Optimal SWAP Gate Insertion
for Nearest Neighbor Quantum Circuits

Robert Wille[1,2,3]   Aaron Lye[1]   Rolf Drechsler[1,2]

[1]Institute of Computer Science   [2]Cyber Physical Systems   [3]Faculty of Computer Science
University of Bremen   DFKI GmbH   Technical University Dresden
28359 Bremen, Germany   28359 Bremen, Germany   01187 Dresden, Germany
e-mail: {rwille,lye,drechsle}@informatik.uni-bremen.de

Abstract—Motivated by its promising applications e.g. for database search or factorization, significant progress has been made in the development of automated design methods for quantum circuits. But in order to keep up with recent physical developments in this domain, new technological constraints have to be considered. Limited interaction distance between gate qubits is one of the most common of these constraints. This led to the development of several strategies aiming at making a given quantum circuit nearest neighbor-compliant by inserting SWAP gates into the existing structure. Usually these strategies are of heuristic nature. In this work, we present an exact approach that enables nearest neighbor-compliance by inserting a *minimal* number of SWAP gates. Experiments demonstrate the applicability of the approach which enabled a comparison of results obtained by heuristic methods to the actual optimum.

I. INTRODUCTION

Quantum computation is an emerging technology that enables the computation of many relevant problems in less complexity than conventional computing paradigms [1]. Prominent examples how quantum circuits outperform conventional solutions include Shor's algorithm for factorization [2] and Groover's database search [3]. For these applications, the respective circuit netlists have manually been derived. But with the increasing interest in this technology, researchers also began to develop automatic solutions for the design of this kind of circuits. This led to a significant amount of contributions particularly for the automatic synthesis of corresponding circuit structures. Approaches addressing quantum circuits directly [4, 5, 6, 7, 8] or exploiting synthesis methods for reversible circuits [9, 10, 11] in combination with proper technology mapping schemes [12, 13] have been proposed for this purpose.

At the same time, the development of physical realizations for quantum computations continued. This led to new technological constraints which have to be considered by these synthesis methods. The emerge of so called *nearest neighbor* quantum circuits is one of the most common ones. The quantum gates of these circuits are restricted to work on *adjacent* circuit signals. While this can easily be achieved by inserting SWAP gates that move the respective gate signals together until they become adjacent, the precise fashion how this is accomplished has a significant effect on the overall costs of the resulting circuit. Accordingly, many approaches improving the SWAP gate insertion for nearest neighbor quantum circuits have been proposed [14, 15, 16, 17, 18]. However, almost all of them are of heuristic nature, i.e. do not guarantee an optimal solution. All this is discussed in more detail later in Section III.

In this work, we are addressing this problem. A SWAP gate insertion scheme for nearest neighbor quantum circuits is presented which keeps the actual number of SWAP gates to be inserted optimal. For this purpose, we are exploiting the deductive power of solvers for *Pseudo Boolean Optimization* (PBO). We are formulating the question which permutations of circuit signals shall be established in order to make all gates of a given quantum circuit adjacent as an satisfiability instance. Afterwards, we apply a cost function to be minimized which incorporates the respective costs of the respectively chosen permutation. By this, a PBO solver not only determines a possible SWAP insertion, but the one with the minimal number of SWAP gates.

Experiments illustrate that, considering the exponential complexity of the addressed problem, the proposed solution indeed is able to efficiently generate optimal results for many quantum circuits. Circuits composed of up to 6 circuit lines or up to 17 non-unary quantum gates can be handled. By this, we were able to prove that previously proposed (heuristic) approaches already determined the minimal number of SWAP gate insertions for some benchmarks. Nevertheless, also an example is shown where these heuristic results can still be further improved.

The remainder of this paper is structured as follows: Section II provides a basic introduction into quantum circuits and Pseudo Boolean Optimization before the SWAP gate insertion for nearest neighbor quantum circuits including existing approaches is reviewed in Section III. The proposed optimal approach is then presented in detail in Section IV and evaluated in Section V. Finally, the paper is concluded in Section VI.

II. BACKGROUND

In order to keep the paper self-contained, this section briefly reviews the basics on the considered circuit technology as well as the solving method utilized to tackle the addressed research problem.

II.A. Quantum Circuits

In contrast to conventional computation, *quantum computation* [1] works on qubits instead of bits. While bits allow binary values only, qubits may assume any superposition of them. More formally, a qubit is a two level quantum system, described by a two dimensional complex Hilbert space. The two orthogonal quantum states $|0\rangle \equiv \binom{1}{0}$ and $|1\rangle \equiv \binom{0}{1}$ are used to represent the Boolean values 0 and 1. Any state of a qubit may be written as $|x\rangle = \alpha |0\rangle + \beta |1\rangle$, where α and β are complex numbers with $|\alpha|^2 + |\beta|^2 = 1$.

Operations on n-qubits states are performed through multiplication of appropriate $2^n \times 2^n$ unitary matrices. Thus, each quantum computation is inherently reversible but manipulates qubits rather than pure logic values. At the end of the computation, a qubit can be measured. Then, depending on the current state of the qubit, either a 0 (with probability of $|\alpha|^2$) or a 1 (with probability of $|\beta|^2$) returns. After the measurement the state of the qubit is destroyed.

<div style="float:left; width:48%">

TABLE I
QUANTUM GATES

Hadamard gate	Pauli-X gate
$\boxed{H}$ $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}$	$\oplus$ $\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$
Pauli-Y gate	Pauli-Z gate
$\boxed{Y}$ $\begin{pmatrix}0 & -i\\i & 0\end{pmatrix}$	$\boxed{Z}$ $\begin{pmatrix}1 & 0\\0 & -1\end{pmatrix}$
V gate	V† gate
$\boxed{V}$ $\frac{1+i}{2}\begin{pmatrix}1 & -i\\-i & 1\end{pmatrix}$	$\boxed{V^\dagger}$ $\frac{1-i}{2}\begin{pmatrix}1 & i\\i & 1\end{pmatrix}$

$|1\rangle = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ $-|1\rangle = \left(\begin{smallmatrix}0\\-1\end{smallmatrix}\right)$ $\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix}-1\\1\end{smallmatrix}\right)$

$|1\rangle = \left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ $\left(\begin{smallmatrix}0\\1\end{smallmatrix}\right)$ $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right)$
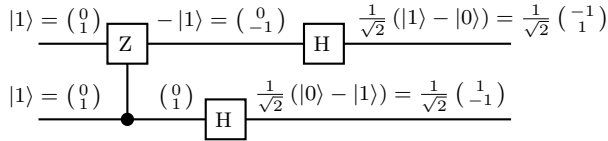
Fig. 1. Quantum circuit

Quantum computations are usually represented by *quantum circuits*. Here, the respective qubits are denoted by solid *circuit lines*. Operations are represented by *quantum gates*. Table I lists common quantum gates together with the corresponding unitary matrices describing their operation. In order to perform operations on more than one qubit, *controlled quantum gates* are applied. These gates are composed of a *target line* $|t\rangle$ and a control line $|c\rangle$ and realize the unitary operation represented by the matrix

$$M = \begin{pmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & & \\0 & 0 & & U\end{pmatrix}$$

where U denotes the operation applied to the target line. That is, if $|c\rangle = |0\rangle$ all states remain unchanged and if $|c\rangle = |1\rangle$ the operation U is applied to the target line $|t\rangle$. In all other cases, the vector $(\alpha_{|c\rangle}\alpha_{|t\rangle}, \alpha_{|c\rangle}\beta_{|t\rangle}, \beta_{|c\rangle}\alpha_{|t\rangle}, \beta_{|c\rangle}\beta_{|t\rangle})$ is applied to M. In the remainder of this work, we use the following formal notation:

Definition 1 *A quantum circuit is denoted by the cascade $G = g_1 g_2 \ldots g_{|G|}$, where $|G|$ denotes the total number of gates. The number of qubits and, thus, the number of circuit lines is denoted by n. The* costs *of a quantum circuit are defined by the number $|G|$ of gates.*

Example 1 *Fig. 1 shows a quantum circuit composed of $n = 2$ circuit lines and $|G| = 3$ gates. This circuit gets $|11\rangle$ as input and transforms the qubits as indicated at the circuit signals.*

II.B. Pseudo Boolean Optimization

Solvers for *Boolean satsifiability* (SAT) and *pseudo-Boolean optimization* (PBO) are core technologies utilized in this work. Both problems are defined as follows:

Definition 2 *The* Boolean satisfiability problem *determines an assignment to the variables of a Boolean function $\Phi : \{0, 1\}^n \to \{0, 1\}$ such that Φ evaluates to 1 or proves that no such assignment exists. The function Φ is thereby given in*

</div>

<div style="float:right; width:48%">

Conjunctive Normal Form *(CNF). Each CNF is a conjunction of clauses where each clause is a disjunction of literals and each literal is a propositional variable or its negation.*

Definition 3 *The* pseudo-Boolean optimization problem *determines a satisfying solution for a pseudo-Boolean function $\Psi : \{0, 1\}^n \to \{0, 1\}$ which – at the same time – minimizes an objective function $\mathcal{F}$. The* pseudo-Boolean function Ψ *is thereby a conjunction of constraints defined by $\sum_{i=1}^{n} c_i \dot{x}_i \geq c_n$, where $c_1 \ldots, c_n \in \mathbb{Z}$ and $\dot{x}_i$ either is a positive or a negative literal. The* objective function $\mathcal{F}$ *is defined by $\mathcal{F}(x_1, \ldots, x_n) = \sum_{i=1}^{n} m_i \dot{x}_i$ with $m_1, \ldots, m_n \in \mathbb{Z}$.*

Example 2 *Let $\Phi = (x_1 + x_2 + \overline{x}_3)(\overline{x}_1 + x_3)(\overline{x}_2 + x_3)$. Then, $x_1 = 1, x_2 = 1$, and $x_3 = 1$ is a satisfying assignment solving the SAT problem.*

Accordingly, let $\Psi = (2x_1 + 3x_2 + \overline{x}_3 \geq 3)(2x_1 + x_2 \geq 2)$ and $\mathcal{F} = x_1 + x_2 + x_3$. Then, $x_1 = 1, x_2 = 0$, and $x_3 = 0$ is a solution to the PBO problem, satisfying Ψ and, at the same time, minimizing $\mathcal{F}$.

Both, SAT and PBO, are well investigated problems. In the past efficient solving algorithms (so called *SAT solvers* or *PBO solvers*, respectively) have been proposed (see e.g. [19, 20]). Instead of simply traversing the complete space of assignments, intelligent decision heuristics, powerful learning schemes, and efficient implication methods are thereby applied. In case of PBO, it is also common to translate the respective instance into a sequence of SAT instances in order to efficiently determine a solution. In the following, we apply these techniques as black boxes delivering a solution for the proposed problem formulation.

III. OPTIMIZING SWAP GATE INSERTION FOR NEAREST NEIGHBOR QUANTUM CIRCUITS

In the past, synthesis of quantum circuits has intensely been considered. While first quantum algorithms such as Grover's Search [3] or Shor's Algorithm [2] have manually been mapped into a circuit netlist, in the meantime also automatic synthesis approaches have been presented [4, 5, 6, 7, 8]. As every quantum circuit inherently is reversible, synthesis methods for reversible circuits [9, 10, 11] in combination with proper mapping methods [12, 13] are utilized for this purpose, too. However, these approaches usually assume a rather general quantum circuit model such as reviewed in Section II.A.

At the same time, new technological constraints emerged that should be considered by circuit designers and synthesis tools. Among the different technological constraints, the limited interaction distance between gate qubits is one of the most common ones. They are motivated by technologies such as ion traps [21], nitrogen-vacancy centers in diamonds [22], quantum dots emitting linear cluster states linked by linear optics [23], laser manipulated quantum dots in a cavity [24], and superconducting qubits [25] and assume that computations are only to be performed between adjacent (i.e. nearest neighbor) signals.

Accordingly, synthesis schemes addressing the nearest neighbor restriction have been introduced recently. While for specific quantum circuits such as quantum Fourier transformation [26], Shor's Algorithm [27], quantum addition [28], or error correction circuits [29] specialized realizations exist, general quantum circuits are usually made nearest neighbor-compliant by following a post-synthesis optimization scheme: First, the desired quantum functionality is realized using e.g. one of the synthesis approaches mentioned above. Afterwards, the resulting circuit is made nearest neighbor-compliant by inserting so called *SWAP gates*.
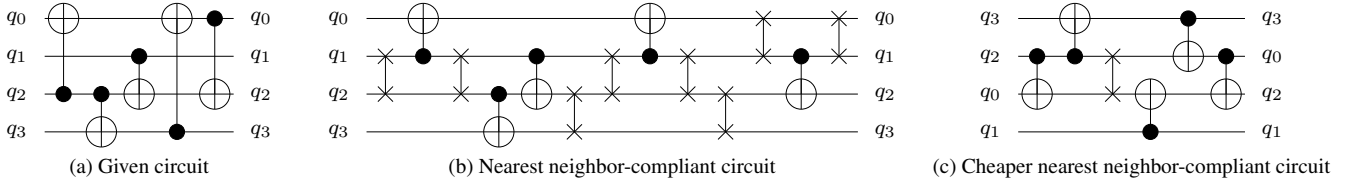
</div>

Fig. 2. Establishing nearest neighbor-compliance

Definition 4 *A* SWAP gate *is a quantum gate* $g(q_i, q_j)$ *including two qubits* q_i, q_j *and maps* $(q_0, \ldots, q_i, \ldots, q_j, \ldots, q_{n-1})$ *to* $(q_0, \ldots, q_j, \ldots, q_i, \ldots, q_{n-1})$. *That is, a* SWAP *gate realizes the exchange of two quantum values.*

More precisely, adjacent SWAP gates are inserted before each gate g with non-adjacent circuit lines in order to "move" the control (target) line of g towards the target (control) line until they become adjacent. Afterwards, SWAP gates are inserted to restore the original ordering of circuit lines.

Example 3 *Consider the circuit depicted in Fig. 2(a). As can be seen, gates* g_1, g_4, *and* g_5 *are non-adjacent. Thus, in order to make this circuit nearest neighbor-compliant, SWAP gates before and behind all these gates are inserted as shown in Fig. 2(b).*

Obviously, inserting SWAP gates increases the cost of the resulting circuits[1]. Moreover, the fashion in which SWAP gates are inserted has a significant effect: The insertion of SWAP gates as shown in Fig. 2(c) (which additionally also exploits different orders of primary inputs/outputs) leads to a much cheaper nearest neighbor-compliant circuit for Example 3. Accordingly, different approaches for optimizing the insertion of SWAP gates have been proposed in the recent past (see e.g. [14, 15, 16, 17, 18]). For this purpose, strategies such as the re-ordering of circuit lines, window-based heuristics, or mapping the problem to a corresponding graph arrangement problem were applied and evaluated. However, all of them are of heuristic nature, i.e. do not guarantee an optimal solution. An exception might be [30] in which the determination of an optimal number of SWAP gate insertion has briefly been discussed by means of an exhaustive enumeration. But this approach has clearly been shown unfeasible[2] and was just used in order to motivate the application of heuristics. Furthermore, also the approach presented in [31] guarantees optimality. But this solution additionally allows for changing the order of the gates so that the results obtained there are not comparable to the previous work discussed above.

In this work, we are aiming to *exactly* determine the best, i.e. minimal, SWAP gate insertion in order to transform a given circuit into a nearest neighbor-compliant representation. For this purpose, the deductive power of solvers for Pseudo Boolean optimization is exploited.

IV. PROPOSED SOLUTION

This section presents an exact solution to the problem sketched above. For this purpose, first the general idea is discussed before details on the precise implementation are presented.

IV.A. General Idea

Given a quantum circuit G, we are looking for a minimal insertion of SWAP gates so that all gates g of G can adjacently be executed. As illustrated above, these SWAP gates are only applied in order to appropriately permute the order of the circuit lines so that all functional gates of the circuit can be executed adjacently. Hence, in order to determine the best possible insertion of SWAP gates, one has to consider

- all possible permutations of circuit lines that, in principle, can be established before each gate g of G and

- the costs (in terms of adjacent SWAP gates) that would be needed in order to create these particular permutations.

The precise cascade of adjacent SWAP gates and, by this, the costs for creating a particular permutation of circuit lines can thereby be calculated using *inversion vectors*. For any permutation π, an inversion vector $\vec{v} = (v_0 v_1 \ldots v_{n-1})$ is defined by $v_i = |\{e \in \pi \mid e > i\}|$ and $0 \le i < n$, i.e. the ith element of $\vec{v}$ is the number of elements in π larger than i to the left of i. Inherently, this is also the number of (adjacent) SWAP operations to be performed in order to move $\pi(i)$ to i. Hence, the total amount of SWAP gates needed for creating a particular permutation of circuit lines can be calculated by summing up the respective entries in the inversion vector[3].

Example 4 *Assume a given circuit line order* $(0, 1, 2, 3)$ *shall be permuted to* $(2, 3, 1, 0)$. *The corresponding inversion vector is* $\vec{v} = (3, 2, 0, 0)$. *Hence,* $3 + 2 + 0 + 0 = 5$ *SWAP gates are required in order to create this permutation.*
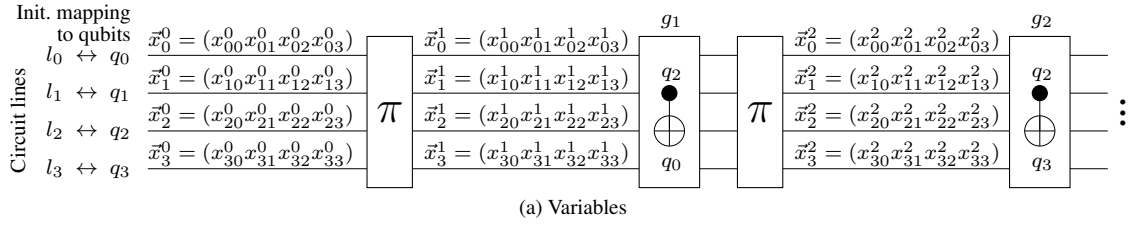
Note thereby that this does not apply in order to permute the circuit lines before the first gate, i.e. before g_1. Here, in accordance with previous work (e.g. [17, 18]), we assume the circuit lines can arbitrarily be permuted with no additional costs just by re-arranging the primary inputs as necessary.

Taking all that into account, a naive approach that ensures minimality of SWAP gate insertion would work as follows:

1. Enumerately consider all possible permutations of circuit lines for all gates of the given circuit G.

2. For each set of permutations which lead to a circuit satisfying the nearest neighbor condition, calculate the costs according to the algorithm described above.

3. After all permutations have been considered, take the one with the smallest costs.

This requires to check all possible permutations for all gates of the circuit, i.e. $n!^{|G|}$ different combinations in total. Although it was shown in [30], that it is sufficient to only consider the permutations between the respective control and target lines of each gate, this remains an exponential complexity. Naive schemes as sketched here or discussed in [30] are

---

[1]In fact, different costs calculations are therefore applied in literature: SWAP gates are either treated as elementary gates with costs of 1 or realized through a cascade of three controlled Pauli-X gates (i.e. with costs of 3).

[2]The approach was aborted for almost all benchmarks due to time constraints.

[3]The same principle has been applied in [30] in order to show that a bubblesort-algorithm generates the minimum number of SWAP gates in order to construct an arbitrary permutation.

(a) Variables

Consistency-constraints:

$$x_{00}^0 + x_{01}^0 + x_{02}^0 + x_{03}^0 = 1$$
$$\wedge x_{10}^0 + x_{11}^0 + x_{12}^0 + x_{13}^0 = 1$$
$$\wedge x_{20}^0 + x_{21}^0 + x_{22}^0 + x_{23}^0 = 1$$
$$\wedge x_{30}^0 + x_{31}^0 + x_{32}^0 + x_{33}^0 = 1$$
$$\wedge x_{00}^0 + x_{10}^0 + x_{20}^0 + x_{30}^0 = 1$$
$$\wedge x_{01}^0 + x_{11}^0 + x_{21}^0 + x_{31}^0 = 1$$
$$\wedge x_{02}^0 + x_{12}^0 + x_{22}^0 + x_{32}^0 = 1$$
$$\wedge x_{03}^0 + x_{13}^0 + x_{23}^0 + x_{33}^0 = 1$$
$$\dots$$

Adjacency-constraints (for g_1 with q_0 and q_2):

$$(x_{00}^1 \wedge x_{12}^1)$$
$$\vee (x_{10}^1 \wedge x_{22}^1)$$
$$\vee (x_{20}^1 \wedge x_{32}^1)$$
$$\vee (x_{02}^1 \wedge x_{10}^1)$$
$$\vee (x_{12}^1 \wedge x_{20}^1)$$
$$\vee (x_{22}^1 \wedge x_{30}^1)$$

Permutation-constraint (for $\pi = (2310)$ and $k = 1$)

$$(\vec{x}_0^0 = \vec{x}_2^1 \ \wedge \ \vec{x}_1^0 = \vec{x}_3^1 \ \wedge \ \vec{x}_2^0 = \vec{x}_1^1 \ \wedge \ \vec{x}_3^0 = \vec{x}_0^1) \Leftrightarrow s_{2310}^1$$

Objective function:

$$\min((0 \cdot s_{0123}^2 + 1 \cdot s_{0132}^2 + 1 \cdot s_{0213}^2 + 2 \cdot s_{0231}^2 + 2 \cdot s_{0312}^2 + 3 \cdot s_{0321}^2 +$$
$$1 \cdot s_{1023}^2 + 2 \cdot s_{1032}^2 + 2 \cdot s_{1203}^2 + 3 \cdot s_{1230}^2 + 3 \cdot s_{1302}^2 + 4 \cdot s_{1320}^2 +$$
$$2 \cdot s_{2013}^2 + 3 \cdot s_{2031}^2 + 3 \cdot s_{2103}^2 + 4 \cdot s_{2130}^2 + 4 \cdot s_{2301}^2 + 5 \cdot s_{2310}^2 +$$
$$3 \cdot s_{3012}^2 + 4 \cdot s_{3021}^2 + 4 \cdot s_{3102}^2 + 5 \cdot s_{3120}^2 + 5 \cdot s_{3201}^2 + 6 \cdot s_{3210}^2)$$
$$+ \dots)$$

(b) Constraints

Fig. 3. Resulting PBO encoding for circuit from Fig. 2(a)

infeasible due to their enumerative nature and the complexity of the problem. Hence, we propose an alternative approach which exploits the deductive power of the state-of-the-art PBO solvers described in Section II.B. Instead of naively enumerating all possible permutations, we are formulating the question which permutation shall be created as an problem of Boolean satisfiability. In addition to that, the costs of the respective permutations are incorporated in an objective function to be minimized. By this, a formulation results that can be passed to a PBO solver. From the solution of this PBO instance, the minimal SWAP insertions can be derived.

IV.B. Implementation

In order to encode the considered problem, we distinguish between the lines of a circuit G (denoted by $l_0, \dots, l_{n-1}$) and their corresponding qubits (denoted by $q_0, \dots, q_{n-1}$). Initially, each qubit corresponds to the circuit line with the same index, i.e. q_i corresponds to l_i for all $0 \leq i < n$. Then, before each gate, we allow an arbitrary permutation (including the identity) which may lead to different mappings of qubits to circuit lines. Following this, Boolean variables are introduced to the PBO encoding representing which qubit currently corresponds to which line.

Definition 5 *Let $G = g_1 g_2 \dots g_{|G|}$ be a circuit over n qubits. Then, variables $\vec{x}_i^k = (x_{i0}^k x_{i1}^k \dots x_{in-1}^k)$ with $0 \leq k < |G|$ and $0 \leq i < n$ are introduced representing which qubit corresponds to circuit line l_i initially (for $k = 0$) and before gate g_k (for $1 \leq k < |G|$). More precisely, a variable x_{ij}^k states whether qubit q_j corresponds to the circuit line l_i ($x_{ij}^k = 1$) or not ($x_{ij}^k = 0$).*

Example 5 *Consider the circuit shown in Fig. 2(a) which works as running example throughout the remainder of this section. Fig. 3 sketches the resulting PBO encoding. The π-blocks denote the positions in which we allow an arbitrary permutation of circuit lines. This leads to a new qubit mapping which is represented by the corresponding $\vec{x}_i^k$-variables in Fig. 3. Here, e.g. the assignment $x_{21}^2 = 1$ states that, before gate g_2, the qubit q_1 corresponds to circuit line l_2.*

Obviously, these mappings cannot arbitrarily be made. In fact, each circuit line must exactly correspond to one qubit and each qubit must exactly correspond to one circuit line. In order to ensure this, the following *consistency*-constraint is added to the PBO instance:

$$\bigwedge_{k=0}^{|G|-1} \left(\bigwedge_{i=0}^{n-1} (\sum_{j=0}^{n-1} x_{ij}^k = 1) \wedge \bigwedge_{i=0}^{n-1} (\sum_{j=0}^{n-1} x_{ji}^k = 1) \right)$$

The left part of this constraint states that, for each permutation position k ($0 \leq k < |G|$) and for each circuit line l_i, the sum $x_{i0}^k + x_{i1}^k + \dots + x_{in-1}^k$ is fixed to 1, i.e. exactly one qubit corresponds to one circuit line. The right part of this constraint states that, for each permutation position k ($0 \leq k < |G|$) and for each qubit q_i, the sum $x_{0i}^k + x_{1i}^k + \dots + x_{n-1i}^k$ is fixed to 1, i.e. exactly one circuit line corresponds to one qubit.

Example 6 *The bottom left of Fig. 3 sketches the consistency constraint for the example from Fig. 2(a).*

Next, we want to ensure that only permutations are applied which satisfy the nearest neighbor condition on all functional gates. As we know the control and target qubits of each gate, this can be enforced through the $\vec{x}_i^k$-variables and the following *adjacency*-constraint:

$$\bigwedge_{g_k(q_c, q_t) \in G} \left(\bigvee_{m=0}^{n-1} (x_{mc}^k \wedge x_{(m+1)t}^k) \vee \bigvee_{m=0}^{n-1} (x_{mt}^k \wedge x_{(m+1)c}^k) \right)$$

This constraint considers all gates $g_k(q_c, q_t)$ from a given circuit G with control qubit q_c and target qubit q_t. For each of these gates, a mapping of qubits to circuit lines is required so that either

- the control qubit q_c corresponds to a circuit line l_m and the target line q_t corresponds to a directly succeeding circuit line l_{m+1} (left part of the constraint) or

- the target qubit q_t corresponds to a circuit line l_m and the control line q_c corresponds to a directly succeeding circuit line l_{m+1} (right part of the constraint).

That is, one of the possible adjacencies between the respective qubits of these gates has to be established.

Example 7 *Consider again the sketch of the encoding shown in Fig. 3. The gate blocks represent the qubits which must be adjacent (derived from Fig. 2(a)). Based on that, the bottom center of Fig. 3 exemplarily shows the resulting adjacency-constraint for gate g_1 with control qubit q_2 and target qubit q_0.*

Finally, the respectively chosen permutation of circuit lines at each position has to be extracted and the corresponding costs for creating it has to be linked to the objective function of the PBO instance. Again, the $\vec{x}_i^k$-variables can be exploited for this purpose. Based on them, it can be derived what permutation is applied before gate g_k in order to change the previous circuit line order. This can be expressed by a *permutation*-constraint as follows:

$$\bigwedge_{k=1}^{|G|-1} \left(\bigwedge_{\pi \in \Pi} (\bigwedge_{i=0}^{n-1} \vec{x}_i^{k-1} = \vec{x}_{\pi(i)}^k) \Leftrightarrow s_\pi^k \right)$$

This constraint considers all possible permutations (denoted by Π) for each position k. If the assignments of $\vec{x}_i^{k-1}$ and $\vec{x}_i^k$ establish a particular permutation $\pi \in \Pi$, then a corresponding new free variable s_π^k is set to 1 (encoded through $\Leftrightarrow$). This states that this particular permutation π has been chosen before gate g_k and, hence, the corresponding costs for it have to be considered. This is eventually incorporated in the objective function

$$min(\sum_{k=1}^{|G|-1} \sum_{\pi \in \Pi} c_\pi s_\pi^k),$$

where c_π denotes the costs (in terms of adjacent SWAP gates) for creating a permutation π using the methods described in Section IV.A.

Example 8 *The permutation-constraint and the objective function for the running example from Fig. 2(a) are sketched at the bottom right of Fig. 3. In particular, the constraints for permutation $\pi = (2, 3, 1, 0)$ and $k = 2$ are shown. As already discussed in Example 4, creating this permutation requires 5 SWAP gates. Accordingly, costs of 5 are assumed in the objective function for this particular permutation. Furthermore, as it is assumed that circuit lines before gate g_1 can arbitrarily be permuted with no additional costs, all variables s_π^1 ($\pi \in \Pi$) are not part of the objective function.*

Combining all these constraints, a PBO instance results which is satisfiabile for all permutations of circuit lines that lead to a nearest neighbor-compliant circuit. The precise permutation to be created at position k can thereby be derived from the assignment to the s_π^k variables. If s_π^k has been assigned 1 by the PBO solver, a permutation π has to be created before gate g_k. By additionally optimizing the objective function, the PBO solver ensures a minimal number of SWAP gates.

Example 9 *Passing the PBO encoding presented above to a PBO solver, an optimal assignment with $s_{\pi_e}^0, s_{\pi_e}^1, s_{(0213)}^2, s_{\pi_e}^3, s_{\pi_e}^4$ set to 1 results (π_e represents the identity permutation). From that, the SWAP insertion as depicted in Fig. 2(c) can be derived. This represents an optimal solution to the SWAP insertion problem for the circuit given in Fig. 2(a).*

TABLE II
EXPERIMENTAL EVALUATION

Benchmark	n	$|G|$	$n!^{|G|}$	Swaps	Time
3_17_13	3	13	$1.3 \cdot 10^{10}$	2	0.1
4gt11_83	5	12	$8.9 \cdot 10^{24}$	6	630.1
4gt11_84	5	7	$3.6 \cdot 10^{14}$	1	8.9
4gt11-v1_85	5	7	$3.6 \cdot 10^{14}$	1	16.6
4gt13-v1_93	5	16	$1.8 \cdot 10^{33}$	6	9808.5
4mod5-v0_19	5	12	$8.9 \cdot 10^{24}$	6	489.2
4mod5-v0_20	5	8	$4.3 \cdot 10^{16}$	2	55.3
4mod5-v1_22	5	9	$5.2 \cdot 10^{18}$	3	45.5
4mod5-v1_24	5	12	$8.9 \cdot 10^{24}$	10	548.7
4mod5-v1_25	5	7	$3.6 \cdot 10^{14}$	1	11.9
alu-v0_27	5	13	$1.1 \cdot 10^{27}$	18	11705.3
alu-v1_29	5	13	$1.1 \cdot 10^{27}$	15	1685.5
alu-v4_37	5	14	$1.3 \cdot 10^{29}$	14	3669.9
decod24-v0_38	4	17	$2.9 \cdot 10^{23}$	4	23.6
decod24-v0_39	4	15	$5.0 \cdot 10^{20}$	5	19.2
decod24-v0_40	4	8	$1.1 \cdot 10^{11}$	3	0.4
decod24-v1_42	4	8	$1.1 \cdot 10^{11}$	2	0.4
decod24-v2_43	4	16	$1.2 \cdot 10^{22}$	3	7.6
decod24-v3_46	4	9	$2.6 \cdot 10^{12}$	3	0.4
graycode6_48	6	5	$1.9 \cdot 10^{14}$	0	4.5
mod5d1_63	5	11	$7.4 \cdot 10^{22}$	10	745.5
mod5d2_70	5	14	$1.3 \cdot 10^{29}$	18	3047.5
mod5mils_71	5	12	$8.9 \cdot 10^{24}$	7	735.7
QFT5	5	10	$6.2 \cdot 10^{20}$	6	2463.2
rd32-v0_66	4	12	$3.7 \cdot 10^{16}$	4	1.6
rd32-v0_67	4	8	$1.1 \cdot 10^{11}$	2	0.3
rd32-v1_68	4	12	$3.7 \cdot 10^{16}$	4	1.6
rd32-v1_69	4	8	$1.1 \cdot 10^{11}$	2	0.3

n: Number of lines $|G|$: Number of gates (does not include unary gates)
$n!^{|G|}$: Search space/complexity   Swaps: Min. number of SWAP gates
Time: Run-time in CPU seconds

V. EXPERIMENTAL EVALUATION

In this section, the results obtained with the proposed approach are presented and discussed. For this purpose, a PBO encoder has been implemented in C++ which takes a given quantum circuit and generates the PBO instance as described in the previous section. Afterwards, the PBO solver *clasp* [20] has been utilized for solving the resulting instance. To evaluate the performance of the proposed solution, quantum circuits from *RevLib* [32] as well as used in [18] have been applied. All evaluations have been conducted on an Intel E6700 Core2 CPU with 2.7 GHz and 4 GB of memory.

The results are summarized in Table II. The first three columns denote thereby the name of the considered benchmarks as well as their number n of circuit lines and their number $|G|$ of gates. Note that the latter does not include any unary gates. As unary gates are inherently nearest neighbor-compliant, they do not have to be considered for SWAP gate insertion and, thus, are ignored. Column $n!^{|G|}$ denotes the size of the search space and, by this, the precise complexity of the problem for the respective benchmark. Finally, the last two columns eventually provide the determined minimal number of SWAP gates to be inserted as well as the run-time (in CPU seconds) needed to obtain the result.

The results confirm the deductive power of the applied PBO solver that helped to determine the optimal SWAP gate insertion for many quantum circuits. In fact, circuits composed of up to 6 circuit lines or up to 17 non-unary quantum gates can be handled. In the most complex case (i.e. for *4gt13-v1_93*) $1.8 \cdot 10^{33}$ possible permutations have been considered – way too much to be tackled by an enumerative solution. Besides that, it can also be observed that the performance differs depending on the respective circuit. For example, *decod24-v2_43* and *mod5d1_63* have about the same complexity. How-

TABLE III
COMPARISON TO HEURISTIC APPROACHES

Benchmark	Minimal	[17]	[18]
3_17_13	2	12	4
4gt11_84	1	3	1
decod24-v3_46	3	4	3
QFT5	6	6	6
rd32-v0_67	2	2	2

ever, an optimal SWAP gate insertion can be determined for *decod24-v2_43* two orders of magnitude faster than for *mod5d1_63*. Such differences can be applied by the different fashions in which the PBO solver traverses the search space.

Using the proposed methodology, we were also able to prove that previously proposed (heuristic) approaches sometimes already determined the minimal number of SWAP gate insertions. Some numbers concerning this are shown in Table III (providing a comparison to some of the results from [17, 18][4]). As can be seen, particularly the approach recently presented in [18] was already able to determine minimal solutions for some benchmarks. However, minimality is not guaranteed in these approaches: As unveiled by our approach, even for the relatively less complex example *3_17_13*, a minimal solution with half the number of SWAP gate insertions exists.

VI. CONCLUSIONS

In this work, we proposed an approach for the optimal determination of SWAP gate insertions needed to make an arbitrary quantum circuit nearest neighbor-compliant. In order to handle the exponential complexity, the deductive power of PBO solvers has been exploited. That is, the given problem has been encoded as a PBO instance and, afterwards, solved by a proper solving method. Experiments confirmed the applicability of the proposed approach. By this, it was possible to compare results obtained by heuristic methods to the actual optimum. Future work focuses on determining the optimal number of SWAP gate insertions for alternative architectures, e.g. nearest neighbor quantum circuits based on 2D architectures [33] or relying on gate libraries particularly suited for nearest neighbor constraints [34].

REFERENCES

[1] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge Univ. Press, 2000.

[2] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. *Foundations of Computer Science*, pages 124–134, 1994.

[3] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Theory of computing*, pages 212–219, 1996.

[4] V. V. Shende, S. S. Bullock, and I. L. Markov. Synthesis of quantum-logic circuits. *IEEE Trans. on CAD*, 25(6):1000–1010, 2006.

[5] W.N.N. Hung, X. Song, G. Yang, J. Yang, and M. Perkowski. Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Trans. on CAD*, 25(9):1652–1663, 2006.

[6] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact synthesis of elementary quantum gate circuits. *Multiple-Valued Logic and Soft Computing*, 15(4):270–275, 2009.

[7] Mehdi Saeedi, Mona Arabzadeh, Morteza Saheb Zamani, and Mehdi Sedighi. Block-based quantum-logic synthesis. *Quant. Info. Comput.*, 11(3&4):262–277, 2011.

[8] P. Niemann, R. Wille, and R. Drechsler. Efficient synthesis of quantum circuits implementing Clifford group operations. In *ASP Design Automation Conf.*, 2014.

[9] R. Wille, D. Große, G.W. Dueck, and R. Drechsler. Reversible logic synthesis with output permutation. In *VLSI Design*, pages 189–194, 2009.

[10] M. Saeedi, M. S. Zamani, M. Sedighi, and Z. Sasanian. Synthesis of reversible circuit using cycle-based approach. *J. Emerg. Technol. Comput. Syst.*, 6(4), 2010.

[11] Mathias Soeken, Robert Wille, Christoph Hilken, Nils Przigoda, and Rolf Drechsler. Synthesis of reversible circuits with minimal lines for large functions. In *ASP Design Automation Conf.*, pages 85–92, 2012.

[12] D. M. Miller, R. Wille, and Z. Sasanian. Elementary quantum gate realizations for multiple-control Toffolli gates. In *Int'l Symp. on Multi-Valued Logic*, pages 288–293, 2011.

[13] R. Wille, M. Soeken, C. Otterstedt, and R. Drechsler. Improving the mapping of reversible circuits to quantum circuits using multiple target lines. In *ASP Design Automation Conf.*, pages 85–92, 2013.

[14] M. Mottonen and J. J. Vartiainen. Decompositions of general quantum gates. *Ch. 7 in Trends in Quantum Computing Research, NOVA Publishers, New York*, 2006.

[15] A. Chakrabarti and S. Sur-Kolay. Nearest neighbour based synthesis of quantum boolean circuits. *Engineering Letters*, 15:356–361, December 2007.

[16] Mozammel H. A. Khan. Cost reduction in nearest neighbour based synthesis of quantum boolean circuits. *Engineering Letters*, 16:1–5, 2008.

[17] M. Saeedi, R. Wille, and R. Drechsler. Synthesis of quantum circuits for linear nearest neighbor architectures. *Quant. Info. Proc.*, 2010.

[18] A. Shafaei, M. Saeedi, and M. Pedram. Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In *Design Automation Conf.*, 2013.

[19] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[20] M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In *Int'l Joint Conference on Artificial Intelligence*, pages 386–392, 2007.

[21] N. H. Nickerson, Y. Li, and S. C. Benjamin. Topological quantum computing with a very noisy network and local error rates approaching one percent. *Nat Commun*, 4:1756, 2013.

[22] N. Y. Yao, Z.-X. Gong, C. R. Laumann, S. D. Bennett, L.-M. Duan, M. D. Lukin, L. Jiang, and A. V. Gorshkov. Quantum logic between remote quantum registers. *Phys. Rev. A*, 87:022306, 2013.

[23] David A. Herrera-Marti, Austin G. Fowler, David Jennings, and Terry Rudolph. Photonic implementation for the topological cluster-state quantum computer. *Phys. Rev. A*, 82:032332, 2010.

[24] N. Cody Jones, Rodney Van Meter, Austin G. Fowler, Peter L. McMahon, Jungsang Kim, Thaddeus D. Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Phys. Rev. X*, 2:031007, 2012.

[25] M. Ohliger and J. Eisert. Efficient measurement-based quantum computing with continuous-variable systems. *Phys. Rev. A*, 85:062318, 2012.

[26] Y. Takahashi, N. Kunihiro, and K. Ohta. The quantum fourier transform on a linear nearest neighbor architecture. *Quant. Info. Comput.*, 7(4):383–391, 2007.

[27] A. G. Fowler, S. J. Devitt, and L. C. L. Hollenberg. Implementation of Shor's algorithm on a linear nearest neighbour qubit array. *Quant. Info. and Comput.*, 4:237–245, 2004.

[28] R. Van Meter and M. Oskin. Architectural implications of quantum computing technologies. *J. Emerg. Technol. Comput. Syst.*, 2(1):31–63, 2006.

[29] A. G. Fowler, C. D. Hill, and L. Hollenberg. Quantum error correction on linear nearest neighbor qubit arrays. *Phys. Rev. A*, 2004.

[30] Yuichi Hirata, Masaki Nakanishi, Shigeru Yamashita, and Yasuhiko Nakashima. An efficient method to convert arbitrary quantum circuits to ones on a linear nearest neighbor architecture. In *International Conference on Quantum, Nano and Micro Technologies*, pages 26–33, Washington, DC, USA, 2009. IEEE Computer Society.

[31] A. Matsuo and S. Yamashita. Changing the gate order for optimal lnn conversion. In *Workshop on Reversible Computation*, pages 89–101, 2011.

[32] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: an online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, pages 220–225, 2008. RevLib is available at http://www.revlib.org.

[33] B.-S. Choi and R. Van Meter. A $\theta(\sqrt{n})$-depth quantum adder on the 2D NTC quantum computer architecture. *J. Emerg. Technol. Comput. Syst.*, 8(3):24, 2012.

[34] Z. Sasanian, R. Wille, and D. M. Miller. Realizing reversible circuits using a new class of quantum gates. In *Design Automation Conf.*, pages 36–41, 2012.

[4]As heuristics do not guarantee optimality, their run-time is usually negligible and, hence, not explicitly reported.