, finding a dominating path of length at most $k$ is NP-complete.*

*Proof.* The problem is in NP, since given a path $P$ in $G$, we can check in polynomial time whether $P$ is dominating and whether its length is at most $k$.

To prove NP-completeness, we provide a polynomial-time reduction from the Set-Cover problem. Let $I = (U, S, k)$ be an instance of Set-Cover, where $U = \{u_1, u_2, \ldots, u_n\}$ is the universe of elements, $S = \{S_1, S_2, \ldots, S_m\}$ is a collection of subsets of $U$, and $k$ is the maximum number of subsets allowed in the cover.

We construct a graph $G = (V, E)$ as follows:

- For each element $u_i \in U$, create a vertex $v_{u_i} \in V$.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

- For each subset $S_j \in S$, create a vertex $v_{S_j} \in V$.
- For each element $u_i \in U$ and each subset $S_j \in S$ such that $u_i \in S_j$, add an edge $\{v_{u_i}, v_{S_j}\} \in E$.
- Add edges to make the subgraph induced by the vertices corresponding to subsets in $S$ a complete graph, i.e., for all $S_j, S_k \in S$ with $j \neq k$, add an edge $\{v_{S_j}, v_{S_k}\} \in E$.

Now, we claim that there exists a set cover of size at most $k$ if and only if there exists a dominating path in $G$ of length at most $k-1$.

($\Rightarrow$) Suppose there exists a set cover $\mathcal{C} \subseteq S$ with $|\mathcal{C}| \leq k$. Construct a path $P$ in $G$ by including the vertices corresponding to the subsets in $\mathcal{C}$ and connecting them in any order. Since $\mathcal{C}$ covers all elements in $U$, every vertex corresponding to an element in $U$ is adjacent to at least one vertex in $P$. Thus, $P$ is a dominating path with length at most $k-1$.

($\Leftarrow$) Conversely, suppose there exists a minimum dominating path $P$ in $G$ with length at most $k-1$. It is easy to see that $P$ cannot contain any vertex corresponding to an element in $U$, since such vertices are leaves in $G$ and including them would not help in dominating other vertices. Therefore, $P$ consists only of vertices corresponding to subsets in $S$. Let $\mathcal{C}$ be the collection of subsets corresponding to the vertices in $P$.

Since $P$ is dominating, every vertex corresponding to an element in $U$ is either in $P$ or adjacent to a vertex in $P$. Therefore, the subsets in $\mathcal{C}$ cover all elements in $U$, and since the length of $P$ is at most $k-1$, we have $|\mathcal{C}| \leq k$. This completes the reduction, and thus the problem is NP-complete.

The following corollary is a direct consequence of our previous result:

**Corollary 1.** *The problem of finding a minimum dominating path is NP-hard.*

## 4 SWAP Networks via Minimum Dominating Paths

This section details the primary contribution of our work: a constructive algorithm for synthesizing SWAP networks on arbitrary connected topologies, grounded in the graph-theoretic concept of minimum Dominating Paths.

Formally, a *swap network* on a graph $G = (V, E)$ is defined as a finite sequence of SWAP operations $\mathcal{S} = (s_1, \ldots, s_k)$. Physically, the execution of a SWAP gate $s_t = (u, v) \in E$ exchanges the quantum states residing on physical qubits $u$ and $v$. This operation effectively permutes the mapping between logical and physical qubits, allowing logical qubits that are initially distant on the graph to be brought into adjacency.

The condition for a valid SWAP network is that for every pair of logical qubit labels $(i, j)$ from the set of qubits $Q$, there exists a time step $t$ wherein the physical vertices occupied by $i$ and $j$ are adjacent:

$$\forall i, j \in Q,\ \exists t \in \{1, \ldots, k\} \text{ such that } \{\varphi_t^{-1}(i), \varphi_t^{-1}(j)\} \in E, \tag{1}$$

where $\varphi_t^{-1}(\cdot)$ denotes the physical location of a logical qubit at time step $t$.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

Since the introduction of SWAP gates significantly increases circuit depth and accumulates error—thereby degrading the overall fidelity of the computation due to decoherence and gate infidelity—our objective is to construct SWAP networks that minimize the total gate count.

*Conjecture 1.* Computing the minimal SWAP network—SWAP count—is NP-hard

This conjecture is mostly supported by the fact that computing the minimal depth SWAP network is NP-hard [10].

Taking the hardware connectivity graph $G$ as input, the proposed method decomposes the routing problem into three modular phases:

1. The *identification* of a Minimum Dominating Path (mDP) to serve as the topological backbone;
2. The *execution* of a swap-and-position procedure along this path;
3. The *insertion and consolidation* of off-path vertices (leaves), which effectively reduces the remaining interactions to a SWAP network on a linear topology.

Note that while each sub-task in our pipeline is solved optimally, the overall four-stage procedure is heuristic. However, our approach remains computationally tractable as the hardware size increases, when exact and optimal algorithm would struggle in producing solutions.

### 4.1 Step 1: Minimum Dominating Path

The first step of our approach is to compute a mDP $P = (v_1, \ldots, v_k)$ of $G$. To do so, we use an Answer Set Programming [20] (ASP) encoding. ASP is a declarative programming paradigm that allows to encode and exactly solve combinatorial problems as logic ones. We refer the reader to [21] for a complete introduction to ASP.

Our ASP encoding can find a mDP in less than a second on a standard laptop, even for the largest architectures provided by IBM. Of course, also the problem of finding a minimal SWAP network can be encoded in ASP. However, our encoding was not able to solve any instances for architectures larger than 8 qubits. Full data is provided in the repository[3]

Given $P$, we define the induced graph $G^P = (V, E^P)$ by retaining: (i) the edges of the path $P$, and (ii) all edges connecting vertices of $P$ to vertices in $V \setminus P$. Edges between non-path vertices are discarded.

Vertices in $P$ are called *mDP nodes*, while vertices in $V \setminus P$ are called *leaves*. The graph $G^P$ is a caterpillar-like [22] structure with $P$ as its spine, possibly augmented with additional spine-to-leaf edges.

Finally, for the sake of completeness, it should be noted that not all graphs allows for the existence of a dominating path. The existence of dominating paths has been investigated in [17], where the authors find sufficient conditions for the existence of dominating path. However, the results obtained require the

[3] https://github.com/RiccardoRomanello/MDP-SN

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

minimum degree of the graph to grow with the number of vertices, and cannot be applied in our setting (as the topologies of quantum devices have a constant degree, usually up to three or four). However, in all the devices considered in the experimental setting (i.e., all the topologies provided by IBM) we were always able to find at least a dominating path. Note that when multiple mDPs exist, the quality of the final solution depends on the selection of the mDP. In particular, distinct mDP may yield slightly different lengths. In principle, it is possible to select the path leading to lowest SWAP count. In this work, we select for simplicity the first mDP returned by the ASP solver.

### 4.2 Step 2: Swap-and-Position on the mDP

Before describing this step, we first build the intuition behind it by considering the case in which the initial graph $G$ coincides with $P$, i.e. $G$ is a line path. First, we observe that routing algorithms on a path graph $P$ can be rigorously reformulated as string manipulations. Since each vertex of $P$ holds exactly one logical qubit, the global state can be represented as a string $\sigma = l_1 l_2 \dots l_n$, corresponding to the sequence of qubit labels read from left to right. Utilizing this abstraction, an optimal SWAP network on a line can be computed by generating a specific sequence of permutations. For instance, considering a path of length $|P| = 4$, a valid evolution of the system state is: $1234 \to 1324 \to 3124 \to 3214$. In this sequence, every pair of labels has eventually achieved adjacency. While the theoretical framework for linear topologies will be formally detailed as a subroutine of our general algorithm, we first address a necessary generalization of this problem relevant to the *Minimum Dominating Path* context.

If we apply the above strategy on our mDP $P$, we obtain that each node interacts with each other node.

However, it is easy to note that this process does not take into account leaves of the mDP. This motivates the addition of the requirement that each label in the mDP must visit each position—so that each one of them also interacts with each leaf. This motivates the following problem definition:

**Definition 4 (Swap-and-Position on a line).** *Given a string $s$ of length $k$, find a sequence of adjacent swaps such that:*

- *every pair of symbols becomes adjacent at least once;*
- *every symbol occupies every position at least once.*

We now provide an algorithm to solve the string Swap-and-Position problem[4], and then we prove its optimality in terms of number of operations.

The idea of the algorithm is the following. Let us assume the input string is $s = 123456$. It splits the string in half, in a left and a right part: $s_L = 123$ and $s_R = 456$. It proceeds by inverting both parts, obtaining $s'_L = 321$ and $s'_R = 654$. This process is done via a simple string inversion with the following

[4] We call this procedure *moonwalk*, since in a naive approach, one could just take the symbol from the rightmost one, and make it go through the string up to the left end.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

logic: $123 \to 213 \to 231 \to 321$. For what concerns the right part, the logic is similar but in the opposite direction: $456 \to 465 \to 645 \to 654$. Subsequently, the two inverted parts are interleaved via a *cross-move* logic. Starting from $s' = 321654$, the cross-move interleaves the two parts as follows: $321654 \to 326154 \to 362154 \to 632154$. The same logic is applied to 1 as well, obtaining $s = 632541$. The algorithm continues by swapping 2 and 5, obtaining 635241. The two symbols are moved to the very end of the string: 563412. At last, the very same idea is applied to 3 and 4 as well, obtaining 456123. This interleaving process is formally implemented in lines 12–13 of Algorithm 1, where the `Move` operations act as the functional *hook* that executes these cross-moves by shifting symbols from the middle to the boundaries of the string. It is clear that every symbol has interacted with every other symbol at least once and has occupied every position in the string at least once.

The odd-length string case is analogous, except that the left part is longer than the right by one. As a result, the algorithm requires a few additional moves at the end to bring the leftmost symbol to the end of the string. The full pseudocode is given in Algorithm 1.

**Algorithm 1** Swap-and-Position Algorithm

```
1: procedure MoonwalkLogic(S, start, end)
2:     n ← end − start
3:     if n ≤ 1 then
4:         return
5:     end if
6:     m ← start + ⌈n/2⌉                    ▷ Left gets extra element if n is odd.
7:     InvertLR(S, start, m)                                  ▷ Invert Left
8:     InvertRL(S, m, end)                                    ▷ Invert Right
9:     if n (mod 2) = 0 then                                  ▷ === Even Case ===
10:                                                           ▷ Interleave via Cross-Move
11:        for k ← 0 to n/2 − 1 do
12:            Move(S[m] → start)                             ▷ Right-Head to Left-Head
13:            Move(S[m] → end − 1)                           ▷ Left-Head to Right-Tail
14:        end for
15:    else                                                   ▷ === Odd Case ===
16:        len_R ← end − m
17:                                                           ▷ Move Right Block to Front
18:        for k ← 0 to len_R − 1 do
19:            Move(S[m + k] → start)
20:        end for
21:        m_new ← start + len_R
22:        InvertRL(S, m_new, end)                            ▷ Re-orient new Right segment
23:    end if
24: end procedure
```

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

We now evaluate the total number of adjacent swaps performed by the algorithm on a string of length $n$. `InvertLR` (and its symmetric counterpart `InvertRL`) inverts a string of length $n' = n/2$, which requires exactly $\frac{n'(n'-1)}{2}$ adjacent swaps—the optimal number for reversing a permutation of $n'$ elements, as observed in [23]. The two `Move` operations together perform $n-1$ swaps per iteration. To see this, consider a string $s = s_1 \dots s_{n/2}\, s_{n/2+1} \dots s_n$. Swapping at the middle moves $s_{n/2+1}$ to position $n/2$, from which it takes $n/2-1$ adjacent swaps to reach position 1. Similarly, $s_{n/2}$ moves to the end in $n/2-1$ swaps. Including the initial middle swap, this totals $n-1$ swaps per iteration. Repeating this for half of the symbols gives $\frac{n}{2}(n-1)$ swaps. Combining the contributions of `InvertLR`, `InvertRL`, and all `Move` operations, the total number of swaps for $n$ even is $3\frac{n^2}{4} - n$. For $n$ odd, the total amount is $\frac{3n^2+1}{4} - n$. Our algorithm solves the Swap-and-Position problem in $S(n) = \left\lceil \frac{3n^2}{4} - n \right\rceil$. It can be proved that $S(n)$ is also the minimum amount of operations required to solve the Swap-and-Position problem. Due to space constraints, we omit the full derivation and instead provide a sketch of the proof. The argument relies on quantifying the total *distance* that the symbols in a string of length $n$ must traverse to satisfy the interaction requirements. We analyze the impact of a single SWAP operation on this metric, demonstrating that each step can reduce the total remaining distance by either 2 units (when both swapped symbols effectively "discover" new necessary positions) or 1 unit (when only one symbol benefits). We formally show that the closed-form expression $S(n)$ corresponds exactly to the minimal sequence of steps required to fully resolve this aggregate distance, thereby proving optimality. Hence, we obtain the following result:

**Theorem 2.** *Algorithm 1 solves the Swap-and-Position problem using* $S(n) = \left\lceil \frac{3n^2}{4} - n \right\rceil$ *operations. Moreover,* $S(n)$ *is the optimal number of operations.*

where the ceiling is used to provide a compact closed-form expression that correctly captures the number of operations for both even and odd path lengths.

This conclude the second step of our pipeline. The only remaining non-interacting pairs are those involving two leaves.

### 4.3 Step 3: Leaves Interactions

Once the Algorithm 1 has been applied to the mDP $P$, pair of labels that have yet to interact are just the ones in which both labels are from the leaves of $G^P$. The core idea is to insert the leaves in $P$, and apply the inversion string to find a SWAP network on the leaves, as described in the last section.

According to the leaves count, we can have two cases. Let $k$ be the length of the mDP $P$ and let $m$ be the number of leaves ($k + m = n$).

We first consider $k > m$. Then, the steps we perform are the following.

- We move all the labels of the leaves inside the mDP $P$;
- We group all such labels together in $P$ (i.e., to make them adjacent);

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

– We apply the inversion string algorithm to find a SWAP network on the leaves.

More in detail, let $H$ be an histogram of leaves count for each node in $P$. Formally, $H_v$, for $v \in P$, is equal to $|\{u \mid (v,u) \in G^P, u \notin P\}|$. Using $H$ we want to compute a new histogram $H'$ that will contain just 0s and 1s and all the ones are grouped together. Informally, the 1s in $H'$ will describe where the leaves of $G^P$ will end up after insertion and grouping. Since $k > m$, we can always fit as many ones in $H'$ as the sum of the elements of $H$. This operation must be performed using the minimum number of adjacent swaps—since all of these labels operation will have to be replicated inside the SWAP network.

This optimization is formally equivalent to the *1-Dimensional Facility Location Problem* with the constraint of non-overlapping destinations, also referred to as the *Soldiers Problem* [24]. The optimal configuration is determined as follows:

1. **Coordinate Expansion:** We interpret the input histogram $H$ as a sorted sequence of source coordinates $X = (x_1, x_2, \ldots, x_m)$, where the index $i$ appears $H[i]$ times in the sequence. This maps the abstract leaf counts to physical locations on the mDP.
2. **Optimal Alignment:** We seek a start index $p$ that minimizes the total transport distance (SWAP count):
$$\min_p \sum_{i=1}^{N} |x_i - (p + i - 1)|$$
By applying the coordinate transformation $y_i = x_i - (i-1)$, the problem reduces to finding a value $p$ that minimizes the $L_1$ norm $\sum |y_i - p|$. It is a fundamental result in order statistics that this sum is minimized when $p$ is the median of the transformed sequence $Y$ [25].
3. **Target Construction:** Given the optimal start index $p$, we construct the output histogram $H'$, which defines the specific locations where the leaves must be placed to form the contiguous block.

Now we obtained that each (former) leaf lies on the initial dominating path $P$. Since all leaves are grouped, we just need to address the all-to-all interaction on a line problem, which can be solved optimally in $\frac{(m-1)(m-2)}{2}$ steps. Let $l_1 \ldots l_{m-1} l_m$ be the string composed by the leaves labels now in the mDP. The algorithm works by making a full inversion of $l_1 \ldots l_{m-1}$ starting from right to left. This is obtained via adjacent swaps and the overall procedure can be done in $\frac{(m-1)(m-2)}{2}$ steps. Moreover, this is the *optimal* amount of operations required for such full inversion [23]. Therefore, we can state the following theorem:

**Lemma 1.** *Minimal SWAP network on a line can be solved in polynomial time.*

This concludes the $k > m$ case. Note that most of the topologies provided by IBM belong to this case. In particular, just three out of the 53 architectures had more leaves than nodes, and always at most 8 qubits. For the sake of completeness, we address the scenario where $k \leq m$. In this regime, the algorithm employs a sequential procedure to accommodate the residual $k - m$ leaves. Specifically, each leaf is individually inserted into the mDP and traverses the path structure exactly once, ensuring that every required physical adjacency is realized.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

### 4.4 Overall Algorithm

Algorithm 2 summarizes the full construction. Once the mDP is computed, all subsequent steps have closed-form SWAP counts and deterministic execution. The resulting SWAP network is circuit-agnostic, topology-dependent, and reusable across different workloads. At this juncture, it is pertinent to highlight two distinctive advantages of our algorithmic framework.

First, regarding computational overhead: although the minimum Dominating Path problem is NP-complete, it does not constitute a practical bottleneck. Empirically, for all hardware topologies tested, the exact solver converges in sub-second time. Furthermore, unlike standard routing strategies that must be re-executed for every input circuit, our approach generates a SWAP network that depends solely on the hardware topology. Consequently, the mDP computation is a *one-time offline cost*. Even if the exact solution required significant runtime, it would be amortized over the lifespan of the device, as the resulting network is reusable for any quantum circuit executed on that architecture.

Second, the derivation of analytical closed forms allows for deterministic performance prediction. If the objective is merely to estimate the gate overhead—without explicitly generating the gate sequence—our method achieves this with *constant time complexity*, $O(1)$, provided the mDP length. This stands in sharp contrast to state-of-the-art methods based on *Simulated Annealing* (SA). Such approaches are inherently stochastic, lacking pre-computable cost functions and offering no guarantees on the gate count prior to full execution.

**Algorithm 2** Full SWAP network algorithm

**procedure** SwapNetwork($G = (V, E)$)
  $P \leftarrow$ MinimumDominatingPath($G$)
  $L \leftarrow V \setminus P$
  $G^P \leftarrow$ ComputeGraphInducedByMDP($G, P$)
  $\varphi_i = i \; \forall i \in V$
  $\varphi \leftarrow$ MoonWalkLogic($P, \varphi$) ▷ Solving Swap-and-Position problem
  $k = |P|, m = |V| - k$
  $\varphi \leftarrow$ InsertAndGroupLeaves($G, P, \varphi$)
  $\varphi \leftarrow$ AcquaintLeaves($G, P$) ▷ all-to-all interaction between leaves in mDP.
  **if** $k < m$ **then**
    **for** $l \in L'$ **do** ▷ $L'$ is the set of yet-to-be-inserted leaves
      FullWalkthrough($l, P, \varphi$) ▷ Leaves enter $P$ using an anchor $u \in P$.
    **end for**
  **end if**
**end procedure**

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

## 5 Experimental Results

We evaluate our proposed pipeline on the architectures of IBM devices [13]. We select all the 53 *"fake"* hardware (including the topologies of all available hardware, such as IBM Torino), ranging from 5 to 133 qubits, and we evaluate both our pipeline and the state-of-the-art approach in [11], based on Simulated Annealing (SA), as a baseline. Our pipeline is implemented in ASP [21] for finding the minimum Dominating Path, and in Python 3.10 for the other steps. As the code for the SA baseline is not available, we re-implemented Python 3.10 to the best of our possibilities, according to the pseudo-code provided in [11]. The source code can be retrieved in the git repository attached to this paper.

In Figure 2 we provide the average reduction of our method compared to the SA baseline in [11]. Note that for low qubit counts the final SWAP counts are very similar. The reason behind this result is that the search space is sufficiently small that SA effectively performs an exhaustive search, reaching the global optimum[5].

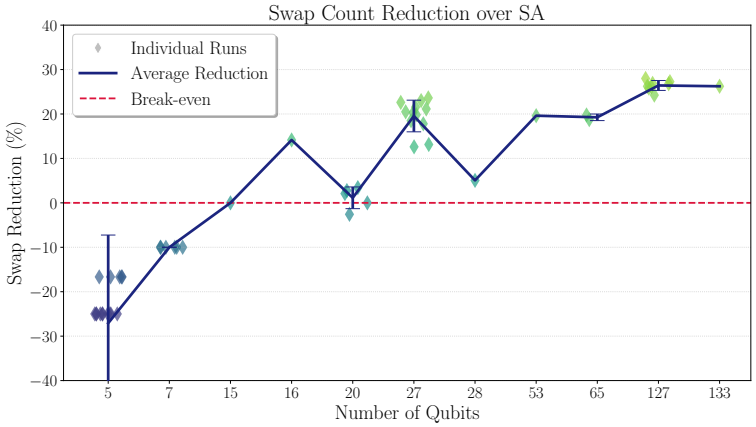


Fig. 2: Percentage reduction in SWAP count for our proposed method compared to [11]. Solid line represents the average (with standard deviation given by error bars) for a given qubit count, while individual runs are represented by diamonds (for 5 qubits, a single run is omitted for readability reasons).

Higher qubit counts are more interesting, as the corresponding hardware are closer to real ones. Note that the *Fake* topologies provided by IBM include also real devices topologies (e.g. IBM Torino). In Table 1, we provide a detailed comparison hardware-by-hardware for each model with 53 qubits or more, for a total of 12 architectures. Our method obtains a consistent advantage, ranging from 18% to 28%. In particular, for architectures with at least 127 qubits, we obtain an advantage of at least 25%.

[5] Qubit counts up to 7 allow for optimal SWAP networks computation with ASP.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

Table 1: Baseline, Total SWAPs, and Percentage Reduction with respect to SA [11], for each device with more than 50 qubits. Isomorphic topologies are reported once, as our method is deterministic and obtains the same amount of SWAPs.

| Backend | Qubits | **SWAPs (this work)** | SWAPs (SA [11]) | Reduction (%) |
|---|---|---|---|---|
| *rochester* | 53 | 1467 | 1825 | 19.62 |
| *brooklyn* | 65 | 2243 | 2796 | 19.78 |
| *brisbane* | 127 | 8837 | 12074 | 26.81 |
| *kawasaki* | 127 | 8799 | 12219 | 27.99 |
| *kyiv* | 127 | 8815 | 12121 | 27.27 |
| *kyoto* | 127 | 8843 | 11988 | 26.23 |
| *osaka* | 127 | 8835 | 12084 | 26.89 |
| *sherbrooke* | 127 | 8909 | 11761 | 24.25 |
| *torino* | 133 | 9726 | 13186 | 26.24 |

## 6 Conclusion

Quantum circuit routing represents a critical stage in the compilation stack, essential for adapting circuits to the connectivity constraints of physical NISQ devices. Currently, this challenge is addressed via dynamic approaches that compute ad-hoc routing solutions for every pair of input circuit and target architecture. In contrast, SWAP networks offer a circuit-agnostic alternative by precomputing a universal schedule of interactions determined solely by the hardware topology, ensuring that every pair of qubits eventually becomes adjacent.

In this work, we advanced the latter paradigm by introducing a novel, deterministic framework centered on the graph-theoretic concept of *Minimum Dominating Paths* (mDPs). By decomposing the hardware topology into a structural backbone and associated leaves, our method systematically constructs efficient swap schedules through a modular pipeline. We provided a rigorous theoretical foundation for our approach, proving the NP-completeness of the mDP problem while demonstrating that the core constituent of our routing strategy—the swap-and-position procedure—achieves optimality in terms of gate count.

Empirical evaluations on IBM quantum architectures ranging from 5 to 133 qubits shows the effectiveness of the proposed method. Our pipeline consistently outperforms state-of-the-art baselines, achieving up to a 28% reduction in total SWAP count on large-scale devices maintaining deterministic execution times.

Future research will focus on further optimizing the leaf integration phase, specifically by investigating batched processing strategies. Additionally, we plan to integrate this routing framework into full-stack quantum compilers to validate its impact on end-to-end circuit fidelity in practical application workflows.

## References

1. R. P. Feynman. Simulating physics with computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5

2. F. et al. Arute. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.
3. J. Preskill. Quantum Computing in the NISQ era and beyond. *Quantum*, 2018.
4. Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. page 1001–1014, 2019.
5. Y. et al. Kim. Evidence for the utility of quantum computing before fault tolerance. *Nature*, 618:500–505, 2023.
6. F. Chong, D. Franklin, and M. Martonosi. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549:180–187, 09 2017.
7. C. Zhu, X. Wu, Z. Yang, J. Wang, A. Wu, S. Zheng, and X. Wang. Quantum Compiler Design for Qubit Mapping and Routing: A Cross-Architectural Survey of Superconducting, Trapped-Ion, and Neutral Atom Systems. 5 2025.
8. W. Tang, Y. Duan, Y. Kharkov, R. Fakoor, E. Kessler, and Y. Shi. Alpharouter: Quantum circuit routing with reinforcement learning and tree search, 2024.
9. I. Shaik and J. van de Pol. Optimal layout synthesis for deep quantum circuits on nisq processors with 100+ qubits, 2024.
10. B. O'Gorman, W. J. Huggins, E. G. Rieffel, and K. B. Whaley. Generalized swap networks for near-term quantum computing, 2019.
11. T. Parella-Dilmé, J. S. Kottmann, and A. Acín. Swap network augmented ansätze on arbitrary connectivity, 2025.
12. H.J. Veldman. Existence of dominating cycles and paths. *Discrete Mathematics*, 43(2):281–296, 1983.
13. Qiskit: An open-source framework for quantum computing, 2023.
14. A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah. On the Qubit Routing Problem. In *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, volume 135 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:32, 2019.
15. E. Farhi, J. Goldstone, and S. Gutmann. A quantum approximate optimization algorithm, 2014.
16. L. Pastori, T. Olsacher, C. Kokail, and P. Zoller. Characterization and verification of trotterized digital quantum simulation via hamiltonian and liouvillian learning. *PRX Quantum*, (3), 2022.
17. H. J. Broersma. Existence of $\delta\lambda$-cycles and $\delta\lambda$-paths. *Journal of Graph Theory*, 12(4):499–507, 1988.
18. G. Kutiel. Hardness results and approximation algorithms for the minimum dominating tree problem, 2018.
19. B. Korte and J. Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Incorporated, 5th edition, 2012.
20. V. Lifschitz. What is answer set programming? AAAI'08. AAAI Press, 2008.
21. N. Leone and F. Ricca. *Answer Set Programming: A Tour from the Basics to Advanced Development Tools and Industrial Applications*, pages 308–326. Springer International Publishing, 2015.
22. M. Khosravani. *Searching for optimal caterpillars in general and bounded treewidth graphs*. PhD thesis, University of Auckland, 2011.
23. D. Kim. Sorting on graphs by adjacent swaps using permutation groups. *Computer Science Review*, 22:89–105, 2016.
24. A. M. Yaglom and I. M. Yaglom. *Challenging Mathematical Problems with Elementary Solutions*, volume 1. Holden-Day, 1964. Problem 64: The Soldiers Problem.
25. T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press, 3rd edition, 2009. See Chapter 9: Medians and Order Statistics.


ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_5