Using πDDs for Nearest Neighbor Optimization of Quantum Circuits

Robert Wille[1,2], Nils Quetschlich[3],
Yuma Inoue[4], Norihito Yasuda[4], and Shin-ichi Minato[4]

[1]Institute for Integrated Circuits, Johannes Kepler University Linz, Austria
[2]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
[3]University of Bremen, 28359 Bremen, Germany
[4]Hokkaido University, Japan
robert.wille@jku.at nquet@informatik.uni-bremen.de
{yuma,yasuda,minato}@ist.hokudai.ac.jp

Abstract. Recent accomplishments in the development of quantum circuits motivated research in Computer-Aided Design for quantum circuits. Here, how to consider physical constraints in general and so-called nearest neighbor constraints in particular is an objective of recent developments. Re-ordering the given qubits in a circuit provides thereby a common strategy in order to reduce the corresponding costs. But since this leads to a significant complexity, existing solutions either worked towards a single order only (and, hence, exclude better options) or suffer from high runtimes when considering all possible options. In this work, we provide an alternative which utilizes so-called πDDs for this purpose. They allow for the efficient representation and manipulation of sets of permutations and, hence, provide the ideal data-structure for the considered problem. Experimental evaluations confirm that, by utilizing πDDs, optimal or almost optimal results can be generated in a fraction of the time needed by exact solutions.

1 Introduction

Quantum computation [1] exploits quantum mechanical phenomena such as superposition, entanglement, etc. and utilizes qubits rather than conventional bits for computation. This allows for solving many practically relevant problems much faster than with conventional circuits. Prominent examples include problems such as factorization (for which Shor's algorithm [2] has been proposed) or database search (for which Groover's iteration [3] has been proposed). While first corresponding quantum circuits have been developed by hand, the design of more complex quantum functionality will require automatic methods – motivating the research in *Computer-Aided Design* (CAD) for quantum circuits. Since each quantum computation is inherently reversible, methods for the design of reversible circuits are frequently utilized for this purpose.

This led to the development of first CAD methods e.g. for the synthesis of reversible circuits [4–12], the corresponding mapping to quantum circuits [13–16], or design schemes which directly address quantum circuit synthesis [17–21]. Besides that, physical constraints and how to already consider them during the design phase has received increasing attention. In particular, the satisfaction of so-called nearest neighbor constraints was an objective of recent developments. Here, the interaction distance between the involved qubits is limited and it is required that computations are performed between adjacent, i.e. nearest neighbor, qubits only. Corresponding CAD-methods addressing this restriction have been proposed e.g. in [22–27].

In this work, we consider the *global reordering scheme* as employed in [22, 26, 27] whose main idea is to determine a qubit order which – applied through the entire circuit – yields the smallest nearest neighbor costs. This often provides the basis for further optimization steps and, hence, constitutes an important part of nearest neighbor optimization. However, since determining the best possible qubit order requires the consideration of $n!$ possible permutations (where n is the number of qubits), existing solutions either

– apply a heuristic which aims for generating a single, dedicated permutation only which, in many cases, is far from optimal or
– apply an exact approach which guarantees an optimal solution but suffers from the underlying complexity.

Motivated by this, we are considering the research question how to optimize heuristic global reordering in order to generate nearly-optimal results while, at the same time, remaining efficient. To this end, we propose the utilization of *Permutation Decision Diagrams* (πDDs, [28]) – a data-structure for the efficient representation and manipulation of sets of permutations. Using πDDs it is possible to consider all permutations at once in an efficient fashion and to subsequently reduce them with respect to the nearest neighbor constraints. This provides an ideal compromise between the existing solutions which directly worked towards a single permutation only and, hence, likely excluded better options or had to deal with an inefficient handling of the complexity. Experimental evaluations confirm the benefits of the proposed approach: In all cases, optimal or almost optimal results are generated in a fraction of the runtime needed for the exact approach.

The remainder of this work is structured as follows: Section 2 reviews the background on quantum circuits and nearest neighbor optimization, while Section 3 reviews the corresponding optimization methods. These sections build the motivation of the proposed approach whose general idea is afterwards presented in Section 4. Then, details on the solution are presented in Section 5. Finally, experimental results are reported and discussed in Section 6 and the paper is concluded in Section 7.

2 Background

In order to keep the paper self-contained, this section briefly reviews the quantum circuit model usually applied in electronic design automation and provides the background on nearest neighbor optimization.

2.1 Quantum Circuits

In contrast to conventional computation, *quantum computation* [1] works on qubits instead of bits. A *qubit* is a two level quantum system, described by a

Table 1. Quantum gates

Hadamard-Gate $-\boxed{\mathrm{H}}-$ $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$	Pauli-Y-Gate $-\boxed{\mathrm{Y}}-$ $\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}$
Pauli-X-Gate $-\boxed{\mathrm{X}}-$ $\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$	Pauli-Z-Gate $-\boxed{\mathrm{Z}}-$ $\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$
V-Gate $-\boxed{\mathrm{V}}-$ $\frac{1+i}{2}\begin{pmatrix}1 & -i\\ -i & 1\end{pmatrix}$	S-Gate $-\boxed{\mathrm{S}}-$ $\begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{2}}\end{pmatrix}$
W-Gate $-\boxed{\mathrm{W}}-$ $\frac{1}{2}\begin{pmatrix}1+\sqrt{i} & 1-\sqrt{i}\\ 1-\sqrt{i} & 1+\sqrt{i}\end{pmatrix}$	T-Gate $-\boxed{\mathrm{T}}-$ $\begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{4}}\end{pmatrix}$

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 the amplitudes 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.

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 1 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

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \multicolumn{2}{c}{\multirow{2}{*}{U}} \\ 0 & 0 & & \end{pmatrix}$$

where U denotes the operation applied to the target line. In the remainder of this work, we use the following formal notation:

Definition 1. *A quantum circuit is denoted by the cascade $G = g_1 g_2 \ldots g_{|G|}$ (in figures drawn from left to right), 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 (also denoted as* quantum cost*) are defined by the number $|G|$ of gates.*

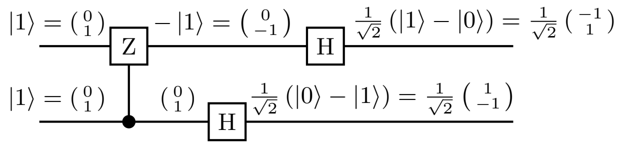

$|1\rangle = \binom{0}{1}$
Z
$-|1\rangle = \binom{0}{-1}$
H
$\frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = \frac{1}{\sqrt{2}}\binom{-1}{1}$
$|1\rangle = \binom{0}{1}$
$\binom{0}{1}$
H
$\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\binom{1}{-1}$

Fig. 1. Quantum circuit

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.

In the following, we do not focus on the dedicated functionality of a quantum circuit, but on the structure and whether it satisfies nearest neighbor constraints (as reviewed next). To this end, we omit unary quantum gates (as they are irrelevant for nearest neighbor optimization) and generically denote quantum gates using the notation $\bullet\!-\!\boxed{\mathsf{U}}$.

2.2 Nearest Neighbor Optimization

In the recent years, researchers proposed several physical realizations for quantum circuits. This led to a better understanding of their physical limitations and constraints, e.g. with respect to the interaction distance, decoherence time, or scaling (see e.g. [29–31]). Besides that, so-called *nearest neighbor constraints* have to be satisfied for many quantum circuit architectures. This particularly holds for technologies based on proposals for ion traps [32–34], nitrogen-vacancy centers in diamonds [35, 36], quantum dots emitting linear cluster states linked by linear optics [37], laser manipulated quantum dots in a cavity [38], and superconducting qubits [39, 40]. Here, nearest neighbor constraints limit the interaction distance between gate qubits and require that computations are performed between adjacent, i.e. nearest neighbor, qubits only.

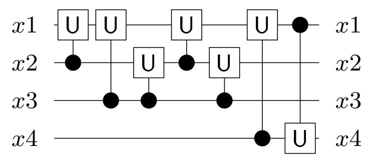
In order to formalize this restriction for electronic design automation, a corresponding metric representing the costs of a quantum circuit to become nearest neighbor compliant has been introduced in [22]. There, the authors defined the *Nearest Neighbor Cost* as follows.

Definition 2. *Assume a 2-qubit quantum gate $g(c, t)$ with a control at the line c and a target at line t where c and t are numerical indices holding $0 \leq c, t < n$. Then, the* Nearest Neighbor Cost *(NNC) for g is calculated using the distance between the target and the control line. More precisely,*

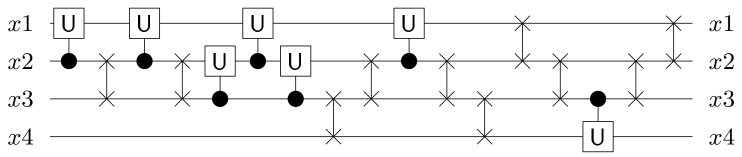
$$NNC(g) = |c - t| - 1.$$

As a result, a single control gate g is termed nearest neighbor compliant *if $NNC(g) = 0$. 1-qubit gates are assumed to have NNC of 0. The resulting NNC for a complete quantum circuit is defined by the sum of the NNC of its gates:*

$$NNC(G) = \sum_{g \in G} NNC(g).$$



(a) Given circuit



(b) Nearest neighbor compliant circuit

Fig. 2. Establishing nearest neighbor compliance

A quantum circuit G is termed nearest neighbor compliant *if $NNC(G) = 0$, i.e. if all quantum gates are 1-qubit gates or adjacent 2-qubit gates.*

Example 2. Consider the circuit G depicted in Fig. 2(a). Gates are denoted by $G = g_1 \ldots g_7$ from the left to the right. As can be seen, gates g_2, g_6, as well as g_7 are non-adjacent and have nearest neighbor costs of $NNC(g_2) = 1$, $NNC(g_6) = 2$, as well as $NNC(g_7) = 2$, respectively. Hence, the entire circuit has nearest neighbor costs of $NNC(G) = 5$.

A naive way to make an arbitrarily given quantum circuit nearest neighbor compliant is to modify it by additional SWAP gates.

Definition 3. *A* SWAP gate *is a quantum gate $g(q_i, q_j)$ including two qubits q_i, q_j and maps $(q_0, \ldots, q_i, q_j, \ldots, q_{n-1})$ to $(q_0, \ldots, q_j, q_i, \ldots, q_{n-1})$. That is, a SWAP gate realizes the exchange of two quantum values (in figures drawn using two connected $\times$ symbols).*

These SWAP gates allow for making all control lines and target lines adjacent and, by this, help to satisfy the nearest neighbor constraint. More precisely, a cascade of adjacent SWAP gates can be inserted in front of each gate g with non-adjacent circuit lines in order to shift the control line of g towards the target line, or vice versa, until they are adjacent. Afterwards, SWAP gates are inserted to restore the original ordering of circuit lines.

Example 3. Consider again the circuit depicted in Fig. 2(a). In order to make this circuit nearest neighbor compliant, SWAP gates in front and after all these gates are inserted as shown in Fig. 2(b).

3 Motivation

Adding SWAP gates in a naive fashion as reviewed in the previous section is a simple way of transforming any given quantum circuit into a nearest neighbor compliant version (in fact, this can be conducted in linear time with respect to the number of gates). But the insertion of SWAP gates obviously increases the quantum cost: For each non-adjacent gate, $2 \cdot (|t-c|-1)$ SWAP gates are additionally inserted to the circuit. In order to minimize these additional costs, researchers investigated how to reduce the number of SWAP gate insertions in order to make a given quantum circuit nearest neighbor compliant.

A broad variety of different approaches has been presented for this purpose – including solutions relying on templates [22], local and global reordering strategies [22], dedicated data-structures [23–25], etc. Also exact approaches, i.e. solutions guaranteeing the minimal number of SWAP gate insertions, have been proposed [26, 27]. The work published in [27] provides a good overview. All these approaches particularly focus on how to properly reorder the qubits in the circuit so that the respective interaction distance (and, hence, the number of required SWAP gates) is reduced.

In this work, we consider *global reordering schemes*, where the main objective is to determine a qubit order which – applied through the entire circuit – yields the smallest nearest neighbor costs. Results obtained from global reordering often provide the basis for further optimization steps and, hence, constitute an important part of nearest neighbor optimization. Unfortunately, determining the best qubit order requires the explicit checking of all possible qubit permutations. For a circuit with n qubits, this yields $n!$ possible combinations – a significant complexity. Two complementary solutions to deal with this complexity represent the current state-of-the-art:

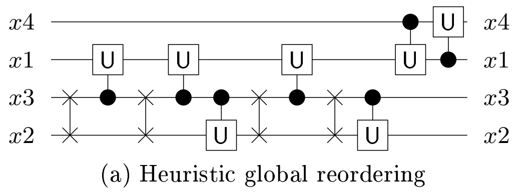
The first one is a heuristic solution proposed in [22]. Here, a good permutation is determined by calculating the *contribution* of each circuit line of a given quantum circuit G. Therefore, for each 2-qubit gate g of G with control line at position c and target line at position t, the NNC value (see Def. 2) is calculated. Afterwards, this value is added to variables imp_c and imp_t which are used to store the "impacts" of the circuit lines c and t on the total NNC, respectively. More precisely, the impact imp_i of the i^{th} circuit line ($0 \leq i < n$) is calculated by

$$imp_i = \sum_{g(c,t)\in G \ | \ c=i \ \vee \ t=i} NNC(g).$$

Using these impacts, the algorithm selects the circuit line with the greatest value and permutes it with the middle circuit line. If the selected line already is the middle line, the one with the next greatest impact is selected. This whole procedure is repeated until no further improvements are achieved.

The second one is an exact solution proposed in [26, 27], which determines the best possible permutation. To this end, the underlying design problem is formulated as an *optimal linear arrangement problem* which, in turn, is formulated as an instance of *pseudo-Boolean Optimization* (PBO, see e.g. [41]). By utilizing corresponding solving engines, the resulting PBO problem is solved.

Example 4. Consider again the circuit depicted in Fig. 2(a). Applying the heuristic of [22], the resulting impacts of the circuits lines are $imp_{x1} = 5$, $imp_{x2} = 0$, $imp_{x3} = 1$, and $imp_{x4} = 4$, respectively. Permuting the line order such that the lines with high impact are located in the middle (descending towards the outer lines) results in the circuit depicted in Fig. 3(a). Compared to the naive method (see result depicted in Fig. 2(b)), this reduces the number of required SWAP



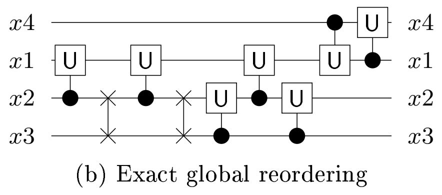


Fig. 3. Global Reordering (applied to the circuit from Fig. 2(a))

gates from 10 to 4. However, significantly further reductions can be achieved if the best permutation is determined (using the exact solution from [26, 27]). Then, a circuit as shown in Fig. 3(b) results which reduces the required number of SWAP gates by another 50% to 2.

Overall, it can be concluded that the heuristic solution provides a very efficient way of further reducing the number of SWAP gates compared to the naive method. But the obtained results are still far from optimal. In contrast, exact methods guarantee minimality with respect to the number of additionally required SWAP gates, but suffer from the significant complexity (and, hence, the resulting run-time and scalability issues). Motivated by this, we are considering the research question how to optimize heuristic global reordering in order to generate nearly-optimal results while, at the same time, remaining scalable to larger quantum circuits.

4 General Idea

Obviously, considering more permutations – ideally all $n!$ possible ones – will allow for the determination of a qubit order that is better than the one determined by the heuristic solution reviewed above. But then, the question remains how to deal with the corresponding complexity? In this work, we are proposing a scheme which utilizes the compact representation of *Permutation Decision Diagrams* (πDDs) for this purpose. In this section, we first review the basics of πDDs. Afterwards, we describe the general idea of utilizing this data-structure and illustrate its potential by means of an example.

4.1 Permutation Decision Diagrams (πDDs)

A πDD is a graph which represents a set of permutations and is based on transposition decomposition [28]. Compared to other representation relying on arrays, πDDs can represent sets of permutations more compactly. Besides that, πDDs are also capable of efficiently conducting operations on the represented sets of permutations. Before introducing the structure of πDDs in detail, we describe the decomposition of a permutation called *transposition decomposition*.

Let $\pi = \pi_1 \dots \pi_n$ be a permutation of length n. Then, π can be considered as a numerical sequence satisfying $\pi_i \in \{1 \dots, n\}$ for $1 \leq i \leq n$ and $\pi_i \neq \pi_j$ for $1 \leq i < j \leq n$. A *transposition* $\tau_{i,j}$ is a swap between two elements π_i and π_j.

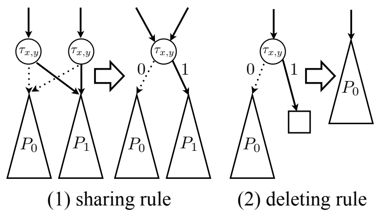


Fig. 4. Two reduction rules of πDDs

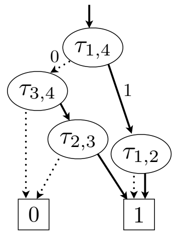


Fig. 5. A πDD repr. $\{2431, 4231, 1423\}$

Any permutation of length n can be uniquely decomposed into a sequence of at most $n-1$ transpositions by conducting the following two steps:

1. Prepare the initial permutation $1 \dots n$.
2. For each k running from n to 1, move π_k to the k-th position by applying a transposition.

Example 5. Consider a permutation $\pi = 2431$ to be decomposed. First, we start with the initial permutation 1234 and set $k = n = 4$. Since $\pi_k = 1$, we swap the first element and the fourth element ($\tau_{1,4}$) and obtain 4231. The third element 3 is at the same position as given by π, i.e. no transposition is needed for $k = 3$. Finally, since $\pi_2 = 4$ is at the first position of 4231, we swap the first element and the second element ($\tau_{1,2}$) and obtain $2431 = \pi$. By this, the given permutation $\pi = 2431$ is uniquely decomposed into a transposition sequence $\tau_{1,4}\tau_{1,2}$.

Following this transposition decomposition, a πDD is defined as follows:

Definition 4. *A πDD is a rooted and directed graph consisting of five types of components: internal nodes, 0-edges, 1-edges, the 0-sink, and the 1-sink. Fig. 5 shows an example of a πDD. Each internal node is labeled with a transposition, and has exactly two out-going edges: a 0-edge and a 1-edge. Each path from a root to the 1-sink corresponds to a permutation held by the πDD as follows: if a 1-edge originates from a node with label $\tau_{x,y}$, the decomposition of the permutation contains $\tau_{x,y}$, while a 0-edge means that the decomposition does not contain $\tau_{x,y}$.*

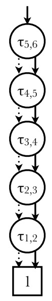
In order to make a πDD compact and canonical, we apply the following two rules called reduction rules (as illustrated in Fig. 4):

– sharing rule: share all nodes which have the same labels and child nodes.
– deleting rule: delete all nodes whose 1-edge points to the 0-sink.

Example 6. Consider the three permutations $\{2431, 4231, 1423\}$. The transposition decomposition easily shows that all these permutations can be realized by the transpositions $\tau_{1,4}\tau_{1,2}, \tau_{1,4}$, and $\tau_{3,4}\tau_{2,3}$. Hence, all of them can be represented by the πDD as shown in Fig. 5.

Although the number of πDD nodes is exponential in the length of permutations in the worst case, in many practical cases, it demonstrates a high compression ratio. For example, Fig. 6 shows an example of an exponentially compact πDD; it represents a set of 2^5 permutations with only 5 internal nodes.

A notable feature of the πDD is that it supports efficient restriction operations that make a πDD representing a restricted subset from the original πDD.



{ 123456, 123465, 123546, 123645,
124356, 124365, 125346, 126345,
132456, 132465, 132546, 132645,
142356, 142365, 152346, 162345,
213456, 213465, 213546, 213645,
214356, 214365, 215346, 216345,
312456, 312465, 312546, 312645,
412356, 412365, 512346, 612345 }

Fig. 6. A πDD with 5 nodes representing 2^5 permutations

An instance of restriction used in the following section is an adjacent restriction; it makes a πDD that only contains permutations such that two elements a and b must be adjacent in the permutation.

Since πDD operations are implemented as recursive procedures on a graph, the computation time of πDD operations depends on the number of πDD nodes, not on the cardinality of the set represented by a πDD. Hence, if a πDD is highly compressed and has a small number of nodes, manipulation on a set of permutations can be efficient.

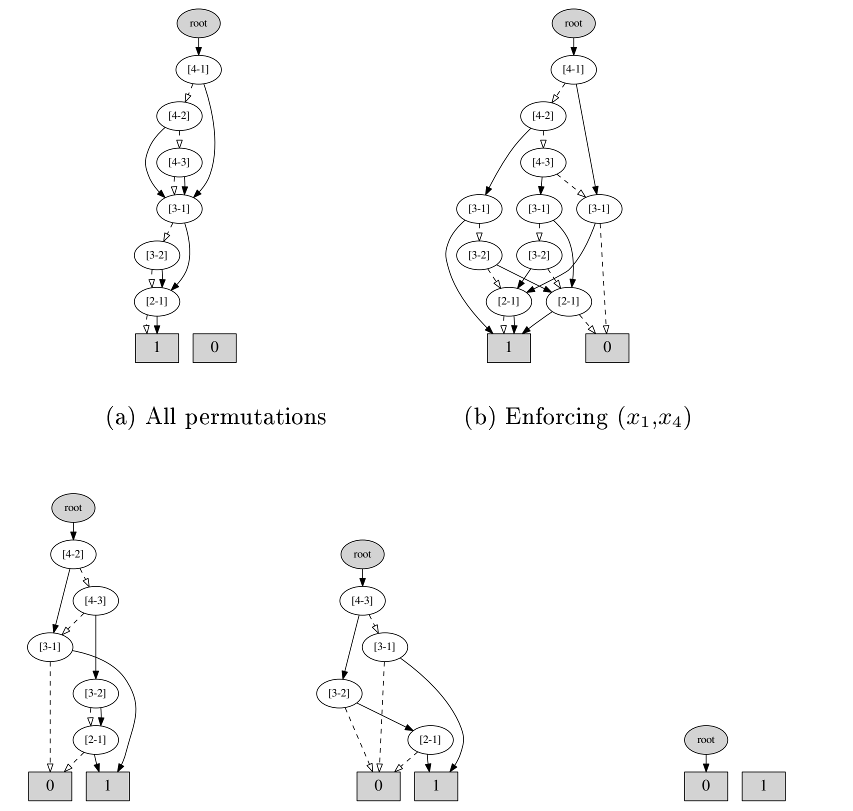
4.2 Proposed Exploitation of πDDs

The concept of πDDs allows one to efficiently represent all $n!$ possible qubit permutations at once. Based on that, the general idea of the proposed nearest neighbor optimization is to iteratively reduce this set of permutations to a subset including *efficient* permutations only. "Efficiency" is thereby defined by the number of SWAP gates that would be required in order to make a given quantum circuit – whose qubits are aligned according to these permutations – nearest neighbor compliant. Hence, permutations are removed which would clearly yield a quantum circuit with high NNC. In the following, the general idea is sketched by means of an example.

Example 7. For the quantum circuit from Fig. 2(a), a qubit order is to be determined. To this end, all $4! = 24$ possible qubit permutations are considered at the beginning. Those are efficiently represented by the πDD depicted in Fig. 7(a). Now, permutations shall be excluded which are clearly not efficient. Obviously, the interactions between qubits x_1 and x_4 dominates in the circuit from Fig. 2(a). Accordingly, we are removing all permutations in which these two qubits are *not* adjacent. This can easily be employed using πDDs and, eventually, yields to a total of 12 remaining permutations represented by the structure shown in Fig. 7(b).

In a similar fashion, further permutations can be removed. This can be continued until either

– all permutations are excluded, i.e. an empty set results, or
– no further restrictions are left to be considered.



(a) All permutations (b) Enforcing (x_1,x_4)

(c) Enforcing (x_1,x_2) (d) Enforcing (x_2,x_3) (e) Enforcing (x_1,x_3)

Fig. 7. Reducing the considered permutations using a πDD representation

In the first case, restrictions have to be loosened again – even if this would yield a quantum circuit which is not nearest neighbor compliant. After all, the representations is satisfying as many of the restrictions as possible. In the second case, no further actions are needed. From the resulting subset, the permutation leading to the lowest NNC is chosen and used in order to realize the circuit. Again, the example illustrates the issue.

Example 8. Using the subset represented by the πDD shown in Fig. 7(b), another restriction is employed, namely that qubits x_1 and x_2 shall be adjacent (this is motivated by the fact that there are 2 gates in which these two qubits interact). Applying this restriction yields the πDD shown in Fig. 7(c). Because of the same reason, x_2 and x_3 are enforced to be adjacent in the next step (yielding the πDD shown in Fig. 7(d)). Finally, x_1 and x_3 is enforced to be adjacent. However, the last restriction yields a πDD representing the empty set (see Fig. 7(e)). Because of that, this restriction is waived (i.e. we backtrack to the πDD shown in

Fig. 7(d)). As no further restrictions are left (all qubit interactions of the original circuit from Fig. 2(a) have been considered), the best permutation regarding its NNC can be calculated and taken from the resulting πDD (shown in Fig. 7(d)). This eventually yields the circuit already shown in Fig. 3(b), i.e. the proposed approach determined a permutation requiring a minimal number of SWAP gates.

Following this scheme aims for keeping permutations that satisfy certain restrictions, while excluding those which are identified as non-efficient (motivated by the interactions of qubits in the circuit). This is a clear improvement compared to the previously proposed heuristic which directly worked towards a single permutation only and, hence, likely excluded better options. The increased complexity caused by considering and manipulating (sub)sets of permutations is tackled through the efficient representation provided by the πDDs. However, the order in which restrictions are applied (and, hence, permutations are excluded) still has an effect on the determined result. The next section deals with how the proposed solution handles this ordering problem.

5 Applying Restrictions to the πDDs

As illustrated in the example from above, the interactions between the qubits provide crucial information on the nearest neighbor compliance of a given permutation. Accordingly, this information builds the basis for deciding what permutations are removed from further consideration. This section describes how this information is obtained, represented and, eventually, applied to the πDD.

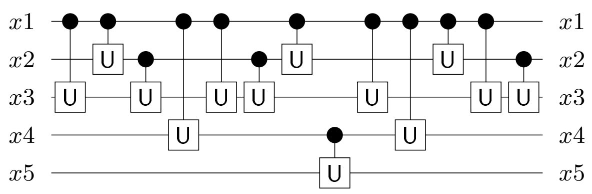
5.1 Obtaining and Weighting Restrictions

In order to store information on the interaction of the qubits (and, eventually, derive restrictions from it), a pre-process is conducted which traverses the entire circuit G. For each gate $g \in G$, the corresponding interaction between the involved qubits is determined and stored. This way, an adjacency matrix is built representing what and how many interactions between qubits are conducted. More formally:

Definition 5. *For a given quantum circuit G with n qubits, an* adjacency matrix *A of size $n \times n$ represents the number of interactions between all qubits. Each entry $a_{i,j} \in A$ contains the number of interactions between the qubits x_i and x_j and between the qubits x_j and x_i. Since no qubit interacts with itself, all entries a_{ii} are left empty. Furthermore, since A is symmetric, only half of the entries has to be considered.*

Example 9. Consider the quantum circuit from Fig. 8(a). The corresponding adjacency matrix is shown in Fig. 8(b).

From this representation, the restrictions to be applied to the πDDs can easily be derived. Each interaction between the qubits x_i and x_j motivate to restrict the set of considered permutation to only those in which x_i and x_j are adjacent. Moreover, the adjacency matrix can be used to assign a weight to each restriction. For example, if the qubits x_i and x_j interact more frequently than the qubits x_k and x_l, then the restriction of having (x_i,x_j) adjacent should be prioritized to the restriction of having (x_k,x_l) adjacent.



(a) Given circuit

	x_1	x_2	x_3	x_4	x_5
x_1	-	3	4	2	0
x_2	-	-	3	0	0
x_3	-	-	-	0	0
x_4	-	-	-	-	1
x_5	-	-	-	-	-

(b) Derived adjacency matrix

Restr.	Involved qubits	Weight
R1	(x_1,x_3)	4
R2	(x_1,x_2)	3
R3	(x_2,x_3)	3
R4	(x_1,x_4)	2
R5	(x_4,x_5)	1

(c) Derived restrictions

Fig. 8. Obtaining and weighting restrictions from a given circuit

Example 10. From the adjacency matrix shown in Fig. 8(b), restrictions and their weights as shown in Fig. 8(c) are derived.

5.2 Applying Resulting Restrictions to the πDD

In an ideal scenario, all restrictions derived above should be applied to the πDD. Then, a subset of permutations would remain in which all qubits that have interactions with each other are adjacent (eventually leading to a nearest neighbor compliant quantum circuit). However, in most of the cases this would yield an empty subset of permutations. Hence, a procedure is required deciding which restrictions are applied and which are not.

The weight which is assigned to each restriction provides an obvious metric for this purpose. But still, options exists how this metric is utilized. The following two possible schemes could be applied:

- In a *greedy* scheme, all restrictions are applied in the order of their weight. That is, the restriction with the highest weight is considered first; afterwards, the restriction with the second highest weight; and so on. This way, stronger restrictions are clearly preferred over weaker restrictions. However, there might be cases in which the application of several restrictions with a relatively small weight outperforms the application of a single restriction with a higher weight.
- This motivates the consideration of an *advanced* scheme which works as follows: First, a threshold e is defined stating the maximum number of restrictions which shall be considered together. Then, *all* possible combinations of the e restrictions with the highest weight are considered (including all e restrictions solely as well as all possible supersets of them). The combination of restrictions which can be applied to the πDD without causing an empty

set of permutations and, additionally, satisfies the highest weight is eventually chosen. Afterwards, all remaining restrictions are applied following the greedy scheme.

The two schemes are illustrated by the following example:

Example 11. Consider again the quantum circuit from Fig. 8(a) and the obtained restrictions as shown in Fig. 8(c). Following the greedy scheme would suggest an application of R1, followed by R2, R3, R4, and R5. This eventually results in a set of permutations whose best one would lead to a circuit requiring 10 SWAP gates.

In contrast, the advance scheme would consider all combinations of the $e = 4$ restrictions with the highest weight, i.e. {R1}, {R2}, {R3}, {R4}, {R1,R2}, {R1,R3}, {R1,R4}, {R2,R3}, {R2,R4}, {R3,R4}, {R1,R2,R3}, etc. This way it can be observed that {R1,R2} can be applied together but not {R1,R2,R3}. Since the combination {R1,R3,R4} (i.e. without R2) yields a higher weight than all other non-empty combinations, this set of restrictions is applied to the πDD. Eventually, this results in a quantum circuit requiring 6 SWAP gates.

6 Experimental Evaluation

The solution proposed in the previous sections has been implemented on top of the πDD-package introduced in [28]. Based on this implementation, the performance of the proposed solution has been evaluated using benchmark quantum circuits taken from *RevLib* [42]. Afterwards, the obtained results have been compared to results obtained by the previously proposed solutions reviewed in Section 3. This section summarizes and discusses the obtained results. All evaluations have been conducted on an Intel i3-4030U machine with 1.9 GHz and 4 GB of memory.

Table 2 summarizes the obtained results. The first columns provide the name of the considered benchmarks as well as its respective number of lines (n) and gates ($|G|$). Afterwards, the number of required SWAP gates are reported if the naive method (reviewed in Section 2.2), the heuristic and exact method (reviewed in Section 3), as well as the proposed method (introduced in Sections 4 and 5 and following the advanced scheme) are applied. In the case of the exact method as well as the proposed method, the required runtime (in CPU seconds) is additionally provided (both, the naive and heuristic approach where able to determine all results in negligible time, i.e. in less than a second). Finally, the last columns provide a comparison of the results obtained by the proposed approach to the respective numbers from the naive, heuristic, and exact approaches.

The results clearly confirm that the proposed approach fulfills the promises discussed in Section 3. By considering sets of permutations (rather than constructing a single one only), significantly better results compared to the previously proposed heuristic can be obtained. Improvements of more than 66% in the best case are reported. Moreover, the proposed solution is capable of generating optimal or almost optimal results (see comparison of the proposed approach to the exact solution). This quality is achieved by requiring only a fraction of the runtime needed for the exact approach thus far. That is, πDDs as utilized in this work allow for determining results of optimal or almost optimal quality while, at the same time, they handle the underlying complexity in an efficient fashion.

Table 2. Experimental evaluation

			Previously proposed solutions				Prop. solution		Comparison		
			Naive	Heuristic	Exact				(wrt. number of SWAP gates)		
Benchmark	n	$\|G\|$	SWAPs	SWAPs	SWAPs	Time	SWAPs	Time	to naive	to heuristic	to exact
3_17_13	3	13	6	6	4	0.1	6	0.1	1.00	1.00	1.50
decod24-v3_46	4	9	18	8	4	0.1	4	0.1	0.22	0.50	1.00
hwb4_52	4	23	28	18	18	0.1	18	0.1	0.64	1.00	1.00
4gt11_84	5	7	14	6	2	0.1	2	0.1	0.14	0.33	1.00
4gt13-v1_93	5	16	52	20	6	0.1	8	0.1	0.15	0.40	1.33
4mod5-v1_23	5	24	50	30	30	0.1	30	0.1	0.60	1.00	1.00
hwb5_55	5	106	230	146	114	0.1	120	0.1	0.52	0.82	1.05
hwb6_58	6	146	358	316	290	0.1	294	0.2	0.82	0.93	1.01
rd32-v0_67	4	8	10	4	4	1.1	4	0.1	0.40	1.00	1.00
rd53_135	7	78	264	194	136	1.8	136	0.3	0.52	0.70	1.00
ham7_104	7	87	204	162	140	1.9	140	0.3	0.69	0.86	1.00
urf2_152	8	25150	90676	83152	71280	22.0	73932	0.4	0.82	0.89	1.04
urf1_149	9	57770	245604	200804	179832	231.3	203836	0.6	0.83	1.02	1.13
urf5_158	9	51380	229568	206288	176284	247.0	179348	0.6	0.78	0.87	1.02
rd73_140	10	76	238	190	150	1579.4	178	0.9	0.75	0.94	1.19
sys6-v0_144	10	62	192	116	114	1586.4	118	0.8	0.61	1.02	1.04
Shor3	10	2076	6710	6710	4802	1846.2	4802	0.8	0.72	0.72	1.00
sym9_148	10	4452	16848	13656	10984	2415.1	12128	0.7	0.72	0.89	1.10
urf3_155	10	132340	663156	491356	453368	3023.6	458476	0.9	0.69	0.93	1.01
4_49_17	4	30	42	32			32	0.1	0.76	1.00	
4gt10-v1_81	5	36	82	74			32	0.1	0.39	0.43	
4gt5_75	5	22	40	40			22	0.1	0.55	0.55	
4mod7-v0_95	5	40	72	68			44	0.1	0.61	0.65	
aj-e11_165	5	59	118	70			52	0.1	0.44	0.74	
alu-v4_36	5	31	70	70			34	0.1	0.49	0.49	
4gt12-v1_89	6	52	172	76			52	0.1	0.30	0.68	
4gt4-v0_80	6	43	66	58			44	0.1	0.67	0.76	
mod5adder_128	6	81	188	148			120	0.2	0.64	0.81	
mod8-10_177	6	108	218	166			156	0.1	0.72	0.94	
hwb7_62	8	2659	8824	7876			7596	0.4	0.86	0.96	
hwb8_118	9	16608	57378	51998			50184	0.6	0.87	0.97	
hwb9_123	10	20405	84630	78266			74086	0.8	0.88	0.95	
cycle10_2_110	12	1212	5272	5272			4500	1.6	0.85	0.85	
plus63mod4096_163	13	29019	155668	144752			144752	1.9	0.93	1.00	
0410184_169	14	82	48	68			48	13.0	1.00	0.71	
plus127mod8192_162	14	65455	376734	362986			349236	14.6	0.93	0.96	
plus63mod8192_164	14	37101	211276	182856			178122	5.6	0.84	0.97	
ham15_108	15	458	3108	2772			1438	2.5	0.46	0.52	
rd84_142	15	112	468	424			348	4.7	0.74	0.82	
urf6_160	15	53700	478068	427344			257604	10.0	0.54	0.60	
cnt3-5_180	16	125	400	400			356	63.9	0.89	0.89	

n: Number of lines $|G|$: Number of gates Time: Runtime in CPU seconds
SWAPs: Number of SWAP gates required when applying the naive method (reviewed in Section 2.2), the heuristic and exact method (reviewed in Section 3), as well as the proposed method (introduced in Sections 4 and 5)
No runtime is provided for the naive and the heuristic method (since all results were obtained in negligible time)

7 Conclusions

In this work, we considered nearest neighbor optimization of quantum circuits using πDDs. Since πDDs allow for an efficient representation and manipulation of sets of permutations, they allow for considering all possible permutations at once and an subsequent reduction of them with respect to the nearest neighbor constraints. This way, an ideal compromise between existing solutions is provided. Experimental evaluations confirmed the efficiency and quality of the obtained results.

Acknowledgments

This work has partially been supported by the EU COST Action IC1405, the JST ERATO Minato Project, as well as JSPS KAKENHI 15H05711 and 15J01665.

References

1. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge Univ. Press (2000)
2. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. Foundations of Computer Science (1994) 124–134
3. Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Theory of computing. (1996) 212–219
4. Gupta, P., Agrawal, A., Jha, N.K.: An algorithm for synthesis of reversible logic circuits. IEEE Trans. on CAD **25**(11) (2006) 2317–2330
5. Maslov, D., Dueck, G.W., Miller, D.M.: Techniques for the synthesis of reversible Toffoli networks. ACM Trans. on Design Automation of Electronic Systems **12**(4) (2007)
6. Saeedi, M., Sedighi, M., Zamani, M.S.: A novel synthesis algorithm for reversible circuits. In: Int'l Conf. on CAD. (2007) 65–68
7. Wille, R., Große, D., Dueck, G., Drechsler, R.: Reversible logic synthesis with output permutation. In: VLSI Design. (2009) 189–194
8. Große, D., Wille, R., Dueck, G.W., Drechsler, R.: Exact multiple control Toffoli network synthesis with SAT techniques. IEEE Trans. on CAD **28**(5) (2009) 703–715
9. Wille, R., Drechsler, R.: BDD-based synthesis of reversible logic for large functions. In: Design Automation Conf. (2009) 270–275
10. Saeedi, M., Sedighi, M., Zamani, M.S.: A library-based synthesis methodology for reversible logic. Microelectronics Journal **41**(4) (2010) 185 – 194
11. Saeedi, M., Zamani, M.S., Sedighi, M., Sasanian, Z.: Synthesis of reversible circuit using cycle-based approach. J. Emerg. Technol. Comput. Syst. **6**(4) (2010)
12. Soeken, M., Wille, R., Hilken, C., Przigoda, N., Drechsler, R.: Synthesis of reversible circuits with minimal lines for large functions. In: ASP Design Automation Conf. (2012) 85–92
13. Barenco, A., Bennett, C.H., Cleve, R., DiVinchenzo, D., Margolus, N., Shor, P., Sleator, T., Smolin, J., Weinfurter, H.: Elementary gates for quantum computation. The American Physical Society **52** (1995) 3457–3467
14. Miller, D.M., Wille, R., Sasanian, Z.: Elementary quantum gate realizations for multiple-control Toffolli gates. In: Int'l Symp. on Multi-Valued Logic. (2011) 288–293
15. Sasanian, Z., Wille, R., Miller, D.M.: Realizing reversible circuits using a new class of quantum gates. In: Design Automation Conf. (2012) 36–41
16. Wille, R., Soeken, M., Otterstedt, C., Drechsler, R.: Improving the mapping of reversible circuits to quantum circuits using multiple target lines. In: ASP Design Automation Conf. (2013) 85–92
17. Shende, V.V., Bullock, S.S., Markov, I.L.: Synthesis of quantum-logic circuits. IEEE Trans. on CAD **25**(6) (2006) 1000–1010
18. Hung, W., Song, X., Yang, G., Yang, J., Perkowski, M.: Optimal synthesis of multiple output Boolean functions using a set of quantum gates by symbolic reachability analysis. IEEE Trans. on CAD **25**(9) (2006) 1652–1663
19. Große, D., Wille, R., Dueck, G.W., Drechsler, R.: Exact synthesis of elementary quantum gate circuits. Multiple-Valued Logic and Soft Computing **15**(4) (2009) 270–275
20. Saeedi, M., Arabzadeh, M., Zamani, M.S., Sedighi, M.: Block-based quantum-logic synthesis. Quant. Info. Comput. **11**(3&4) (2011) 262–277
21. Niemann, P., Wille, R., Drechsler, R.: Efficient synthesis of quantum circuits implementing Clifford group operations. In: ASP Design Automation Conf. (2014) 483–488

22. Saeedi, M., Wille, R., Drechsler, R.: Synthesis of quantum circuits for linear nearest neighbor architectures. Quant. Info. Proc. **10**(3) (2011) 355–377
23. Khan, M.H.: Cost reduction in nearest neighbour based synthesis of quantum Boolean circuits. Engineering Letters **16**(1) (2008) 1–5
24. Hirata, Y., Nakanishi, M., Yamashita, S., Nakashima, Y.: An efficient method to convert arbitrary quantum circuits to ones on a linear nearest neighbor architecture. In: Conf. on Quantum, Nano and Micro Technologies. (2009) 26–33
25. Shafaei, A., Saeedi, M., Pedram, M.: Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures. In: Design Automation Conf. (2013) 41–46
26. Wille, R., Lye, A., Drechsler, R.: Optimal SWAP gate insertion for nearest neighbor quantum circuits. In: ASP Design Automation Conf. (2014) 489–494
27. Wille, R., Lye, A., Drechsler, R.: Exact reordering of circuit lines for nearest neighbor quantum architectures. IEEE Trans. on CAD **33**(12) (2014) 1818–1831
28. Minato, S.: πDD: A new decision diagram for efficient problem solving in permutation space. In: Conf. on Theory and Applications of Satisfiability Testing. (2011) 90–104
29. Fowler, A.G., Devitt, S.J., Hollenberg, L.C.L.: Implementation of Shor's algorithm on a linear nearest neighbour qubit array. Quant. Info. and Comput. **4** (2004) 237–245
30. Meter, R.V., Oskin, M.: Architectural implications of quantum computing technologies. J. Emerg. Technol. Comput. Syst. **2**(1) (2006) 31–63
31. Ross, M., Oskin, M.: Quantum computing. Comm. of the ACM **51**(7) (2008) 12–13
32. Amini, J.M., Uys, H., Wesenberg, J.H., Seidelin, S., Britton, J., Bollinger, J.J., Leibfried, D., Ospelkaus, C., VanDevender, A.P., Wineland, D.J.: Toward scalable ion traps for quantum information processing. New Journal of Physics **12**(3) (2010) 033031
33. Kumph, M., Brownnutt, M., Blatt, R.: Two-dimensional arrays of radio-frequency ion traps with addressable interactions. New Journal of Physics **13**(7) (2011) 073043
34. Nickerson, N.H., Li, Y., Benjamin, S.C.: Topological quantum computing with a very noisy network and local error rates approaching one percent. Nat Commun **4** (2013) 1756
35. Devitt, S.J., Fowler, A.G., Stephens, A.M., Greentree, A.D., Hollenberg, L.C.L., Munro, W.J., Nemoto, K.: Architectural design for a topological cluster state quantum computer. New Journal of Physics **11**(8) (2009) 083032
36. Yao, N.Y., Gong, Z.X., Laumann, C.R., Bennett, S.D., Duan, L.M., Lukin, M.D., Jiang, L., Gorshkov, A.V.: Quantum logic between remote quantum registers. Phys. Rev. A **87** (2013) 022306
37. Herrera-Martí, D.A., Fowler, A.G., Jennings, D., Rudolph, T.: Photonic implementation for the topological cluster-state quantum computer. Phys. Rev. A **82** (2010) 032332
38. Jones, N.C., Van Meter, R., Fowler, A.G., McMahon, P.L., Kim, J., Ladd, T.D., Yamamoto, Y.: Layered architecture for quantum computing. Phys. Rev. X **2** (2012) 031007
39. Ohliger, M., Eisert, J.: Efficient measurement-based quantum computing with continuous-variable systems. Phys. Rev. A **85** (2012) 062318
40. DiVincenzo, D.P., Solgun, F.: Multi-qubit parity measurement in circuit quantum electrodynamics. New Journal of Physics **15**(7) (2013) 075001
41. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: Int'l Joint Conference on Artificial Intelligence. (2007) 386–392
42. Wille, R., Große, D., Teuber, L., Dueck, G.W., Drechsler, R.: RevLib: an online resource for reversible functions and reversible circuits. In: Int'l Symp. on Multi-Valued Logic. (2008) 220–225 RevLib is available at http://www.revlib.org.