Application of Permutation Group Theory in Reversible Logic Synthesis

Dmitry V. Zakablukov
Dep. of Information Security, Bauman Moscow State Technical University, Moscow, Russian Federation
E-mail: dmitriy.zakablukov@gmail.com
July 11, 2016

Abstract

The paper discusses various applications of permutation group theory in the synthesis of reversible logic circuits consisting of Toffoli gates with negative control lines. An asymptotically optimal synthesis algorithm for circuits consisting of gates from the NCT library is described. An algorithm for gate complexity reduction, based on equivalent replacements of gates compositions, is introduced. A new approach for combining a group-theory-based synthesis algorithm with a Reed–Muller-spectra-based synthesis algorithm is described. Experimental results are presented to show that the proposed synthesis techniques allow a reduction in input lines count, gate complexity or quantum cost of reversible circuits for various benchmark functions.

Keywords: reversible logic, synthesis, permutation group theory.

1 Introduction

Reversible logic circuits have been studied in many recent papers [1], [10], [12], [14], [18]. On the one hand, the interest in these circuits is caused by the theoretically possible reduction of energy consumption in digital devices due to the reversibility of all computations [4]. On the other hand, all quantum computations are necessarily reversible. Hence, with the help of a reversible circuit, one can model a quantum circuit.

One important research area is the development of new efficient and fast synthesis algorithms, which can produce a reversible circuit with low gate complexity and depth. However, for the purpose of a comparison between different synthesis algorithms, we should first choose a library of gates, from which a synthesized circuit will consist. One such gate library is one that includes NOT (inversion gate), CNOT (Feynman gate) and C$^2$NOT (Toffoli gate). We will refer to it as the NCT library. Another popular gate library is the GT library, which includes generalized Toffoli gates with positive and negative control input lines. Both libraries are functionally complete in terms of the ability to construct a reversible circuit that implements a desired even permutation from the alternating group $A(2^n)$ without using additional inputs. An odd permutation
from the symmetric group $S(B^n)$ can always be realized in a reversible circuit without additional inputs in the GT, but not in the NCT library.

For many proposed synthesis algorithms, an upper bound for the gate complexity of a reversible circuit in the worst case is proved. Though it was proved that the worst case requires $\Omega(n2^n / \log n)$ gates from the NCT library [15], almost all these bounds are of the form $O(n2^n)$ in the NCT library [9].

Recently, the first asymptotically optimal in NCT library synthesis algorithm was introduced with the gate complexity $L(S) \leq 3n2^{n+4} / \log_2 n$ of a reversible circuit in the worst case [18]. In Section 2, we briefly describe this cycle-based algorithm. Section 3 contains descriptions of the replacement rules from [17] and of a “moving and replacing” algorithm for reducing the gate complexity of a reversible circuit in NCT and GT libraries with the help of these rules. In Section 4, we discuss various approaches of reducing the gate complexity during the synthesis process. In Section 5, we introduce a novel technique for combining a cycle-based synthesis algorithm with a Reed–Muller-spectrum-based one. Experimental results of benchmark functions synthesis are presented in Section 6; all new circuits were obtained with the help of our open source software ReversibleLogicGenerator [19] that implements all synthesis techniques described in this paper. All results we present here (except Section 2 and the first part of Section 3) are new.

We use the following notation for a generalized Toffoli gate with negative control input lines.

**Definition 1.** A generalized Toffoli gate $TOF(I; J; t) = TOF(i_1; \ldots; i_r; j_1; \ldots; j_s; t)$ is a reversible gate, which defines a transformation $f_{I; J; t}: B^n \rightarrow B^n$ as follows:

$$f_{I; J; t}(x_1, \ldots, x_n) = (x_1, \ldots, x_t \oplus x_{i_1} \land \cdots \land x_{i_r} \land \bar{x}_{j_1} \land \cdots \land \bar{x}_{j_s}, \ldots, x_n),$$

where $I = \{i_1, \ldots, i_r\}$ is a set of indices of positive control input lines, $J = \{j_1, \ldots, j_s\}$ is a set of indices of negative control input lines, and $t$ is an index of a controlled output line, $I \cap J = \emptyset$, $t \notin I \cup J$.

In the case of the absence of negative control input lines, a generalized Toffoli gate will be referenced as $TOF(I; t)$, and in the case when a generalized Toffoli gate has no control input lines at all, it will be referenced as $TOF(t)$. In other words, $TOF(t) = TOF(\emptyset; \emptyset; t)$ and $TOF(I; t) = TOF(I; \emptyset; t)$. Using this notation, we can refer to a NOT gate as $TOF(a)$, to a CNOT gate as $TOF(b; a)$ and to a C$^2$NOT gate as $TOF(b, c; a)$.

## 2 Asymptotically optimal synthesis algorithm

In [18] a cycle-based synthesis algorithm that can produce a reversible circuit with the asymptotically optimal in NCT library gate complexity for any even permutation on the set $B^n$, was described. It is the first and currently (as far as we know) the only asymptotically optimal non-search synthesis algorithm for the NCT library. Our software [19] is based on it, so we are going to briefly describe the essence of the algorithm.

Let’s consider an even permutation $h \in A(B^n)$. The main idea is a decomposition of $h$ into a product of transpositions in such a way that all of them can
be grouped by $K$ independent transpositions:

$$h = G_1 \circ G_2 \circ \cdots \circ G_t \circ h',$$

where $G_i = (x_{i,1}, y_{i,1}) \circ \cdots \circ (x_{i,K}, y_{i,K})$ is an $i$-th group of $K$ independent transpositions, $x_{i,j}, y_{i,j} \in \mathbb{B}^n$ and $h'$ is a residual permutation.

Using vectors of a group $G_i$, we construct a matrix $A_i$ as follows:

$$A_i = [x_{i,1} \ y_{i,1} \ \cdots \ x_{i,K} \ y_{i,K}]^T.$$

The matrix $A_i$ is a $2K \times n$ binary matrix. If $2^K < n$, then some columns in it are equal to one another. These duplicated columns can be zeroed-out in the matrix, using CNOT gates, with the help of conjugation; this results in a new matrix $A_i^{(1)}$.

Note that the matrix $A_i$ defines a permutation $\pi_i \in S(\mathbb{B}^n)$ and every gate $e$ from the NCT library defines a permutation $h_e \in S(\mathbb{B}^n)$, for which $h_e^{-1} = h_e$. Therefore, a conjugation of a permutation $\pi$ by a permutation $h_e$, denoted as $\pi^{h_e} = h_e^{-1} \circ \pi \circ h_e$, corresponds to attaching the gate $e$ to the front and back of a current sub-circuit. For example, if the first two columns in the matrix $A_i$ are equal, we can zero-out the second column with the help of two TOF($1;2$) gates.

Next, we fix all pairwise distinct nonzero columns $\{c_{j_1}, \ldots, c_{j_d}\}$ in the matrix $A_i^{(1)}$ and choose an index of a controlled output $t$ from the set $\{j_1, \ldots, j_d\}$. After that we transform the matrix $A_i^{(1)}$ to the canonical form $A_i^{(2)}$ with the help of conjugation, where an $l$-th row, $l$ is odd, differs from the $(l+1)$-th row only in $t$-th element.

And finally, we transform the matrix $A_i^{(2)}$ to the final form $A_i^{(3)}$ with the help of TOF($j$) gates, where $j \notin \{j_1, \ldots, j_d\}$. In [18] it was proved that the matrix $A_i^{(3)}$ can be realized by the single gate TOF$(\{1, \ldots, n\} \setminus \{j_1, \ldots, j_d\}; t)$. The gate can be represented as a composition of C$^2\text{NOT}$ gates if $K > 1$ (the number of independent transpositions in a group $G_i$).

A synthesized reversible circuit $\mathfrak{S}$, produced by the algorithm, has the gate complexity $L(\mathfrak{S}) \leq 3n2^{n+4}/\log_2 n$, if $K = O(\log_2 n - \log_3 \log_2 n - \log_2 \phi(n))$, where $\phi(n) < n/\log_2 n$ is an arbitrarily slowly growing function, and the gate complexity $L(\mathfrak{S}) \leq 6n2^n$, if $K = 2$. These results were proved in [18].

In our software [19], we can change the parameter $K$ to achieve the best synthesis result in a particular case. But in practice, when the number of input lines in a circuit is large, it is almost always the best option to use $K = \lceil \log_2 n \rceil$ during the synthesis process.

The time complexity of the synthesis algorithm is $T(A) = O(n2^n/\log_2 n)$ in the worst case.

3 Generalized replacement rules for gate compositions

One of the most widely used gate complexity reduction techniques is an applying gate compositions templates to a reversible circuit. For example, such templates

---

1Hereinafter a multiplication of permutations is left-associative: $(f \circ g)(x) = g(f(x))$. 

3
were considered in [9]. This approach involves storing templates and finding them in a circuit. But we can interchange some adjacent gates of NCT and GT libraries in a reversible circuit without changing the resulting transformation, defined by the circuit. We call such gates independent.

In [5] the necessary and sufficient conditions for the independence of two $TOF(I_1; t_1)$ gates were proved. However, for the gates from the GT library we can supplement these conditions.

Lemma 1. Gates $TOF(I_1; J_1; t_1)$ and $TOF(I_2; J_2; t_2)$ are independent iff at least one of the following condition holds (see Fig. 1):

1. $t_1 \notin I_2 \cup J_2$ and $t_2 \notin I_1 \cup J_1$ (in particular, $t_1 = t_2$);
2. $I_1 \cap J_2 \neq \emptyset$ or $I_2 \cap J_1 \neq \emptyset$.

Proof of the Lemma 1 was partly given in [17]. Even though the first condition of gate independence was already known before [5] (see Fig. 1a–1b), the second one cannot be derived from it (see Fig. 1c).

In [3] rule-based optimization techniques based on Karnaugh maps for the optimization of sub-circuits with common targets were described. The main disadvantage of this approach is the restricted scalability for circuits with the large number of input lines. On the other hand, the advantage of using negative control Toffoli gates for the simplification of reversible circuits and reducing their quantum cost was shown by the authors.

In [17] we proposed generalized replacement rules for the case of an arbitrary number of input lines. Moreover, we were able to obtain a new rule for interchanging two gates with changing the polarity of a control line for one of these gates (see the last rule in Table 4). Our replacement rules are essentially templates of small length. But the advantage of using them is in changing the set of negative control input lines in a gate. This makes it possible to obtain independent gates instead of dependent ones in some cases, interchange or move them in a reversible circuit to new places and apply other replacement rules. “Moving and replacing” algorithm will be described later.

A similar approach was used in [11], though replacement rules in that paper differ from ours.

Figure 1: Examples of independent gates: (a)–(b) NCT library specific; (c) GT library specific.
Let’s consider a composition of two dependent gates $e_1 \ast e_2$. Let $h_{e_1}$ and $h_{e_2}$ be permutations defined by them respectively. If we want to obtain an equal composition $S_1 \ast e_1 = e_1 \ast e_2$, then the circuit $S_1$ must implement the permutation $h_1 = h_{e_1}^{h_{e_1}}$. And if we want to obtain an equal composition $e_2 \ast S_2 = e_1 \ast e_2$, then the circuit $S_2$ must implement the permutation $h_2 = h_{e_2}^{h_{e_2}}$.

All our replacement rules can be classified as follow:

1. Representing a gate from the GT library as a composition of gates from the NCT library.

2. Merging two gates into one.

3. Reducing the negative control lines number.

4. Interchanging two dependent gates.

For the sake of clarity, we will organize the detailed description of our rules in the form of Tables 1–4, one for each rule “class”. The left column of the tables contains a gate composition before applying replacement rule, and the right column contains the result of the replacement. For every rule a picture goes first (for understanding the concept of a rule), then, a text description of the rule, and finally, a condition for applying the rule.

Now we can describe the “moving and replacing” algorithm implemented in our software [19], which may reduce the gate complexity of a circuit.

Let a reversible circuit $S$ be a composition of $l$ gates from the GT library: $S = \bigstar_{i=1}^{l} e_i$. If a gate composition $e_i \ast e_j$ satisfies the condition of a replacement, where $i < j$, and there is such an index $s$, $i \leq s < j$, that gates $e_i$ and $e_k$ are independent for every $i < k \leq s$, and gates $e_j$ and $e_k$ are independent for every $s < k < j$, then the gates $e_i$ and $e_j$ can be removed from the circuit and a result of the replacement for $e_i \ast e_j$ can be inserted between gates $e_s$ and $e_{s+1}$.
So, the “moving and replacing” algorithm first searches a pair of gates, the composition of which satisfies the condition of a replacement. After that the algorithm checks if they can be moved to each other, using Lemma 1. If yes, it implements a replacement as described above. In the case, when the gate complexity is not reduced after replacement, but there are new gates in a circuit, the algorithm continues to work, until the gate complexity is reduced or there are no new gates.

The time complexity of the proposed “moving and replacing” algorithm $T(A) \geq R \cdot l^2$, where $R$ is the number of replacement rules, $l$ is the gate complexity of an original circuit. It is almost the same as the time complexity of any template based optimization algorithm. At the same time, our “moving and replacing” algorithm seems to be more flexible than a template-based approach, because proposed replacement rules do not depend on the number of inputs in a reversible circuit, therefore there is no need to store a large number of templates and search them in a library.
Table 3: Reducing the negative control lines number.

| $\text{TOF}(I_1; J_1; t) \ast \text{TOF}(I_2; J_2; t)$ | $\text{TOF}(I_1; J_3; t) \ast \text{TOF}(I_2; J_3; t)$ |
|---------------------------------------------------|---------------------------------------------------|
| $J_3 = J_1 \setminus \{ q \} = J_2 \setminus \{ p \}$ |

Condition: there are such $p$ and $q$, that $p \in I_1 \cap J_2$, $q \in J_1 \cap I_2$.

$I_2 = I_1 \setminus \{ p \} \cup \{ q \}$, $J_2 = J_1 \setminus \{ q \} \cup \{ p \}$.

| $\text{TOF}(I_1; J_1; t) \ast \text{TOF}(I_2; J_2; t)$ | $\text{TOF}(I_1 \cup \{ p \}; J_1 \setminus \{ p \}; t) \ast \text{TOF}(I_2 \cup \{ q \}; J_2 \setminus \{ q \}; t)$ |
|---------------------------------------------------|---------------------------------------------------|

Condition: there are such $p$ and $q$, that $J_1 = J' \cup \{ p \}$, $J_2 = J' \cup \{ q \}$, $J_1 \cap J_2 = J'$, $I_1 = I_2$.

4 Boolean hypercube search

Let’s consider the following permutation:

$$h = ((1,0,0,0,0), (1,0,1,0,1)) \circ (1,0,0,1,0), (1,0,1,0,0)) \circ (1,0,0,1,1), (1,0,1,1,1)) \circ (1,0,0,1,1), (1,0,1,1,0)) .$$

As we can see, vectors in every transposition of permutation $h$ presented above differ only in the 3-rd and 5-th coordinates. There are four transpositions total. Hence, a set of all vectors in these transpositions represents a Boolean 3-cube $\mathbb{B}^{5,1,2}$ contained in a Boolean 5-cube $\mathbb{B}^5$. This 3-cube can also be denoted as $\langle 1,0,*,*,*\rangle$. Therefore, the permutation $h$ can be implemented by a composition of gates $\text{TOF}(1; 2; 3) \ast \text{TOF}(1; 2; 5)$.

Let’s assume we can represent a permutation $h \in A(\mathbb{B}^n)$ as a product of transpositions in such a way that a set of all vectors of first $k$ transpositions in this product represents a Boolean $(1 + \log_2 k)$-cube. In the case, when we use only our cycle-based approach for the synthesis, we have to divide these $k$ transpositions into groups and synthesize them separately. This approach can lead to significant gate complexity of a produced reversible circuit. On the other hand, any Boolean hypercube contained in $\mathbb{B}^n$ can be implemented by a composition of no more than $n$ generalized Toffoli gates $\text{TOF}(I; J; t)$.

For example, a transformation $f((x_1, x_2, \cdots , x_n)) = (x_1, x_2 \oplus x_1, x_3, \ldots , x_n)$
Table 4: Interchanging two dependent gates.

\[
\begin{align*}
T(1; J; t_1) \ast T(1; J; t_1) &= T(1; J; t_1) \ast T(1; J; t_1) \\
T(1; J; t_2) \ast T(1; J; t_2) &= T(1; J; t_2) \ast T(1; J; t_2)
\end{align*}
\]

\[
\begin{align*}
I_3 &= I_1 \cup I_2 \setminus \{ t_1 \}, \quad J_3 = J_1 \cup J_2 \setminus \{ t_1 \}
\end{align*}
\]

Condition: gates are dependent, \( t_1 \in I_2 \cup J_2 \) and \( t_2 \notin I_1 \cup J_1 \).

\[
\begin{align*}
T(1; J; t_1) \ast T(1; J; t_1) &= T(1; J; t_1) \ast T(1; J; t_1) \\
T(1; J; t_2) \ast T(1; J; t_2) &= T(1; J; t_2) \ast T(1; J; t_2)
\end{align*}
\]

\[
\begin{align*}
I_3 &= (I_2 \setminus \{ t_1 \}) \cup (J_2 \cap \{ t_1 \}), \quad J_3 = (J_2 \setminus \{ t_1 \}) \cup (I_2 \cap \{ t_1 \})
\end{align*}
\]

Condition: gates are dependent, \( I_1 \subseteq I_2 \) and \( J_1 \subseteq J_2 \).

The table illustrates the interchanging of two dependent gates. The conditions are as follows:

1. \( I_3 = I_1 \cup I_2 \setminus \{ t_1 \} \) and \( J_3 = J_1 \cup J_2 \setminus \{ t_1 \} \).
2. \( I_3 = (I_2 \setminus \{ t_1 \}) \cup (J_2 \cap \{ t_1 \}) \) and \( J_3 = (J_2 \setminus \{ t_1 \}) \cup (I_2 \cap \{ t_1 \}) \).

These conditions ensure that the interchanging is valid and does not violate the dependency constraints.

Can be implemented by a reversible circuit, produced by our main synthesis algorithm, with the gate complexity \( O(n^{2^n}) \), and it can be implemented by a single gate \( T(1; 2) \), because there is a Boolean 1-cube \( \langle 1, *, \cdots, * \rangle \).

It is obvious that searching a Boolean hypercube can take a significant amount of time and can be inefficient for large functions. But this approach makes it possible to obtain better synthesis results in some cases.

4.1 Effective disjoints of cycles

To find a larger Boolean hypercube, we should somehow effectively represent a permutation \( h \) as a product of specific transpositions. Let’s consider a permutation \( h = (a, b, c, e, f, g) \), where the Hamming distances \( d(a, e) = d(b, g) = d(c, f) = \Delta \) and the Hamming distance for any other two elements of \( h \) is not equal to \( \Delta \). We have the two possible representations of \( h \) as a product of cycles:

1. \( h = (a, e) \circ (a, f, g) \circ (c, b, c) \).
2. \( h = (b, g) \circ (c, f) \circ (a, b) \circ (c, g) \circ (e, f) \).

We can see that in the first case only the cycle \((a, e)\) has the two elements with the Hamming distance equal to \( \Delta \). But in the second case there are two cycles \((b, g)\) and \((c, f)\) that have the two elements with the Hamming distance equal to \( \Delta \). Therefore, we can assume that the set \( \{ b, g, c, f \} \) may contain a larger
Boolean hypercube, compared to the set \{a, e\}, and we will call the second representation of \(h\) an **effective disjoint of cycles**.

There is a simple linear algorithm for an effective disjoint of cycles of a permutation for a given Hamming distance \(\Delta\). In the first pass, the algorithm searches all pairs of elements in a cycle with the Hamming distance equal to \(\Delta\). In the second pass, the algorithm calculates for a found pair \(p\) how many other pairs would be broken, if we disjoint the cycle by the pair \(p\). In the third pass, the algorithm chooses a pair \(p\), for which the number of broken pairs is minimal. And finally, the algorithm disjoints the cycle by the chosen pair. After that we don’t have to repeat all steps for obtained cycles, because we can simply remove broken pairs and use previous results for further disjoints.

For our example above, we have the three pairs with the Hamming distance equal to \(\Delta\): \((a, e)\), \((b, g)\) and \((c, f)\). If we choose the pair \((a, e)\), it would break two pairs \((b, g)\) and \((c, f)\). And if we choose either \((b, g)\) or \((c, f)\), they would break only the pair \((a, e)\). Hence, an effective disjoint will be for the pair \((b, g)\) or \((c, f)\). It is not difficult to show that the proposed algorithm for an effective disjoint of cycles doesn’t depend on the order of elements in a cycle. The disjoint result will be the same for the permutation \(h = (a, b, c, e, f, g)\) and for the permutation \(h' = (c, e, f, g, a, b)\).

The time complexity of a single disjoint operation for a cycle of length \(l\) is no more than \(O(l \log_2 l)\).

### 4.2 Left and right multiplication

Until now we used a left multiplication for a cycle disjoint. But we can also use a right multiplication. E. g., a cycle \((a, b, c)\) can be represented in two ways for the transposition \((a, b)\):

1. Left multiplication: \((a, b, c) = (a, b) \circ (a, c)\).
2. Right multiplication: \((a, b, c) = (b, c) \circ (a, b)\).

We can see that the results of the multiplications are different. This difference can lead to significantly different synthesis results.

There is no way to find out on an \(i\)-th step of our basic synthesis algorithm, whether the left or right multiplication would be the best in the end. The only thing we can do is to make both left and right multiplications on an \(i\)-th step and choose the one which leads to the greater permutation reduction and to the lower gate complexity of a current reversible circuit. This approach doubles the synthesis time, but it also leads to better reversible circuits in some cases.

And finally, another area for optimizations is the constructing of a bijective transformation for a given non-bijective one. We believe that in terms of reversible logic synthesis the best result can be achieved, when this bijective transformation has minimal Hamming distances between inputs and outputs.

### 5 Combining cycle-based and RM-spectra based algorithms

In [13] a hybrid framework was proposed, which combines a cycle-based and a RM-spectra based algorithms. Unfortunately, this combination is only the choice of a better reversible circuit synthesized by one or another algorithm.
We propose a new approach for combining a cycle-based and a RM-spectra based algorithms. In [9] a RM-spectra based synthesis algorithm was described. For a reversible specification \( f : \mathbb{B}^n \rightarrow \mathbb{B}^n \) the algorithm successively transforms the truth table \( T_n \) to the truth table that corresponds to the identity transformation. This is done by changing an \( i \)-th row in \( T_n \), for which \( T_n[i] \neq i \), to the form \( T_n[i] = i \) for every \( i = 0, \ldots, (2^n - 1) \). Every row \( j < i \) is not changed after a transformation of an \( i \)-th row: \( T_n[j] = j \).

Our combining approach allows us to modify the truth table \( T_n \) in such a way that for the first row \( i \), for which \( T_n[i] \neq i \), \( T_n[j] = j \), \( j < i \), each row \( k \leq i \) in a modified truth table \( T'_n \) will be equal to itself: \( T'_n[k] = k \).

On an \( i \)-th step of the RM-spectra based synthesis algorithm from [9] a reversible circuit \( S \) which we synthesize is of the form:

\[
S = S_l \ast S_{T_n} \ast S_r,
\]

where sub-circuits \( S_l \) and \( S_r \) were constructed on the previous steps and a circuit \( S_{T_n} \) is unknown, it implements a transformation described by the truth table \( T_n \), for which \( T_n[j] = j \), \( j < i \), \( T_n[i] \neq i \). The original RM-spectra based synthesis algorithm appends gates to the \( S_l \) or \( S_r \) after modifying the \( i \)-th row in \( T_n \).

Let \( h, h_l, h_t, h_r \) be the permutations, defined by the circuits \( S, S_l, S_{T_n} \) and \( S_r \) respectively. This implies that

\[
h = h_l \circ h_t \circ h_r.
\]

Let’s assume \( T_n[i] = k \) and \( T_n[l] = i \), where \( k, l > i \). We can state that

\[
\begin{align*}
h = & h_l \circ h'_t \circ (i, k) \circ h_r = h_l \circ h'_t \circ h_r \circ (i, k)^h_r, \\
h = & h_l \circ (i, l) \circ h'_r \circ h_r = (i, l)^h_r^{-1} \circ h_l \circ h'_r \circ h_r,
\end{align*}
\]

where a permutation \( h'_t \) is defined by the truth table \( T'_n \), \( T'_n[j] = j \) for every \( j \leq i \).

From this it follows that we can “push” one transposition \((i, k)\) or \((i, l)\) from the permutation \( h_t \) to the right or to the left, conjugate it by the permutation \( h_r \) or \( h_r^{-1} \) respectively and “skip” the transformation of the \( i \)-th row in the truth table \( T_n \) by the original RM-spectra based synthesis algorithm. After that we can move to the next row and repeat this process.

After a RM-spectra based synthesis algorithm finishes its work, we can use a cycle-based synthesis algorithm to synthesize pushed transpositions. There are several approaches to decide, whether an \( i \)-th row is pushed or not and where it will be pushed (left or right). For example, we can push an \( i \)-th row only when the Hamming weight of \( i \) is greater or equal to a predefined threshold \( w \). It is equivalent to processing all monomials of degree \( d < w \) in a Reed–Muller polynomial with a RM-spectra based synthesis algorithm. All other monomials will be processed by a cycle-based synthesis algorithm.

We realized the proposed combining approach in our software [19] with the ability to choose a “push policy” and a weight threshold. This allowed us to find a reversible circuit implementing \texttt{rd53} function with 7 inputs and with the gate complexity equal to 11 in the GT library (see Fig. 2). The weight threshold was equal to one during the synthesis process.
Figure 2: Realization of \textbf{rd53} function in a reversible circuit with 7 inputs and 11 gates in the GT library.

Table 5: Benchmark functions synthesis (new circuits with less input count).

| Function | New circuits | Existing circuits |
|----------|--------------|-------------------|
|          | lines$_{min}$ | GC | QC | T-count | lines$_{min}$ | GC | QC | T-count |
| $g_2^3$mult | 7 | 73 | 740 | 632 | 9 | 11 | 47 | 63 |
| $g_2^3$mult | 7 | 79 | 712 | 632 | 12 | 19 | 83 | 112 |
| $g_2^4$mult | 9 | 415 | 47649 | 10838$^*$ | 10 | 7522 | 17975 | 14193 |
| $n_{thprime9}$ | 9 | 3942 | 19313 | 15234 | 14 | 28 | 98 | 147 |
| $rd73$ | 9 | 296 | 43421 | 8765$^*$ | 10 | 7522 | 17975 | 14193 |
| $rd73$ | 9 | 835 | 4069 | 3521 | 10 | 20 | 64 | 98 |
| $rd84$ | 11 | 679 | 359384 | 25364$^*$ | 15 | 28 | 98 | 147 |
| $rd84$ | 11 | 2560 | 12397 | 8772$^*$ | 15 | 28 | 98 | 147 |

* an ancillary line is required

6 Experimental results

We developed an open source software \textit{ReversibleLogicGenerator} [19], which implements the basic cycle-based synthesis algorithm from [18] and all the gate complexity reduction techniques, described in this paper. Using our software, we conducted a series of experiments on reversible benchmark functions synthesis. The results are presented in Tables 5–7. The synthesis time in the worst case was a matter of seconds.

We were able to obtain more than 40 new reversible circuits, which have less input count, less gate complexity or less quantum cost compared to existing circuits. All specifications for benchmark functions and their names were taken from the \textit{Reversible Logic Synthesis Benchmarks Page} [7] and from the \textit{RevLib} site [16]. We use the following conventions in the tables: \textit{lines} is the number of inputs in a reversible circuit $S$, GC is the gate complexity $L(S)$ of this circuit and QC is its quantum cost $W(S)$; \textit{T-count} is the number of \textit{T} gates in a decomposition of the circuit into Clifford+$T$ gates.

The quantum cost of obtained circuits was calculated with the help of the software \textit{RCViewer+} [2]. Its calculation is based on the paper [6], according to which a generalized Toffoli gate with negative control lines may have the same
Table 6: Benchmark functions synthesis (new circuits with less gate complexity).

| Function | New circuits | Existing circuits |
|----------|--------------|-------------------|
|          | lines | GC\(_{\text{min}}\) | QC | T-count | lines | GC\(_{\text{min}}\) | QC | T-count |
| 2of5     | 6    | 9   | 268 | 191*    | 6    | 15  | 107 | 119 |
| 2of5     | 6    | 10  | 118 | 135     |       |      |     |     |
| 2of5     | 7    | 11  | 32  | 42      | 7    | 12  | 32  | 49  |
| 317      | 3    | 4   | 14  | 14      | 3    | 6   | 12  | 14  |
| 317      | 3    | 5   | 13  | 14      |       |      |     |     |
| 4b15g2   | 4    | 12  | 57  | 55*     | 4    | 15  | 31  | 35  |
| 4b15g4   | 4    | 12  | 49  | 45*     | 4    | 15  | 35  | 31* |
| 4b15g5   | 4    | 14  | 72  | 54*     | 4    | 15  | 29  | 42  |
| 4mod5    | 5    | 4   | 13  | 14      | 5    | 5   | 7   | 7   |
| 5mod5    | 6    | 7   | 429 | 294*    | 6    | 8   | 84  | 70* |
| 6sym     | 7    | 14  | 1308 | 628*  | 7    | 36  | 777 | 741 |
| 6sym     | 7    | 15  | 825  | 624*   |       |      |     |     |
| 9sym     | 10   | 73  | 61928 | 7004* | 10   | 129 | 6941 | 5484* |
| 9sym     | 10   | 74  | 31819 | 6788* |       |      |     |     |
| ham7     | 7    | 19  | 77  | 85      | 7    | 25  | 49  | 42  |
| hwbl2    | 12   | 42095 | 134316 | 98482 | 12   | 55998 | 198928 | 134131 |
| nth_prime7_inc | 7   | 427  | 10970 | 5403* | 7    | 1427 | 3172 | 2837 |
| nth_prime7_inc | 7   | 474  | 10879 | 5403* |       |      |     |     |
| nth_prime7_inc | 7   | 824  | 2269 | 1906    |       |      |     |     |
| nth_prime8_inc | 8   | 977  | 10218 | 7359* | 8    | 3346 | 7618 | 5985 |
| nth_prime8_inc | 8   | 1683 | 6330 | 5213    |       |      |     |     |
| nth_prime9_inc | 10  | 2234 | 22181 | 17292 | 10   | 7522 | 17975 | 14193 |
| nth_prime10_inc | 11  | 5207 | 50152 | 38261 | 11   | 16626 | 40299 | 30315 |
| nth_prime11_inc | 12  | 11765 | 124408 | 92937 | 12   | 35335 | 95431 | 68255 |
| rd53     | 7    | 11  | 96  | 100     | 7    | 12  | 120 | 124 |

* an ancillary line is required
**Table 7: Benchmark functions synthesis (new circuits with less quantum cost).**

| Function | New circuits | Existing circuits |
|----------|--------------|-------------------|
|          | lines | GC | QC | T-count | lines | GC | QC | T-count |
| 2of5     | 7     | 12 | 31 | 42      | 7     | 12 | 32 | 49      |
| 6sym     | 7     | 41 | 206| 184     | 7     | 36 | 777| 714     |
| 9sym     | 10    | 347| 1975| 1680    | 10    | 210| 4368| 4368    |
| hwb7     | 7     | 603| 1728| 1400    | 7     | 331| 2611| 2245*   |
| hwb8     | 8     | 1594| 4852| 3748    | 8     | 2710| 6940| 5201    |
| hwb9     | 9     | 3999| 12278| 10220  | 9     | 6563| 16173| 12150   |
| hwb10    | 10    | 8247| 26084| 20368  | 10    | 12288| 35618| 25939   |
| hwb11    | 11    | 21432| 69138| 52922  | 11    | 32261| 90745| 63430   |
| hwb12    | 12    | 42095| 134316| 98482  | 12    | 55998| 198928| 134131  |
| nth_prime7.inc | 7  | 824 | 2269 | 1906 | 7 | 1427 | 3172 | 2837 |
| nth_prime8.inc | 8  | 1683 | 6330 | 5213 | 8 | 3346 | 7618 | 5985 |
| rd53     | 7     | 12 | 82 | 92      | 7     | 12 | 120| 124     |
| rd53     | 7     | 12 | 95 | 100     | 7     | 11 | 96 | 100     |

* an ancillary line is required

Quantum cost as the corresponding generalized Toffoli gate without negative control lines. Also, we included the T-count cost measure for all circuits in the tables (cost calculation was based on the paper [8]). Despite the fact that this cost measure is very popular for fault-tolerant circuits in the literature, it is not universal in the case of limited ancillary lines availability. According to [8], there is a circuit of Toffoli gates that cannot be implemented, using Clifford+T gates, without an ancillary line. We marked such T-count metrics by the asterisk symbol in the tables.

Tables 5, 6 and 7 contain results for obtained reversible circuits with less input count (column \( \text{lines}_{\text{min}} \)), less gate complexity (column \( \text{GC}_{\text{min}} \)) and less quantum cost (column \( \text{QC}_{\text{min}} \)) compared to existing circuits respectively. Since we compare our circuits only with circuits consisting of gates from NCT and GT libraries and since the NCT library is a part of the GT library, such comparison made by us is correct.

We have not included circuits with more than 12 inputs in the tables just because of the limited format of the paper. One can easily synthesize such circuits with the help of our software.

With the help of developed software we were able to find a reversible circuit with 7 input lines and with 11 gates from the GT library for one of the most popular benchmark functions \textbf{rd53} (see Fig. 2). This circuit and all other circuits described in the tables above can be freely downloaded in TFC and REAL formats from the cite [19] as well as ReversibleLogicGenerator software itself.
7 Conclusion

A reversible circuit with \( n \) inputs necessarily defines a permutation from the symmetric group \( S(B^n) \). Permutations that correspond to all the gates NOT, CNOT and \( C^2 \) NOT generate the alternating group \( A(B^n) \) if \( n > 3 \), and permutations that correspond to all the gates \( C^k \) NOT, \( 1 \leq k \leq n \) generate the symmetric group. This implies that we can use the permutation group theory to successfully synthesize a reversible circuit for a given reversible specification.

In the paper, we briefly described the first asymptotically optimal in NCT library synthesis algorithm, based on the permutation group theory, which makes it possible to obtain a reversible circuit without additional inputs. We also suggested the “moving and replacing” algorithm for gate complexity reduction for circuits consisting of the gates from the GT library; the algorithm is based on equivalent replacements of gate compositions and on conditions of independence for the gates with negative control lines.

We described some gate complexity reduction techniques that use the permutation group theory. Among them are the search of a Boolean hypercube and an effective cycle disjoint. We presented experimental results for benchmark functions synthesis, which include more than 40 reversible circuits consisting of gates from the GT library, obtained with the help of developed open source software that implements all described techniques.

We believe that the permutation group theory may allow us to obtain better reversible circuits for all benchmark functions, and we hope that this paper will motivate other researchers to improve our results.

Acknowledgments.

The reported study was partially supported by RFBR, research project No. 16-01-00196 A.

References

[1] N. Abdessaied, R. Wille, M. Soeken, and R. Drechsler. Reducing the depth of quantum circuits using additional circuit lines. In G. W. Dueck and D. M. Miller, editors, Reversible Computation, volume 7948 of LNCS, pages 221–233. Springer, 2013.

[2] M. Arabzadeh and M. Saeedi. RCVviewer+ — a viewer/analysier for reversible and quantum circuits. http://ceit.aut.ac.ir/QDA/RCV.htm, 2013.

[3] M. Arabzadeh, M. Saeedi, and M. S. Zamani. Rule-based optimization of reversible circuits. In Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific, pages 849–854, Jan. 2010.

[4] C. H. Bennett. Logical reversibility of computation. IBM J. Res. Dev., 17(6):525–532, Nov. 1973.

[5] K. Iwama, Y. Kambayashi, and S. Yamashita. Transformation rules for designing CNOT-based quantum circuits. In Proc. 39th Annual Design
[6] D. Maslov, G. W. Dueck, D. M. Miller, and C. Negrevergne. Quantum circuit simplification and level compaction. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 27(3):436–444, Mar. 2008.

[7] D. A. Maslov. Reversible logic synthesis benchmarks page. [http://webhome.cs.uvic.ca/~dmaslov/](http://webhome.cs.uvic.ca/~dmaslov/), 2011.

[8] D. A. Maslov. On the advantages of using relative phase Toffolis with an application to multiple control Toffoli optimization. *CoRR*, abs/1508.03273, 2016.

[9] D. A. Maslov, G. W. Dueck, and D. M. Miller. Techniques for the synthesis of reversible toffoli networks. *ACM Trans. Des. Autom. Electron. Syst.*, 12(4), Sept. 2007.

[10] D. M. Miller, R. Wille, and R. Drechsler. Reducing reversible circuit cost by adding lines. In *Proc. 40th IEEE International Symposium on Multiple-Valued Logic (ISMVL’10)*, pages 217–222, May 2010.

[11] M. Z. Rahman and J. E. Rice. Templates for positive and negative control Toffoli networks. In S. Yamashita and S.-i. Minato, editors, *Reversible Computation*, volume 8507 of *LNCS*, pages 125–136. Springer International Publishing, 2014.

[12] M. Saeedi and I. L. Markov. Synthesis and optimization of reversible circuits — a survey. *ACM Comput. Surv.*, 45(2):21:1–21:34, Mar. 2013.

[13] M. Saeedi, M. S. Zamani, M. Sedighi, and Z. Sasanian. Reversible circuit synthesis using a cycle-based approach. *ACM J. Emerg. Technol. Comput. Syst.*, 6(4):13:1–13:26, Dec. 2010.

[14] B. Schaeffer and M. A. Perkowski. A cost minimization approach to synthesis of linear reversible circuits. *CoRR*, abs/1407.0070, 2014.

[15] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 22(6):710–722, June 2003.

[16] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In *Proc. 38th IEEE International Symposium on Multiple-Valued Logic (ISMVL’08)*, pages 220–225, 2008.

[17] D. V. Zakablukov. Reduction of the reversible circuits gate complexity without using the equivalent replacement tables for the gate compositions. *BMSTU J. Sci. Educ.*, (3):275–289, Mar. 2014. in Russian.

[18] D. V. Zakablukov. On asymptotic gate complexity and depth of reversible circuits without additional memory. *CoRR*, abs/1504.06876, 2015.

[19] D. V. Zakablukov. ReversibleLogicGenerator software. [https://github.com/dmitry-zakablukov/ReversibleLogicGenerator](https://github.com/dmitry-zakablukov/ReversibleLogicGenerator), 2015.