A reversible circuit synthesis algorithm with progressive increase of controls in generalized Toffoli gates

Edinelço Dalcumune
(Universidade Federal dos Vales do Jequitinhonha e Mucuri, Brazil
Universidade Federal do Rio de Janeiro, Brazil
edcomune@ufvjm.edu.br)

Luis Antonio Kowada
(Universidade Federal Fluminense, Brazil
luis@ic.uff.br)

André da Cunha Ribeiro
(Instituto Federal Goiano, Brazil
andre.cunha@ifgoiano.edu.br)

Celina Miraglia Herrera de Figueiredo
(Universidade Federal do Rio de Janeiro, Brazil
celina@cos.ufrj.br)

Franklin de Lima Marquezino
(Universidade Federal do Rio de Janeiro, Brazil
franklin@cos.ufrj.br)

Abstract: We present a new algorithm for synthesis of reversible circuits for arbitrary n-bit bijective functions. This algorithm uses generalized Toffoli gates, which include positive and negative controls. Our algorithm is divided into two parts. First, we use partially controlled generalized Toffoli gates, progressively increasing the number of controls. Second, exploring the properties of the representation of permutations in disjoint cycles, we apply generalized Toffoli gates with controls on all lines except for the target line. Therefore, new in the method is the fact that the obtained circuits use first low cost gates and consider increasing costs towards the end of the synthesis. In addition, we employ two bidirectional synthesis strategies to improve the gate count, which is the metric used to compare the results obtained by our algorithm with the results presented in the literature. Accordingly, our experimental results consider all 3-bit bijective functions and twenty widely used benchmark functions. The results obtained by our synthesis algorithm are competitive when compared with the best results known in the literature, considering as a complexity metric just the number of gates, as done by alternative best heuristics found in the literature. For example, for all 3-bit bijective functions using generalized Toffoli gates library, we obtained the best so far average count of 5.23.

Key Words: design of algorithms, reversible computing, circuit synthesis, cycle representations of permutations

Category: B.2, B.6, F.1, F.2

1 Introduction

In the last decades, the synthesis of reversible circuits has received considerable attention due to the possibility of applications in several areas of science, such as signal processing, cryptography, quantum computing, low-power design, computer graphics, optical computing, DNA computing, bioinformatics, nano and photonic circuits [Saeedi and Markov, 2013]. One of the main motivations for reversible computing is that quantum computing has as one of its foundations the reversibility of all gates, that is, quantum computing circuit models are reversible. This fact stems from the evolution postulate of quantum mechanics, where a unitary operator describes the time-evolution of the state of a closed quantum system [Nielsen and Chuang, 2000]. Another main motivation is the important physical consequences for reversible computing. Landauer [Landauer, 1961] proved that using irreversible logic gates necessarily leads to power dissipation regardless of the underlying technology. The reason is that each erased bit leads to at least $kT \ln 2$ energy dissipation, where k is Boltzmann's constant and T is the absolute temperature of the circuit. On the other hand, Bennett [Bennett, 1973] showed how to use reversible logic gates to reduce or even eliminate power dissipation in a circuit. Indeed, in a reversible circuit—classical or quantum—we can retrieve all information from the circuit input using what was obtained at the circuit output. That is, in this process no input information is erased. Moreover, Bennett also showed that zero power dissipation in circuits is only possible if their calculation is reversible.

An important problem in reversible computing that has been intensively studied for the last decades is the synthesis of reversible circuits. The problem can be stated as: Given a bijective function f, find the best possible reversible circuit that implements f [Shende et al., 2003, Miller et al., 2003, Kowada et al., 2006, Maslov et al., 2007, Golubitsky et al., 2010, Saeedi and Markov, 2013].

In this work, we present a new algorithm for synthesis of reversible circuits using multiple-control Toffoli gates [Toffoli, 1980, Iwama et al., 2002] with any number of positive or negative controls, also known as generalized Toffoli gates [Maslov and Dueck, 2004, Zakablukov, 2016]. Our method is an upgrade of the cycle-based synthesis (CBS) algorithm [Ribeiro et al., 2015]. Besides using only totally controlled gates—gates with controls on all lines except for the target line—we also include partially controlled Toffoli gates [Iwama et al., 2002]. An important contribution from our method is the fact that the obtained circuits use low cost gates first, and consider increasing costs towards the end of the synthesis. For 3 bits, the circuits achieved by our new synthesis algorithm use less gates than the circuits achieved by the CBS algorithm, and by recent algorithms in Zhu *et al.* [Zhu et al., 2018] and Cheng *et al.* [Cheng et al., 2012] that rely on partially controlled Toffoli gates with negative controls. Previous works [Wille et al., 2012, Datta et al., 2013, Rahman and Rice, 2014] have

shown that gate libraries including negative controlled Toffoli gates may be more efficient for circuit synthesis. Our proposed algorithm and previous algorithms found in [Maslov and Dueck, 2004, Maslov et al., 2005] and [Ribeiro et al., 2015] rely on the Hamming distance between the permutation that needs to be synthesized and the identity, and try to minimize by making the moves (gate assignments) that do not increase the Hamming distance. We start with gates with few controls, and then increase the number of controls until we finally have totally controlled gates, in which case we can apply CBS algorithm [Ribeiro et al., 2015], which explores properties of the cycle representation of permutations. An alternative cycle representation has been used in [Saeedi et al., 2010], where the authors consider the permutation as a product of small not necessarily disjoint cycles and explores the properties of certain kinds of products, such as products of transpositions, for instance.

Synthesis can be performed optimally or heuristically, we refer to the survey on synthesis and optimization of reversible circuits [Saeedi and Markov, 2013]. We compare our heuristic results for the synthesis of all 3-bit bijective functions with the main current results using the Generalized Toffoli library obtained by Zhu *et al.* [Zhu et al., 2018] and Cheng *et al.* [Cheng et al., 2012]. Zhu *et al.* cycle-decomposition-based synthesis algorithm is specific to 3-bit bijective functions, and its main idea consists in representing the permutation in disjoint cycles and then using the so-called Head-Pointer-Adjust, which, over all cycles, takes the greatest Hamming distance between two consecutive elements. On the other hand, Cheng *et al.* synthesis algorithm also specific to 3-bit bijective functions uses templates and bidirectional method. It is important to mention that, differently from the algorithms of Zhu *et al.* and Cheng *et al.*, our algorithm works for arbitrary n-bit bijective functions. Besides that, our algorithm achieves circuits with smaller number of gates in average, a cost model adequate when staying within the traditional reversible framework.

There are optimal algorithms restricted to 3 bits [Wille et al., 2012] and to 4 bits [Szyprowski and Kerntopf, 2012]. These methods mainly formulate the synthesis problem as a sequence of instances of standard decision problems, such as Boolean satisfiability. We remark that only a small number of qubits can be handled by these methods.

Furthermore, we perform a series of experiments on benchmark bijective functions. We compare the results obtained by our synthesis algorithm with the best results in the literature, due to Zakablukov [Zakablukov, 2016] and Maslov *et al.* [Maslov et al., 2005, Maslov et al., 2007], for the gate count problem.

The present work is organized as follows. In Sec. 2, we briefly introduce the key concepts of reversible computing. In Sec. 3, we present our synthesis algorithm. In Sec. 4, we present the experimental results that we obtained. Finally, in Sec. 5, we present our final remarks and propose further related questions.

2 Preliminary notions

In the combinational model, a circuit using elementary logical gates AND, OR or NOT can be used to implement a function $f : \{0,1\}^n \to \{0,1\}^m$, where n and m denote the numbers of bits necessary to represent the input and output size, respectively. If f is a bijective function, then it can be represented by a permutation with 2^n values. Otherwise, the function f can be transformed into a new bijective function, by adding ancilla bits. For instance, the function $f = \{(0,7),(1,4),(2,1),(3,0),(4,3),(5,2),(6,6),(7,5)\}$ can be represented by the permutation $[7,4,1,0,3,2,6,5]$.

There are gates that implement bijective functions $f(x_0,\ldots,x_{n-1})$ called *reversible gates*, that is, such gates compute the function f, given the input $x_0,\ldots,x_{n-1}$. The Multiple-Control Toffoli (MCT) gates are some important reversible gates [Toffoli, 1980, Iwama et al., 2002]. These gates, also known as $\mathrm{C^kNOT}(x_{e_0},\ldots,x_{e_k})$, where $0 \leq e_0,...,e_k < n$, keep the $k<n$ lines related to $x_{e_0},\ldots,x_{e_{k-1}}$ (the control lines) unchanged and flip the bit in the line related to x_{e_k} (the target line) if and only if all control lines carry the 1 value. In particular, for $k = 0, 1$ and 2 the gates are named NOT, CNOT and Toffoli, respectively. These reversible gates together with $\mathrm{C^3NOT}$ are depicted in Fig. 1.

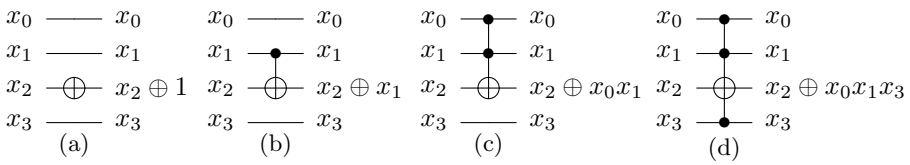

Figure 1: Representation of (a) $\mathrm{NOT}(x_2) = x_2 \oplus 1$, (b) $\mathrm{CNOT}(x_1,x_2) = (x_1, x_2 \oplus x_1)$, (c) $\mathrm{Toffoli}(x_0,x_1,x_2) = (x_0,x_1,x_2 \oplus x_0x_1)$ and (d) $\mathrm{C^3NOT}(x_0,x_1,x_3,x_2) = (x_0,x_1,x_3,x_2 \oplus x_0x_1x_3)$.

An n-bit *reversible circuit* is defined as a sequence of reversible gates each acting on at most n lines. This type of circuits can be used to implement bijective functions. More specifically, since each elementary logical gate AND, OR or NOT can be simulated by a constant number of NOT, CNOT, and Toffoli gates, reversible circuits form a computational model equivalent to combinational circuits.

The *problem of reversible circuit synthesis* is defined as, given a bijective function $f : \{0,1\}^n \to \{0,1\}^n$, find an n-bit reversible circuit that computes f. The input of the problem is a permutation π of S_{2^n} and the output is the reversible circuit that transforms the identity permutation into π. We aim to find the optimal reversible circuit.

The C^{k}NOT gates, with positive and negative controls, called Generalized Toffoli (GT) gates, allow positive and negative controls to flip the bit in the target line. These gates flip the bit in the target line if and only if each positive (or negative) control line carries the 1 (or 0) value. We indicate a negative control line with $'$ after the label of the respective parameter. The values on the remaining lines (without control) do not affect any output. If $k = n-1$, we have the generalized Toffoli gates that are totally controlled, called Totally GT gates. On the other hand, if $k < n-1$, we have the generalized Toffoli gates that are partially controlled, called Partially GT gates.

Definition 1. The *Hamming distance* $d_H(i,j)$ between two values i and j in binary representation is the number of bit positions that differ. Two values i and j are *adjacent* if $d_H(i,j) = 1$.

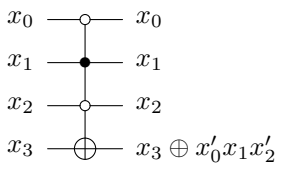

Figure 2: Totally GT gate representing C^{3}NOT$(x_0', x_1, x_2', x_3) = (x_0', x_1, x_2', x_3 \oplus x_0' x_1 x_2')$. This gate changes the sequence $x_3x_2x_1x_0 = 0010$ into 1010 (*in binary representation*) and vice versa. The bottom line denotes the most significant bit. An equivalent representation is $X(2, 10)$.

An equivalent representation for the Totally GT gate is $X(i,j)$ to indicate that it swaps values i and j, where i and j should be adjacent. We may also understand the $X(i,j)$ notation as follows. Let us consider $i < j$ and t the bit position that they differ. Let $b_{n-1} \cdots b_{t+1}\mathbf{0}b_{t-1}...b_0$ and $b_{n-1} \cdots b_{t+1}\mathbf{1}b_{t-1}...b_0$ be the binary representations of i and j, respectively, where b_{n-1} is the most significant bit. In an n-bit circuit, the $X(i,j)$ gate is equivalent to C^{n-1}NOT$(b_0, \ldots, b_{t-1}, b_{t+1}, \ldots, b_{n-1}, b_t)$. Note that the target line must be indicated as the last parameter in a C^{n-1}NOT gate, and the order of the other parameters, which indicate the control lines, is not relevant. Throughout the text we shall use the control parameters in crescent order of the indices. See Fig. 2 for an example of a Totally GT gate, and see Fig. 3 for an example of a reversible circuit that transforms the identity permutation into π. To clarify our use of equivalent representations, we include a corresponding Table 1 with the explicit sequence of used gates. The choice of the example is justified in Sec. 3.3, where we present in Figs. 5 and 6 some optimized equivalent circuits returned by our proposed algorithm.

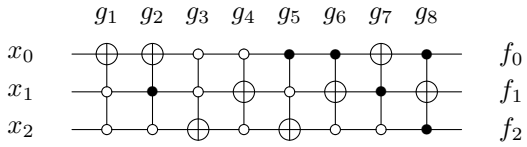

Figure 3: Example of a reversible circuit that transforms the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ using only Totally GT gates. The sequence of gates (reading from left to right) is $g_1 = X(0,1)$, $g_2 = X(2,3)$, $g_3 = X(0,4)$, $g_4 = X(0,2)$, $g_5 = X(1,5)$, $g_6 = X(1,3)$, $g_7 = X(2,3)$ and $g_8 = X(5,7)$. We consider the representation of the numbers with x_0 being the least significant bit and x_2 being the most significant bit. The output bits are denoted by $f_2 f_1 f_0$.

Table 1: Sequence of gates $g_1 = X(0,1)$, $g_2 = X(2,3)$, $g_3 = X(0,4)$, $g_4 = X(0,2)$, $g_5 = X(1,5)$, $g_6 = X(1,3)$, $g_7 = X(2,3)$ and $g_8 = X(5,7)$ that transforms the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ using only Totally GT gates (See Fig. 3). In bold, the bit changes introduced by the corresponding gates. The result of application of the gate on the bottom of each column is shown in the next column while reading from left to right.

	$x_2x_1x_0$								$f_2f_1f_0$
	000	00**1**	001	001	001	**1**01	101	101	1**1**1 = 7
	001	00**0**	000	**1**00	100	100	100	100	100 = 4
	010	010	01**1**	011	011	011	0**0**1	001	001 = 1
	011	011	01**0**	010	0**0**0	000	000	000	000 = 0
	100	100	100	**0**00	0**1**0	010	010	01**1**	011 = 3
	101	101	101	101	101	**0**01	0**1**1	01**0**	010 = 2
	110	110	110	110	110	110	110	110	110 = 6
	111	111	111	111	111	111	111	111	1**0**1 = 5
Gates:	g_1 ↗	g_2 ↗	g_3 ↗	g_4 ↗	g_5 ↗	g_6 ↗	g_7 ↗	g_8 ↗	

Definition 2. Let π and σ be permutations with n elements each. The *Hamming distance* $d_H(\pi,\sigma)$ is the sum of $d_H(\pi_i,\sigma_i)$ where π_i and σ_i are the elements in position i of π and σ respectively, for $0 \leq i < n$. We denote by $d_H(\pi)$ the Hamming distance between π and the identity ι.

For example, the Hamming distance $d_H(\pi)$ between the permutation $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ and the identity is the sum $3+2+2+2+3+3+0+1 = 16$. A circuit that transforms the identity into π in given in Fig. 3.

Our algorithm uses the cycle representation of a permutation. Let $\pi = [\pi_0\ \pi_1\ \ldots\ \pi_{2^n-1}]$ be a permutation and let $\pi = C_1C_2\ldots C_m$ be its corresponding cycle representation. Each cycle C can be represented by $C = (c^1c^2\ldots c^{|C|})$,

where $c^{i+1} = \pi_{c^i}$ with $c^0 = c^{|C|}$. Therefore, in case $c^i = \pi_j$, we have $d_H(c^{i-1}, c^i) = d_H(j, \pi_j)$. We denote by $d_H(\pi_j)$ the Hamming distance between π_j and j, and we shall use both names $d_H(c^{i-1}, c^i)$ and $d_H(\pi_j)$ interchangeably. A permutation π is unicyclic if it can be represented by a single cycle.

Given a cycle C of length $|C|$ in a permutation π, we define

$$S(C) = \sum_{i=1}^{|C|} d_H(c^{i-1}, c^i), \tag{1}$$

with $c^0 = c^{|C|}$. It follows immediately from the definition that

$$d_H(\pi) \equiv d_H(\pi, \iota) = \sum_{\ell=1}^{m} S(C_\ell), \tag{2}$$

where m is the number of cycles of π.

For example, for the permutation π given by Fig. 3 the cycle representation is (0 7 5 2 1 4 3)(6), and $d_H(\pi) = 16 + 0 = 16$.

3 Algorithm

Our algorithm is divided in two parts. The first part which is described in Subsection 3.1 consists in searching Partially GT gates with the goal of finding the gate that gives us the greatest decrease in Hamming distance, increasing progressively the number of controls. The permutation associated with each gate is obtained in a preprocessing step and stored in an array. The second part which is described in Subsection 3.2 consists only in trying to apply Totally GT gates. Finally in Subsection 3.3, our Synthesis algorithm with progressive increase of controls in GT gates is described using the algorithms obtained in the First and Second parts.

3.1 First Part: Partially GT gates

The goal is to decrease the Hamming distance between the current permutation and the identity, using as few controls as possible for the generalized Toffoli gates at the beginning of the algorithm. Hence, we first try the NOT gates, while it is possible to decrease the Hamming distance, then we try the CNOT gates, then we try the gates with two controls, and so on, until we get to the Totally GT gates.

In a preprocessing part, we build an array with all possible logical gates for the Generalized Toffoli library for n-bit circuits. The number of possible GT gates is $n3^{n-1}$. Indeed, for each line that the target bit occupies among the n possible lines, the $n-1$ remaining lines can be occupied in three different ways:

positive control, negative control or uncontrolled. The construction of the array begins with the Totally GT gates, i.e., gates with $n-1$ controls. From these gates, we build Toffoli gates with $n-2$ controls, which in turn are used to build the Toffoli gates with $n-3$ controls, and so on.

The permutations associated with the Totally GT gates are of the form $(i\ j)$ in the cyclic notation, which corresponds to one transposition. In turn, the permutations associated with the GT gates with $n-2$ controls are of the form $(i\ j)(k\ l)$, while the permutations associated with the GT gates with $n-3$ controls are formed by four transpositions, and so on.

We present in Fig. 4 an example for 4 bits, where the composition of two Totally GT gates corresponding to (4 12) and (5 13) is equivalent to one GT gate (4 12)(5 13) with 2 controls.

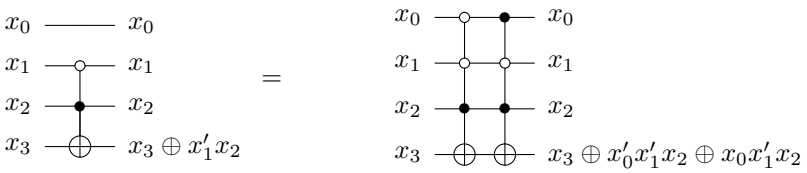

Figure 4: The $\mathrm{C^2NOT}(x_1', x_2, x_3)$ gate is equivalent to the composition of gates $\mathrm{C^3NOT}(x_0', x_1', x_2, x_3)$ and $\mathrm{C^3NOT}(x_0, x_1', x_2, x_3)$.

We say a transformation T over a permutation π is a *d-move* if $d_H(\pi) - d_H(T(\pi)) = d$, or equivalently, $d_H(\pi, T(\pi)) = d$.

Lemma 3. *If T is a d-move corresponding to a* GT *gate, with n bits and c controls, where $0 \leq c < n$, then $|d| \leq 2^{n-c}$ and $|d|$ is even.*

Proof. Let π be a permutation. For each d-move T corresponding to a GT gate, with n bits and c controls, where $0 \leq c < n$, we have $d_H(\pi \oplus \sigma, 0) = 2^{n-c}$, where $\sigma = T(\pi)$ and $\oplus$ is the bitwise XOR operation. Therefore, exactly 2^{n-c} bits (in distinct elements) are swapped by T. Suppose that among the 2^{n-c} bits swapped for T there were q bits that before the swap differed from the corresponding bits of the ι identity permutation. When effecting the swap, such q bits will coincide with the corresponding bits of the ι. On the other hand, the $p = 2^{n-c} - q$ bits that initially coincided with the corresponding bits in ι will now be different. So, d-move will be such that $d = q - (2^{n-c} - q) = 2(q - 2^{n-c-1})$. Therefore, d is even and $|d| = |q - p| = |2(q - 2^{n-c-1})| \leq 2^{n-c}$. □

Thus, in particular, if T is a transformation corresponding to a Totally GT gate, then T is either a 2-move, or a 0-move, or a -2-move.

For example, let $\pi = (0\ 7\ 6\ 1\ 2\ 11\ 13\ 12\ 3\ 15\ 14\ 8\ 10\ 5\ 9\ 4)$ be a unicyclic permutation. The Partially GT gate $\mathrm{C^2NOT}(x_1, x_3', x_0)$, which corresponds to $(2\ 3)(6\ 7)$ applied to π, returns $\sigma = (0\ 6\ 1\ 3\ 15\ 14\ 8\ 10\ 5\ 9\ 4)(2\ 11\ 13\ 12)(7)$ and $d_H(\pi) - d_H(\sigma) = 34 - 30 = 4$, so gate $\mathrm{C^2NOT}(x_1, x_3', x_0)$ applied to π is a 4-move. Gate $\mathrm{C^2NOT}(x_1, x_3', x_0)$ applied to σ is a -4-move. The Partially GT gate $\mathrm{C^2NOT}(x_1', x_3, x_0)$, which corresponds to $(8\ 9)(12\ 13)$ applied to π, returns $\lambda = (0\ 7\ 6\ 1\ 2\ 11\ 12\ 3\ 15\ 14\ 9\ 4)(5\ 8\ 10)(13)$ and $d_H(\pi) - d_H(\lambda) = 34 - 36 = -2$, so gate $\mathrm{C^2NOT}(x_1', x_3, x_0)$ is a -2-move. Gate $\mathrm{C^2NOT}(x_1', x_3, x_0)$ applied to λ is a 2-move.

Definition 4. A d-move is a *useful move* if and only if $d \geq 0$.

So, in the first part of our new algorithm for synthesis of reversible circuits we use Partially GT gates. Initially, we apply GT gates with few controls, starting with NOT gates. First, all NOT gates are tested. The NOT gate that results in the larger d-move will be applied, where d is even and $0 < d \leq 2^n$. This procedure will be applied while the Hamming distance between the current permutation for identity decreases. Then, all CNOT gates are tested, and the CNOT gate that results in the largest d-move is applied, where d is even and $0 < d \leq 2^{n-1}$. Again, such a procedure will be applied while the Hamming distance between the current permutation for identity decreases. The same procedure is applied for the gates $\mathrm{C^2NOT}$ and for all other gates with more controls, until only Totally GT gates are applied in the second part of the algorithm. In this case, the applied d-move is such that $d = 0$ or $d = 2$.

Algorithm 1 below is the first part of our main algorithm, its library is composed only of GT gates with at most $n - 2$ controls. The $L(c)$ is the library of Toffoli gates with exactly c controls.

3.2 Second Part: Totally GT gates

The next lemmas will be used in the second part of our new algorithm, when only Totally GT gates will be used.

Lemma 5. *Let σ be the permutation resulting from the application of the $X(i, j)$ gate to π, and let $m(\sigma)$ be the number of cycles in its cycle representation. If i and j are on the same cycle in π, then $m(\sigma) = m(\pi)+1$. Otherwise, $m(\sigma) = m(\pi)-1$.*

Proof. In case $i = c^s$ and $j = c^t$ belong to the same cycle C, with $s < t$, the gate $X(i, j)$ splits C into two cycles, one defined by the sequence of elements $c^s, \ldots, c^{t-1}$, and the other defined by the remaining elements of C. On the other hand, in case $i = c_1^s$ and $j = c_2^t$ belong to distinct cycles C_1 and C_2, the gate $X(i, j)$ joins C_1 and C_2 into one cycle C defined by the sequence so that c_1^{s-1}

Algorithm 1: First Part: Partially GT gates

Input: a permutation π_0 representing a bijective function.
Output: a permutation π such that $d_H(\pi) \leq d_H(\pi_0)$ and a circuit G that transforms π_0 into π.

```
1  π ← π_0;
2  G ← ∅;
3  for c ← 0 to n − 2 do
4  |   continue ← True;
5  |   while continue do
6  |   |   d_max ← 0;
7  |   |   for g ∈ L(c) do
8  |   |   |   d ← d_H(π) − d_H(g(π));
   |   |   |   /* See Lemma 3                                  */
9  |   |   |   if d > d_max then
10 |   |   |   |   d_max ← d;
11 |   |   |   |   g_max ← g;
12 |   |   |   end
13 |   |   end
14 |   |   if d_max = 0 then
15 |   |   |   continue ← False;
16 |   |   else
17 |   |   |   update π and add g_max in G;
18 |   |   end
19 |   end
20 end
21 return π and G;
```

precedes c_2^t, c_2^{t-1} precedes c_1^s and the other elements remain according to the order of C_1 and C_2. □

Given three elements i, j and k of a permutation, we say k is in a *minimum path* between i and j if $d_H(i,j) = d_H(i,k) + d_H(k,j)$. Moreover, notice that if $d_H(s,t) = 1$, then for every element u, either s is in a minimum path between u and t, or t is in a minimum path between s and u. Notice that, in case values i and j are adjacent, we have $d_H(i,j) = 1$ and for every other element u, $|d_H(i,u) - d_H(j,u)| \leq 1$.

Lemma 6. *Let π be a permutation such that π_i and π_j are adjacent values and consider the following possible situations: (a) π_j is in a minimum path between i and π_i; (b) π_i is in a minimum path between j and π_j. If both conditions (a) and (b) are simultaneously satisfied, then gate $X(\pi_i, \pi_j)$ applied to π is a 2-move. If only a single condition (a) or (b) is satisfied, then gate $X(\pi_i, \pi_j)$ applied to π*

is a 0*-move. If neither condition (a) nor condition (b) are satisfied, then gate* $X(\pi_i, \pi_j)$ *applied to* π *is a* -2*-move.*

Proof. Let π' be the permutation resulting from the application of $X(\pi_i, \pi_j)$ to the permutation π. The only elements modified by the application of this gate are π_i and π_j, therefore $d_H(\pi, \pi') = d_H(i, \pi_j) + d_H(j, \pi_i) - d_H(i, \pi_i) - d_H(j, \pi_j)$.

First, let us consider the case in which both conditions (a) and (b) are simultaneously satisfied. In this case, $d_H(i, \pi_i) = d_H(i, \pi_j) + d_H(\pi_j, \pi_i)$ and $d_H(j, \pi_j) = d_H(j, \pi_i) + d_H(\pi_i, \pi_j)$. Therefore, $d_H(i, \pi_i) = d_H(i, \pi_j) + 1$ and $d_H(j, \pi_j) = d_H(j, \pi_i)$ $+1$. Thus, $d_H(\pi, \pi') = 2$, indicating that $X(\pi_i, \pi_j)$ is a 2-move.

Now, let us consider the case in which condition (a) is satisfied and condition (b) is not. In this case, $d_H(i, \pi_i) = d_H(i, \pi_j) + d_H(\pi_j, \pi_i)$ and π_j is in the minimum path between j and π_i. Then, $d_H(j, \pi_i) = d_H(j, \pi_j) + d_H(\pi_j, \pi_i)$. Therefore, $d_H(i, \pi_j) = d_H(i, \pi_i) - 1$ and $d_H(j, \pi_i) = d_H(j, \pi_j) + 1$. Thus, $d_H(\pi, \pi') = 0$, indicating that $X(\pi_i, \pi_j)$ is a 0-move.

The analyses of the remaining cases are analogous. □

Lemma 7. *If* $\pi \neq \iota$ *is not unicyclic, then there is a useful move corresponding to some Totally* GT *gate which joins cycles.*

Proof. The only permutation that does not allow a 0-move nor a 2-move is the identity. Let us consider a not unicyclic permutation $\pi \neq \iota$ which does not allow a 2-move. This implies that π has more than one cycle and there is at least one cycle with two distinct elements. Then, there is a cycle in π in which there are two consecutive elements c^i and c^{i+1} such that there is an element c' in another cycle that belongs to the minimum path between c^i and c^{i+1}. □

For example, the gate $X(2, 3)$ applied to $\pi = (0\ 3\ 1)(2)$ returns $\lambda = (0\ 2\ 3\ 1)$.

Lemma 8. *Let* $C = (c^1 c^2 \cdots c^{|C|})$ *be a cycle with at most one pair of neighbor elements that are not adjacent. Then there is a sequence of* $|C| - 1$ *Totally* GT *gates which transforms this cycle into* $|C|$ *unitary cycles.*

Proof. Without loss of generality, consider a cycle $C = (c^1 c^2 \cdots c^{|C|})$ such that the only possible not adjacent pair is $(c^{|C|} c^1)$. Applying Totally GT gate $X(c^1, c^2)$, this cycle C is split in two cycles $C_1 = (c^1)$ and $C' = (c^2 \cdots c^{|C|})$. After applying the sequence of gates $X(c^2, c^3)$, $X(c^3, c^4), \ldots, X(c^{|C|-1}, c^{|C|})$, we obtain the unitary cycles $(c^1), (c^2), ..., (c^{|C|})$. □

For example, let $\pi = (0\ 1\ 3\ 7\ 6)(2)(4)(5)$. Then, the sequence of gates $X(0, 1)$, $X(1, 3)$, $X(3, 7)$ and $X(6, 7)$ transforms π into ι.

Given a cycle C of length $|C|$ in a permutation π, we define

$$P(C) = \frac{S(C)}{|C|}, \tag{3}$$

and

$$P(\pi) = \sum_{i=1}^{m} P(C_i). \tag{4}$$

For example, let $\pi = C_1C_2 = (0\ 3\ 1)(2)$. We have that $S(C_1) = 4$ and $S(C_2) = 0$, therefore $P(C_1) = 1.33$, $P(C_2) = 0$ and $P(\pi) = 1.33$.

Lemma 9. *Let σ be the permutation resulting from the application of gate $X(i, j)$ to π. If $X(i, j)$ is a useful move that joins cycles in π, then we have that $P(\sigma) < P(\pi)$.*

Proof. Let $X(i, j)$ be a d-move gate that can be applied to π such that $i \in C'$ and $j \in C'' \neq C'$, creating a new cycle C''' with the union of the two cycles. Let $P(C') = S(C')/|C'|$ and $P(C'') = S(C'')/|C''|$ and $P(C''') = S(C''')/|C'''|$. Since $S(C''') = S(C') + S(C'') - d$ and $|C'''| = |C'| + |C''|$, we have that

$$\begin{aligned} P(C''') &= \frac{S(C') + S(C'') - d}{|C'| + |C''|} \\ &= \frac{P(C') \cdot |C'|}{|C'| + |C''|} + \frac{P(C'') \cdot |C''|}{|C'| + |C''|} - \frac{d}{|C'| + |C''|}. \end{aligned} \tag{5}$$

Considering that $|C'| < |C'|+|C''|$ and $|C''| < |C'|+|C''|$, if $d = 0$ or $d = 2$, then we have that $P(C''') < P(C') + P(C'')$. Since the only cycles modified in π were C' and C'', we have that $P(\sigma) = P(\pi) + P(C''') - P(C') - P(C'')$. Therefore $P(\sigma) < P(\pi)$. □

Definition 10. A sequence of useful moves is a *useful sequence* if it decreases $d_H(\pi)$ or $P(\pi)$.

In order to finish the analysis of the algorithm 2 convergence, we need to see the case in which the permutation is unicyclic and there is no 2-move. For this goal, we use the next procedures and lemmas.

Lemma 11. *Procedure 1 puts j in the position of i in π, going through all the elements of a minimum path between i and j.*

Proof. In each step of the loop, in which the condition of 'if' is true, i' has the same bits of i except bit k, and $d_H(i', j) = d_H(i, j) - 1$, therefore i' is in a minimum path between i and j. After applying the gate $X(i, i')$, i' stays in position of π occupied by i. After applying $d_H(i, j)$ gates, the element j occupies the place of original i. The last i' is j. □

Procedure 1: Replace(i, j)

Input: $i = i_{n-1}...i_0, j = j_{n-1}...j_0 \in \pi$ $\{i_k$ is the bit k of $i\}$
Output: $T(\pi) = X(i, j)\pi$.

```
for k ← 0 to n − 1 do
    if i_k ≠ j_k then
        apply the gate X(i, i'), where i' is the adjacent element of i s.t.
          i'_k = j_k;
        i ← i';
    end
end
```

Lemma 12. *Let i and j be elements of π, such that i is successor of j in a cycle, then after Replace(i, j) there is a unitary cycle (j).*

Proof. If i is successor of j in a cycle then i is in position j ($\pi_j = i$). Therefore, Replace(i, j) puts j in position j. □

Procedure 2: Useful sequence with 2-move

Input: Current permutation π.
Output: A useful sequence with 2-move.

```
j ← 2^n − 1;
repeat
    let i be successor of j in the respective cycle, Replace(i, j);
    j ← j − 1;
until Replace(i, j) has a 2-move;
```

Lemma 13. *Procedure 2 outputs a useful sequence.*

Proof. If j is in a unitary cycle then successor of j is j and so Replace(i, j) has no move, else all moves are useful moves. Indeed, in each gate of Replace(i, j), the present i' is in a minimum path between j and π_j so, by Lemma 6, this move is a useful move.

This Procedure converges because, in each iteration, the elements greater than j are in unitary cycles and they do not belong to the minimum path between i and j used by Replace(i, j). □

Algorithm 2 is used when it is not possible to apply partially controlled GT gates that decrease the Hamming distance of the permutation.

Algorithm 2: Second Part: Totally GT gates

Input: a permutation π_0 representing a bijective function.
Output: the circuit G that transforms π_0 into ι.

$\pi \leftarrow \pi_0$;
$G \leftarrow \emptyset$;
while $d_H(\pi) > 0$ **do**
  if *there is a* 2*-move that joins cycles* **then**
    choose $X(c^i, j)$ corresponding to the 2-move;
    apply the corresponding reversible gate;
    update π and G;
  else if *there is a* 2*-move that splits cycles* **then**
    apply the corresponding reversible gate;
    update π and G;
  else if *there is a sequence of gates that splits cycle in unitary cycles* **then**
    apply the sequence of gates;
    update π and G;
    /* See Lemma 8 */
  else if *there is a* 0*-move that joins cycles* **then**
    choose $X(c^i, j)$ corresponding to the 0-move;
    apply the corresponding reversible gate;
    update π and G;
    /* Necessarily decreases the value of $P(\pi)$, see Lemma 9 */
  else
    /* π is unicyclic, see Lemma 7 */
    apply Procedure 2;
    update π and G;
  end
end
return G;

Theorem 14. *Correctness of Algorithm 2.*

Proof. At each step, the algorithm either applies a 2-move or a 0-move minimizing $P(\pi)$, see Lemmas 5 to 9 and Lemma 13. Therefore, at each step the Hamming distance between π and ι is kept constant or decreases. When the Hamming distance is kept constant, the function $P(\pi)$ decreases until the permutation is unicyclic, see Lemmas 5, 7 and 9. In this case, the algorithm applies a useful sequence that applies a 2-move, see Lemma 13. Thus, the permutation always converges to the identity permutation ι. □

3.3 Algorithm with progressive increase of controls in GT gates

Our synthesis method uses gates with $0, 1, ..., n-1$ controls, increasing the number of controls progressively. The synthesis algorithm with progressive increase of controls in GT gates (Algorithm 3) summarized uses Algorithms 1 and 2. See Algorithm 3 for a sketch of our synthesis method using gates with $0, 1, ..., n-1$ controls.

Algorithm 3: Synthesis algorithm with progressive increase of controls in GT gates

Input: a permutation π_0 representing a bijective function.
Output: the circuit G that transforms ι into π_0.
1 $(\pi, G_1) \leftarrow$ apply Algorithm 1 to π_0;
2 $G_2 \leftarrow$ apply Algorithm 2 to π;
3 $G \leftarrow reverse(G_1 G_2)$;
4 return G;

Theorem 15. *Correctness of Algorithm 3.*

Proof. In the first steps, for each quantity k of controls, $k \in \{0, 1, \ldots, n-2\}$, the algorithm applies d-moves ($d > 0$, largest possible, according to Lemma 3), such that the Hamming distance between the current permutation for identity decreases as much as possible. When arriving at $k = n-1$ controls, just use Theorem 14. □

Figure 5 depicts the circuit that transforms identity into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, for the case where the input of Algorithm 3 is $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, considered in the example in Figure 3. Note that the order of the returned gates is reversed in relation to the order in which the gates are obtained.

Algorithm 3 produces the circuit by selecting generalized Toffoli gates that manipulate the output side of the circuit. Since the permutation is reversible, we can define strategies that use Algorithm 3 to improve the gate count.

Strategy 1 (Synthesis of the Inverse) *Apply an algorithm to the input permutation and its inverse. Then, choose the circuit with the least number of gates.*

Strategy 2 (Bidirectional Synthesis) *Apply an algorithm simultaneously in both directions. Choose to add input-side or output-side gates of the circuit at each step of the synthesis.*

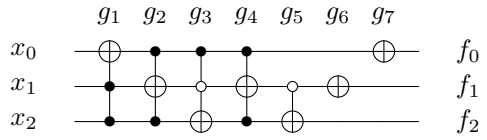

Figure 5: Example of a reversible circuit returned by Algorithm 3, that transforms the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. The output bits are denoted by $f_2 f_1 f_0$. The sequence of gates (reading from left to right) is $g_1 = \text{Toffoli}(x_1, x_2, x_0)$, $g_2 = \text{Toffoli}(x_0, x_2, x_1)$, $g_3 = \text{Toffoli}(x_0, x_1', x_2)$, $g_4 = \text{Toffoli}(x_0, x_2, x_1)$, $g_5 = \text{CNOT}(x_1', x_2)$, $g_6 = \text{NOT}(x_1)$ and $g_7 = \text{NOT}(x_0)$, where $g_7 g_6 g_5$ and $g_4 g_3 g_2 g_1$ are the circuits returned by Algorithms 1 and 2 respectively.

Figure 6 depicts the circuits that transform the identity into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$, obtained with Strategy 1 and Strategy 2, respectively, where the input of Algorithm 3 is $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. As we did before, to clarify our use of equivalent representations, we include a corresponding Table 2 with the explicit sequence of used gates corresponding to the circuit depicted in Fig. 6b.

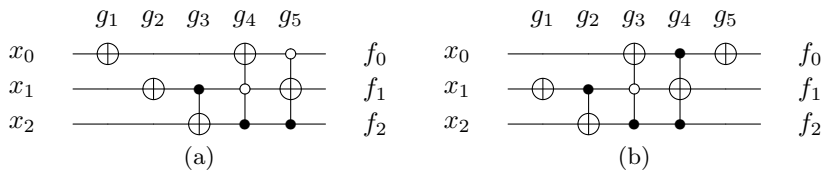

Figure 6: Example of reversible circuits, obtained from Algorithm 3 with (a) Strategy 1 and (b) Strategy 2, respectively, that transform the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$. The output bits are denoted by $f_2 f_1 f_0$. (a) The sequence of gates is $g_1 = \text{NOT}(x_0)$, $g_2 = \text{NOT}(x_1)$, $g_3 = \text{CNOT}(x_1, x_2)$, $g_4 = \text{Toffoli}(x_1', x_2, x_0)$ and $g_5 = \text{Toffoli}(x_0', x_2, x_1)$. (b) The sequence of gates is $g_1 = \text{NOT}(x_1)$, $g_2 = \text{CNOT}(x_1, x_2)$, $g_3 = \text{Toffoli}(x_1', x_2, x_0)$, $g_4 = \text{Toffoli}(x_0, x_2, x_1)$ and $g_5 = \text{NOT}(x_0)$.

4 Experimental Results

Firstly, we have considered all 3-bit bijective functions and compared our results with different synthesis algorithms. The results are in Table 3. Column **Size** shows the number of gates of the circuit. The main current results appear in two columns. Column **CGWZ** shows the results of synthesis simplification algorithm by Cheng *et al.* [Cheng et al., 2012]. Column **ZLZPZ** shows

Table 2: Sequence of gates $g_1 = \text{NOT}(x_1)$, $g_2 = \text{CNOT}(x_1, x_2)$, $g_3 = \text{Toffoli}(x_1', x_2, x_0)$, $g_4 = \text{Toffoli}(x_0, x_2, x_1)$ and $g_5 = \text{NOT}(x_0)$ that transforms the identity permutation into $\pi = [7\ 4\ 1\ 0\ 3\ 2\ 6\ 5]$ using the Algorithm 3 and Strategy 2, see the corresponding circuit in Fig. 6b. In bold, the bit changes introduced by the corresponding gates. The result of application of the gate on the bottom of each column is shown in the next column while reading from left to right.

	$x_2x_1x_0$					$f_2f_1f_0$
	000	0**1**0	**1**10	110	110	11**1** = 7
	001	0**1**1	**1**11	111	1**0**1	10**0** = 4
	010	0**0**0	000	000	000	00**1** = 1
	011	0**0**1	001	001	001	00**0** = 0
	100	1**1**0	**0**10	010	010	01**1** = 3
	101	1**1**1	**0**11	011	011	01**0** = 2
	110	1**0**0	100	10**1**	1**1**1	11**0** = 6
	111	1**0**1	101	10**0**	100	10**1** = 5
Gates:	g_1 ↗	g_2 ↗	g_3 ↗	g_4 ↗	g_5 ↗	

the results of cycle-decomposition-based synthesis optimized algorithm by Zhu *et al.* [Zhu et al., 2018]. The results of applying Algorithm 3 to all 3-bit bijective functions are in columns **(a)**, **(b)** and **(c)** according to different strategies as follows:

- In column (a), application of Algorithm 3.
- In column (b), application of Algorithm 3 improved by Strategy 1.
- In column (c), application of Algorithm 3 improved by Strategy 1 plus Strategy 2, followed by choosing the minimum value between the two results.

The improved Algorithm 3 considered in column (c) obtained the best number 5.23 of gates in average. The result 5.38 obtained by the algorithm of Cheng *et al.* [Cheng et al., 2012] also considers Strategy 1 of their simplification algorithm, in addition to the use of templates. Our Algorithm 3 considered in column (a) obtained 5.82 gates in average, improving the result of Zhu *et al.* [Zhu et al., 2018], which is also a unidirectional algorithm. Moreover, Zhu *et al.* algorithm is specific to 3 bits while our Algorithm 3 works for an arbitrary number of bits. The main idea in Zhu *et al.* algorithm is to decompose cycles in permutations and use the so-called Head-Pointer-Adjust, which takes the greatest Hamming distance between elements of the cycle.

Our method is an upgrade of the cycle-based synthesis (CBS) algorithm of Ribeiro *et al.* [Ribeiro et al., 2015], including Partially GT gates at the library,

Table 3: Number of bijective functions using a specified number of gates, considering all 3-bit bijective functions for different synthesis algorithms as indicated by citations. Our results are listed in columns (a), (b) and (c). The main current results using GT library are in columns **CGWZ** [Cheng et al., 2012] and **ZLZPZ** [Zhu et al., 2018]. Row AG reports the average number of gates necessary to synthesize a circuit.

	Algorithm 3			Current literature	
Size	(a)	(b)	(c)	CGWZ	ZLZPZ
0	1	1	1	1	1
1	27	27	27	27	15
2	309	369	369	369	129
3	1797	2505	2601	2633	753
4	5376	7586	8114	7624	3100
5	9758	12932	12994	11263	8409
6	10529	9940	10066	10258	13405
7	7046	4523	4652	5963	10506
8	3922	2166	1450	1716	3625
9	1206	213	24	372	369
10	319	54	18	93	8
11	30	4	4	1	0
AG	5.82	5.32	5.23	5.38	6.03

and improves the average number of gates, as follows. The average number of gates necessary to synthesize a circuit is reduced from 6.74 with unidirectional CBS algorithm to 5.82 with Algorithm 3 considering column (a), and is reduced from 6.64 with CBS algorithm applying Strategy 1 to 5.32 with Algorithm 3 applying Strategy 1 considering column (b).

Secondly, we have performed a series of experiments on reversible benchmark function synthesis. The results are in Table 4. The first and the second columns show the name of each **benchmark function** and its **size** (number of variables) considered in the literature, respectively. The third column shows the results obtained by the proposed approach, using the Algorithm 3 and strategies 1 and 2. The fourth column shows the best results for the gate count of synthesis algorithm by Zakablukov [Zakablukov, 2016] which uses properties of Permutation Group Theory to reduce gate complexity and combines cycle-based and Reed-Muller-spectra-based algorithms. The last two columns show the best results for the gate count of MMD method without/with template matching by Maslov *et al.* [Maslov et al., 2005], and the MMD method with the Reed-Muller spectra by Maslov *et al.* [Maslov et al., 2007], respectively. All specifications for benchmark function and their names were taken from the *RevLib site*[1] [Wille et al., 2008]

[1] available at http://www.revlib.org

and from the *Reversible Logic Synthesis Benchmarks Page*[2].

Table 4: Benchmark Function Synthesis. Our results are listed in column **Algorithm 3**. The main current results are in columns **Z-16** [Zakablukov, 2016], **MMD-05** [Maslov et al., 2005] and **MMD-07** [Maslov et al., 2007].

benchmark function	size	**Algorithm 3**	Z-16	MMD-05	MMD-07
3_17	3	6	4	6/6	6
4_49	4	14	-	16/16	12
4b15g_2	4	17	12	-	-
4b15g_4	4	19	12	-	-
4b15g_5	4	18	14	-	-
aj−e11	4	9	-	13*/−	-
ham3	3	6	-	6/5	5
hwb4	4	18	-	18/17	11
hwb5	5	43	-	57/55	24
hwb6	6	103	-	134/126	42
hwb7	7	282	603	302/289	236
hwb8	8	697	1594	688/−	614
hwb9	9	2633	3999	1625/−	1541
mod5adder	6	17	-	37/−	15
mod5mils	5	3	-	5/−	-
nth_prime4_inc	4	13	-	-	12**
nth_prime5_inc	5	38	-	-	25**
nth_prime6_inc	6	79	-	-	55**
nth_prime7_inc	7	231	427	-	-
nth_prime8_inc	8	627	977	-	-

* Result based on MMD method executed by [Große et al., 2009] to generate a heuristic result.
** Result available at http://webhome.cs.uvic.ca/∼dmaslov/

Analyzing the results obtained in Table 4, we see that our approach obtained in more than half of cases, best results in relation to those obtained by Zakablukov [Zakablukov, 2016] and Maslov *et al.* [Maslov et al., 2005] for benchmark function synthesis. We were able to get 9 (out of 17) reversible circuits, which have less or equal gate count in relation to the two cited references. Comparing our results with those obtained by Zakablukov, we see that we obtained better results in 5 out of 9 comparisons, being that, our synthesis algorithm performed better as we increased the number of bits. On the other hand, comparing with the results obtained by Maslov *et al.*, we see that we obtained better results in 7 out of 12 comparisons (1 equal), being that, in general our perfor-

[2] available at http://webhome.cs.uvic.ca/∼dmaslov/

mance was better for functions up to 7 bits. However, the results obtained by Maslov *et al.* [Maslov et al., 2007] based on MMD method [Maslov et al., 2005] using templates with Reed-Muller spectra are the best results so far.

Thirdly, we have made available at https://github.com/Marquezino/dkmfr an implementation of our main result to the community. In order to get a sense to what extent 8-bit permutations can be handled, please see in Figure 7 a graph where we have depicted the corresponding actual obtained running times. The obtained linear graph in the base-10 logarithmic scale agrees with the expected exponential behavior as we increase the number of bits.

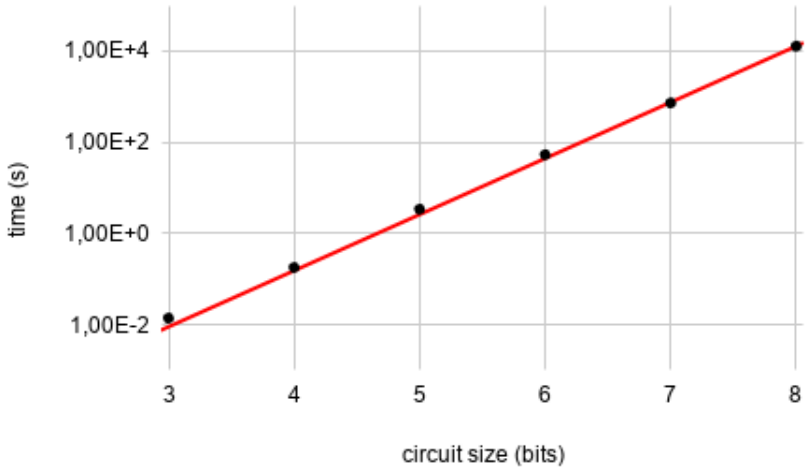

Figure 7: Actual obtained running times (in seconds) of Algorithm 3. The time-axis is represented in the base-10 logarithmic scale. We depict in red the trend line with slope 6/5.

5 Concluding remarks

In this work, we presented a new algorithm of reversible circuit synthesis using generalized Toffoli gates (GT gates). Our Algorithm 3 is an upgrade of CBS algorithm [Ribeiro et al., 2015], including Partially GT gates at the library. The new approach progressively increases the number of Toffoli gate controls. Besides that, our Algorithm 3 explores properties of the cycle representations of permutations as part of the process and uses bidirectional strategies. The circuits achieved by our new synthesis algorithm in general use less gates than the circuits achieved

by the CBS algorithm. Our Algorithm 3 works for arbitrary n-bit bijective functions, whereas the exact synthesis found in the literature work only for the cases of $n = 3$ and $n = 4$ bits [Wille et al., 2012, Szyprowski and Kerntopf, 2012]. Considering that the quantum cost [Barenco et al., 1995] of each GT gate is proportional to the number of control bits, the quantum cost of circuits generated by Algorithm 3 is, in average, less than the quantum cost of circuits generated by CBS algorithm. In the present work, we consider as a complexity metric just the number of gates, a cost model frequently used in the literature and considered adequate when staying within the traditional reversible framework [Saeedi and Markov, 2013].

The main contributions include the best so far average gate count for all 3-bit bijective functions for the GT library. Besides that, we present experimental results for reversible benchmark function synthesis, which include twenty reversible circuits consisting of gates from GT library, that are competitive when compared with the best results for the gate count of reversible synthesis heuristics in the literature.

As future research, possible directions include using templates to replace by shorter ones certain sequences of gates in the circuits returned by Algorithm 3. Moreover, we could use some other approaches such as permutation group theory and Reed-Muller spectra [Maslov et al., 2007, Zakablukov, 2016] in order to find better gate sequences. We could also apply similar techniques to the synthesis of quantum circuits [de Almeida et al., 2019], taking into account that moving to the level of quantum circuits requires a more fine-grained gate set than considered here and optimizations that align with error-correction needs.

Acknowledgements

The authors wish to thank the editor for handling our submission during the pandemic, and the two anonymous referees for the diligent reading and several useful suggestions. This work was partially supported by CAPES, CNPq and FAPERJ.

References

[Barenco et al., 1995] Barenco, A., Bennett, C. H., Cleve, R., DiVincenzo, D. P., Margolus, N., Shor, P., Sleator, T., Smolin, J. A., and Weinfurter, H. (1995). Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467.

[Bennett, 1973] Bennett, C. H. (1973). Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532.

[Cheng et al., 2012] Cheng, X., Guan, Z., Wang, W., and Zhu, L. (2012). A simplification algorithm for reversible logic network of positive/negative control gates. In *9th International Conference on Fuzzy Systems and Knowledge Discovery*, pages 2442–2446.

[Datta et al., 2013] Datta, K., Rathi, G., Wille, R., Sengupta, I., Rahaman, H., and Drechsler, R. (2013). Exploiting negative control lines in the optimization of reversible circuits. In Dueck, G. W. and Miller, D. M., editors, *Reversible Computation*, pages 209–220, Berlin, Heidelberg. Springer.

[de Almeida et al., 2019] de Almeida, A. A. A., Dueck, G. W., and da Silva, A. C. R. (2019). Efficient realization of Toffoli and NCV circuits for IBM QX architectures. In *Reversible Computation - 11th International Conference, RC 2019, Lausanne, Switzerland, June 24-25, 2019, Proceedings*, pages 131–145.

[Golubitsky et al., 2010] Golubitsky, O., Falconer, S. M., and Maslov, D. (2010). Synthesis of the optimal 4-bit reversible circuits. In *Design Automation Conference*, pages 653–656.

[Große et al., 2009] Große, D., Wille, R., Dueck, G. W., and Drechsler, R. (2009). Exact multiple-control Toffoli network synthesis with SAT techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(5):703–715.

[Iwama et al., 2002] Iwama, K., Kambayashi, Y., and Yamashita, S. (2002). Transformation rules for designing CNOT-based quantum circuits. In *Proceedings of the 39th Annual Design Automation Conference*, pages 419–424, New York, NY, USA. ACM.

[Kowada et al., 2006] Kowada, L. A. B., Portugal, R., and de Figueiredo, C. M. H. (2006). Reversible Karatsuba's algorithm. *Journal of Universal Computer Science*, 12(5):499–511.

[Landauer, 1961] Landauer, R. (1961). Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191.

[Maslov and Dueck, 2004] Maslov, D. and Dueck, G. W. (2004). Reversible cascades with minimal garbage. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(11):1497–1509.

[Maslov et al., 2005] Maslov, D., Dueck, G. W., and Miller, D. M. (2005). Toffoli network synthesis with templates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):807–817.

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

[Miller et al., 2003] Miller, D. M., Maslov, D., and Dueck, G. W. (2003). A transformation based algorithm for reversible logic synthesis. In *Proceedings of the 40th Annual Design Automation Conference*, pages 318–323, New York, NY, USA. ACM.

[Nielsen and Chuang, 2000] Nielsen, M. A. and Chuang, I. L. (2000). *Quantum Computation and Quantum Information*. Cambridge University Press, New York, NY, USA.

[Rahman and Rice, 2014] Rahman, M. Z. and Rice, J. E. (2014). Templates for positive and negative control Toffoli networks. In Yamashita, S. and Minato, S.-I., editors, *Reversible Computation*, pages 125–136, Cham. Springer International Publishing.

[Ribeiro et al., 2015] Ribeiro, A. C., Kowada, L. A. B., Marquezino, F. L., and Figueiredo, C. M. H. (2015). A new reversible circuit synthesis algorithm based on cycle representations of permutations. *Electronic Notes in Discrete Mathematics*, 50:187–192. LAGOS'15–VIII Latin-American Algorithms, Graphs and Optimization Symposium.

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

[Saeedi et al., 2010] Saeedi, M., Zamani, M. S., Sedighi, M., and Sasanian, Z. (2010). Synthesis of reversible circuit using cycle-based approach. *J. Emerg. Technol. Comput. Syst*, 6(4):13.1–13.26.

[Shende et al., 2003] Shende, V. V., Prasad, A. K., Markov, I. L., and Hayes, J. P. (2003). Synthesis of reversible logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(6):710–722.

[Szyprowski and Kerntopf, 2012] Szyprowski, M. and Kerntopf, P. (2012). A study of optimal 4-bit reversible circuit synthesis from mixed-polarity Toffoli gates. In *12th*

IEEE International Conference on Nanotechnology (IEEE-NANO), pages 1–6.

[Toffoli, 1980] Toffoli, T. (1980). Reversible computing. In de Bakker, J. and van Leeuwen, J., editors, *Automata, Languages and Programming*, page 632, New York. Springer. MIT Technical Memo No. MIT/LCS/TM-151, 1980 (unpublished).

[Wille et al., 2008] Wille, R., Große, D., Teuber, L., Dueck, G. W., and Drechsler, R. (2008). Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic*, pages 220–225.

[Wille et al., 2012] Wille, R., Soeken, M., Przigoda, N., and Drechsler, R. (2012). Exact synthesis of Toffoli gate circuits with negative control lines. In *IEEE 42nd International Symposium on Multiple-Valued Logic*, pages 69–74.

[Zakablukov, 2016] Zakablukov, D. V. (2016). Application of permutation group theory in reversible logic synthesis. In Devitt, S. and Lanese, I., editors, *Reversible Computation*, pages 223–238, Cham. Springer International Publishing.

[Zhu et al., 2018] Zhu, W., Li, Z., Zhang, G., Pan, S., and Zhang, W. (2018). A reversible logical circuit synthesis algorithm based on decomposition of cycle representations of permutations. *International Journal of Theoretical Physics*, 57(8):2466–2474.