A Fast and Scalable Qubit-Mapping Method for Noisy Intermediate-Scale Quantum Computers

Sunghye Park, Daeyeon Kim, Minhyuk Kweon, Jae-Yoon Sim, and Seokhyeong Kang
Department of Electrical Engineering
Pohang University of Science and Technology, Pohang, Korea
shkang@postech.ac.kr

ABSTRACT

This paper presents an efficient qubit-mapping method that redesigns a quantum circuit to overcome the limitations of qubit connectivity. We propose a recursive graph-isomorphism search to generate the scalable initial mapping. In the main mapping, we use an adaptive look-ahead window search to resolve the connectivity constraint within a short runtime. Compared with the state-of-the-art method [15], our proposed method reduced the number of additional gates by 23 % on average and the runtime by 68 % for the three largest benchmark circuits. Furthermore, our method improved circuit stability by reducing the circuit depth and thus can be a step forward towards fault tolerance.

KEYWORDS

Noisy Intermediate-Scale Quantum (NISQ) Hardware, Qubit Mapping, Scalable Quantum Computing

1 INTRODUCTION

Nowadays, after the doom of Moore's law, quantum computing has attracted substantial attention. Quantum computing performs computations by exploiting quantum phenomena such as superposition and entanglement instead of classical physics [1]. As a result, quantum computing can provide essential speed advantages [2, 3, 4] and an enormous increase in computational capacity compared to classical computations. Quantum algorithms use specific quantum paradigms or components [5] that have no classical equivalents. For example, quantum algorithms [2, 6] that rely on the *quantum Fourier transform* achieve exponential speedup. In general, a quantum circuit consisting of multiple logical quantum bits (qubits) and a sequence of quantum logic gates is used to describe quantum algorithms.

To enable quantum computations, a quantum computer must operate the quantum logic gates step-by-step. At that time, a pair of qubits interacting in the quantum logic gate must be physically connected on the quantum hardware. However, modern quantum computers such as IBM Q [7] and Google [8] are noisy intermediate-scale quantum (NISQ) devices, which have limitations in operating quantum logic gates; because not every physical qubit is always connected. Therefore, to make the quantum circuit executable on a target quantum hardware, quantum algorithms must be modified; this procedure is known as qubit mapping (Fig. 1).

DAC '22, July 10–14, 2022, San Francisco, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9142-9/22/07…$15.00
https://doi.org/10.1145/3489517.3530402

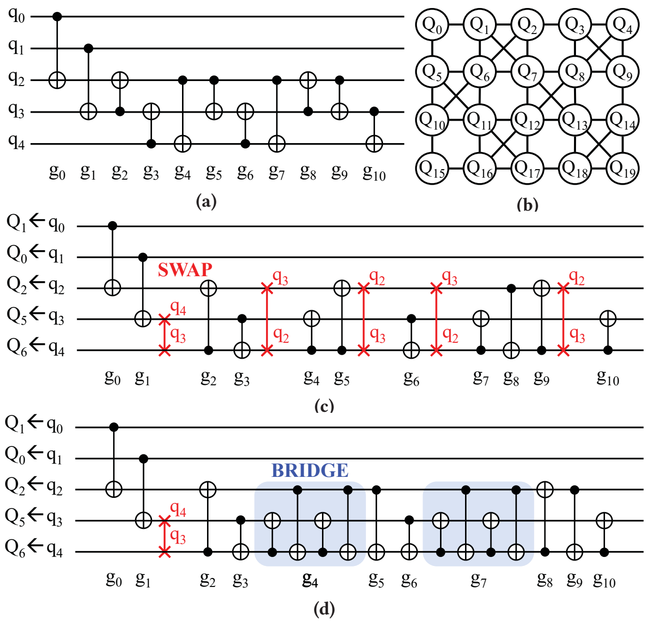


Figure 1: Overall flow of the qubit-mapping problem; (a) Real quantum circuit: *4mod_v1_22.* (b) Coupling graph of quantum hardware: *IBM Q 20 Tokyo.* (c) Modified circuit after qubit mapping (a) to (b) using only SWAP gates. (d) Modified circuit after qubit mapping (a) to (b) using SWAP and BRIDGE gates.

In qubit mapping, additional quantum gates, e.g., SWAP or BRIDGE operations, are used to modify the quantum circuit. However, these extra quantum gates can increase the circuit depth and induce errors. The additional gates have higher error rate than do existing gates [9], so the computation result of the algorithm eventually fails due to accumulation of errors. The increased circuit depth also decreases the algorithmic accuracy due to physical limits in NISQ devices [10]. Therefore, a qubit-mapping algorithm that can efficiently reduce the inserted gate count (i.e., 'overhead') is required.

In this work, we propose an efficient heuristic qubit-mapping algorithm with linear time complexity. We introduce a recursive graph-isomorphism search to find an initial mapping at short runtime. In the main mapping stage, we perform an adaptive look-ahead window search to calculate the heuristic cost with reduced search space. Then we use SWAP and BRIDGE gates to solve the gate constraints by using the obtained cost. We also perform post-processing to increase the quality of the results. With the proposed method, we reduced circuit overheads and runtime during the qubit mapping. The main contributions of this paper are as follows:

- With a recursive graph-isomorphism search using multiple centers, we found a scalable initial solution between logical qubits and physical qubits quickly.

- We propose a heuristic cost function that uses an adaptive window that considers the discount factor to give more reward to the front than the backside of the circuit, so we reduced the search space and runtime in the main mapping.
- Our proposed method has linear time complexity according to the gate size, and has the merit of minimizing the circuit depth.
- Compared to exact methods [12, 13], we provide cost-effective results for small circuits in an extremely short runtime.

The remainder of this paper is organized as follows: Section II summarizes the qubit-mapping problem and the previous work. Section III explains the proposed algorithm for initial mapping and main mapping with examples. Section IV reports results and analysis, and followed by the conclusion described in Section V.

2 PRELIMINARIES

2.1 The Qubit-Mapping Problem

The qubit-mapping problem is to map a set q of logical qubits in a quantum circuit to a set Q of physical qubits in quantum hardware. During this procedure, additional quantum gates are incorporated into the input circuit to overcome the limitation of connection between the physical qubits. This work uses two types of additional gates (Fig. 2). SWAP gate exchanges the state of the two neighboring logical qubits on the quantum hardware. BRIDGE gate refers that single non-neighboring CNOT gate with a physical distance of "2" can be transformed to four adjacent CNOT gates. The objective is to minimize the number of additional quantum gates.

Quantum circuits are modified after the qubit-mapping procedure from the quantum circuit (*4mod_v1_22*) to quantum hardware (*IBM Q 20 Tokyo* [7]) (Fig. 1). In the example, a random initial mapping ($q_i \mapsto Q_j$) between logical qubits and physical qubits is used. Then the quantum hardware sequentially executes the original quantum circuit from quantum gates g_0 to g_{10}. When executing g_2, the logical qubits $q_2 \mapsto Q_2$ and $q_3 \mapsto Q_5$ are not directly connected on the hardware. To make the quantum gate executable, we must perform a SWAP operation in front of g_2 (Fig. 1c). Also, to further reduce the circuit overhead, a BRIDGE operation must be exploited along with the SWAP operation (Fig. 1d).

2.2 Related Work

2.2.1 Exact Algorithm. Exact algorithms [11, 12, 13] find an optimal qubit-mapping solution that satisfies all constraints at a minimum cost. They use problem solvers such as dynamic programming to meet constraints, which are defined as the states between the layout and each gate. However, they are slow and require a large amount of memory. Also, they do not easily scale to more than five qubits, and therefore are only used for small quantum circuits.

2.2.2 Heuristic Algorithm. Heuristic methods [13, 14, 15, 16, 17, 18, 19] were introduced as practical approaches to overcome the limited speed and scalability of exact algorithms. They first obtain an initial mapping solution and then execute a main mapping stage to solve the connectivity constraint with the additional gates.

• **Initial Mapping:** Li et al. [14] have updated the initial mapping solution using reversible quantum characteristics repeatably. But, the number of additional gates is not gradually reduced over iterations. Zhu et al. [15] exploit the graph isomorphism between the quantum circuit and hardware, but has limitations in using only one center for graph-matching. Some methods [16, 17] use mathematical approaches such as mixed-integer programming and simulated annealing, but are much slower than other algorithms.

• **Main Mapping:** Zhu et al. [15] have selected the SWAP gate with the maximum consecutive positive effect, but do not consider reducing the search space when calculating the cost. Another method [13] exploits the BRIDGE gate along with the SWAP gate to solve the coupling problem. However, the policy of choosing between two gates to use is not optimized.

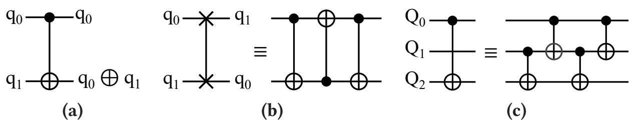
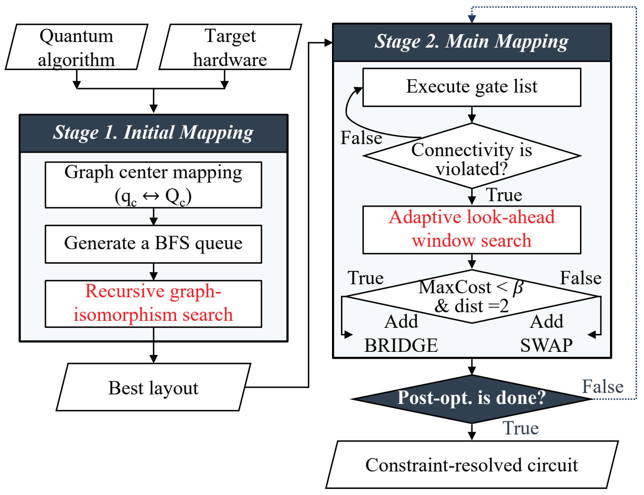


Figure 2: Quantum gates used in this work; (a) CNOT. (b) SWAP. (c) BRIDGE.



Quantum algorithm
Target hardware
Stage 1. Initial Mapping
Graph center mapping ($q_c \leftrightarrow Q_c$)
Generate a BFS queue
Recursive graph-isomorphism search
Best layout
Stage 2. Main Mapping
Execute gate list
Connectivity is violated?
False
True
Adaptive look-ahead window search
True
MaxCost < β & dist =2
False
Add BRIDGE
Add SWAP
Post-opt. is done?
False
True
Constraint-resolved circuit

Figure 3: Overall flow of the proposed qubit-mapping method.

3 PROPOSED ALGORITHM

Our proposed method (Fig. 3) consists of two stages - initial mapping and main mapping. The solution quality of initial mapping is crucial, because the initial layout determines the gate constraints and then affects the additional gate counts. So we generate a good-quality initial mapping by using our proposed recursive graph-matching approaches. Next, the main mapping stage resolves the connectivity problem by using SWAP or BRIDGE gates, by consulting our heuristic cost function. In calculating the cost, we apply a discount factor to consider each portion of a quantum circuit differently. Finally, the post-optimization stage improves the mapping quality by further reducing the number of additional CNOT gates.

For the efficient search in our proposed method, we generate the data structure (Fig. 4). We use two weighted graphs to determine the search order for the initial mapping: (a) *interaction-weight graph* and (b) *interaction-order graph.* The nodes in both graphs represent logical qubits. The weighted edges of (a) indicate the number of CNOTs, and of (b) indicate the order of the first CNOT gates that appeared. During the main mapping procedure, we generate a set of linked lists of a quantum circuit to determine the execution order: (c) *gate list.* It has n-linked lists as the number of logical qubits and indicates topological relationships between the gates.

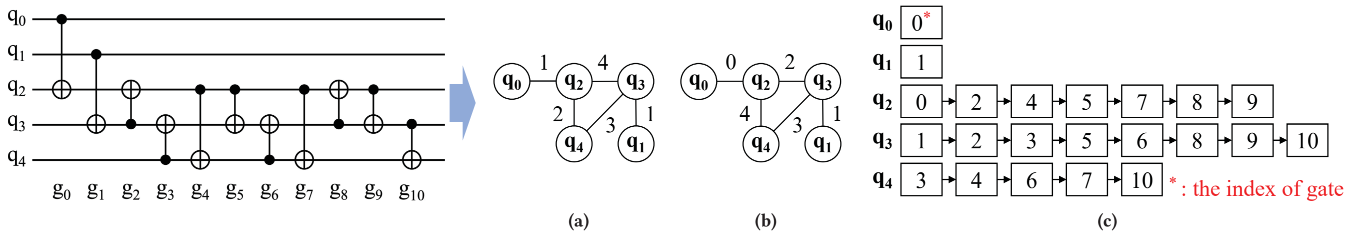
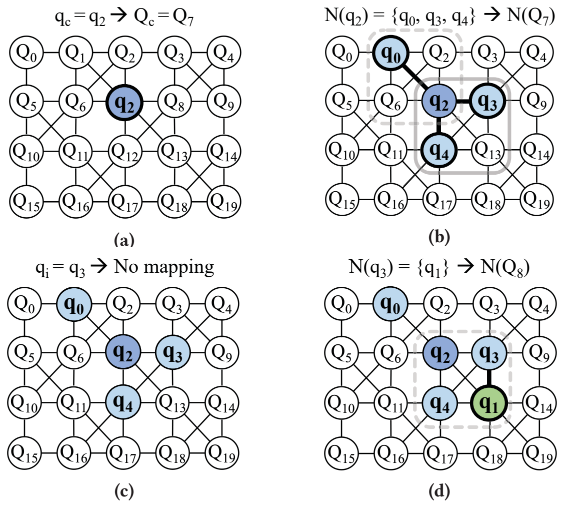


Figure 4: Data structures for quantum circuit in Figure 1a; (a) Interaction-weight graph. (b) Interaction-order graph. (c) Gate list.

```
Algorithm 1: Recursive graph-isomorphism search
   inputs  : Interaction-weight graph $G_w$, Interaction-order graph $G_o$,
             Coupling graph $G_c$ and distance table $dist[\,][\,]$
   output  : An Initial mapping $layout[\,]$ ;
 1 Initialize $layout[\,] = -1$ ;
 2 /* $layout[q_i]$ = -1, logical qubit $q_i$ is not mapped on any physical qubit */
 3 $Q_c \leftarrow$ CalGraphCenter($G_c$) ;
 4 $q_c \leftarrow$ CalGraphCenter($G_w$) ;
 5 $layout[q_c] \leftarrow$ UpdateLayout($Q_c$) ;
 6 $q_cQueue[\,] \leftarrow$ BfsQueueGen($G_w, q_c$) ;
 7 while !$q_cQueue[\,]$.empty() do
 8     $q_i \leftarrow q_cQueue$.front() ;
 9     $Q_i \leftarrow$ MakeCandidateLocation($q_i, G_c$) ;
10     if $layout[q_i] = -1$ then
11         $layout[q_i] \leftarrow$ UpdateLayout($Q_i$) ;
12     end
13     $SubTree(q_i) \leftarrow$ MakeSubtree($q_i, G_o$) ;
14     $N(q_i) \leftarrow$ GetChildNodes($SubTree(q_i)$) ;
15     $N(Q_i) \leftarrow$ GetDistanceOnes($Q_i, G_c$) ;
16     $layout[N(q_i)] \leftarrow$ UpdateLayouts($N(Q_i)$) ;
17     $q_cQueue$.pop() ;
18 end
19 return $layout[\,]$
```


$q_c = q_2 \rightarrow Q_c = Q_7$
$N(q_2) = \{q_0, q_3, q_4\} \rightarrow N(Q_7)$
$q_i = q_3 \rightarrow$ No mapping
$N(q_3) = \{q_1\} \rightarrow N(Q_8)$
(a)
(b)
(c)
(d)
Figure 5: (a) Step #1 ($q_c = q_2$). (b) Step #2 (neighbor nodes of q_c). (c) Step #3 ($q_i = q_3$). (d) Step #4 (neighbor nodes of q_3).

3.1 Initial Mapping with Recursive Graph-Isomorphism Search

The best initial mapping solution to reduce the gate count is an isomorphic graph between the quantum circuit and hardware, because every interaction in logical gates can be connected on the hardware. However, as the maximum degree of the coupling graph is limited, determining an exact graph matching is difficult. Therefore, the goal of our proposed initial mapping stage, *recursive graph-isomorphism search*, is to find the isomorphic graph or alternative subgraph that is expected to minimize the overhead (Algorithm 1).

The algorithm starts to calculate the physical graph-center [20], which has the minimum topological distance from all nodes the coupling graph. When calculating the logical center, we use the *interaction-weight graph* to consider qubits in descending order of number of CNOT connections. Then we map the logical center to the physical center (Lines 3-5). Afterward, the subgraph can be expanded from the center that has the highest degree of freedom.

In the recursive graph-isomorphism search, we iterate the mapping of each logical center and the mapping of child nodes in the sub-tree. To determine the order of center qubits, we perform a breadth-first search (BFS) around the first logical center q_c (Line 6). Using the derived order, we map the new logical center q_i to the physical qubit Q_i (Lines 8-12). Then we create a sub-tree that has a parent node q_i (Line 13). An *interaction-order graph* is used to consider gate constraints related to the parent node sequentially. Finally, the child nodes in the sub-tree are mapped to neighbor locations that have distance "1" to the physical qubit (Lines 14-17). If all graph matching is completed, the final layout is printed as output (Line 19) and then used as input for the main mapping stage.

Example 1. Continuing with the example (Fig. 1), we show the proposed graph-isomorphism procedure (Fig. 5). The final *layout* is printed as $\{q_0, q_1, q_2, q_3, q_4\} \mapsto \{Q_1, Q_{13}, Q_7, Q_8, Q_{12}\}$. Then we can map the input circuit without any additional gates.

The algorithmic complexity of calculating graph center and BFS is $O(n^2)$, and updating the layout is $O(\log n)$, where n is the number of logical qubits. The complexity of generating physical candidates is $O(m^2)$, where m is the number of physical qubits. Our proposed algorithm is performed recursively, so the total complexity is $O(n^2) + O(n+1)(\log n + m^2) = O(m^2 n + n^2)$. Compared to the exact graph-matching method [21], which has a computational complexity of $exp(n^{1/2})$, our approach provides an alternative subgraph in the reasonable time complexity.

3.2 Main Mapping with Adaptive Look-ahead Window Search

In the main mapping stage, we must resolve all connectivity constraints by inserting additional gates. In this work, we propose an *adaptive look-ahead window search* with a heuristic cost function (Algorithm 2). By considering the calculated cost, we find the best SWAP gate to execute the current gate. If the selected SWAP is not continuously effective for the backside, we choose the BRIDGE gate as a substitute. As the search size is efficiently reduced, the main mapping procedure is finished at fast runtime regardless of the circuit size.

Algorithm 2: Adaptive look-ahead window search

```
   inputs  : Original quantum circuit $gateList[\,]$, Initial mapping $layout[\,]$,
             Coupling graph $Gc$ and distance table $dist[\,][\,]$
   output  : Constraint-resolved circuit $FinalCircuit[\,]$
1  while !$gateList[\,]$.empty() do
2      while do
3          $headGates[\,] \leftarrow$ UpdateHeadGates($gateList[\,]$) ;
4          $pairGates[\,] \leftarrow$ CheckPairs($headGates[\,]$) ;
5          $execGates[\,], nonexecGates[\,] \leftarrow$
             CheckExecutable($pairGates[\,], layout[\,]$) ;
6          if $execGates[\,]$.empty() then break
7          foreach $execGate$ where $execGates[\,]$ do
8              $FinalCircuit[\,]$.push_back($execGate$) ;
9          end
10     end
11     $candiSwap[\,] \leftarrow$ GetSWAPcandi($dist[\,][\,], nonexecGates[\,], G_c$) ;
12     $costSwap[\,] \leftarrow$ CalCost($candiSwap[\,], dist[\,][\,], gateList[\,], \alpha$) ;
13     /* $\alpha$ is a discount factor for calculating the heuristic cost ($0 \le \alpha \le 1$)   */
14     $maxCost, bestSWAP \leftarrow$ FindBestSWAP($costSwap[\,]$) ;
15     if GetDist($nonexecGate$) = 2 and $maxCost \le \beta$ then
16         $BRIDGE \leftarrow$ ConvertBridge($nonexecGate$) ;
17         $FinalCircuit[\,]$.push_back($BRIDGE$) ;
18     end
19     /* $\beta$ is minimum cost to choose the BRIDGE gate instead of SWAP gate   */
20     else
21         $FinalCircuit[\,]$.push_back($bestSWAP$) ;
22         $FinalCircuit[\,]$.push_back($nonexecGate$) ;
23         $layout[\,] \leftarrow$ UpdateLayout($bestSWAP$) ;
24     end
25 end
26 return $FinalCircuit[\,]$
```

To execute the gates sequentially, we first designate the front-most gate of each logical qubit as the *head gates* by using the *gate list* (Line 3). If an arbitrary gate is a head gate for a pair of logical qubits, then the corresponding gate becomes a candidate to be mapped now; *pair gate* (Line 4). The reason is that all the predecessor gates of the CNOT have been executed. After that, we divide the pair gates into two cases: *executable gates* that can be executed immediately, and *non-executable gates* that cannot be executed due to hardware constraints. If executable gates exist, we add them to the final circuit. Then we repeat this procedure until the current layout contains no more executable gates (Lines 5-9). In this manner, we reduced the number of cost calculations to resolve constrained gates.

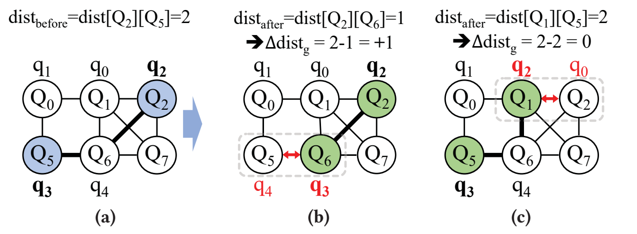
Example 2. In the example (Fig. 4c), the initial *head gates* are $\{g_0, g_1, g_3\}$. Then g_0 (g_1) is defined as a *pair gate*, because the gate simultaneously becomes the *head gate* for both q_0 (q_1) and q_2 (q_3).

To solve the connectivity problem of a non-executable gate, the proposed SWAP policy selects the best SWAP gate based on the heuristic cost. The pair of physical qubits that are connected with the distance "1" from the control (or target) qubit of the non-executable CNOT gate can be candidates for SWAP (Line 11). Then we consider the effect of the candidate SWAP gate to one CNOT as:

$$\Delta dist_g(\text{SWAP}) = dist_{before}[Q_c][Q_t] - dist_{after}[Q_c][Q_t] \quad (1)$$

where Q_c and Q_t are respectively the physical locations for the control and target qubits of non-executable CNOT gate g. The effect of the SWAP gate is calculated as the difference of the distance before and after the SWAP application. Thus, a positive value indicates that the SWAP gate solves the connection problem of a gate.

Example 3. If we consider the g_2 in the example circuit (Fig. 1a), then the first SWAP gate is selected to resolve the constraint (Fig. 6). Because the first SWAP gate has a positive effect; but using the second SWAP gate keeps the distance the same (zero effect).



dist_before=dist[Q2][Q5]=2
dist_after=dist[Q2][Q6]=1
➔Δdist_g = 2-1 = +1
dist_after=dist[Q1][Q5]=2
➔Δdist_g = 2-2 = 0
q1 q0 q2
Q0 Q1 Q2 Q5 Q6 Q7
q3 q4
(a)
(b)
(c)

Figure 6: (a) Original gate distance in the current layout. (b) Gate-effect for $SWAP(q_3, q_4)$. (c) Gate-effect for $SWAP(q_2, q_0)$.

When calculating the heuristic cost, the size of the considered gate-set affects the quality of the selected SWAP gate. Therefore, we propose an adaptive window in calculating the cost (Lines 12-13).

By applying an exponential discount factor α, we give a greater reward to solving the overhead of the *non-executable gates* than at the back of the circuit. The final heuristic cost for one SWAP is:

$$cost(\text{SWAP}) = \sum_{g' \in G} \alpha^{dist[g'][\text{SWAP}]} \cdot \Delta dist_{g'} (0 \le \alpha \le 1) \quad (2)$$

where G represents the total gates from the g to subsequent gates, and α is exponentially proportional to the distance between g' and SWAP gate. Then we choose the SWAP gate that has the highest cost among the candidates as the best gate (Line 14).

In addition, the appropriate use of BRIDGE gates can further reduce the overall gate count (Lines 15-24). During qubit mapping, the SWAP gate solves the connectivity constraints and changes the layout. However, the changed layout affects all gates in the circuit after swapping, so additional SWAP gates may be required to perform the remaining gates. Therefore, we designate the lower limit of the SWAP cost as β. If the calculated cost is not larger than β, the SWAP gate was considered to be of little use in solving the posterior connectivity problem. So, in this case, we use the BRIDGE gate as a substitute without changing the layout.

If all gates are processed, the algorithm is terminated, and *FinalCircuit* is returned (Line 26). Our proposed algorithm can efficiently reduce the search size, so the main mapping procedure is finished in short runtime regardless of the circuit size.

4 EXPERIMENTAL RESULT

This section describes the experimental setup and results. The proposed approach was implemented in C++. We performed experiments with benchmark circuits [11, 14, 18]. *IBM Q 20 Tokyo* was used for the target hardware. The evaluation considers the number of additional CNOT gates and runtime. After preliminary experiments, we empirically set $\alpha = 0.8$ and $\beta = 3$ (Algorithm 2).

4.1 Result of Proposed Qubit-Mapping

We compared our qubit-mapping method to three previous methods [14, 15, 17] for analysis of *Benchmark 1* (Table 1). Compared with SABRE [14], we reduced the number of CNOTs by 39 % and the runtime by 95 %. Compared to HA [17], we reduced the gate count by 7 % on average, and the runtime by approximately 99 %. For intuitive comparison, we also present experimental results normalized to our results (Fig. 7a). Compared to previous studies, we achieved gate reductions of 39 %, 20 %, and 3 %, and corresponding speed-ups of 21.8×, 2.5×, and 90.4×, respectively.

We also compared the proposed method with MCPE [15] with a larger size of circuits (*Benchmark 2*). Our approach reduced the

Table 1: Comparison of total qubit-mapping method (for *Benchmark 1* and *2*)

Benchmark			Circuit info.			SABRE [14]		MCPE [15]		HA [17]		Ours			Reduction	
name	*1*	*2*	n	g	d	CNOT	RT	CNOT	RT	CNOT	RT	CNOT	RT	Depth	ΔCNOT (%)	ΔRT (%)
4mod5-v1_22		V	5	21	12	-	-	**0**	**0**	-	-	**0**	**0.00**	13	0 (N/A)	0.00 (N/A)
mod5mils_65		V	5	35	21	-	-	**0**	**0**	-	-	**0**	**0.00**	22	0 (N/A)	0.00 (N/A)
alu-v0_27		V	5	36	21	-	-	**3**	**0**	-	-	**3**	**0.00**	25	0 (N/A)	0.00 (N/A)
decod24-v2_43		V	4	52	30	-	-	**0**	**0**	-	-	**0**	**0.00**	31	0 (N/A)	0.00 (N/A)
4gt13_92		V	5	66	38	-	-	21	**0**	-	-	**0**	**0.00**	39	21 (100)	0.00 (N/A)
ising_model_10	V	V	10	480	70	0	**0**	0	**0**	**0**	0.005	**0**	0.01	70	0 (N/A)	-0.01 (N/A)
ising_model_13	V	V	13	633	71	0	**0**	0	**0**	**0**	0.010	**0**	0.02	71	0 (N/A)	-0.02 (N/A)
ising_model_16	V	V	16	786	71	3	**0**	0	**0**	**0**	0.020	**0**	0.01	71	0 (N/A)	-0.01 (N/A)
qft_10	V	V	10	200	63	54	0.103	39	**0**	**36**	0.015	39	0.01	133	0 (N/A)	-0.01 (N/A)
qft_13	V	V	13	403	84	93	0.036	96	**0.015**	**78**	0.043	105	0.03	249	-9 (-9)	-0.02 (-100)
qft_16	V	V	16	512	105	186	0.084	192	**0.031**	174	0.090	**147**	0.06	291	45 (23)	-0.03 (-94)
qft_20		V	20	970	133	-	-	**360**	**0.062**	-	-	423	0.11	530	-63 (-18)	-0.05 (-77)
rd84_142		V	15	343	110	-	-	108	**0.109**	-	-	**81**	0.02	199	27 (25)	0.09 (82)
adr4_197	V	V	13	3439	1839	1614	0.49	1224	**0.046**	**882**	1.410	972	0.15	2948	252 (21)	-0.10 (-226)
radd_250	V	V	13	3213	1781	1275	0.48	1047	0.218	**840**	1.240	897	**0.11**	2670	150 (14)	0.11 (50)
z4_268	V	V	11	3073	1644	1365	0.44	855	0.186	801	1.130	**651**	**0.09**	2389	204 (24)	0.10 (52)
sym6_145	V	V	14	3888	2187	1272	0.56	1017	0.202	786	1.710	**606**	**0.11**	2879	411 (40)	0.09 (46)
misex1_241	V	V	15	4813	2676	1251	0.89	1098	0.249	**942**	2.570	1065	**0.15**	3841	33 (3)	0.10 (40)
rd73_252	V	V	10	5321	2867	2133	0.94	2193	0.343	1635	3.190	**1320**	**0.18**	4418	873 (40)	0.16 (48)
cycle10_2_110	V	V	12	6050	3386	2622	1.35	1968	0.348	1719	4.020	**1536**	**0.20**	5065	432 (22)	0.15 (43)
square_root_7	V	V	15	7630	3847	2598	1.5	1788	0.406	**828**	5.660	1512	**0.21**	5156	276 (15)	0.20 (48)
sqn_258	V	V	10	10223	5458	4344	3.52	3057	0.563	2712	11.700	**2277**	**0.27**	8037	780 (26)	0.29 (52)
rd84_253	V	V	12	13658	7261	6147	5.39	5697	0.892	3843	21.800	**3774**	**0.34**	11489	1923 (34)	0.55 (62)
co14_215	V	V	15	17936	8570	8982	9.51	5061	1.062	6429	36.000	**4737**	**0.47**	13571	324 (6)	0.59 (56)
sym9_193	V	V	11	34881	19235	16653	30.17	13746	2.091	11553	138.300	**9234**	**0.75**	29542	4512 (33)	1.34 (64)
* urf5_158		V	9	164416	86145	-	-	58947	9.312	-	-	**47079**	**3.47**	143792	11868 (20)	5.84 (63)
* hwb9_119		V	10	207775	116199	-	-	89355	12.909	-	-	**50427**	**4.24**	171770	38928 (44)	8.67 (67)
* urf4_187		V	11	512064	264330	-	-	168366	38.42	-	-	**119166**	**10.04**	393559	49200 (9)	28.38 (74)
Normalized to [14] with *Benchmark 1*						1	1	0.76	0.12	0.63	4.14	**0.61**	**0.046**			
Normalized to [15] with *Benchmark 2*								1	1			**0.78**	**0.32**		23%	18% (* 68%)

*: the three largest benchmarks of *Benchmark 2*; n, g, d: number of qubits and gates, and circuit depth of each benchmark circuit; CNOT, RT: number of additional CNOT gates and runtime [s] after qubit mapping; ΔCNOT: $\text{CNOT}_{[15]}$-CNOT_{Ours}; ΔRT: $\text{RT}_{[15]}$-RT_{Ours}.

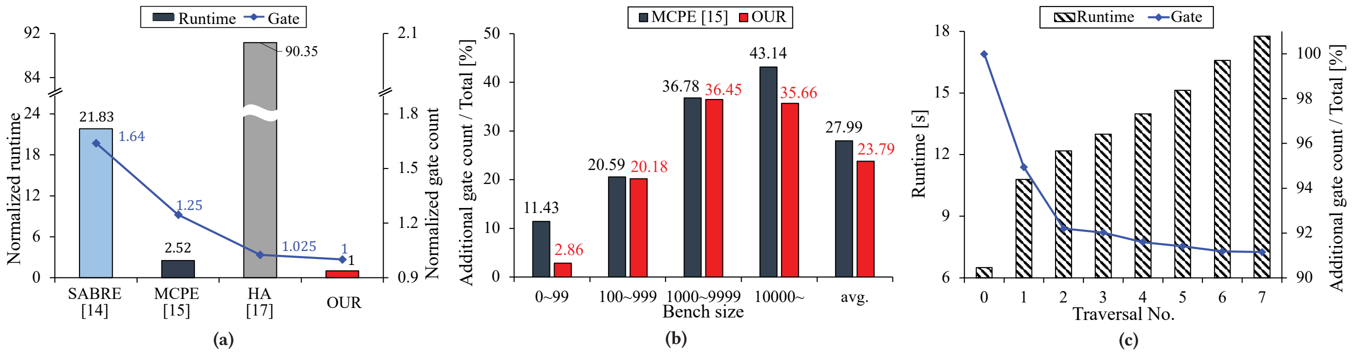


Figure 7: (a) The average runtime for big-3 benches and the number of additional gates. (b) Results of initial mapping according to the size of benchmarks. (c) Results of post optimization according to the number of forward-reverse traversals.

number of additional CNOT gates by 23 % on average and reduced the runtime by 68 % for the three largest benches. Thus, the best qubit-mapping results were found with fewer additional CNOTs at reduced runtime than previous methods. A comparison of individual steps, scalability and circuit depth will be detailed in the following subsection.

4.2 Step Analysis

4.2.1 Initial Mapping. For fair comparisons with [15], only SWAP gates were used to solve the connectivity constraints. Also, we only performed one forward traversal (without post optimization).

Our proposed algorithm can provide a scalable initial solution to the bench size concerning the overheads (Fig. 7b). For small benchmark circuits, we found an initial mapping close to the optimal solution. In addition, the reduction was 17 % on average for the three largest benches, which have more than 100,000 gates. These results imply that the number and order of gate constraints are well considered using the two interaction graphs.

4.2.2 Main Mapping. To evaluate Algorithm 2, we used the identity mapping (i.e., $\{q_i\} \mapsto \{Q_i\}$) as initial mapping. At this time, we used SWAP and BRIDGE together and did not perform post optimization.

We reduced the number of additional gates by 24 % on average, except for "*qft_10*" (1.13 % increase). These results indicate that our proposed cost function is suitable for the main mapping; so the inserted operation does not severely affect the behind of the circuit and the selection between the SWAP and BRIDGE is optimized.

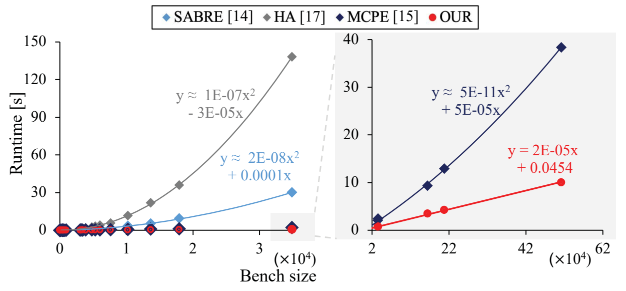
4.2.3 Post Optimization. We evaluated the effectiveness of post optimization with our proposed method. Like the previous methods [14, 15], we traversed the entire circuit 15 times (seven forward-backward traversals and one forward traversal) for post-processing.

The post-optimized results (Fig. 7c) show that our mapping can gradually reduce the additional gate counts, whereas the previous method [15] could not. Also, even if the same number of iterations was performed, the runtime for big-3 bench is decreased by 68 % on average. Therefore, the optimization procedure that uses our algorithm produces better results than the previous methods.

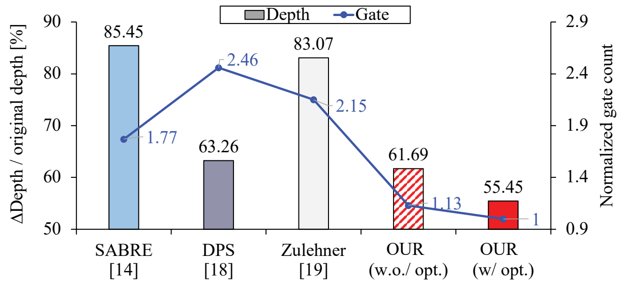
Table 2: Experimental results in exact qubit-mapping method

n	Coupling	Qiskit gate	Qiskit RT (s)	[12] gate	[12] RT (s)	[13] gate	[13] RT (s)	Ours gate	Ours RT (s)
5>	ibmqx4	13.00	-	7.48	119.86	-	-	**0.57**	**0.0053**
	LNN	-	-	-	-	-	-	4.14	0.0057
5	ibmqx4	21.76	-	9.26	175.22	7.20	<60	**2.06**	**0.0054**
	LNN	-	-	-	-	19.50	<60	**6.94**	**0.0058**

Coupling: coupling graph; gate: the average number of additional SWAP and BRIDGE gates; LNN: n-qubit linear nearest neighboring topology.



(a)



(b)

Figure 8: (a) Runtime results according to the benchmark size. (b) Results of the circuit depth and additional gate counts.

4.2.4 Comparison with Exact Method. For small-sized circuits, we conducted comparative experiments with some exact methods [12, 13] (Table 2). Based on the experiments, cost-effective results were obtained in a fairly short runtime, even if we did not focus on the exact method. Therefore, it was confirmed that the proposed heuristic search is also applicable to hardware that has limited size.

4.3 Scalability and Stability Analysis

4.3.1 Scalability. Runtime results (Fig. 8a) for each gate size show that the proposed approach achieved linear time complexity $O(g)$ with gate size g for total qubit mapping. In contrast, previous methods have a polynomial complexity of $O(g^2)$. The reason is that our graph-isomorphism search have determined an initial mapping with less overhead than the existing methods. In addition, by using the adaptive look-ahead window, we quickly selected additional gates to solve connectivity constraints.

4.3.2 Stability. Recent NISQ computers have short coherence time, so the circuit depth affects the circuit stability. Thus, we compared the circuit depth with previous works [19, 14, 18] on 23 benchmark circuits used in [18]. The depth was maintained in a reasonably low range, even without post-optimization (Fig. 8b). Moreover, we achieved depth reductions of 12.3 % and corresponding gate reductions of 2.46×, compared to DPS [18], which is the state-of-the-art depth-aware qubit-mapping method. To summarize, we can manage the circuit depth and thereby ensure operational stability.

5 CONCLUSION

In this paper, we propose a qubit-mapping method for modern quantum computers, which have limited ability to accommodate quantum circuits. For initial mapping, our recursive graph-isomorphism search produces optimal results for small quantum circuits and is algorithmically scalable for large circuits. The adaptive window search for the main mapping achieves cost-effective results in reducing the number of additional gates, while reducing runtime.

The proposed qubit-mapping method achieved a runtime that has linear time complexity, and that is drastically shorter than the previous work [17]. Compared with the state-of-the-art algorithm [15], we reduced the number of additional gates by 23 % and runtime by 68 %. In addition, by reducing the depth, the output circuit is less affected by the physical limitations of the NISQ computer.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea(NRF) grant (No.2019R1A5A1027055), and Institute of Information communications Technology Planning Evaluation (IITP) grant (No.2021-0-00754, Software Systems for AI Semiconductor Design).

REFERENCES

[1] Michael A Nielsen and Isaac Chuang. 2002. Quantum computation and quantum information. (2002).
[2] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41, 2, 303–332.
[3] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *Proc. of STOC*, 212–219.
[4] Benjamin P Lanyon, James D Whitfield, Geoff G Gillett, Michael E Goggin, Marcelo P Almeida, Ivan Kassal, Jacob D Biamonte, Masoud Mohseni, Ben J Powell, Marco Barbieri, et al. 2010. Towards quantum chemistry on a quantum computer. *Nature chemistry*, 2, 2, 106–111.
[5] Carlos A Pérez-Delgado and Pieter Kok. 2011. Quantum computers: definition and implementations. *Physical Review A*, 83, 1, 012303.
[6] Chris Lomont. 2004. The hidden subgroup problem-review and open problems. *arXiv preprint quant-ph/0411037.*
[7] IBM. 2019. *IBM Q Experience Device.* https://www.reasearch.ibm.com/ibm-q/technology/experience/ [Online].
[8] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. 2019. Quantum supremacy using a programmable superconducting processor. *Nature*, 574, 7779, 505–510.
[9] IBM. 2019. *Cramming more power into a quantum device.* https://www.ibm.com/blogs/research/2019/03/power-quantum-device/ [Online].
[10] John Preskill. 2018. Quantum computing in the nisq era and beyond. *Quantum*, 2, 79.
[11] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Caroline Collange, and Fernando Magno Quintão Pereira. 2018. Qubit allocation. In *Proc. CGO*, 113–125.
[12] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. 2019. Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations. In *Proc. DAC*. IEEE, 1–6.
[13] Toshinari Itoko, Rudy Raymond, Takashi Imamichi, and Atsushi Matsuo. 2020. Optimization of quantum circuit mapping using gate transformation and commutation. *Integration*, 70, 43–50.
[14] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proc. ASPLOS*, 1001–1014.
[15] Pengcheng Zhu, Zhijin Guan, and Xueyun Cheng. 2020. A dynamic look-ahead heuristic for the qubit mapping problem of nisq computers. *IEEE TCAD*, 39, 12, 4721–4735.
[16] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. 2014. Qubit placement to minimize communication overhead in 2d quantum architectures. In *Proc. ASPDAC*. IEEE, 495–500.
[17] Siyuan Niu, Adrien Suau, Gabriel Staffelbach, and Aida Todri-Sanial. 2020. A hardware-aware heuristic for the qubit mapping problem in the nisq era. *IEEE TQE*, 1, 1–14.
[18] Chi Zhang, Yanhao Chen, Yuwei Jin, Wonsun Ahn, Youtao Zhang, and Eddy Z Zhang. 2020. A depth-aware swap insertion scheme for the qubit mapping problem. *arXiv preprint arXiv:2002.07289.*
[19] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. An efficient methodology for mapping quantum circuits to the ibm qx architectures. *IEEE TCAD*, 38, 7, 1226–1236.
[20] James A McHugh. 1990. *Algorithmic graph theory.* Volume 68056. Citeseer.
[21] László Babai. 2016. Graph isomorphism in quasipolynomial time. In *Proc. STOC*, 684–697.