Look-ahead Schemes for Nearest Neighbor Optimization of 1D and 2D Quantum Circuits

Robert Wille\textsuperscript{1,2} Oliver Keszocze\textsuperscript{2,3} Marcel Walter\textsuperscript{3} Patrick Rohrs\textsuperscript{3} Anupam Chattopadhyay\textsuperscript{4} Rolf Drechsler\textsuperscript{2,3}
\textsuperscript{1}Institute for Integrated Circuits, Johannes Kepler University Linz, A-4040 Linz, Austria
\textsuperscript{2}Cyber Physical Systems, DFKI GmbH, 28359 Bremen, Germany
\textsuperscript{3}Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
\textsuperscript{4}Nanyang Technological University, Singapore

robert.wille@jku.at \{keszocze,mwalter,proehrs,drechsle\}@informatik.uni-bremen.de anupam@ntu.edu.sg

Abstract—Ensuring nearest neighbor compliance of quantum circuits by inserting SWAP gates has heavily been considered in the past. Here, quantum gates are considered which work on non-adjacent qubits. SWAP gates are applied in order to "move" these qubits onto adjacent positions. However, a decision how exactly the SWAPs are "moved" has mainly been made without considering the effect a "movement" of qubits may have on the remaining circuit. In this work, we propose a methodology for nearest neighbor optimization which addresses this problem by means of a look-ahead scheme. To this end, two representative implementations are presented and discussed in detail. Experimental evaluations show that, in the best case, reductions in the number of SWAP gates of 56\% (compared to the state-of-the-art methods) can be achieved following the proposed methodology.

I. INTRODUCTION

Fascinating progress is regularly being reported in the domain of quantum computation. In particular, recent accomplishments such as the realization of a highly coherent qubit [1] (the quantum-equivalent to a conventional bit) or the implementation of a scalable and fault-tolerant qubit architecture [2] received interest. Nevertheless, many obstacles prevent the realization of a practical and affordable quantum computer.

Among those, fault-tolerance stands out to be a prominent one. To address that, error correcting codes, such as Steane or surface code is widely being accepted [2]. But these, however, depend on so called nearest-neighbor interactions, i.e. allow for operations on adjacent qubits only. Furthermore, also recent progresses in two-qubit gate operations [3], quantum simulators [4], quantum error correction [5], or circuit implementations [6], [7] unanimously assume nearest-neighbor interactions. While there is an effort to enable interaction between remote qubits [8], it assumes the underlying architecture to be nearest-neighbor compliant in order to leverage on its fault-tolerance ability. Hence, how to ensure nearest neighbor interaction on qubits is an important issue in the design of quantum circuits.

In order to ease these developments, researchers recently developed various EDA methods for nearest neighbor optimization. To this end, several schemes have been proposed which insert so-called SWAP gates in order to "move" non-adjacent qubits onto adjacent positions. In [9], it has been shown that the underlying problem is computationally very expensive (in fact, exact approaches are applicable to rather small circuits only). As a consequence, researchers focused on heuristics in order to efficiently obtain a nearest neighbor, but also compact circuit (a review on related work is later provided in Section III-A). However, all these works inherit the drawback that the decision how to explicitly place the SWAP gates has mainly been made without considering the effect the resulting "movement" of qubits may have to the remaining circuit.

This work introduces a heuristic which aims for tackling that problem by additionally incorporating a look-ahead methodology. The general idea is to determine all possible options on how to "move" qubits together in order to make them adjacent. Then, the one is chosen which has the best impact to the following gates. In order to evaluate the proposed methodology, two representative implementations are presented and discussed which allow for an efficient but also meaningful estimate about the effect of applied SWAP gates.

We provide experimental validation to claim that the proposed heuristic significantly outperforms the state-of-the-art heuristic algorithms for a wide range of benchmarks. At the same time, we show that the precise implementation of the look-ahead methodology has a significant impact on what improvements can be gained. Overall, following the proposed methodology, reductions in the number of SWAP gates of 56\% can be achieved in the best case. Evaluations have been conducted on 1D as well as 2D quantum architectures.

In the following, Section II revisits the basics on nearest neighbor quantum circuits. Afterwards, the considered problem is motivated before the general idea of the proposed methodology is sketched in Section III. Section IV describes the schemes including two implemented schemes of the look-ahead methodology. Limitations as well as how to overcome those are discussed in Section V before results of our experimental evaluations are summarized in Section VI. Finally, the paper is concluded in Section VII.

II. NEAREST NEIGHBOR QUANTUM CIRCUITS

Quantum computing relies on manipulating quantum bits (qubits) rather than conventional bits. A qubit may assume any state represented by the linear superposition of the basis states $|0\rangle$ and $|1\rangle$. The manipulations are performed by applying specific unitary operations $U$. Commonly used quantum operations include the Hadamard operation $H$ (setting a qubit into superposition), the phase shift operation $S$, as well as the NOT operation $X$. Details on these operations are not relevant in the remainder of this work, but can be found e.g. in [10]. The application of a unitary operation is eventually represented by means of quantum gates, i.e. an $k$-qubit quantum gate applies a $2^k \times 2^k$ unitary matrix to the corresponding qubits. This leads to the following definition of a quantum circuit used in this work.

Definition 1. A quantum circuit is a cascade $C = g_1 g_2 \ldots g_n$ of quantum gates $g_i$, where $|C|$ denotes the total number of gates. The number of qubits is denoted by $n$. Usually, quantum circuits are composed of unary gates simply applying the respective unitary operation on a single qubit or controlled quantum gates over two qubits. In the latter case, a gate operates on a control qubit at position $c$, and a target qubit at position $t_i$.

Example 1. Consider the circuit depicted in Fig. 1(a) composed of four gates, i.e. $C = g_1 g_2 g_3 g_4$. Control qubits are denoted by a black circle, while target qubits are denoted by $U$.

Originally, qubits in a quantum circuit have been arranged in a 1-dimensional (i.e. linear) fashion where each qubit is placed next to each other. Fig. 1(a) shows an example of such a circuit. However, recent technological developments (e.g. [11], [12], [13]) also lead to the consideration of 2-dimensional arrangements where qubits are placed according to a grid-structure. In this case, the circuit from Fig. 1(a) would be realized as sketched in Fig. 1(b). Such arrangements can be extended to higher dimensions eventually leading to multi-dimensional quantum circuits.
Moreover, technological constraints for certain technologies and applications (see e.g. [2], [3]) limit the interaction distance between the qubits and, hence, only allow the application of gates between adjacent (i.e., nearest neighbor) qubits. For the 1D circuit depicted in Fig. 1(a) and the 2D circuit depicted Fig. 1(b), this only holds for gate $g_4$. All other gates have to be made nearest neighbor-compliant. To this end, so-called SWAP gates can be utilized.

**Definition 2.** A SWAP gate over two qubits $q_1$, $q_j$ transforms $(q_1, q_2, q_3, q_4)$ to $(q_2, q_1, q_3, q_4)$, i.e., simply swaps the value of the two qubits $q_1$ and $q_j$.

More precisely, nearest neighbor-compliance of a quantum circuit can be achieved by adding adjacent SWAP gates in front of each gate $g_i$ with non-adjacent control and target qubits to “move” the control (target) qubit of $g_i$ towards the target (control) qubit until they become adjacent. Afterwards, SWAP gates are added to restore the original ordering of the qubits.

**Example 2.** Fig. 2(a) exemplary shows the circuit previously considered in Fig. 1(a) which has been made nearest neighbor-compliant by inserting additional SWAP gates (SWAP gate connections are denoted by $\times$). In a similar fashion, this can be applied to the 2D circuit from Fig. 1(b) as shown in Fig. 2(b).

### III. Motivation and General Idea

Ensuring nearest neighbor compliance by inserting SWAP gates has heavily been considered in the past and is an established procedure to satisfy the underlying technological constraints. Various solutions have been proposed for this purpose. In this section, we briefly outline the related work and discuss their main drawback to be addressed in this paper. Based on this, we describe the general idea of an alternative nearest neighbor optimization methodology which overcomes this drawback.

**A. Related Work and Motivation**

According to [9], approaches for nearest neighbor optimization of quantum circuits can roughly be divided into approaches following a global reordering scheme (see e.g., [9], [14]) and approaches following a local reordering scheme (see e.g., [15], [14], [16], [17], [18], [9]). Since the local reordering scheme allows for better results (also confirmed in [9] by means of exact results), we consider this optimization scheme in the following.

Here, SWAP gates may be applied before each non-adjacent gate in order to change the order of the qubits. However, in contrast to the naive scheme sketched before by means of Fig. 2, no SWAP gates are added to restore

---

1Note that between gates $g_2$ and $g_3$ as well as between gates $g_5$ and $g_4$, the tuple of identical SWAP gates have been omitted due to space limitations.

2Note that it is of course also possible to move the qubits to (adjacent) positions outside the areas sketched in Fig. 4. But this would unnecessarily increase the (local) costs and, hence, is not further considered.
neighbor costs are defined as $nnc$. Consider the circuit depicted in Fig. 1(a). The first gate has nearest neighbor costs of $nnc$. When calculating $nnc$, the costs of establishing the considered permutation $\pi$ (i.e. the number of SWAP gates required to generate $\pi$) has to additionally be taken into account.

Following the proposed look-ahead methodology provides an alternative which may overcome the drawbacks of previously proposed solutions for nearest neighbor optimization. But details on how to apply the scheme – in particular, how to observe the following gates and how to decide for an option based on these observations – remain to be addressed. In fact, several implementations of a look-ahead methodology exists. In the following, we explicitly propose and discuss two representatives of this scheme. As the evaluation later in Section VI shows, the precise implementation will have a significant effect on what improvements can be gained from the methodology.

IV. LOOK-AHEAD SCHEMES FOR NEAREST NEIGHOR OPTIMIZATION

The following sections describe two schemes that employ different implementations of the proposed look-ahead methodology. That is, different schemes on (1) how to “look-ahead” (i.e. how to observe the following gates) and (2) how to add SWAP gates into the circuit are proposed. The first scheme follows thereby a joint consideration of the following gates, while the second schemes applies an iterative approach.

A. Joint Consideration Scheme

The main idea of the first scheme is to consider all gates $C' = g_{i+1} \ldots g_{|C|}$ following $g_i$ in a joint fashion. From the available options of how to “move” the qubits of $g_i$ together, the one is chosen which has the least negative effect to $|C'|$ (with respect to nearest neighbor costs).

To describe the algorithm in more detail, the following notation of a “circuit-tail” is applied.

Definition 4. For a given circuit $C = g_1 g_2 \ldots g_{|C|}$, a circuit-tail of $C$ after gate $g_i$ is defined as $C_i := g_{i+1}, g_{i+2}, \ldots, g_{|C|}$.

The algorithm proceeds as follows: For a given circuit $C$, iterate over all gates $g_i$. If the gate is not nearest neighbor-compliant (i.e. $nnc(g_i) > 0$), consider all possible permutations within the SWAP region (see Fig. 4) which would make the gate nearest neighbor-compliant. Next, determine the impact of all these possibilities to the circuit-tail $C_i$. The permutation that results in the smallest value of $nnc(C_i)$ is chosen and appropriate SWAP gates are added in front of $g_i$. This procedure is repeated until the end of the circuit is reached.

While doing that, two further issues have to be addressed:

- The permutation leading to the smallest value of $nnc(C_i)$ is not necessarily unique. Hence, if more than one permutation achieving the smallest value is obtained, the directly following gate $g_{i+1}$ is additionally considered.

- When calculating $nnc(C_i)$, the costs of establishing the considered permutation $\pi$ (i.e. the number of SWAP gates required to generate $\pi$) has to additionally be taken into account.

Overall, this leads to a more formal description of the scheme as follows:

1) Initialize the circuit’s permutation $\pi_C$ with the identity function.
2) Iterate over all gates $g_i \in C$. If $nnc(g_i) > 0$ then:
   a) Store all permutations $\pi$ with $nnc(\pi) = 0$ from $g_i$’s SWAP region in $\Pi$.
   b) Determine a permutation $\pi^* \in \Pi$ that minimizes the nearest neighbor costs of the circuit-tail, i.e. $\pi^* = \min_{\pi \in \Pi} \{nnc(\pi)\}$.
   c) Insert SWAP gates in front of $g_i$ that establishes the permutation $\pi^*$.
   d) Update $\pi_C$ to incorporate $\pi^*$ (i.e. the new circuit permutation is $\pi_C = \pi_C \circ \pi^*$).
3) Return the nearest neighbor-compliant circuit $C$.
is established, the algorithm continues until precisely, all gates does not decide for or against an option by jointly considering 1

Example 5. Consider again the circuit C shown in Fig. 1(a) with \( \pi_C = (q_1, q_2, q_3, q_4) \). The first gate has costs of \( \text{nnc}(g_1) = 1 \). Therefore, a permutation to make this gate nearest neighbor-compliant needs to be found. There are two options within the SWAP region that achieve this: \( \pi_1 \) that swaps \( q_1 \) and \( q_2 \) and \( \pi_2 \) that swaps \( q_4 \) and \( q_3 \). As \( \pi_1 \) would lead to costs of 1 (see Example 4) and \( \pi_2 \) would lead to costs of 1, \( \pi_2 \) is chosen. After inserting SWAP gates so that \( \pi_2 \) is established, the algorithm continues until \( g_3 \). Here the procedure is applied again eventually leading to the circuit already shown before in Fig. 5.

B. Iterative Scheme

As an alternative scheme, we propose an approach which does not decide for or against an option by jointly considering the following gates at once, but in an iterative fashion. More precisely, all gates \( g_i \) of the circuit-tail \( C_i \) following the currently considered gate \( g_i \) are considered one after another. For each gate \( g_j \), it is checked whether the positions of the qubits of \( g_j \) motivate a particular option to be applied for “moving” the qubits of gate \( g_j \). For this purpose, we can distinguish between three different cases which are sketched in Fig. 6 for 1D circuits\(^3\) (\( q_1 \) being the currently considered gate):

1) The qubits of \( q_1 \) are positioned completely outside of the SWAP region defined by gate \( g_1 \) (as sketched by gate \( g_2 \) in Fig. 6). Then, the gate \( g_j \) does not motivate a particular option of how to “move” the qubits of gate \( g_j \) and, hence, \( g_j \) is ignored.

2) The qubits of \( g_j \) are positioned completely inside of the SWAP region defined by gate \( g_i \) (as sketched by gate \( g_3 \) in Fig. 6). Then, the set of possible options to be applied for gate \( g_i \) reduces to only those permutations which reduce the costs of \( g_i \). In the sketch from Fig. 6, “moving” the target qubit of \( g_1 \) towards the control qubit is still an option (as it reduces the nearest neighbor costs for both, \( q_1 \) and \( q_3 \)), while “moving” the control qubit of \( g_1 \) towards the target qubit is not an option anymore (as this would increase the nearest neighbor costs of \( g_3 \)).

3) The qubits of \( g_j \) are positioned partially inside of the SWAP region defined by gate \( g_i \) (as sketched by gates \( g_4 \) and \( g_5 \) in Fig. 6). Here, it depends on the precise configuration of \( g_j \). In the case of gate \( g_4 \), “moving” the target qubit of \( g_1 \) towards the control qubit would be a preferred option (as, again, it reduces the nearest neighbor costs for both, \( g_1 \) and \( g_4 \)). However, this is not the case in this scenario of gate \( g_5 \). Hence, this “movement” would not be used in this case.

Eventually, this leads to the following more formal description of the scheme:

1) Initialize the circuit’s permutation \( \pi_C \) with the identity function.
2) Iterate over all gates \( g_i \in C \). If \( \text{nnc}_{g_i}(g_i) > 0 \) then
   a) Set \( \text{source} = c, \text{target} = t, S = \emptyset, j = i + 1 \)
   b) Retrieve SWAP gates making \( g_i \) nearest neighbor-compliant by applying the following sub-algorithm
      i) If either source and target are equal or nearest neighbor-compliant, return \( \emptyset \)
      ii) If the end of the circuit is reached, choose a direct permutation from source to target, and return its corresponding SWAP gates.
   c) Let \( S \) be the permutation established by \( g^*_i \).
   d) Update \( \pi_C \) to incorporate the permutation \( \pi \) that corresponds to the SWAP gates retrieved from step (b) (i.e., the new circuit permutation is \( \pi \circ \pi_C \)).
3) Return the nearest neighbor-compliant circuit \( C \).

Example 6. Consider the circuit C shown in Fig. 6. The gate \( g_1 \) has costs of 1. The following gate \( g_2 \) is not within the SWAP region and, therefore, skipped. Swapping qubits \( q_1 \) and \( q_5 \) makes gate \( g_3 \) nearest neighbor-compliant as it is within the region. Therefore, the permutation \( \pi_C \) is updated to incorporate the corresponding SWAP gate. Now, two new paths are investigated. One is empty (there is no qubit between the former target of \( g_5 \) and the end of the SWAP region, i.e. \( R = \emptyset \) and the other one is between the updated target of \( g_3 \) and the control of \( g_1 \). For this path, the SWAP region does not contain any further gates. This allows to directly move \( g_1 \)’s control qubit downwards using a single SWAP gate (i.e., “consists of a single gate only”). The resulting SWAP cascade for \( g_1 \) is shown in Fig. 7.

V. LIMITATIONS OF THE LOOK-AHEAD METHODOLOGY

Following the look-ahead methodology proposed above leads to a nearest neighbor optimization approach which bases its decision how to “move” qubits of a gate \( g_i \) together on its circuit-tail \( C_i \). As motivated by the discussions and examples from above, this often proves to be beneficial. However, the general idea that the circuit-tail should be taken into account has its limitations when gates of \( C_i \) which follow rather late after \( g_i \) suddenly influence how qubits are “moved” together. This becomes particularly crucial when large circuits composed of hundreds or thousands of gates are considered. For example, it is clearly not beneficial anymore to consider e.g. a gate \( g_{1429} \) in order to evaluate a decision e.g. for gate \( g_{141} \).

In order to address that, the proposed look-ahead schemes shall only be applied to a restricted circuit-tail. To this end, we apply the concept of a circuit window:

Definition 5. For a given circuit \( C = (g_1, g_2, \ldots, g_{|C|}) \), a circuit window of size \( w \) starting at position \( i \) is defined as \( C_{i,w} = (g_{i+1}, g_{i+2}, \ldots, g_{i+w}) \).

Applying the proposed look-ahead schemes not on the entire circuit-tail \( C_i \) but on a circuit window \( C_{i,w} \) restricted by \( w \) avoids the negative impacts of gates which follow rather late after \( g_i \).
Example 7. Consider the circuit shown in Fig. 8(a) and let’s assume that the look-ahead scheme proposed in Section IV-A is applied in order to evaluate the options how to make gate $q_1$ nearest neighbor-compliant. Then, Fig. 8(b) shows the resulting circuit when the entire circuit-tail is considered. In contrast, a much cheaper circuit (as shown in Fig. 8(c)) results if only a circuit window $C_{i,2}$ restricted by $w = 2$ is considered.

Of course, an open question remains to what value to set the size $w$ of the respective circuit windows. Together with a comparison to previously proposed approaches for nearest neighbor optimization, this question is covered in the next section.

VI. EXPERIMENTAL EVALUATION

In this section, we report and discuss the most important results obtained by our experimental evaluation of the proposed look-ahead methodology for nearest neighbor optimization. For this purpose, the representative schemes as described in Section IV have been implemented in Java. Evaluations have been performed by means of benchmark circuits (taken from RevLib [19]) which also have been applied in respective related work (e.g. [14], [20]). In a first series of evaluations, we consider the effect of the window sizes to the look-ahead scheme as introduced in Section V. Afterwards, a numerical comparison to results of previously proposed solutions (namely [14], [20]) is provided. In contrast to most of the related work, we consider both, 1D and 2D architectures of quantum circuits.

All evaluations have been conducted on an Ubuntu machine with a 2.6 GHz Intel Core i5 CPU and 8 GB of main memory. All results have been obtained in negligible run-time (i.e. no experiment required significantly more than one CPU-minute) which is why, for sake of clarity, we omitted a detailed run-time discussion in the following.

A. Effect of Windows

As discussed in Section V, the window size, which is applied for the look-ahead scheme, may have a significant impact on the final result. Hence, in a first series of experiments we evaluate this effect. To this end, the approaches presented in Section IV have been applied with different window sizes. Afterwards, the respectively obtained number of added SWAP gates has been compared to the results obtained by the approaches with no consideration of windows.

Table I summarizes the results for the scheme proposed in Section IV-A. The first two columns provide the name of the respectively considered benchmark circuits (as given in RevLib [19]) as well as its total number $d$ of gates (note that unary gates are not considered here as they are irrelevant for nearest neighbor optimization). Afterwards, the results obtained by the scheme for the window sizes are reported for $d = 5, 10, 20, 30,$ and $d$ (i.e. without any window). In order to ease the evaluation, we used the configuration without the consideration of window circuits as baseline (i.e. provide absolute values). All remaining results are given in proportional relation to that baseline. Results are presented for both 1D and 2D architectures (in case of 2D architectures, the dimension of the respectively considered 2D grid is additionally provided).

The results clearly show that the consideration of window circuits has a significant effect on the number of required SWAP gates. By this, the discussions conducted in Section V are confirmed. In the best cases, improvements of up to 45% can be achieved when not all following gates are respectively considered but just a subset defined by the window size. Concerning the best window size, the results slightly differ. But overall, it can be concluded that the window sizes should be rather small – in most of the cases, no significant further improvements are observed with window constituted with more than 20 gates.

B. Comparison to Related Work

In a second series of experiments, we compared the results obtained by our approach to the state of the art. Since corresponding evaluations have been conducted focusing on 1D architectures or 2D architectures only, we consider the approach proposed in [14] for 1D architectures and proposed in [20] for 2D architectures. Both represent recently proposed methodologies for nearest neighbor architectures for the respective architectures and, hence, guarantee an up-to-date evaluation.

The results of the comparisons are summarized in Table II. Again, the first two columns provide the name of the respectively considered benchmark circuits as well as its total number $d$ of gates. Afterwards, the best results obtained by the approaches presented in Section IV as well as the best results obtained by the approaches from [14], [20] are listed – aggregated per $d$ as well as 2D architecture. Additionally, we list the proportional improvement of the proposed approaches.

As can be seen, the look-ahead methodology proposed in this work leads to substantial improvements in the majority of
In this work, we proposed a methodology for nearest neighbor optimization of quantum circuits which, in contrast to previous work, incorporates a look-ahead scheme. By this, SWAP gates – added to make a particular gate nearest-neighbor-compliant – are chosen with respect to their impact to the following gates. Two different schemes are presented and discussed as possible implementations of the proposed methodology. An experimental evaluation confirmed that the precise implementation of the proposed look-ahead methodology has a significant impact on what improvements can be gained. Overall, improvements of up to 56% have been obtained in the best case. The proposed methodology has been evaluated for 1D and 2D quantum circuits.

**ACKNOWLEDGMENTS**

This work has partially been supported by the European Union through the COST Action IC1405 and the Nanyang Technological University through grant M4081438.

---

**REFERENCES**


