Search Space Reduction for Efficient Quantum Compilation

Amisha Srivastava
amisha.srivastava@utdallas.edu
The University of Texas at Dallas
Richardson, TX, USA

Chao Lu
chao.lu@utdallas.edu
The University of Texas at Dallas
Richardson, TX, USA

Navnil Choudhury
navnil.choudhury@utdallas.edu
The University of Texas at Dallas
Richardson, TX, USA

Ayush Arunachalam
arunachalamayush@utdallas.edu
The University of Texas at Dallas
Richardson, TX, USA

Kanad Basu
kanad.basu@utdallas.edu
The University of Texas at Dallas
Richardson, TX, USA

ABSTRACT

Quantum computers have demonstrated exponential speedup for certain computational tasks like integer factorization, molecular simulation, and machine learning, compared to the classical computers. One of the most challenging problems in quantum computing is quantum compilation, which involves the translation of a quantum circuit into a representation that adheres to the constraints imposed by the quantum hardware. However, this process of mapping the logical qubits to physical qubits incurs a significantly large search space, which needs to be analyzed to obtain the optimal mapping. A non-optimal mapping or compilation strategy introduces additional hardware overhead, thereby rendering inefficiency. Recently, researchers have proposed a technique to reduce the search space for efficient quantum compilation. However, this approach focuses on a generic solution involving only the physical architecture, and hence, as shown in our paper, often fails to incorporate the optimal solution in the reduced search space. To this end, we propose PERM and SGO (PAS), which, to the best of our knowledge, is the first quantum compilation strategy that facilitates a reduced search space comprising a more optimal solution in terms of additional CNOT gates compared to the existing technique. Our experimental evaluation using the MQT benchmarks demonstrates the efficacy of our approach, which furnishes up to 428× reduction compared to the unoptimized search space, and 57.1× reduction compared to existing research, while providing savings in terms of additional CNOT gates by up to 53.85%.

CCS CONCEPTS

• **Hardware → Software tools for EDA**.

KEYWORDS

Quantum compilation, Quantum computing, Quantum Circuit

ACM Reference Format:
Amisha Srivastava, Chao Lu, Navnil Choudhury, Ayush Arunachalam, and Kanad Basu. 2023. Search Space Reduction for Efficient Quantum Compilation. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583781.3590223

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA

ACM ISBN 979-8-4007-0125-2/23/06.
https://doi.org/10.1145/3583781.3590223

1 INTRODUCTION

Recent research demonstrates the prowess of quantum algorithms, owing to their exponential speedup over their classical counterparts in certain computational tasks, including cryptography, machine learning, and molecular simulation [3, 5, 6]. The performance improvement is attributed to the ability of quantum computers to detect multiple solutions concurrently via quantum specifications (such as superposition and entanglement). This is in contrast to classical computers, which analyze solutions sequentially [7]. For instance, Shor's Algorithm has been demonstrated to perform integer factorization in polynomial time on a quantum computer [9]. Furthermore, recent research has facilitated improved fidelity and qubit availability in quantum computers, which might potentially lead to their common-user adoption in the near future [4].

During quantum computation, it is necessary to translate the quantum circuit into a representation that adheres to the constraints imposed by the physical quantum hardware. This process is known as Quantum Compilation [10]. Specifically, it entails the mapping of logical qubits constituting the Logical Quantum Circuit (LQC) to the physical qubits present in quantum hardware. A critical issue encountered during such mapping is the lack of connectivity between the physical qubits, whose logical counterparts are connected directly. In such a scenario, quantum SWAP gates can be introduced to establish a direct connection between the logical qubits.

For example, consider Fig. 1a. As illustrated here, although the logical qubits, q_1 and q_3 are connected via a CNOT gate, a direct connectivity between their physical counterparts, P_1 and P_3 in the Physical Architecture Graph (PG) (as shown in Fig. 1b), is not feasible. In order to execute the CNOT gate, we need to incorporate a SWAP gate, which is used to interchange the positions of two logical qubits according to the permitted connections in the physical architecture. This SWAP gate can be inserted in two ways: either by switching q_1 and q_2, or by interchanging q_3 and q_2. Among these configurations, we consider the configuration with the minimum number of SWAP gates as the optimal solution. The next step is associated with updating the LQC by the insertion of a CNOT gate between the revised positions of q_1 or q_3. Afterwards, the updated LQC is considered for all subsequent quantum gate operations according to the desired physical architecture. However, inserting a SWAP gate in the LQC results in an increase in hardware overhead, along with the possible induction of additional noise [8, 13].

In order to reduce the overhead, it is imperative to minimize the number of SWAP gate insertions. To this end, optimal quantum compilation techniques, which rearrange the logical qubits in a favorable manner, are required. There are various ways to perform the aforementioned rearrangement, all of which are permutations constituting the search space (S), defined by: $S = m * n!$. Here, m and n correspond to the number of gates in the LQC and qubits in the PG, respectively. As evident, S grows appreciably with n, which could result in non-scalability and increased space complexity.

Existing state-of-the-art quantum compilation approaches utilize this unoptimized search space to determine the optimal solution, *i.e.*. the qubit mapping with the least overhead. This, in turn, leads to an enhanced time complexity [2, 10]. In order to reduce this overhead of traversing through a large search space, researchers have proposed search space reduction techniques [2]. However, this approach is limited by utilization of only the physical qubit layout of the quantum hardware and not the LQC. This leads to reduced search spaces that might not include the optimal compilation solution, as explained in Section 2.

In this paper, we propose a novel solution to this problem, PERM and SGO (PAS), which addresses the limitations of existing research. PAS, to the best of our knowledge, is the first quantum compilation strategy that generates a reduced search space containing a more optimal mapping compared to other strategies. This technique incorporates a priority rank-based approach for the aforementioned permutations. Our search space comprises permutations that generate an ideal initial mapping to obtain the minimum number of SWAP gate insertions (SGI), and thus, reduce the overhead. In our approach, the logical qubit, which is connected with the maximum number of CNOT gates, gets mapped to the physical qubit with the highest connectivity in the physical architecture of the quantum hardware. Other qubits are subsequently mapped in a similar manner. The proposed technique limits the search space, which contains the permutation that results in the least possible number of SGI. Next, in order to determine the optimal mapping, each permutation in the search space is analyzed by using a Breadth First Search (BFS) algorithm, which traverses the physical architecture graph to find the shortest path between two vertices, and SWAP gates are introduced where necessary [1]. Since we utilize both the physical architecture and the LQC, our proposed approach is efficient in producing a significant reduction over the unoptimized search space, as shown in Table 1. The proposed approach can be employed as a starting point for future quantum compilation algorithms, in order to determine the optimal solution from a reduced search space. The major contributions of our paper are:

- This paper proposes PERM and SGO (PAS), which, to the best of our knowledge, is the first strategy to generate a reduced search space containing a more optimal solution for quantum compilation in comparison to other state-of-the-art techniques.
- We perform a novel rank-based approach to generate the mapping set (*i.e.* the search space) of permutations of logical qubits mapped to physical qubits.
- We design a BFS-based algorithm to identify an optimal quantum compilation solution, in terms of SGI, from the reduced search space.

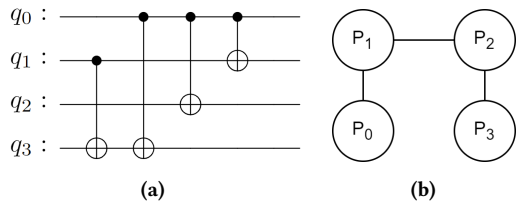


Figure 1: (a) Logical circuit and (b) 4-qubit linear architecture.

- The proposed approach, when evaluated on the MQT benchmarks, furnishes up to 428× reduction of the unoptimized search space and 57.1× reduction compared to the state-of-the-art technique. Moreover, we also obtain up to 53.85% savings in cost, in terms of additional CNOT gates.

The rest of the paper is organised as follows. Section **??** reviews prior related work. Section 2 provides a background on quantum computing and motivates our problem. Section 3 describes the proposed methodology. Section 4 evaluates the proposed approach using experiments. The paper is concluded in Section 6.

2 MOTIVATION

In this section, we provide an example to illustrate the limitations faced by the existing state-of-the-art approach that addresses search space reduction [2]. To this end, we use the LQC from Fig. 1a and physical architecture from Fig. 1b. Both architectures are composed of four qubits, which results in an unoptimized search space of 96 states. On application of the approach proposed by [2], we obtain a reduced set of search space P, consisting of 8 elements (permutations), as follows. P = {0312, 1032, 1203, 0231, 2013, 0132, 1023, 0213}, where each element (*e.g.*, 0312) represents a specific mapping strategy. For example, 0312 represents the mapping q_0-P_0, q_3-P_1, q_1-P_2, q_2-P_3. This is followed by adding permutations for every gate in the circuit. Therefore, the overall search space becomes S = 8 × m, where m is the number of gates (which is four in this example). Hence, the resulting search space has 32 elements in total. The minimum number of SWAP gates that can be obtained through these mappings is two, as shown in Fig. 2a.

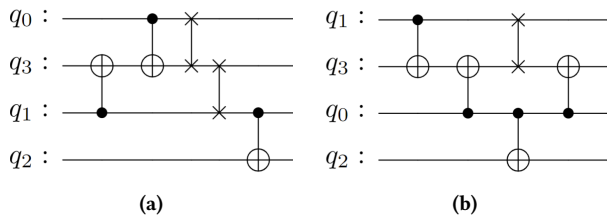


Figure 2: Mapped circuit for (a) 0312 and (b) 1302.

On the other hand, when we apply our proposed method PAS, as described in Section 3, to the same LQC and physical architecture, we obtain a reduced search space (using the PERM algorithm), denoted by P' = {1302, 2013, 3012, 1032, 2031, 2103, 3102, 2301}. This is followed by performing the SGO algorithm which adds the permutations obtained for all the possible SGIs (required to obtain the optimal solution) to the search space. In the example shown in Fig. 2b, the SWAP gate required for the fourth CNOT gate can be inserted by moving either the control qubit q_0 or the target qubit q_1,

resulting in two permutations. In this manner, the size of the search space obtained using PAS consists of 10 permutations, which is 3.2× less than [2]. Moreover, it is also 9.6× less than the unoptimized search space. The permutation 1302 in P' results in only one SWAP gate (as opposed to two for [2]), as shown in Fig. 2b. Therefore, it is evident that the reduced space obtained using PAS consists of a more optimal solution, as compared to [2].

3 PROPOSED PAS TECHNIQUE

Our proposed technique, PAS, comprises two steps: Permutation (PERM) and SWAP Gate Optimization (SGO). First, the PERM algorithm determines the number of permutations in the reduced search space. Following this, we utilize the SGO algorithm to determine the optimal number and position of SWAP gates required to execute the quantum circuit.

3.1 Permutation Algorithm (PERM)

Algorithm 1 Permutation Algorithm

Input: LQC, PG
Output: Mapping Set (MS)

1: **for** CNOT gate in LQC **do**
2: Control Qubit =Q_x
3: Target Qubit =Q_y
4: *Extract* **Gate_Coordinates** (Q_x,Q_y)
5: *Graph* **add_vertex**(Q_x)
6: *Graph* **add_vertex** (Q_y)
7: **Add_Edge** (Q_x,Q_y)
8: **end for**
9: **for** P_i in PG **do**
10: **Degree** P_i = $\sum$ *Edges of* P_i
11: **end for**
12: **for** Q_i in Graph **do**
13: **Degree** Q_i = $\sum$ *Edges of* Q_i
14: **end for**
15: **for** Q_i in Graph, P_i in PG **do**
16: MS = [Max(Degree in Q_i) → Max(Degree in P_i)]
17: **end for**
18: **return** MS

In this section, we describe the proposed PERM algorithm, which involves the mapping of frequently-used logical qubits to the physical qubits associated with a high number of adjacent connections in the PG. This approach, demonstrated in Algorithm 1, utilizes both the PG and the LQC as input arguments. This is in contrast to existing approaches, as mentioned in Section **??**, which utilize only the PG for determining the reduced search space [2]. In the PG, the physical qubits (P_i) and the connections between them correspond to vertices and edges, respectively. For the LQC, the single qubit gates are ignored, since they do not form any connections with other qubits, which is a crucial criterion used to determine the number of permutations. The first step is to define the control qubit (Q_x) and target qubit (Q_y) for each CNOT gate, as seen in lines 2 and 3 of Algorithm 1, respectively. Subsequently, each CNOT gate is extracted from the LQC and represented in a graphical form, as indicated in line 4. Following this, Q_x and Q_y are added as vertices to the LQC graph, from which the corresponding edges are generated, as indicated in lines 5 to 7. In the next steps, we determine the degree of connectivity for each qubit in both PG and LQC graph, as represented in lines 9 to 14. The degree of connectivity for a qubit (defined as the number of edges associated with that qubit/vertex), increases along with the number of connections. Based on this, we assign a rank, *i.e.*, label, to each logical and physical qubit to indicate their priority. Next, we map each logical qubit to a physical qubit of the same rank. During the first iteration, the logical and physical qubits with the highest rank (*i.e.*, 1) will get mapped to each other. This is followed by the mapping of the logical qubit in the subsequent rank (*i.e.*, 2) to its corresponding physical counterpart in the next iteration. These iterations will terminate once the mapping has been accomplished for all the qubits from each rank, as shown in lines 15-17. Moreover, if the number of logical qubits is less than the physical qubits, we add additional logical qubits, which are considered idle, *i.e.*, vertices in the LQC graph without any connections. Specifically, these idle qubits will always be paired with the physical qubits having the least connectivity. In this manner, all the possible combinations of the mapping of the logical to the physical qubits are generated, which we term as the Mapping Set (MS). This constitutes the output of the algorithm.

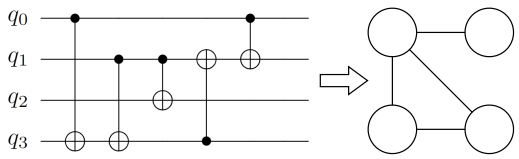


Figure 3: Logical Circuit to Graph.

We further explain our strategy using an illustrative example. The LQC in consideration is from Fig. 3 and the PG to be used is from Fig. 1b. The qubits in PG are associated with two ranks: (i) P_1 and P_2 in rank one, since they are connected with two other qubits; (ii) P_0 and P_3 in rank two, since they only have one qubit connection. The same strategy also applies for the LQC, and the degree for each of the logical qubits is determined from LQC graph, as shown in Fig. 3: degree(q_0) = 2, degree(q_1) = 3, degree(q_2) = 1, degree(q_3) = 2. The resulting order of logical qubits becomes q_1, q_0/q_3, q_2. Finally, the LQC vertex with the highest degree, q_1, gets mapped to the PG vertex with the highest degree, P_1 or P_2, and the process is continued until all qubits follow the rank-based system. Here, we only need eight different permutation groups, as opposed to the unoptimized search space, which is 120. These permutations are denoted as [3102, 2103, 0312, 2130, 3012, 2013, 0312, 2310], where the permutation [3102] is the abbreviation of the mapping strategy of [q_3-P_0, q_1-P_1, q_0-P_2, q_2-P_3]. The time complexity of the PERM algorithm is O(n), where n is number of the physical qubits. Therefore, its runtime increases at most linearly with the size of the input.

3.2 SWAP Gate Optimization Algorithm (SGO)

The second aspect of our PAS approach is to find the optimal number of SWAP gates for each of the permutations in Mapping Set (MS), generated using Algorithm 1. We call this the SGO algorithm,

Algorithm 2 SWAP Gate Optimization Algorithm

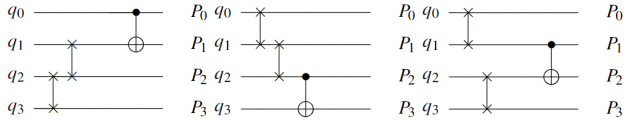
Input: LQC, PG, MS
Output: SWAP, LQC′
Initial Mapping ∈ MS

```
1:  for CNOT Gate in LQC do
2:      if Physical Qubits connected directly then
3:          Insert CNOT Gate in LQC′
4:      else
5:          n ← Distance between Physical Qubits
6:          do Swap_List(n, Q_c, Q_t )
7:          return list(Swap_Type)
8:          SWAP = Choose_Swap(Swap_Type, LQC, PG )
9:          Insert SWAP gate in LQC′
10:     end if
11: end for
12: return SWAP, LQC′
```

as shown in Algorithm 2. The inputs to Algorithm 2 are LQC, PG, and MS, obtained from Algorithm 1. The physical qubits in PG corresponding to each CNOT gate in LQC are examined for direct connectivity, as seen in line 1. If they are connected directly, the CNOT gate can be inserted in the updated LQC (LQC′), as indicated in lines 2 and 3. If any direct connection cannot be established between them, SWAP gates need to be added in LQC′ prior to inserting the CNOT gate. These SWAP gates can be inserted between the two logical qubits (control qubit or Q_c and target qubit or Q_t) in various ways, depending on the distance (n) between the corresponding physical qubits (P_x and P_y). In order to achieve this, we use the function *Swap_List*, which furnishes the list of all possible SWAP gate insertions (SGI). Q_c, Q_t, and n are provided as inputs to *Swap_List*, as shown in lines 5 and 6. Next, we define the function *Choose_swap* to identify the optimal number of SWAP gates from all the possible combinations for P_x and P_y (line 8). In this function, we provide *LQC*, *PG*, and *Swap_type* (an element obtained from *Swap_List*) as input arguments. The SWAP evaluation is performed by investigating all possible updated logical circuits, which are obtained after performing each type of SGI. The comparison of SGI is based on a list of conditions, as detailed below.

(1) Condition 1: P'_x and P'_y are not connected directly by any of the possible SGIs.
(2) Condition 2: P'_x and P'_y are connected directly by only one type of SGI.
(3) Condition 3: P'_x and P'_y are connected directly by more than one type of SGI, as follows:
   (a) Sub Condition 1: P''_x and P''_y are connected directly by only one type of SGI.
   (b) Sub Condition 2: P''_x and P''_y are connected directly by more than one type of SGI.
   (c) Sub Condition 3: P''_x and P''_y are not connected directly by any SGI.

For these conditions, we consider the $(N+1)^{th}$ CNOT gate in the circuit, with control qubit, Q'_c, and target qubit, Q'_t, where N is the position of the current CNOT gate. Also, we refer to the physical qubits mapped to Q'_c and Q'_t as P'_x and P'_y. Similarly, we also consider the $(N+2)^{th}$ CNOT gate in the circuit, associated with Q''_c, Q''_t as P''_x, and P''_y. If condition 1 is true, we perform Breadth First Search (BFS) on P'_x and P'_y, for all the types of SGI. BFS is a graph searching strategy which guarantees finding the shortest path between two nodes [1]. BFS traverses all the vertices of PG and identifies the shortest path between two vertices, which in this case are P'_x and P'_y. The selection from these SGI types is contingent upon the path length, which is obtained by performing BFS. The SGI with the minimum path length is selected. For identical path lengths between multiple SGIs, we can choose any of the available SGI. If condition 2 is satisfied, we consider that type of SGI as the best option. The sub-conditions 1 to 3 are examined if condition 3 is satisfied. If sub-condition 1 is fulfilled, the corresponding type of SGI is considered to be optimal. If sub-condition 2 is true, any one type of SGI can be selected as the best alternative. However, if sub-condition 3 is true, then we conduct BFS on P''_x and P''_y for each type of SGI that satisfies condition 3. In this scenario as well, the SGI with the minimum path length is selected. We repeat this process for each CNOT gate in the LQC, and update the circuit (LQC′) based on the optimal SGI type, and count the number of SWAP gates inserted. SGIs can be achieved in n number of ways, where n is the distance between the logical qubits involved. Since we search through all of these ways (*i.e.* all the SGIs) for the optimal solution, the permutation obtained after performing each SGI is added to the search space. Therefore, the size of the search space becomes $S = p \times n$, where p is the size of the permutation set obtained from the PERM algorithm.This can be explained using an illustrative example, as shown in Fig. 4. The LQC and PG considered here are as shown in Fig. 3 and Fig. 1b, respectively. When the iteration for LQC commences, all possible types of SGIs for the first CNOT gate are considered. The logical qubits for the gate are q_0, q_3, and the corresponding physical qubits are P_0, P_3, as illustrated in Fig. 3. Here, we observe a distance of three between P_0 and P_3 in PG, and thus, three types of SGIs are possible. In this example, P'_x and P'_y correspond to P_1 and P_3 respectively. For SGI type 2, two SWAP gates are inserted between P_0, P_1, and P_2, P_3 respectively. This, however, does not result in any connection between P_3 and P_1. In SGI type 3, SWAP gates are inserted between P_0, P_1, and P_1, P_2 respectively. This type of SGI also does not result in any connection between P_3 and P_1. On the other hand, SGI type 1 is the only one that furnishes a direct connection between P1 and P3. Thus, it satisfies condition 2 and is implemented for the first CNOT gate of the LQC. Since we search for the solution in three types of SGIs, the search space also increases by three for the first CNOT gate. This process terminates when all CNOT gates in LQC have been inserted into LQC′. Moreover, for every CNOT gate, the permutation obtained for each type of SGI is added to the search space until the process ends. The time complexity of the SGO algorithm is $O(n^2)$. However, it can be reduced to $O(nlog(n))$, if lines 6-8 are implemented using a priority queue [11].



(a) Type 1 **(b) Type 2** **(c) Type 3**

Figure 4: SGI Types.

4 RESULTS

4.1 Experimental Setup

In this section, we evaluate our proposed approach on the MQT benchmarks by utilizing the open-source JKQ toolkit designed for quantum computing [12]. Since the benchmark suite predominantly comprises circuits of five physical qubits, we have primarily considered a corresponding IBMQ London architecture (five physical qubits) for our experiments. However, our solution also accounts for scenarios where the number of logical qubits is less than their physical counterparts, such as *miller*_11 and $rd32-v1\_68$, which have 3 and 4 qubits, respectively. We utilize the IBM quantum computing platform to simulate the circuits.

4.2 Analysis of Results

In this section, we analyze the performance of the proposed PAS technique, as demonstrated in Table 1. Column 1 in the table provides details about the benchmarks used and consists of three sub-columns, with the first one referring to the name of the benchmark. The second and third sub-columns correspond to the number of qubits and gates present in the circuit, respectively. Column 2 of the table refers to the search space for each benchmark, and is further divided into three sub-columns. The first sub-column lists the number of permutations (PRM) that generate the search space for PAS, whereas the second one refers to the search space furnished by existing state-of-the-art approaches [2]. Finally, the third sub-column corresponds to the size of the unoptimized search space.

Column 3 corresponds to the reduction of the search space furnished by PAS over other techniques. It is further divided into two sub-columns, where the first one denotes the reduction in search space over [2] using Equation 1.

$$Reduction\ in\ search\ space = \frac{\text{search space size in [2]}}{\text{search space size in PAS}} \quad (1)$$

The second sub-column corresponds to the reduction in the unoptimized search space, which can be calculated by replacing the numerator in Equation 1 with the size of the unoptimized search space. Column 4 represents the number of additional CNOT gates, *i.e.*, the cost required to map the logical circuit for two strategies, and is therefore divided into two sub-columns. The first sub-column denotes the cost for our proposed method, PAS, and the second refers to cost from [2]. Finally, the last column refers to the percentage savings in cost compared to [2] when augmented by PAS strategy, as indicated by Equation 2. This cost is a measure of the number of additional CNOT gates required to map a logical circuit to the physical architecture.

$$Savings\ in\ cost = 1 - \frac{\text{cost in PAS}}{\text{cost in [2]}} \quad (2)$$

As seen from the second column of Table 1, the maximum number of permutations possible for PAS is 1162, which is appreciably less compared to the unoptimized search space as well as the state-of-the-art existing approach [2]. As observed in Table 1, PAS requires 320 permutations to map to the IBMQ London architecture in benchmark *hwb*4_49, which is significantly better compared to the unoptimized search space of 27960 permutations and 3728 permutations furnished by [2]. The values from the first sub-column in column 3 demonstrate up to 57.1× reduction in search space size for our strategy over [2]. Furthermore, it is evident from the second sub-column that for each benchmark, the reduction is at least 20.73× compared to the unoptimized search space. From these results, we can infer that Algorithm 1 is successful in significantly reducing the search space. Results from column 4 indicate that, although limited, our search space still provides a more optimal solution compared to the state-of-the-art technique. As seen from column 5 of Table 1, our additional SWAP gate cost is also minimized, when deployed in the reduced search space. For instance, the cost reduces to as low as 153 from 213 for the *hwb*4_49 circuit, compared to [2]. However, there are some cases where the cost remains the same, as seen for the benchmark qe_qft_5. The reason is attributed to the relatively small size of the circuit, when compared to other circuits, for this benchmark. Overall, our method outperforms [2] (with the least number of SWAP gates) by optimizing the cost by up to 53.85%. The reduction in number of additional CNOT gates as compared to [2] for the benchmarks indicates that Algorithm 2 successfully optimizes the cost of the circuits. Moreover, it also demonstrates the prowess of PAS in incorporating a more optimal solution in the reduced search space compared to [2]. PAS can be utilized as a starting point by other approaches, which attempt to find an optimal quantum compilation mapping strategy.

5 ACKNOWLEDGMENT

This research is supported by NSF grant #2228725.

6 CONCLUSION

This paper proposes PERM and SGO (PAS), a novel search space reduction strategy that facilitates efficient quantum compilation. Most existing approaches have primarily focused on identifying the optimal compilation strategy, without optimizing the search space. Although recent research has explored search space reduction to address this issue, the proposed approach only considers the physical architecture and obtains a less optimal solution in the reduced search space. This paper, to the best of our knowledge, is the first solution that addresses both issues by: (1) Developing a permutation algorithm, PERM, that furnishes a significant reduction in the unoptimized search space; (2) Formulating a SWAP Gate Optimization (SGO) algorithm, which identifies a reduced number of SWAP gates required to establish the quantum circuit functionality, compared to the existing methodology [2]. Our experimental analysis furnishes promising results of up to 428× reduction in unoptimized search space and 57.1× reduction over the state-of-the-art search space reduction technique [2], while incurring a savings of up to 53.85% in the number of additional CNOT gates compared to [2]. Although experimented on the IBMQ London Architecture, the proposed approach can be incorporated on any physical architecture layout. Furthermore, our solution can augment other methods that primarily focus on minimizing the number of SWAP gate insertions in quantum circuits.

REFERENCES

[1] Alan Bundy et al. 1984. Breadth-first search. In *Catalogue of artificial intelligence tools*. Springer, 13–13.
[2] Lukas Burgholzer et al. 2022. Limiting the Search Space in Optimal Quantum Circuit Mapping. In *IEEE ASP-DAC*.

Table 1: Analysis of Search Space Reduction and Additional Cost.

Benchmark			Search Space			Search Space Reduction over		Additional CNOT Gates		Savings in Cost [%]
Name	#Qubits	#Gates	#PRM in PAS	#PRM in [2]	#PRM in Unoptimized	[2]	Unoptimized	PAS	[2]	
4g11_82	5	27	38	432	3240	11.4×	85.27×	21	27	22.22
alu-v3_35	5	37	84	592	4440	7.1×	52.86×	27	30	10
3_17_13	3	36	180	576	4320	3.2×	24×	18	18	0
4gt10-v1_81	5	148	654	2368	17760	3.7×	27.16×	102	111	8.11
4_49_16	5	217	426	3472	26040	8.2×	61.13×	150	207	27.54
hwb4_49	5	233	320	3728	27960	11.7×	87.38×	153	213	28.17
mod10_171	5	244	348	3904	29280	11.3×	84.14×	177	285	37.89
mini-alu_167	5	288	1162	4608	34560	4×	29.75×	204	330	38.18
one-two-three-v0_97	5	290	1124	4640	34800	4.2×	30.97×	192	234	17.95
alu-v2_32	5	163	944	2608	19560	2.8×	20.73×	105	117	10.26
alu-v2_31	5	451	484	7216	54120	15×	111.82×	348	375	7.2
decod24-v3_45	5	150	170	2400	18000	14.2×	105.89×	102	156	34.62
aj-e11_165	5	151	240	2416	18120	10.1×	75.5×	90	129	30.23
4mod7-v1_96	5	164	186	2624	19680	14.2×	105.81×	114	123	7.32
one-two-three-v0_98	5	146	376	2336	17520	6.3×	46.6×	99	108	8.33
one-two-three-v1_99	5	132	154	2112	15840	13.8×	102.86×	90	108	16.67
4gt5_77	5	131	268	2096	15720	7.9×	58.66×	96	99	3.03
4gt13_91	5	103	196	1648	12360	8.5×	63.07×	57	93	38.71
miller_11	3	50	252	800	6000	3.2×	23.81×	27	27	0
alu-v4_36	5	115	124	1840	13800	14.9×	111.3×	66	87	24.14
4gt5_76	5	91	208	1456	10920	7×	52.5×	66	84	21.43
decod24-v1_41	5	85	160	1360	10200	8.5×	63.75×	57	84	32.14
decod24-v2_43	4	52	132	832	6240	6.4×	47.28×	27	27	0
4mod5-v1_23	5	69	88	1104	8280	12.6×	94.1×	48	66	27.27
4gt13_92	5	66	78	1056	7920	13.6×	101.54×	36	78	53.85
rd32_270	5	84	108	1344	10080	12.5×	93.34×	51	54	5.56
4mod5-v0_18	5	69	144	1104	8280	7.7×	57.5×	42	48	12.5
mod5d2_64	5	53	56	848	6360	15.2×	113.58×	33	42	21.43
alu-v1_28	5	37	84	592	4440	7.1×	52.86×	27	30	10
rd32-v1_68	4	36	40	576	4320	14.4×	108×	18	18	0
rd32-v0_66	4	34	40	544	4080	13.6×	102×	18	18	0
4gt13_90	5	107	200	1712	12840	8.6×	64.2×	60	114	47.37
mod5mils_65	5	35	34	560	4200	16.5×	123.53×	21	24	12.5
alu-v4_37	5	37	84	592	4440	7.1×	52.86×	27	30	10
one-two-three-v3_101	5	70	72	1120	8400	15.6×	116.67×	48	66	27.27
alu-v3_34	5	52	98	832	6240	8.5×	63.68×	33	51	35.29
decod24-v0_38	4	51	120	816	6120	6.8×	51×	27	27	0
qe_qft_5	5	107	30	1712	12840	57.1×	428×	9	9	0
4mod5-v0_19	5	35	34	560	4200	16.5×	123.53×	21	30	30

[3] Yudong Cao et al. 2019. Quantum chemistry in the age of quantum computing. *Chemical reviews* 119, 19 (2019), 10856–10915.

[4] Jerry Chow et al. 2021. IBM Quantum breaks the 100-qubit processor barrier. *IBM Research Blog* (2021).

[5] Carlo Ciliberto et al. 2018. Quantum machine learning: a classical perspective. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 474, 2209 (2018), 20170551.

[6] Craig Gidney et al. 2021. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum* 5 (2021), 433.

[7] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. In *ACM TC*. 212–219.

[8] Murali et al. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *ASPLOS*.

[9] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* (1999).

[10] Bochen Tan et al. 2020. Optimal layout synthesis for quantum computing. In *IEEE/ACM ICCAD*.

[11] Jean Vuillemin. 1978. A data structure for manipulating priority queues. *Commun. ACM* 21, 4 (1978), 309–315.

[12] Robert Wille et al. 2020. JKQ: JKU tools for quantum computing. In *IEEE ICCAD*.

[13] James R Wootton et al. 2021. Teaching quantum computing with an interactive textbook. In *IEEE QCE*. 385–391.