Reversible Logic Synthesis Example using a Transformation Based Algorithm

B. Gergel
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
Email: barry.gergel@uleth.ca

J. Rice
Department of Mathematics and Computer Science
University of Lethbridge
Lethbridge, Alberta, Canada
Email: j.rice@uleth.ca

***Abstract*— Interest in quantum computing, nanotechnology, and low power consumption circuits is fueling the expanding research in reversible logic synthesis. A reversible function is one that generates a unique output vector for every input vector. Traditional circuits are designed with AND and OR gates that allow fan-outs and loop back results which often results in inefficient energy consumption. In contrast, reversible circuits must preserve the number of outputs to the number of inputs and can only be represented as a cascade of reversible gates. Thus traditional synthesis techniques are not adequate for working with reversible functions. Miller *et al.* have created a novel approach for synthesizing the functions into near-optimal reversible networks. The aim of this paper is to provide a thorough example demonstrating their template matching algorithm, and summarize their work in reversible logic synthesis.**

I. Introduction

Research in the synthesis of reversible logic circuits is a growing area of study and is motivated by a number of factors. Digital systems have played a major role in the development of the modern world in the last seventy years. The ability to manipulate discrete values in an ordered fashion is a hallmark of these systems, which are traditionally represented by logical functions containing AND and OR gates. These circuits are one directional and contain logic that allows for fan-outs and loopbacks. These traditional irreversible circuits have been shown to result in the dissipation of energy [1]. With modern circuits shrinking in size and increasing in density, the dispersion of the extra energy has become a problem. In order for a circuit to retain its energy, Bennett [2] showed that the circuit must be constructed with reversible gates. Another area of interest is that of quantum computing. The realization of a quantum computer has reversible circuits as a key component [3]. Also stimulating interest in reversible logic is the area of nanotechnology [4] as power management on such a small scale becomes even more challenging.

As we have noted above traditional synthesis techniques are inadequate for reversible logic applications. Thus the creation of methods to synthesize reversible logic is of great interest. In reversible circuits fan-outs and loopbacks are not allowed. Therefore, reversible functions cannot be represented using gates such as AND and OR, but instead they must be represented by a cascade of reversible gates [5]. There are a large number of libraries of reversible gates, but in this paper we focus on $n * n$ Toffoli gates [6]. There have been a number of approaches for synthesizing reversible functions to circuit representations. These methods have included those based on dynamic programming [7], transformation-based synthesis [8], and heuristics methods [9], [10]. The early solutions struggled with a couple problems. Some suffer in performance due to extensive search requirements, and others are not guaranteed to converge while generating the reversible circuit.

The algorithm examined in this paper uses a two-step method to generate the reversible circuit. First the reversible function is transformed into a circuit. Next, a series of templates are matched against the generated circuit, and any matches allow for the reduction of the circuit size by replacing the matched template with a reduced chain of gates that generates the same as the template. Key features of this algorithm are that it is guaranteed to converge, does not require excessive search time, and generates near optimal results [11].

In the following section, background information covering the definitions required to understand the algorithm are presented. This is followed by presentation in section III and analysis of the algorithm in section IV. A complete example demonstrating the realization of a 3-input, 3-output reversible function is shown in section V. Finally, the conclusion looks at some of the strengths and weaknesses of the algorithm. We also touch on some of the work that has followed which uses this algorithm as a foundation.

II. Background

The following definitions outline material required to understand the synthesis of reversible logic.

Definition 1: A **reversible function**, $f(x_1, x_2, \ldots, x_n)$, maps each input vector to a unique output vector[12]. The function, which must

have the same number of inputs n as outputs n^0, can be shown [13] as a bijective mapping of the set of integers $\{0, 1, \ldots, 2^n - 1\}$ onto itself, or as a standard truth table as shown in Table I.

Definition 2: A **reversible logic gate** has k-inputs and k-outputs, and transforms a input into a unique pattern. As each reversible gate is also invertible, which means each gate has an inverse gate [14]. There are many proposed reversible gate libraries. In this work we restrict our choice to 3-gate subset of the Toffoli family of gates as done by Miller *et al.* [13].

Definition 3: For the set of domain variables $\{x_1, x_2, \ldots, x_n\}$ the **generalized Toffoli gate** [11] has the form $Tof(C;T)$, where $C = \{x_{i_1}, x_{i_2}, \ldots, x_{i_k}\}$, $T = \{x_j\}$, and $C \cap T = \emptyset$. Such a gate maps the Boolean pattern $(x_1^0, x_2^0, \ldots, x_n^0)$ to $(x_1^0, x_2^0, \ldots, x_{j-1}^0, x_j^0 \oplus x_{i_1}^0 x_{i_2}^0 \ldots x_{i_k}^0, x_{j+1}^0, \ldots, x_n^0)$. The set C which controls the change of the j-th bit is called the set of control lines and T is called the target.

For the purposes the example demonstrated in this paper, we only consider a subset of three gates from the Toffoli family as shown in Figure 1.

i) inverts the value t but has no control lines. This is a traditional NOT gate, and we represent it in this paper as $Tof(;t)$.
ii) inverts t only if c_i is also 1, and the gate is written as $Tof(c_i;t)$. This gates is commonly known as a CNOT gate or a Feynman gate [15]
iii) represents a 3-input gate where the target t is transformed if both the controls lines are 1. It is represented as $Tof(c_1, c_2;t)$ which is often called the Toffoli gate [6]

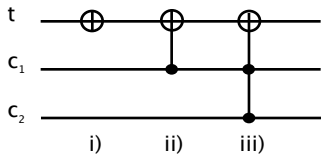

Fig. 1. i) $Tof(;t)$ ii) $Tof(c_i;t)$, and iii) $Tof(c_1, c_2;t)$ Toffoli gates

Definition 4: A **reversible circuit** is a well-formed acyclic combinational logic circuit in which all the gates are reversible and interconnected [16]. Because reversible circuits do not allow fan-outs or loop backs, they may only be represented as a cascade of gates [5] will have n input and n output lines.

Definition 5: The **cost of realization** of a reversible circuit is defined as the number of gates required to bring the reversible function to its identity [17].

Definition 6: The **complexity** $C(f)$ of a reversible function f is the sum of all 2^n input/output Hamming distances added together [13].

III. Algorithm

The goal of an algorithm that synthesizes a reversible circuit is to transform a reversible function into an optimal network of reversible gates. The basic process involves generating the circuit by applying Toffoli gates to bring the reversible function to its identity function. An algorithm generating gates to represent a reversible circuit should converge to ensure that a circuit is created. If a valid reversible function is supplied to the algorithm, it must create a valid representation of the function. Another important consideration is the ability of the algorithm to generate reversible circuits with the smallest gate count possible. Cost savings for reversible circuits can be further enhanced by algorithms that do not generate garbage lines to realize a circuit. Reducing the costs of circuit realization is a primary motivation for all forms of logic synthesis, and reversible logic is no different.

In this paper we chose to evaluate and demonstrate the algorithm presented by Miller *et al.* [13]. Their algorithm is guaranteed to converge with a maximum of $(n-1)2^n + 1$ Toffoli gates to represent a reversible function. Through the application of template matching, they have been able to achieve near optimal results with this algorithm. Another reason for choosing this particular algorithm is that it is the basis for further research [17], [5], [11]. The algorithm uses a truth table to represent a $n * n$ reversible function. An example of the composition of a truth table representation of the reversible function is shown in Table I.

A. Stage 1 – Circuit Realization

The first goal of the algorithm is to create a sequence of Toffoli gates that represents the reversible function. Miller *et al.* have created two algorithms that converge and generate a reversible circuit with an upper bound of $(n-1)2^n + 1$ Toffoli gates. A $3 * 3$ reversible function would require at most 17 gates to reach its identity function as demonstrated in the $3\_17.plafunction$ [11]. The basic algorithm greedily generates the cascade of gates by the application of Toffoli gates to the $f()$ side of the truth table only. The second method synthesizes the circuit by applying Toffoli gates simultaneously to both sides of the truth table. The result of the bidirectional algorithm can realize the function with fewer gates in the circuit [13]. T As the reversible function is transformed during realization, the following notation f^+ to represent the current function specification.

Algorithm 1 Simple Realization Algorithm

```
{First make sure 0 = f(0)}
if f(0) ≠ 0 then
  for all 1-bit in f(0) do
    Apply Tof(; x_i) to transform f(0) = 0
  end for
end if
{Progress through each input/output value until f+ is the identity}
for i = 1 to (2^n − 1) do
  if f+(i) = i then
    continue;
  else
    {Find all the incorrect bits to realize i = f+(i)}
    for j = 0 to (n − 1) do
      if f_j+(i) = 0 and i_j = 1 then
        p_j ⇐ 1
      else
        p_j ⇐ 0
      end if
    end for
    for k = 0 to (n − 1) do
      if f_k+(i) = 1 and i_k = 0 then
        q_k ⇐ 1
      else
        q_k ⇐ 0
      end if
    end for
    {Apply Toffoli Gates to realize i = f+(i)}
    for all p_j = 1 do
      C_j ⇐ {x_i : i_j = 1}
      Apply Tof(C_j; p_j) gate to truth table
      Circuit ⇐ insert Tof(C_j; p_j) at head of list
    end for
    for all q_k = 1 do
      C_k ⇐ {x_i : f+(i_k) = 1}
      Apply Tof(C_k; q_k) gate to truth table
      Circuit ⇐ insert Tof(C_k; q_k) at head of list
    end for
  end if
end for
return Circuit {Cascade of gates representing f()}
```

1) Simple Realization: To synthesize the reversible function, the basic algorithm applies Toffoli gates to the output side of the specification. It starts at 0, and progresses through each value in the specification applying gates to transform each $f(i)$ to i. The choice of gate applied becomes important to ensure convergence by not undoing previously applied transformations. Because each gate that is applied to modify a value must be applied to all the values in the output section, making use of the Toffoli family of gates ensures that the algorithm will converge.

Step one of the algorithm ensures that $0 = f(0)$ by applying a $Tof(; x_j)$ gate to each 1 bit. After the first value is transformed the algorithm progresses from 1 to $(2^n - 1)$ transforming each $i = f(i)$ through the application of Toffoli gates. The gates are inserted to the head of a list to preserve the ordering of the gates. Algorithm 1 shows how the basic algorithm is able to convert a reversible function into a reversible circuit.

2) Bidirectional Realization: The bidirectional algorithm, shown in Algorithm 2, realizes a reversible function by applying Toffoli gates to both the input and output sides of the specification. The effect is that the algorithm applies transformations based on the number of gates required, instead of how close they map to the identity. The net result is that the created circuit is equal to or smaller than the circuit generated by the basic algorithm. In most cases, smaller circuits are realized by the bidirectional algorithm.

At each i in the truth table, the number of transformations required for the input and output side is calculated. If the number of transformations for the output side is less than or equal to the transformations required for input, then the method used in the basic algorithm is applied to the output side of the specification. Otherwise, Toffoli gates are applied to input values. This results in the ordering of the input side no longer being in the proper sequence, but $i = f(i)$ on some line greater than i. Therefore, sorting the input values of the specification restores proper ordering and $i = f(i)$ is located in its proper location. An example of the differences between the effectiveness of the bidirectional algorithm versus the basic algorithm would be the generation of the reversible function $f(7, 0, 1, 2, 3, 4, 5, 6)$ [13]. For this function, the basic algorithm generated a circuit consisting of 7 gates while the bidirectional algorithm generated a circuit of only 3 gates. In our example, our reversible function also generates a smaller circuit using the bidirectional realization algorithm.

B. Stage 2 – Template Matching

The reversible circuits generated during stage 1 of the algorithm are possibly not optimal. The goal of the second stage of the algorithm is to further reduce the cost of the reversible circuit. This is achieved through the matching of a template to part of the circuit. Miller *et al.* describe a library of templates which they have divided into 5 classes [13]. Each template consists of two circuits. The first circuit of the template is a commonly found in circuits generated during the realization stage. The second circuit is a smaller gate sequence that is substituted for the first. An example template is shown in Figure 5. The algorithm checks all the templates against

Algorithm 2 Bidirectional Realization Algorithm

for $i = 0$ to $(2^n - 1)$ **do**
  $x \Leftarrow$ bit changes to bring output side to $f^+(i) = i$
  search for $f^+(i) \rightarrow j$ where $i < j \leq 2^n - 1$
  $y \Leftarrow$ bits changes to bring input side to $f^+(i) = i$
  if $x \leq y$ **then**
    Apply Toffoli gates to transform $f^+(i)$ until $f^+(i) = i$
    RightCircuit $\Leftarrow$ add gate to head of list
  else
    Apply Toffoli gates to transform i until $i = f^+(i)$
    LeftCircuit $\Leftarrow$ add gate to end of list
    sort truth table on first column in ascending order
  end if
end for
Circuit $\Leftarrow$ LeftCircuit
Circuit $\Leftarrow$ append RightCircuit to the end of the list
return Circuit {Cascade of gates representing $f()$}

Algorithm 3 Template Matching Algorithm

for all Template classes **do**
  search Circuit for gates in templates of the current class
  {Match gates to the largest possible template}
  if all gates in a template are found in the Circuit **then**
    if Gates of template are not in sequence **then**
      Apply **moving rule**
    end if
    if gates are in sequence of template **then**
      Replace matched template with reduced sequence
    end if
    if two adjacent gates are equal **then**
      Apply **deletion rule**
    end if
  end if
end for
return Circuit { Cascade of gates representing $f()$}

the realized circuit, and any matches further reduce the cost of the reversible circuit.

When the algorithm is checking the realized circuit for the presence of template it does so by comparing gates of the circuit with the gates contained in template of the current class. The algorithm, shown in Algorithm 3, searches for the presence of gates for each template and choose to replace the largest template discovered in the realized circuit. Once it has identified that the gates are present that match a template, the gates of the realized circuit might need to be moved in order to match the template. Gates can be moved in a circuit based on the *moving rule* which allows the order of two gates, $Tof(C_1, t_1)$ and $Tof(C_2, t_2)$, to be interchanged if, and only if, $C_1 \cap t_2 = \emptyset$ and $C_2 \cap t_1 = \emptyset$ [17]. The realized circuit may also be reduced using the **deletion rule** which allows two adjacent gates that are equal to be removed. Once the template matching has finished the resulting reversible circuit should be near optimal, and Miller *et al.* show these results when comparing all the reversible functions of 3-input/3-output size [13].

IV. Algorithm Analysis

An analysis of the algorithm yields an overall time cost of $O(n \cdot 2^n)$ and a space requirement of $O(2^n)$. The cost of both the simplified realization and bidirectional realization are the most costly part of the algorithm, and this is due to the representation of the reversible function as a truth table. The template matching stage of the algorithm is much less expensive at $O(n^3)$. The benefit of a guaranteed convergence when realizing a reversible function combined with the near optimal result after template matching is complete make the algorithm very effective. Techniques to reduce the costs of the representation as truth table would greatly enhance the ability of the algorithm to process larger reversible functions.

V. Example

One of the main goals of this paper is to demonstrate the presented algorithm to synthesize a 3-input, 3-output reversible function into a valid reversible circuit of Toffoli gates. A small function of size $n = 3$ is used in this example to show how each part of the algorithm works. We demonstrate both circuit realization algorithms as well as applying the template-matching step to the reversible circuit generated by the bidirectional algorithm. The reversible function we chose to demonstrate is $\{5, 2, 0, 4, 7, 3, 1, 6\}$ as shown in Table I. The function was randomly chosen from the set of reversible functions that have not been showcased in [5], [11], [13], [17].

cba	$c^0b^0a^0$	
0 0 0	1 0 1	(5)
0 0 1	0 1 0	(2)
0 1 0	0 0 0	(0)
0 1 1	1 0 0	(4)
1 0 0	1 1 1	(7)
1 0 1	0 1 1	(3)
1 1 0	0 0 1	(1)
1 1 1	1 1 0	(6)

TABLE I
Truth Table for reversible function $\{5, 2, 0, 4, 7, 3, 1, 6\}$

A. Simple Realization

The algorithm for the simple realization of the reversible circuit is the first example demonstrated. The circuit that is generated by the algorithm is shown in Figure 2, and a table representing the application of the Toffoli gates is shown in Table II. The bits modified by the application of a Toffoli gate are highlighted in each column that represents the current state of the output side. The following steps review which gates are added.

Step 1: The first step in the algorithm is to make sure that the mapping $f(0) \rightarrow 0$. As our function is $f(5) \rightarrow 0$, we need to apply a $Tof(;a)$ gate and a $Tof(;c)$ gate. This results in the a and c bits being inverted for the whole truth table.

Step 2: The next step involves transforming the next row of the specification, $f^+(7) \rightarrow 1$. This is going to require the application of two Toffoli gates to reduce the output side to equal the input side, but we must not change the values of $f^+(0) \rightarrow 0$. In order to do this, we make use of the value of a to serve as a control line for the Toffoli gates. We achieve the desired result using a $Tof(a;b)$ and a $Tof(a;c)$ gate.

Step 3: The a bit in $f^+(3) \rightarrow 2$ specification is 1, which is incorrect. Again, to not undo our earlier modifications we make use of a control line, in this case b, and apply a $Tof(b;a)$ gate to transform the specification as required. This is the first level in the specification where we only need to apply a single gate to gain the transformation required.

Step 4: The transformation of $f^+(6) \rightarrow 3$ requires the application of a $Tof(c,b;a)$ gate and a $Tof(a,b;c)$ to bring the mapping to $f^+(3) \rightarrow 3$.

Step 5: The mapping $f^+(4) \rightarrow 4$ requires a $Tof(a,c;b)$ gate and a $Tof(c;a)$ gate to correct the mapping of $f^+(7) \rightarrow 4$.

Step 6: Next the mapping of $f^+(7) \rightarrow 5$ must be transformed to $f^+(5) \rightarrow 5$, and this is achieved with the application of a $Tof(c,a;b)$ gate.

Step 7: In this final step, the mapping $f^+(7) \rightarrow 6$ requires a $Tof(c,b;a)$ gate to correct the mapping. This also transforms $f^+(6) \rightarrow 7$ to the correct mapping. This final gate transforms our reversible function into the identity function.

With the application of the last gate, our circuit has been generated. The circuit shown in Figure 2 should be read from right to left. This is because while transforming the function to its identity, we applied the gates to the output side of the function to bring it to the input side. The simple realization algorithm converged successfully and generated a circuit containing 11 Toffoli gates.

B. Bidirctional Realization

The realization of a reversible circuit using the bidirectional realization algorithm should hopefully generate a circuit containing fewer gates than the previous algorithm. In the application of this algorithm both the input and output sides of the function specification are modified. This makes creating a table as used in the previous example harder to follow. Thus, for the first gate that is applied to the input side we use a smaller table as shown in Table III to illustrate modification to the initial inputs followed by sorted version of the specification. After this all the gate transformations are shown in Table IV where the direction of the gate is shown. The Toffoli gates that are applied against the input side are presented in the sorted form. Because the transformations affect both input and output but we are only displaying the output side in Table IV, the bits which are modified by each gate are not highlighted.

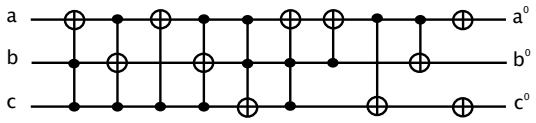

Fig. 2. Circuit generated by the **Simple Algorithm**

Step 1: At each iteration of the bidirectional algorithm, the number of bits k of the input to map $j \rightarrow i$ which results in the correct mapping of $f_+(i) \rightarrow i$ is compared against the number of bits l required on the output side to bring $f_+(i) \rightarrow i$. If $l \leq k$ then we apply gates to the output side. Otherwise, apply the Toffoli gates required to the input side and then sort the specification to reorder the input side values. In this step $k = 1$ because of $f(2) \rightarrow 0$, and $l = 2$ because of $f(5) \rightarrow 0$. Because $k < l$, we apply a $Tof(;b)$ gate to the input side and then sort the table. This results in the required mapping of $f^+(0) \rightarrow 0$.

cba	$c^1b^1a^1$	cba	$c^1b^1a^1$
0 **1** 0	1 0 1	0 0 0	0 0 0
0 **1** 1	0 1 0	0 0 1	1 0 0
0 **0** 0	0 0 0	0 1 0	1 0 1
0 **0** 1	1 0 0	0 1 1	0 1 0
1 **1** 0	1 1 1	1 0 0	0 0 1
1 **1** 1	0 1 1	1 0 1	1 1 0
1 **0** 0	0 0 1	1 1 0	1 1 1
1 **0** 1	1 1 0	1 1 1	0 1 1

TABLE III
BIDIRECTIONAL ALGORITHM – FIRST INPUT SIDE GATE ←TOF(;B)

Step 2: In this step $f^+(1) \rightarrow 4$ making $k = 2$ while $f^+(4) \rightarrow 1$ makes $l = 2$. Because $k = l$, we apply a $Tof(c;a)$ gate and a $Tof(a;c)$ gate against the output side instead of modifying the input side. This saves the cost of having to complete two sorts and results in the

cba	$c^1b^1a^1$	$c^2b^2a^2$	$c^3b^3a^3$	$c^4b^4a^4$	$c^5b^5a^5$	$c^6b^6a^6$	$c^7b^7a^7$	$c^8b^8a^8$	$c^9b^9a^9$	$c^{10}b^{10}a^{10}$	$c^{11}b^{11}a^{11}$
000	**0** 0 1	0 0 **0**	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
001	**1** 1 0	1 1 **1**	1 **0** 1	**0** 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1
010	**1** 0 0	1 0 **1**	1 **1** 1	**0** 1 1	0 1 **0**	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0
011	**0** 0 0	0 0 **1**	0 **1** 1	**1** 1 1	1 1 **0**	1 1 **1**	**0** 1 1	0 1 1	0 1 1	0 1 1	0 1 1
100	**0** 1 1	0 1 **0**	0 1 0	0 1 0	0 1 0	0 1 **1**	**1** 1 1	1 **0** 1	1 0 **0**	1 0 0	1 0 0
101	**1** 1 1	1 1 **0**	1 1 0	1 1 0	1 1 **1**	1 1 **0**	1 1 0	1 1 0	1 1 **1**	1 **0** 1	1 0 1
110	**1** 0 1	1 0 **0**	1 0 0	1 0 0	1 0 0	1 0 0	1 0 0	1 0 0	1 0 **1**	1 **1** 1	1 1 **0**
111	**0** 1 0	0 1 **1**	0 **0** 1	**1** 0 1	1 0 1	1 0 1	1 0 1	1 **1** 1	1 1 **0**	1 1 0	1 1 **1**
	$T(;c)$	$T(;a)$	$T(a;b)$	$T(a;c)$	$T(b;a)$	$T(c,b;a)$	$T(a,b;c)$	$T(a,c;b)$	$T(c;a)$	$T(c,a;b)$	$T(c,b;a)$

TABLE II

RESULTS OF APPLICATION OF SIMPLE REALIZATION ALGORITHM

required $f^+(1) \rightarrow 1$ mapping.

Step 3: The mapping of $f^+(2) \rightarrow 3$ makes $k = 1$, while $l = 2$ due to the mapping of $f^+(4) \rightarrow 2$. A $Tof(b;a)$ gate is applied to the input and the specification is sorted. The required mapping of $f^+(2) \rightarrow 2$ is satisfied.

Step 4: The next required mapping $f^+(3) \rightarrow 3$ has $k = 2$ with $f^+(3) \rightarrow 5$, and $l = 3$ with $f^+(4) \rightarrow 3$. Therefore, by applying a $Tof(c,a;b)$ gate and a $Tof(a,b;c)$ gate to the input side and sorting, the proper mapping is realized.

Step 5: With $l = 1$ from $f^+(5) \rightarrow 4$ and $k = 2$ from $f^+(4) \rightarrow 7$, a $Tof(c;a)$ gate is applied to the output side. This generates the $f^+(4) \rightarrow 4$ mapping.

Step 6: The $f^+(5) \rightarrow 7$ mapping sets $k = 1$ and $f^+(7) \rightarrow 5$ sets $l = 1$. Because $k = l$, a $Tof(c,a;b)$ gate is applied to the output side generating the required mapping of $f^+(5) \rightarrow 5$.

Step 7: Because both $f^+(6) = 6$ and $f^+(7) = 7$, the algorithm does generate any further gates as the function is fully realized.

The synthesis of our reversible function $\{5, 2, 0, 4, 7, 3, 1, 6\}$ is realized with 8 gates using the bidirectional algorithm which is better than the 11 gates generated by the simple realization algorithm. A diagram of this realization is shown in Figure 3. In reading this diagram it is important to realize that gates applied to the input side go from left to right, while the gates applied to the output side go from right to left.

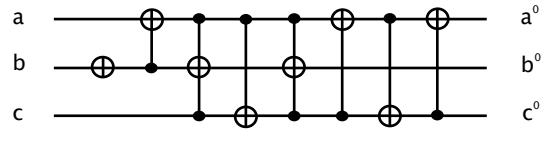

Fig. 3. Circuit generated by the **Bidirectional Algorithm**

C. Template Matching

Finally we get to the point of testing our generated reversible circuit against the template library to identify any further reductions that might be made. We apply the algorithm for template matching to our circuit using the template library [13], and this yields two templates that match. *Template 1:* The first template match is template 1.2 which swaps two values and is shown in Figure 4. The application of this template removes 3 gates from the right side of the circuit by swapping the lines for a and c.

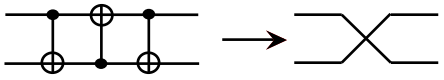

Fig. 4. Template 1.2

Template 2: The second template that is discovered in our circuit is the inverse of template 5.1 (shown in Figure 5). It is important to note that this template does not reduce the gate count but reduces the number of control lines from 2 to 1 for two of the gates. The physical cost of implementing is less for gates with fewer control lines, and this make this template reduction desirable for the synthesis of a reversible circuit.

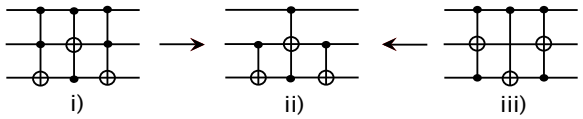

Fig. 5. Template 5.1: i) input template, ii) replacement gates, and iii) input inverse template

The final representation of our circuit is shown in Figure 6. It is here that we see the circuit has been reduced to only 5 gates along with the a and c crossing over.

cba	$c^2b^2a^2$	$c^3b^3a^3$	$c^4b^4a^4$	$c^5b^5a^5$	$c^6b^6a^6$	$c^7b^7a^7$	$c^8b^8a^8$
0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0 0
0 0 1	1 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1	0 0 1
0 1 0	1 0 0	1 0 0	0 1 0	0 1 0	0 1 0	0 1 0	0 1 0
0 1 1	0 1 0	0 1 0	1 0 0	1 0 0	0 1 1	0 1 1	0 1 1
1 0 0	0 0 1	1 0 1	1 0 1	1 0 1	1 0 1	1 0 0	1 0 0
1 0 1	1 1 1	0 1 1	0 1 1	1 1 0	1 1 0	1 1 1	1 0 1
1 1 0	1 1 0	1 1 0	1 1 1	1 0 1	1 1 1	1 1 0	1 1 0
1 1 1	0 1 1	1 1 1	1 1 0	0 1 1	1 0 0	1 0 1	1 1 1
	$\rightarrow Tof(c;a)$	$\rightarrow Tof(a;c)$	$\leftarrow Tof(b;a)$	$\leftarrow Tof(c,a;b)$	$\leftarrow Tof(a,b;c)$	$\rightarrow Tof(c;a)$	$\rightarrow Tof(c,a;b)$

TABLE IV

BIDIRECTIONAL ALGORITHM AFTER INITIAL GATE APPLICATION

Impressively, by the use of the bidirectional realization algorithm and template matching, we have been able to reduce our circuit representation of $\{5, 2, 0, 4, 7, 3, 1, 6\}$ from 11 gates down to 5 gates with a swap of the a and c lines.

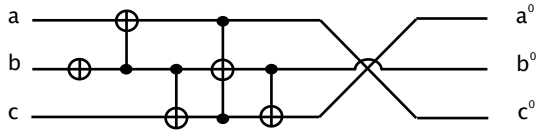

Fig. 6. Circuit after template matching reductions

VI. CONCLUSION

We have reviewed and demonstrated the template matching algorithm of Miller *et al.* for the synthesis of a reversible function into a near optimal reversible circuit [13]. The algorithm as shown only uses a limited set of Toffoli gates to represent the circuit. In later work, using gates such as the Fredkin [18], it has been shown that further reduction of gate counts can be achieved [5]. The largest hinderance of the presented algorithm is the exponential costs in time and size once functions with larger input/output are considered. Techniques such as the application of Reed-Muller decomposition [12] hold potential for providing improvements that will allow this technique to synthesize a larger reversible functions.

REFERENCES

[1] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM J. Research and Development*, vol. 5, pp. 183–191, 1961.

[2] C. Bennett, "Logical reversibility of computation," *IBM J. Research and Development*, vol. 17, pp. 525–532, 1973.

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

[4] R. C. Merkle, "Two types of mechanical reversible logic," *Nanotechnology*, vol. 4, pp. 114–131, Apr. 1993.

[5] D. Maslov, G. W. Dueck, and D. M. Miller, "Fredkin/toffoli templates for reversible logic synthesis," in *International Conference on Computer-Aided Design (ICCAD'03)*. San Jose, California, USA: IEEE Computer Society / ACM, Nov. 2003.

[6] T. Toffoli, "Reversible computing," MIT Lab for Comp. Sci, Tech. Rep., 1980.

[7] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Reversible logic circuit synthesis," in *International Conference on Computer Aided Design (ICCAD)*, San Jose, California, USA, Nov. 2002, pp. 125–132.

[8] K. Iwama, K. Kambayashi, and S. Yamashita, "Transformation rules for designing cnot-based quantum circuits," in *39th Design Automation Conference (DAC)*. IEEE, June 2002, pp. 419–424.

[9] D. Maslov and G. W. Dueck, "Reversible function synthesis with minimum garbage outputs," in *International Symposium on Representations and Methodology of Future Computing Technologies (RM2003)*, Trier, Germany, Mar. 2003.

[10] D. M. Miller and G. W. Dueck, "Spectral techniques for reversible logic synthesis," in *International Symposium on Representations and Methodology of Future Computing Technologies (RM2003)*, Trier, Germany, Mar. 2003.

[11] D. Maslov, G. W. Dueck, and D. M. Miller, "Transformation-based synthesis of networks of toffoli/fredkin gates," in *CCECE 2003 – CCGEI 2003 Proceedings*, vol. 1. Montreal, Canada: IEEE Canada, May 2003, pp. 211–215.

[12] A. Agrawal and N. K. Jha, "Synthesis of reversible logic," in *Design, Automation and Test in Europe (DATE)*, Feb. 2004, pp. 21 384–21 385.

[13] D. Maslov, G. W. Dueck, and D. M. Miller, "A transformation based algorithm for reversible logic synthesis," in *Design Automation Conference (DAC)*, June 2003, pp. 318–323.

[14] J. W. Bruce, M. A. Thorton, L. Shivakumaraiah, P. S. Kokate, and X. Li, "Efficient adder circuits based on conservative reversible logic gate," in *Computer Society Annual Symposium on VLSI*, IEEE, Ed., IEEE. Pittsburgh, Pennsylvania: IEEE, Apr 2002, pp. 83–87.

[15] R. Feynman, "Quantum mechanical computers," *Optic News*, pp. 11–20, 1985.

[16] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes, "Synthesis of reversible logic circuits," *IEEE Transactions on CAD*, vol. 22, no. 6, pp. 710–722, June 2003.

[17] D. Maslov, G. W. Dueck, and D. M. Miller, "Simplification of toffoli networks via templates," in *Integrated Circuits and Systems Design (SBCCI)*, Sao Paulo, Brazil, Sept. 2003, pp. 53–58.

[18] E. Fredkin and T. Toffoli, "Conservative logic," *International Journal of Theoretical Physics*, vol. 21, pp. 219–253, 1982.