# Search-Based Transformation Synthesis for 3-Valued Reversible Circuits

D. Michael Miller[1(✉)] and Gerhard W. Dueck[2]

[1] University of Victoria, Victoria, Canada
mmiller@uvic.ca
[2] University of New Brunswick, Fredericton, Canada
gdueck@unb.ca

**Abstract.** A novel bounded search transformation-based synthesis approach is presented that finds a reversible circuit implementation for a given reversible function. Methods for simplifying the circuit post-synthesis are presented. Quantum implementation constraints are also considered. Experimental results for all 2-input 3-valued functions show the effectiveness of the new approaches compared to earlier transformation-based synthesis approaches. Other examples are given to show both the effectiveness and limitations of the new approach which point to a number of key areas for further research.

## 1 Introduction

An $r$-valued $n$-variable reversible logic function maps each of the $r^n$ input patters to a unique output pattern. Hence the function has $n$ outputs. The synthesis problem is to realize a reversible function by a cascade of basic reversible gates. In this paper we present a novel bounded search method for this synthesis problem as well as systematic approaches to circuit simplification. Quantum circuit implementation is considered with respect to a variety of practical constraints.

Reversible functions and circuits have the interesting property that if one has a circuit for a function $f$, reversing the order of the gates and replacing each by the gate implementing the inverse operation yields a circuit realizing $f^{-1}$. Consequently, one can synthesize a circuit for $f$ and a second circuit for $f^{-1}$ and choose the better circuit as the basis to realize $f$.

Transformation-based synthesis was introduced in [4] for Boolean reversible functions and extended to MVL functions in [3,5]. A study of the MVL reversible logic synthesis including the transformation-based approach appears in [1]. The method introduced here employs a bounded recursive search to more extensively explore alternative circuits. It employs the basic pattern transform operation of earlier transformation-based synthesis approaches. The bound is based on the best circuit found to date.

Empirical results for 3-valued functions show the new search method produces significantly better circuits. Since the new method is a search, significantly more CPU time is required but this is justified by the improvement in the synthesized circuits. Limitations of the approach are discussed and issues for further research are identified.

I. Lanese and M. Rawski (Eds.): RC 2020, LNCS 12227, pp. 218–236, 2020.
https://doi.org/10.1007/978-3-030-52482-1_13

## 2 Background

### 2.1 Reversible Functions, Gates and Circuits

**Definition 1.** *An $n$-input, $n$-output, (written $n \times n$) totally-specified $r$-valued function is* reversible *if it maps each input assignment to a unique output assignment. We use $x_0, x_1, ..., x_{n-1}$ to denote the function inputs and $x_0^+, x_1^+, ..., x_{n-1}^+$ to denote the corresponding outputs. A reversible function defines a permutation of the input patterns. There are $r^n!$ $r$-valued, $n \times n$ reversible functions.* □

An $r$-valued $n \times n$ reversible function can be specified as a list $F$ with $r^n$ entries $F_0, F_1, ..., F_{r^n-1}$ where the $n$ digit $r$-valued expansion of each $F_i$ specifies the output pattern corresponding to the input pattern which is the $n$ digit $r$-valued expansion of $i$. The specification list for the identity function has the $i^{th}$ entry equal to $i$ for all $i$.

**Definition 2.** *For a given $f$ with specification $F$, the* distance *between $f$ and the identity function is given by*

$$\triangle(f) = \sum_{j=0}^{r^n-1} d(j, F_j) \qquad d(a, b) = \sum_{k=0}^{n-1} |a_k - b_k|$$

*where $a_k$ and $b_k$ denote the $k^{th}$ digit in the $r$-valued expansion of $a$ and $b$, respectively.* □

**Definition 3.** *A* reversible gate *has $p$ inputs and $p$ outputs and realizes a $p \times p$ reversible function.* □

In this work, we employ 3-valued reversible gates given by the following definition:

**Definition 4.** *A* 3-valued $p \times p$ controlled unary reversible gate *passes $p-1$* control *lines through unchanged, and applies a specified unary operator to the $p^{th}$ line, the* target *line, if the control lines assume particular specified values. Otherwise the target line is passed through unaltered. The permitted unary operators are the five listed in Table 1. Note that a gate must have a single target and the case of 0 controls ($p = 1$) is permitted in which case the gate is said to be* uncontrolled. □

**Definition 5.** *A reversible circuit realizing an $n \times n$ reversible function is a cascade of reversible gates with no fanout or feedback [7]. The circuit has $n$ inputs and $n$ outputs and is thus identified as $n \times n$.* □

The synthesis problem considered here is how to realize a given reversible function specification as a circuit using a basic set of reversible gates. The presentation focuses on 3-valued functions and circuits but the methods can be readily extended to a higher radix.

We will use circuit diagrams where targets are boxes labeled by the appropriate unary operation and controls are shown as circles containg the control value for reversible circuits and as • for quantum circuits.

**Table 1.** 3-valued unary operators

| $x$ | $C_1[x]$ | $C_2[x]$ | $N[x]$ | $D[x]$ | $E[x]$ |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 2 | 0 | 1 |
| 1 | 2 | 0 | 1 | 2 | 0 |
| 2 | 0 | 1 | 0 | 1 | 2 |

Consider the operators in Table 1. $C_1$ and $C_2$ are inverses of each other. $D$, $E$ and $N$ are each self-inverse. The following readily verified identities are used in the circuit simplification techniques discussed later in this paper.

$$C_2[C_2[x]] = C_1[x] \tag{1}$$
$$C_1[C_1[x] = C_2[x] \tag{2}$$
$$D[x] = E[C_1[x]] = N[C_2[x]] \tag{3}$$
$$E[x] = D[C_2[x]] = N[C_1[x]] \tag{4}$$
$$N[x] = D[C_1[x]] = E[C_2[x]] \tag{5}$$

## 2.2 Quantum Circuits

Reversible circuits may be implemented in a variety of technologies. Here we are interested in potential quantum circuit implementations [6–8,10]. The objective is to map a reversible circuit composed of reversible gates as defined in Definition 4 to a circuit composed of gates directly implementable in a given quantum technology.

Muthukrishnan and Stroud [6] introduced a family of elementary ternary quantum gates (MS gates) widely used in the quantum MVL circuit literature which for the ternary case can be defined as follows:

**Definition 6.** *A Muthukrishnan and Stroud (MS) gate is a gate as defined in Definition 4 with at most one control.*

Muthukrishnan and Stroud considered ion trap technology for implementing these gates and for the ternary case required that all control values be 2. Here, we do not assume a particular underlying technology and consider a number of possible scenarios. In particular, we consider situations where only a subset of the MS gates are physically available since in some technologies certain MS gates are readily implemented while others are more costly or may not be implementable. In addition, we require that all controls in a quantum circuit have the same *global control value* ($cv$). We will consider cases with $cv = 1$ or 2.

It is clear from equations (1) to (5) that a single cycle gate, $C_1$ or $C_2$, and at least one of $D$, $E$ or $N$, is sufficient as the other gates can be implemented by suitable gate pairings. In this work, we distinguish between the gates that are *logically* available during circuit synthesis and the gates that are *physically* available for the quantum circuit with the assumption that all physically available gates are available for use during the synthesis process. We also assume that both $C_1$ and $C_2$ are physically, and therefore logically, available as a cycle gate is implemented as a rotation the difference between $C_1$ and $C_2$ being the direction of rotation. A technology that supports one type of cycle can reasonably be expected to support the other.

Given a set of physically available MS gates, we next consider how to implement reversible gates with more than 1 control. A realization for 2 controls as given in [2] is shown in Fig. 1[1]. $\alpha$ can be any of $C_1$, $C_2$, $D$, $E$ or $N$ that are physically available. $h$ is a helper line which is initialized to 0. Given that, a gate with three controls can be implemented as shown in Fig. 2. Two helper lines, $h_1$ and $h_2$ are required and must both be initialized to 0. Gates with more controls can be implemented following a similar strategy.

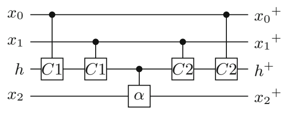


**Fig. 1.** Implementation for $\alpha[x_2, x_1 = 2, x_0 = 2]$ with helper line $h$

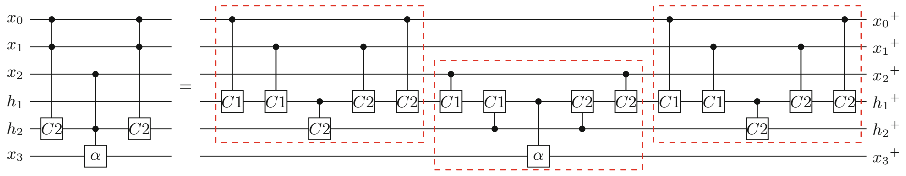


**Fig. 2.** Implementation for $\alpha[x_3, x_2 = 2, x_1 = 2, x_0 = 2]$ with helper lines $h_1$ and $h_2$

The 2 control case requires 5 MS gates whereas the 3 control case requires 15. We have shown the circuits where the control values are 2. These are readily changed to 1, but recall that we assume all controls in a quantum circuit have the same global value. If a control line to a gate $g$ is required to have a different value, a simple solution is to place uncontrolled gates on that line before and after gate $g$. This leads to the following definition of quantum cost.

**Definition 7.** *A gate which applies $\alpha \in \{C_1, C_2, D, E, N\}$ with 0, 1, 2, 3 controls has a base cost of 1, 1, 5, 15, respectively. In general, the base cost for a $k$-control gate, $k > 2$, is $5 + 2\times$the cost of a gate with $k - 1$ controls. If the gate type $\alpha$ is not physically available as a single MS gate a pair of gates is required and the cost increases by 1. The cost increases by 2, for each control that does not have the global control value.*

Adding uncontrolled gates before and after each control not equal to the global control value can obviously be very costly. A more efficient approach will be introduced in Sect. 5.

[1] The gate notation is $type[target, controls]$.

**Table 2.** Transition options

| Transition | Options | Transition | Options |
|---|---|---|---|
| $0 \rightarrow 1$ | $C_1$ $E$ | $0 \rightarrow 2$ | $C_2$ $N$ |
| $1 \rightarrow 0$ | $C_2$ $E$ | $1 \rightarrow 2$ | $D$ |
| $2 \rightarrow 0$ | $C_1$ $N$ | $2 \rightarrow 1$ | $D$ |

## 3 Transformation-Based Synthesis

Transformation-based synthesis was introduced by the current authors and D. Maslov in [4] for Boolean functions and in [3,5] for MVL, specifically 3-valued, functions. The core operation, TRANSFORM$(a, b)$, is the identification of an ordered list of reversible gates to map an $n$ digit pattern $a$ to an $n$ digit pattern $b$ where $a > b$. The gates are chosen so they will not affect any pattern $< b$. We here describe TRANSFORM$(a, b)$ for the 3-valued case. Note that we use $\leftarrow$ to denote assignment, **break** to indicate exiting the current loop and **continue** to indicate going back to the top of the current loop.

```
 1: procedure TRANSFORM($a, b$)
 2:     $list \leftarrow \phi$
 3:     for $i$=$0, 1, ..., n-1$ do
 4:         if $a_i$=$b_i$ then
 5:             continue
 6:         end if
 7:         create a new gate $G$ with target $x_i$
 8:         use $a_i \rightarrow b_i$ to get the type for $G$ using Table 2
 9:            there may be a choice in which case both are recorded
10:         $c \leftarrow a$ and then set $c_i \leftarrow 0$
11:         for $j$=$0, 1, ..., n-1$ do
12:             if setting $c_j$ to 0 results in $c < b$ then
13:                 break
14:             else
15:                 $c_j \leftarrow 0$
16:             end if
17:         end for
18:         the nonzero digits in $c$ identify the controls for $G$
19:         append $G$ to the end of $list$
20:         $a_i \leftarrow b_i$
21:     end for
22:     return $list$
23: end procedure
```

TRANSFORM generates a gate for each $i$ such that $a_i \neq b_i$. The gates generated use only control values 1 and 2. 0 controls are generated by an optimization discussed in Sect. 5. The for $j$ loop in 11–17 reduces the number of controls for the gate while ensuring the gate will not affect any pattern $< b$. TRANSFORM is readily extended to other radices by revising the options in Table 2.

As noted, TRANSFORM can generate a choice of gate depending on the transition required and also of course which gates are available to be used in a particular synthesis. Our approach to resolving a choice is to choose the one that when applied moves the function closest to the identity. Note that $D$ gates are specified as single choices in the table for transitions $1 \rightarrow 2$ and $2 \rightarrow 1$. $C_1$ and $C_2$ cannot be used alone for those two cases as they would always affect an entry earlier in the specification.

The basic bidirectional transformation-based synthesis approach [5] is shown as METHOD1. TRANSFORM is as just described. INVERSE computes the specification of the inverse of a reversible function which by definition must always exist. Applying a gate to a function specification $F$ means to apply the gate to update each of the entries in $F$.

```
 1: procedure METHOD1($F$)
 2:     set gate lists $C_{in} \leftarrow C_{out} \leftarrow \phi$
 3:     $FI \leftarrow$ INVERSE($F$)
 4:     for $i$=$0, 1, ..., r^n - 2$ do
 5:         if $F_i \neq i$ then
 6:             $T_{out} \leftarrow$TRANSFORM($F_i, i$)
 7:             apply the gates in $T_{out}$ to update $F$
 8:             $T_{in} \leftarrow$TRANSFORM($FI_i, i$)
 9:             apply the gates in $T_{in}$ to update $FI$
10:             if $|T_{out}| < |T_{in}|$ or
11:                $|T_{out}|$=$|T_{in}|$ and $\triangle F < \triangle FI$ then
12:                 append the gates in $T_{out}$ to the end of $C_{out}$
13:                 $FI \leftarrow$ INVERSE($F$)
14:             else
15:                 append the gates in $T_{in}$ to the end of $C_{in}$
16:                 $F \leftarrow$ INVERSE($FI$)
17:             end if
18:         end if
19:     end for
20:     reverse the order of the gates in $C_{out}$
21:     replace each gate in $C_{out}$ by its inverse
22:     form the *circuit* by appending $C_{out}$ to the end of $C_{in}$
23:     return *circuit*
24: end procedure
```

The operation of METHOD1 is straightforward. The basic idea is to find a circuit that will map $F$ to the identity. The inverse circuit will of course map the identity to the desired $F$. For each $i$ starting at 0, the method determines the output-side gates that will map the $i^{th}$ output side entry to $i$. Separately, it finds the input-side gates that will map the appropriate input pattern to match the output pattern $i$. The latter is expressed in terms of $FI$, the inverse of $F$, so that lines 6–7 and 8–9 are similar and can be implemented with common code which is more efficient that treating the output and input sides of $F$ separately. Recall, that TRANSFORM is such that the gates chosen will not alter an entry $F_j, j < i$. Also note that entry $i = r^n - 1$ need not be considered since as the last entry aligning all previous entries to the identity means it is also mapped to the identity.

For each $i$, the method chooses to use the output-side or input-side transformation based on the number of gates required and if those factors are the same based on which choice leads to a specification closest to the identity using the $\triangle$ operator. In the event of a tie, the output-side transform is used. A more fulsome description of this algorithm can be found in [5].

METHOD1 uses either all input or all output gates for each iteration as it only considers two possibilities. An extension to this approach, developed in [9] for the 2-valued case, is to for each $i$, consider all $j$, $i \leq j \leq r^n - 1$ and for each apply TRANSFORM$(F_j, i)$ to find output-side gates and TRANSFORM$(FI_j, i)$ to find input-side gates. From the results for each $j$, the one is chosen that requires the fewest gates (input plus output) and if there is a tie the lowest $j$ resulting in an $F$ closest to the identity is used. We call this approach METHOD2. A full description can be found in [9]. The bounded search transformation-based synthesis method introduced in the next section is derived from this approach.

## 4 Bounded Search Transformation-Based Synthesis

METHOD3 is the new bounded search approach introduced in this paper. The basic idea is to search through the options to transform each entry of $F$ in order $0, 1, ..., r^n - 1$ in a recursive manner. A bounded search is performed where the bound is based on the best circuit found to that point in the search. Any search path where the circuit to that point has a higher cost than the best circuit found so far is abandoned.

A key question is how to set the initial bound. One could use a cost of $\infty$ but that can lead to excessive searching. The approach we adopt is to apply METHOD1 to find an initial circuit and to use it as the initial best circuit which is retained in a global called *BestCircuit*. The invocation of METHOD1 to set the initial bound is made prior to calling METHOD3 to initiate the search. In the initial call to METHOD3 a value of 0 should be provided for parameter $k$ and an empty list for parameter *circuit*.

COST is a function that computes the cost of a circuit which can be selected to be either (a) the number of gates in the circuit, or (b) the sum of the quantum costs of the gates in the circuit as specified in Definition 7.

SIMPLIFY is a procedure that applies the post-synthesis simplifications process described in Sect. 5.

Function MAP used in METHOD3 orders the alternatives to be considered in a special way. If $j = k$ it returns $k$. If $j = k + 1$, it returns the value $p$ such that $F_p = k$. Those two cases are considered first since the first one only requires output side gates and the second only requires input side gates. Those cases quite often have the cheapest incremental costs. For $j > k + 1$ in order, function MAP returns the other choices from $k$ to $R^n - 1$ in ascending order.

```
 1: procedure METHOD3($F, k, circuit$)
 2:     if $k{=}r^n - 1$ then
 3:         SIMPLIFY($circuit$)
 4:         if COST($circuit$) <COST($BestCircuit$) then
 5:             $BestCircuit \leftarrow circuit$
 6:         end if
 7:         return
 8:     end if
 9:     for $j{=}k, k+1, ..., r^n - 1$ do
10:         $i \leftarrow$ MAP($j, k$)
11:         copy $F$ into $FC$
12:         $T_{out} \leftarrow$TRANSFORM($F_i, k$)
13:         apply the gates in $T_{out}$ to update $FC$
14:         $T_{in} \leftarrow$TRANSFORM($i, k$)
15:         if $|T_{in}| + |T_{out}| + |circuit| \geq |BestCircuit|$ then
16:             return
17:         end if
18:         $FIC \leftarrow$ INVERSE($FC$)
19:         apply the gates in $T_{in}$ to update $FIC$
20:         $FC \leftarrow$ INVERSE($FIC$)
21:         append gates in $T_{out}$ reversed and inverted
22:                     to the end of $circuit$
23:         prepend gates in $T_{in}$ to the front of $circuit$
24:         METHOD3($FC, k+1, circuit$)
25:         remove gates in $T_{in}$ and $T_{out}$ from $circuit$
26:     end for
27:     return
28: end procedure
```

Lines 2–8 in METHOD3 is the terminal case for the recursive search. The current circuit is compared to the best circuit found so far and replaces it if it is cheaper. Lines 15–17 implement the bound on the search by comparing the number of gates in the circuit being built to the number of gates in the best circuit found so far. This bound is used as it has been found to better bound the search in terms of computational time than using the quantum cost. Line 24 is the recursive call to move to the next entry in $F$ and line 25 removes the gates generated for one alternative before iterating to consider the next.

## 5 Post-synthesis Circuit Simplification

Suggestions for circuit simplification were made in [5] with a hand-worked example. Here we present two procedures for circuit simplification of 3-valued circuits. The first, REDUCE, accepts an ordered list of gates $G = G_0 G_1 ... G_{ngates-1}$ and returns a modified list of gates. The following four definitions are employed.

**Definition 8.** *Two gates are* inverses *of each other if they have the same target, the same control variables and control values and either they are both $D$, $E$ or $N$ gates, or one is a $C_1$ gate and the other is a $C_2$ gate.* □

**Definition 9.** *Two $C_k$ gates are* mergeable *if they have the same target, the same control variables and control values. The merge into a single gate has the given target, control variables and control values and is of type $C_{3-k}$.* □

**Definition 10.** *Two gates $G_i$ and $G_j$ are* control reducible *if they are of the same type, have the same target and controls and matching control values except for one control $x_k$. The gates can be modified by removing $x_k$ from $G_i$ and setting the control value for $x_k$ for $G_j$ to $3 - s$ where $s$ is the sum of the original $x_k$ control values for the two gates. If the gates are $C$ gates, $G_j$ is replaced by its inverse.* □

The commonly used rule [4] for whether two adjacent gates $G_i$ and $G_{i+1}$ can be interchanged is to check that the target for $G_i$ is not a control for $G_{i+1}$ and the target for $G_{i+1}$ is not a control for $G_i$. Here we introduce a more flexible definition which permits more optimization possibilities.

**Definition 11.** *Two adjacent gates $G_i$ and $G_{i+1}$ can be* interchanged *unless:*

1. *The two gates have the same target but the gates are not both of the same type ($C$, $D$, $E$ or $N$);*
2. *If $G_i$ has type $t \in \{C, D, E, N\}$, the target of $G_i$ is a control for $G_{i+1}$ with control value $v$ and $t = C$, or $t = D, E, N$ and $v \neq 0, 2, 1$ respectively; or*
3. *If $G_{i+1}$ has type $t \in \{C, D, E, N\}$, the target of $G_{i+1}$ is a control for $G_i$ with control value $v$ and $t = C$ or $t = D, E, N$ and $v \neq 0, 2, 1$ respectively.* □

(1) in the above definition states two gates cannot be interchanged if they have the same target but potentially conflicting gate operations. (2) and (3) state two gates cannot be interchanged if the target for one gate is a control for the second gate and the target operation could affect the corresponding control value. This allows more gate movement than the simple blocking rule [4].

REDUCE, Fig. 3, implements our gate simplification strategy. It should be noted that the procedure looks for two gates $G_i$ and $G_j$ that could be moved to be adjacent but does not actually move them. Lines 8–9, 18–19 and 30–31 follow the removal or modification of $G_i$ and start the reduction search over to look for simplifications that may have been previously blocked.

Our second simplification procedure, *insert_C*, Fig. 4, accepts $G$, an ordered list of gates and a global control value $cv = 1$ or 2, and inserts uncontrolled $C_1$ and $C_2$ gates so that all control values in the circuit will be $cv$. Effort is made to reduce the number of gates inserted, *i.e.* it does not insert a gate before and after every control that differs from the global value.

Post-synthesis circuit simplification used in this work involves four steps:

1. apply *reduce* to the circuit produced by the chosen synthesis method;
2. if the target is a quantum circuit, apply *insert_C* to add the required uncontrolled $C$ gates to map all gate control values to the desired value;
3. perform any logical gate substitutions for $D$, $E$ or $N$ gates depending on which types of gate substitution have been specified for the current synthesis.
4. if step 2 and/or step 3 has been applied, apply *reduce* a second time to identify any possible reductions arising from steps 2 and 3.

```
procedure REDUCE(G)
    $i \leftarrow 0$
    while $i < ngates - 1$ do
        $j \leftarrow i + 1$
        while $j < ngates$ do
            if $G_i$ and $G_j$ are inverses of each other then
                remove $G_i$ and $G_j$ from $G$
                $i \leftarrow -1$
                break
            end if
            if $G_i$ are $G_j$ are mergeable $C$ gates then
                remove $G_i$ from $G$
                if $G_j$ is a $C_1$ then
                    change $G_j$ to a $C_2$
                else
                    change $G_j$ to a $C_1$
                end if
                $i \leftarrow -1$
                break
            end if
            if $G_i$ and $G_j$ are control reducible gates then
                let $x_k$ be the control difference variable
                set the control value for $x_k$ for $G_j$ to
                     the value unused for control $x_k$
                     by $G_i$ and $G_j$
                remove $x_k$ from the controls for $G_i$
                if the gates are type $C$ then
                    replace $G_j$ by its inverse
                end if
                $i \leftarrow -1$
                break
            end if
            if $G_i$ and $G_j$ cannot be interchanged then
                break
            end if
            $j \leftarrow j + 1$
        end while
        $i \leftarrow i + 1$
    end while
end procedure
```

**Fig. 3.** Gate reduction procedure

Step 3 requires some explanation. If a $D$, $E$ or $N$ gates is logically available, *i.e.* available during synthesis, but not physically available for the final circuit, it must be substituted by other gates. Step 3 is a *logical* substitution where the gate is replaced during the simplification process at which point the substituted gates become candidates for reduction. The alternative is to do a physical gate substitution in the final quantum circuit as suggested in Definition 7.

```
procedure INSERT_C($G, cv$)
    if $cv = 1$ then
        $p \leftarrow 2,\ q \leftarrow 1$
    else
        $p \leftarrow 1,\ q \leftarrow 2$
    end if
    for i=0,1,...,n-1 do
        $a_i \leftarrow cv$
    end for
    $i \leftarrow 0$
    while i<ngates do
        if $g_i$ is a $D$, $E$ or $N$ gate and $a_k \neq cv$ where $x_k$ is
                    the target for $G_i$ then
            if $a_k$=0 then
                insert a $C_p[x_k]$ gate before $G_i$
            else
                insert a $C_q[x_k]$ gate before $G_i$
            end if
            $a_k \leftarrow 1$
            for each control $x_k$ for $G_i$ do
                $c \leftarrow$ control value for $x_k$ in $G_i$
                if $c \neq a_k$ then
                    $v = (a_k + 2 \times c) mod 3$
                    insert a $C_v[x_k]$ gate before $G_i$
                    $j \leftarrow j + 1$
                end if
                set control value for $x_k$ for $G_i$ to $cv$
            end for
            $j \leftarrow j + 1$
        end if
    end while
    for i=0,1,...,n-1 do
        if $a_i \neq cv$ then
            if $a_i$=0 then
                append a $C_p[x_i]$ to the end of $G$
            else
                append a $C_q[x_i]$ to the end of $G$
            end if
        end if
    end for
end procedure
```

**Fig. 4.** Insertion of uncontrolled $C$ gates

## 6 Experimental Results

We have implemented the above techniques in C using the gcc compiler with optimization level -O3. Experiments were run on a computer with an Intel i5 650 CPU @ 3.20 GHz and 3 GB of RAM.

Our first experiment generated reversible circuits for the $9! = 362,880$ 2-variable 3-valued reversible functions. Table 3 shows the results for methods 1, 2 and 3 with and without applying REDUCE. Since the target is reversible circuits, INSERT_C is not applied and gate count is used for circuit cost.

To make the results comparable to [5], $D$ gates are used as individual gates, *i.e.* without substitution, and $E$ gates are not used. In each case, results are shown for synthesizing the function alone and synthesizing the function and its inverse and choosing the better circuit. The table shows the average gate count and total CPU time in seconds for each scenario. The CPU time, here and for the other experiments, includes the time required to verify the circuits. The table shows that METHOD2 provides quite small improvement over METHOD1.

METHOD3 yields substantial improvement but at a high increase in computational cost. For each METHOD3 scenario, the table shows the average number of circuits examined per function which is a good indicator of where the computational cost comes from. For example, for the gate reduction using $f$ and $f^{-1}$ scenario, a total of 21,798,180 circuits were generated in finding solutions for the 362,880 functions an average of 60.07.

As an aside, noted by one of the referees, the METHOD1 search considering $f$ and $f^{-1}$ with no gate reduction took 5.31 CPU sec. whereas the same search in 2004 [3] took several CPU minutes on a then modern desktop computer. An interesting illustration of the tremendously increased computing power that is now available.

**Table 3.** 2-variable 3-valued functions: average gate count

| Method | No gate reduction | | | | | |
|---|---|---|---|---|---|---|
| | $f$ | | | $f$ and $f^{-1}$ | | |
| | Avg. Gates | CPU Sec. | Avg. Circ. per Func. | Avg. Gates | CPU Sec. | Avg. Circ. per Func. |
| 1 | 7.160 | 2.51 | | 6.957 | 5.31 | |
| 2 | 7.078 | 12.67 | | 6.860 | 25.28 | |
| 3 | 6.125 | 86.42 | 21.727 | 6.083 | 171.63 | 43.450 |
| impr. 3 vs 1 | 14.46% | | | 12.56% | | |
| **Method** | **Gate reduction** | | | | | |
| | $f$ | | | $f$ and $f^{-1}$ | | |
| | Avg. Gates | CPU Sec. | Avg. Circ. per Func. | Avg. Gates | CPU Sec. | Avg. Circ. per Func. |
| 1 | 7.077 | 2.67 | | 6.855 | 5.51 | |
| 2 | 6.989 | 12.73 | | 6.753 | 25.65 | |
| 3 | 5.983 | 103.45 | 30.030 | 5.919 | 209.25 | 60.070 |
| impr. 3 vs 1 | 15.46% | | | 13.65% | | |

Our second experiment again considers all 2-variable 3-valued functions. The results are shown in Table 4. Each row of the table represents a particular scenario regarding the use of $D$, $E$ and $N$ gates and choice for the global control value $cv$. Recall that $C_1$ and $C_2$ are assumed to be physically, and therefore logically, available in all the scenarios. In every case, the best circuit found by considering $f$ and $f^{-1}$ is used. The table is ordered by ascending cost for METHOD3 with $cv = 2$ which most often yields the best results.

Column Synth. identifies which of $D$, $E$ and $N$ are available during the synthesis process. As noted above, $D$ must always be available. Column Sub. identifies what gate substitutions are performed. A 1 denotes logical substitution during the synthesis process *i.e.* gate substitution during the circuit simplification step. A 2 denotes physical substitution in the final circuit. Results are shown for the three methods with $cv$, the circuit wide control value, equal 1 and 2. Each trial for METHOD1 required 5–6 CPU sec. For METHOD2 and METHOD3 the CPU usage per trial is around 35–40 s and 3–3.5 min, respectively.

As before, METHOD3 shows very significant improvement over the other methods at a rather high computational cost. It is interesting that, as one would tend to expect, the best performance (shown in bold) for all methods, except for METHOD3 with $cv = 2$, is for the scenario where $D$, $E$ and $N$ are available during synthesis with no logical or physical substitution, *i.e.* all three gate types are available with lowest cost. For the case of METHOD3 with $cv = 2$, the best result is when only $D$ and $N$ are available for synthesis and direct implementation. This is reflective of the fact our methods are heuristic and even the search based method relies on heuristic choices as to which gates are best to use at each step of the synthesis.

In all cases for METHOD1 and METHOD2 a $cv$ value of 2 leads to lower average quantum cost than does a $cv$ value of 1. For METHOD3, there are some exceptions (shown in italics). In those cases the differences are rather small. This effect arises from the fundamental property that the transformation-based synthesis methods process the specification in a fixed order and tend to produce more gate control values of 2 than 1. Consequently, fewer uncontrolled $C$ gates need to be inserted to map all control values in the circuit to 2 than to 1.

The results also strongly suggest that logical substitution for $D$, $E$ and $N$ is more effective than physical substitution. This is because for logical substitution, the substituted gates are considered during the circuit simplification process.

To further illustrate the effectiveness of the circuit reductions performed by procedure REDUCE, consider the best result in Table 4 – the scenario using METHOD3 with $D$ and $N$ gates with no gate substitution, $cv = 2$, and considering $f$ and $f^{-1}$. If one turns off reductions but leaves insertion of uncontrolled $C$ gates on, the average quantum cost rises from 7.837 to 8.771 an increase of 11.9%.

Our third experiment considers a 3-valued full adder which is an irreversible function with three inputs, here identified as $x_0, x_1, x_2$ and two outputs *sum* and *carry*. To make it reversible, requires an additional input and two additional outputs. As noted in [3], experience with a reversible binary full adder is helpful and leads to the following specification which behaves as a full adder when

$x_3 = 0$.

$$x_0^+ = sum[x_0, x_1, x_2)\ x_1^+ = x_1 \oplus x_2\ x_2^+ = x_2\ x_3^+ = carry[x_0, x_1, x_2) \oplus x_3 \qquad (6)$$

Figure 5 shows the reversible adder circuit found using METHOD1 forward synthesis with circuit reduction but no gate substitutions. This circuit has 17

**Table 4.** 2 variable 3-valued functions: average quantum cost

| Synth. | Sub. | | | METHOD1 | | METHOD2 | | METHOD3 | |
|---|---|---|---|---|---|---|---|---|---|
| | *D* | *E* | *N* | CV = 2 | CV = 1 | CV = 2 | CV = 1 | CV = 2 | CV = 1 |
| *D N* | | | | 9.950 | 11.667 | 9.896 | 11.279 | **7.837** | 8.488 |
| *D E N* | | | | **9.701** | **10.884** | **9.623** | **10.553** | 7.963 | **8.048** |
| *D* | | | | 10.193 | 11.995 | 10.143 | 11.617 | 8.057 | 8.684 |
| *D E* | | | | 10.001 | 11.301 | 9.917 | 10.935 | 8.163 | *8.154* |
| *D N* | | | 1 | 10.416 | 12.224 | 10.355 | 11.823 | 8.275 | 8.934 |
| *D E N* | | 1 | | 10.386 | 11.713 | 10.307 | 11.404 | 8.327 | 8.730 |
| *D E N* | | | 1 | 10.244 | 11.409 | 10.157 | 11.063 | 8.477 | *8.345* |
| *D N* | | | 2 | 10.614 | 12.328 | 10.546 | 11.923 | 8.486 | 9.082 |
| *D E N* | | 2 | | 10.543 | 11.772 | 10.464 | 11.459 | 8.502 | 8.800 |
| *D E N* | | | 2 | 10.338 | 11.513 | 10.245 | 11.164 | 8.545 | 8.507 |
| *D E* | | 1 | | 10.653 | 11.988 | 10.591 | 11.674 | 8.566 | 8.786 |
| *D E* | | 2 | | 10.762 | 12.109 | 10.697 | 11.783 | 8.722 | 8.978 |
| *D E N* | 1 | | | 11.489 | 12.215 | 11.345 | 11.876 | 8.799 | *8.670* |
| *D E N* | | 1 | 1 | 10.898 | 12.212 | 10.813 | 11.892 | 8.859 | 9.085 |
| *D E N* | 2 | | | 11.627 | 12.769 | 11.479 | 12.346 | 8.879 | *8.809* |
| *D N* | 1 | | | 12.208 | 13.355 | 12.086 | 12.975 | 8.887 | 9.237 |
| *D E N* | | 2 | 1 | 11.003 | 12.326 | 10.916 | 11.996 | 8.977 | 9.251 |
| *D E N* | | 1 | 2 | 11.075 | 12.289 | 10.982 | 11.968 | 9.045 | 9.185 |
| *D N* | 2 | | | 12.484 | 14.180 | 12.37 | 13.723 | 9.130 | 9.643 |
| *D E N* | | 2 | 2 | 11.181 | 12.403 | 11.086 | 12.072 | 9.175 | 9.357 |
| *D E* | 1 | | | 12.077 | 13.132 | 11.892 | 12.701 | 9.197 | *8.979* |
| *D E N* | 1 | 1 | | 12.211 | 13.072 | 12.071 | 12.755 | 9.285 | 9.475 |
| *D E N* | 2 | 1 | | 12.350 | 13.626 | 12.204 | 13.227 | 9.386 | 9.640 |
| *D E* | 2 | | | 12.316 | 13.575 | 12.13 | 13.083 | 9.406 | *9.189* |
| *D E N* | 1 | | 1 | 12.040 | 12.992 | 11.886 | 12.598 | 9.443 | *9.135* |
| *D E N* | 1 | 2 | | 12.365 | 13.129 | 12.225 | 12.807 | 9.478 | 9.548 |
| *D E N* | 1 | | 2 | 12.133 | 13.093 | 11.973 | 12.697 | 9.518 | *9.305* |
| *D E N* | 2 | | 1 | 12.179 | 13.304 | 12.023 | 12.867 | 9.519 | *9.219* |
| *D E N* | 2 | 2 | | 12.507 | 13.685 | 12.362 | 13.282 | 9.586 | 9.718 |
| *D E N* | 2 | | 2 | 12.274 | 13.408 | 12.112 | 12.968 | 9.597 | *9.393* |

gates. $D$ and $E$ gates were allowed. METHOD2 finds the same circuit. Figure 6 shows the reversible adder circuit found using METHOD3 forward synthesis. This circuit has 10 gates.

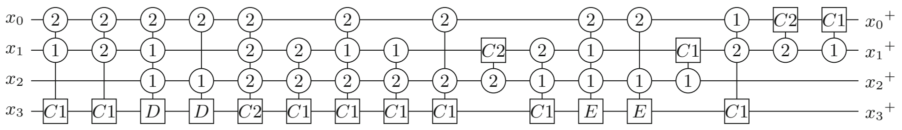


**Fig. 5.** METHOD1 forward synthesis: 17 gate reversible adder circuit

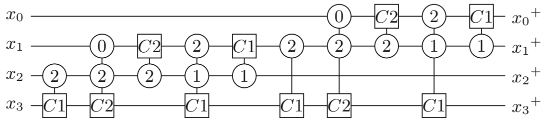


**Fig. 6.** METHOD3 forward synthesis: 10 gate reversible adder circuit

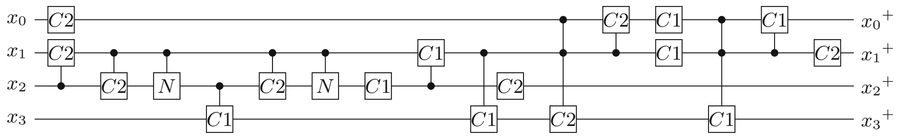


**Fig. 7.** METHOD3 forward synthesis with simplification: 18 gate full adder quantum circuit, cost 26, $cv = 2$

Table 5 shows our full experimental results for the full adder. Uncontrolled $C$ gates were inserted for a global control of 2 and logical substitution of $D$ and $E$ gates was employed. The top half of the table shows the results for the reversible full adder and the bottom half shows the results for the inverse function. Here the new search method (METHOD3) significantly outperforms the basic transformation-based synthesis methods. Note that METHOD3 is quite quick for the full adder but takes significantly longer for the inverse case.

It is interesting to contrast the results in the top half of Table 5 to the discussion in [5] where an initial 16 gate solution was hand optimized to a circuit with 23 gates with a quantum cost of 96. that circuit contains four 3-control $C$ gates which are quite expensive in terms of elementary quantum operations. In particular, METHOD3 forward synthesis followed by circuit simplification as described in Sect. 5 produces the circuit in Fig. 7 which has 18 gates and a quantum cost of 26. There are only two 2-control gates (shown in bold) in this circuit, the rest having 1 or 0 controls.

**Table 5.** Ternary full adder

| Method | No E Gates | | | E Gates | | |
|---|---|---|---|---|---|---|
| | Gates | Cost | CPU | Gates | Cost | CPU |
| Forward synthesis | | | | | | |
| 1 | 30 | 106 | 0.047 | 37 | 157 | 0.047 |
| 2 | 30 | 106 | 0.062 | 37 | 157 | 0.078 |
| 3 | 18 | 26 | 0.438 | 18 | 26 | 0.703 |
| impr. 3 vs. 1 | | 75.5% | | | 83.4% | |
| Reverse synthesis | | | | | | |
| 1 | 44 | 180 | 0.063 | 59 | 289 | 0.078 |
| 2 | 44 | 180 | 0.094 | 59 | 289 | 0.125 |
| 3 | 18 | 34 | 379.8 | 18 | 34 | 593.7 |
| impr. 3 vs. 1 | | 81.1% | | | 88.2% | |

Next we consider a 5-variable function with inputs $x_4, x_3, ..., x_0$ which acts as a controlled counter. The 4-digit 3-valued number represented by $x_3, x_2, x_1, x_0$ is incremented by the value of $x_4$ and that result is taken modulo $3^4$. In specifying $F$ for this problem $x_4$ is treated as the most significant variable and $x_0$ as the least significant. The results are shown in Table 6. Note that the same circuits are found regardless of whether E gates are used. The synthesis scenario and circuit simplification used are as described for the adder.

The results are significantly better for METHOD3 compared to the other methods but once again at a high computation cost. The 11 gate quantum cost 29 circuit is shown in Fig. 8.

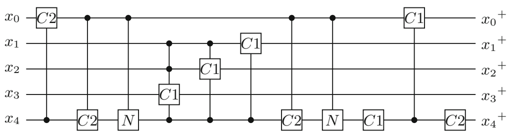


**Fig. 8.** METHOD3 Forward synthesis: 11 gate quantum counter circuit, cost 29, $cv = 2$

The previous two examples, the adder and counter, are arithmetic functions. It is constructive to consider a different type of example. We consider a reversible function with four inputs $x_3, x_2, x_1, x_0$ which rotates the order of $x_2, x_1, x_0$ based on the value of $x_3$. Specifically, the output is $x_3, x_2, x_1, x_0$ if $x_3 = 0$, $x_3, x_1, x_0, x_2$ if $x_3 = 1$, and $x_3, x_0, x_2, x_1$ if $x_3 = 2$.

Applying METHOD1 using $N$, $D$ and $E$ gates with $D$ and $E$ gate logical substitution and the simplification procedure outlined above yields a reversible circuit with 76 gates and a quantum cost of 358. Applying the same approach to the inverse function yields a circuit with 75 gates and quantum cost 358.

**Table 6.** Controlled counter

| Method | No E Gates | | | E Gates | | |
|---|---|---|---|---|---|---|
| | Gates | Cost | CPU | Gates | Cost | CPU |
| Forward synthesis | | | | | | |
| 1 | 15 | 137 | 0.03 | 15 | 137 | 0.05 |
| 2 | 15 | 137 | 0.11 | 15 | 137 | 0.13 |
| 3 | 11 | 29 | 182.17 | 11 | 29 | 290.64 |
| impr. 3 vs. 1 | | 78.8% | | | 78.8% | |
| Reverse synthesis | | | | | | |
| 1 | 17 | 137 | 0.03 | 17 | 137 | 0.03 |
| 2 | 17 | 137 | 0.11 | 17 | 137 | 0.13 |
| 3 | 11 | 29 | 210.96 | 11 | 29 | 272.52 |
| impr. 3 vs. 1 | | 78.8% | | | 78.8% | |

The same results are found if METHOD2 is used. About 0.125 CPU seconds is required for each synthesis.

Applying METHOD3 directly to this function is problematic. Unlike the previous examples, the search takes a truly inordinate amount of time. We have implemented two changes to METHOD3 to make it a bit more reasonable for this problem: (1) A check is inserted between lines 8 and 9 to test if $F_k = k$ and if it does to accept that 0 gate case without exploring all other alternatives i.e. $j = k+1...r^n - 1$. (2) The search is aborted if a preset circuit cost is reached.

Applying the modified METHOD3 with a cost limit of 170 to the inverse of the rotation function a circuit was found with 50 gates and quantum cost 170 – a bit less than half the cost found using METHOD1. The search took 3.8 CPU hours and considered 1,551 circuits. The question is whether this is a good result.

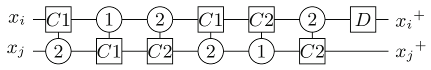
Consider a gate $swap[x_i, x_j]$ that interchanges the values of the two inputs and assume that controls can be applied to such a gate. This is a generalization of the well-known binary Fredkin gate [7]. Given such a gate, the input rotation function as described above can be realized as shown in (7) which can be simplified by applying control reduction to the first and third swaps, which is possible because swaps two and three can be reordered, yielding the circuit in (8).

$$swap[x_0, x_2, x_3 = 1]\, swap[x_1, x_2, x_3{=}1]\, swap[x_0, x_2, x_3 = 2]\, swap[x_0, x_1, x_3{=}2] \quad (7)$$

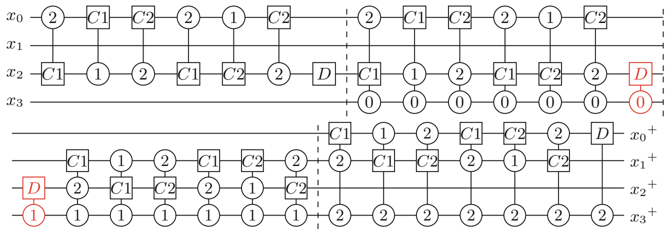
$$swap[x_0, x_2]\, swap[x_0, x_2, x_3 = 0]\, swap[x_1, x_2, x_3 = 1]\, swap[x_0, x_1, x_3 = 2] \quad (8)$$

Using *method*3 with quantum cost as the cost metric and without post-synthesis simplification yields the circuit in Fig. 9(a) for the uncontrolled swap of $x_i$ and $x_j$. Substituting this circuit for each of the swaps in (8) with $x_3$ control added appropriately we find the circuit in Fig. 9(b) where the lines separate the swap gate implementations. Note that the inverse circuit has been used for the third swap so that the two $D$ gates in red are brought together. This is possible since the swap operation is self-inverse.

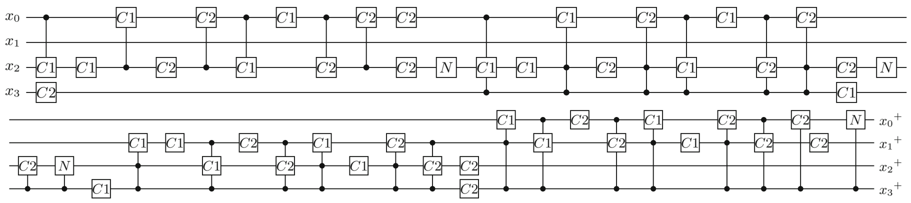
Applying the circuit simplification procedures from Sect. 5 yields the circuit in Fig. 9(c). That circuit has 50 gates and quantum cost 122 which is a reduction of 28.2% from the 170 found by applying METHOD3 directly to the rotation function specification.



(a) METHOD3 no simplification: uncontrolled swap of $x_i$ and $x_j$



(b) reversible rotation circuit by substituting swap circuit in (a) into eqn. 8



(c) 50 gate quantum rotation circuit found by simplification of the circuit in (b), quantum cost 122, $cv = 2$

**Fig. 9.** Rotation circuit derived from 4 swap gate circuit

## 7 Conclusions and Future Work

This paper has presented a novel bounded search transformation-based synthesis method as well as circuit simplification and quantum mapping procedures. The discussion of the rotation function example shows the limitation of the search based approach on its own but also the potential to use it within a broader approach using alternative gates and decomposition techniques. It is an issue for further study why METHOD3 takes so much longer for the rotation function compared to the adder and counter.

Our implementation accepts and handles $F$, the specification of a reversible function, in tabular form. Other researchers [9] have explored alternative more compact representations in the Boolean case. It would be interesting to consider how those approaches might be used in our search based synthesis approach.

The search based approach has been described in terms of $r$-valued functions, but our implementation concentrates on 3-valued functions. The procedure TRANSFORM and the circuit simplification techniques need to be extended if MVL functions with $r > 3$ are to be considered.

We have considered various circuit simplification techniques but not the optimization of the final quantum circuit after substitution of the realizations of gates with more than one control. That is an interesting area for further research.

Lastly, we have developed the new methods for the MVL case. It would be interesting to adapt them to the Boolean case, which is simpler, and see how they compare to other Boolean reversible circuit synthesis approaches.

**Acknowledgement.** The authors gratefully acknowledge the comments and suggestions by the reviewers, particularly the suggestion that we better clarify issues regarding quantum circuits which has led to inclusion of much broader experimental results.

## References

1. Barbieri, C., Moraga, C.: A complexity analysis of the cycles-based synthesis of ternary reversible circuits. In: 13th International Workshop on Boolean Problems (2018)
2. Kole, A., Rani, P.M.N., Datta, K., Sengupta, I., Drechsler, R.: Exact synthesis of ternary reversible functions using ternary toffoli gates. In: Proceedings of the International Symposium on Multiple-Valued Logic, pp. 179–184 (2017)
3. Miller, D.M., Dueck, G.W., Maslov, D.: A synthesis method for MVL reversible logic. In: Proceedings of the International Symposium on Multiple-Valued Logic, pp. 74–80 (2004)
4. Miller, D.M., Maslov, D., Dueck, G.W.: A transformation-based algorithm for reversible logic synthesis. In: Proceedings of the IEEE/ACM Design Automation Conference (DAC), pp. 318–323 (2003)
5. Miller, D.M., Maslov, D., Dueck, G.: Synthesis of quantum multiple-valued circuits. J. Multiple-Valued Logic Soft Comput. **12**(5–6), 431–450 (2006)
6. Muthukrishnan, A., Stroud, C.R.: Multivalued logic gates for quantum computation. Phys. Rev. A **62**, 052309 (2000)
7. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
8. Resch, S., Karpuzcu, U.R.: Quantum computing: An overview across the system stack. arXiv:1905.07240v3 [quant-ph] (2019)
9. Soeken, M., Dueck, G.W., Rahman, M.M., Miller, D.M.: An extension of transformation-based reversible and quantum circuit synthesis. In: Proceedings of the International Symposium on Circuits and Systems, pp. 2290–2293 (2016)
10. Töormä, P.: Realizations of quantum computing using optical manipulations of atoms. Nat. Comput. **1**, 199–209 (2002)