Optimal Synthesis of Linear Reversible Circuits

Ketan N. Patel, Igor L. Markov and John P. Hayes
University of Michigan, Ann Arbor 48109-2122
{knpatel,imarkov,jhayes}@eecs.umich.edu

Abstract

In this paper we consider circuit synthesis for n-wire linear reversible circuits using the C-NOT gate library. These circuits are an important class of reversible circuits with applications to quantum computation. Previous algorithms, based on Gaussian elimination and LU-decomposition, yield circuits with $O\left(n^2\right)$ gates in the worst-case. However, an information theoretic bound suggests that it may be possible to reduce this to as few as $O\left(n^2/\log n\right)$ gates.

We give an algorithm that is optimal up to a multiplicative constant, and $\Theta(\log n)$ times faster than previous methods. While our results are primarily asymptotic, simulation results show that even for relatively small n our algorithm is faster and yields smaller circuits than the standard method. The proposed algorithm has direct applications to the synthesis of stabilizer circuits, an important class of quantum circuits. Generically our algorithm can be interpreted as a matrix decomposition algorithm, yielding an asymptotically efficient decomposition of a binary matrix into a product of elementary matrices.

1 Introduction

A reversible circuit is one that implements a bijective function, or loosely, a circuit where the inputs can be recovered from the outputs and all output values are achievable. A major motivation for studying reversible circuits is the emerging field of quantum computation [10]. A quantum circuit implements a unitary function, and is therefore reversible. Circuit synthesis for reversible computations is an active area of research [3, 7, 11, 13, 9]. The goal in circuit synthesis is, given a gate library, to synthesize a small circuit performing a desired computation. In the quantum context, the individual gates correspond to physical operations on quantum states called qubits, and therefore reducing the number of gates in the synthesized circuit generally leads to a more efficient implementation.

Linear reversible classical circuits form an important subclass of quantum circuits, which can be generated by a single type of gate called a C-NOT gate (see Figure 1c). This gate is an important primitive for quantum computation because it forms a universal gate set when augmented with single qubit rotations [8]. Moreover, current quantum circuit synthesis algorithms can generate circuits with blocks of C-NOT gates, and therefore, synthesis methods that reduce the size of these sub-blocks would in turn reduce the size of the overall quantum circuit as well.

Very recently Aaronson and Gottesman [1] showed that the important class of quantum circuits known as stabilizer circuits can be synthesized by a short sequence of blocks of C-NOT gates, phase gates, and Hadamard gates. The Hadamard and phase gate blocks can be synthesized with a small number of gates: $O(n)$ gates for an n-wire stabilizer circuit. The gate count is consequently dominated by the C-NOT blocks. Therefore, synthesizing these linear reversible blocks is critical. Stabilizer circuits are used for a number of important quantum applications, including quantum teleportation [4], super-dense coding [5], and quantum error-correction [10, Chap. 10].

In this paper we consider the problem of synthesizing an arbitrary linear reversible circuit on n wires using as few C-NOT gates as possible. This problem can be mapped to the problem of row reducing an $n \times n$ binary matrix. Until now the best synthesis methods have been based on standard row reduction methods such as Gaussian elimination and LU-decomposition, which yield circuits with $O(n^2)$ gates [6]. However, the best lower bound leaves open the possibility that synthesis with as few as $O\left(n^2/\log n\right)$ gates in the worst case may exist [13].

We present a new synthesis algorithm that meets the lower bound, and is therefore asymptotically optimal up to a multiplicative constant. Furthermore, our algorithm is also asymptotically faster than previous methods. Empirical results show that the proposed algorithm outperforms previous methods even for relatively small n. Generically our algorithm can be interpreted as a matrix decomposition algorithm, that yields an asymptotically efficient elementary matrix decomposition of a binary matrix. Generalizations to matrices over larger finite fields are straightforward.

2 Background

We can represent the action of an n-input m-output logic gate as a function mapping the values of the inputs to those of the outputs: $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, where f maps each element of $\mathbb{F}_2^n$ to an element in $\mathbb{F}_2^m$. Here $\mathbb{F}_2$ is the two-element field, and $\mathbb{F}_2^n$ is the set of all n-dimensional vectors over this field. A gate is *reversible* if this function is bijective, that is, f is one-to-one and onto. Intuitively, this means that the inputs can be uniquely determined from the outputs and all output values are achievable. For example, the AND gate (Figure 1a) is not reversible since it maps three input values to the same output value. The NOT gate (Figure 1b), on the other hand, is reversible since both possible input values yield unique output values, and both possible output values are achievable. The *controlled-NOT* or C-NOT

a b	a'
0 0	0
0 1	0
1 0	0
1 1	1

a	a'
0	1
1	0

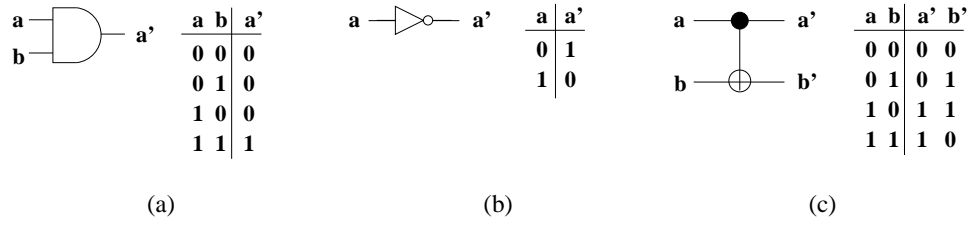| a b	a' b'
0 0	0 0
0 1	0 1
1 0	1 1
1 1	1 0

(a) (b) (c)

Figure 1: Examples of reversible and irreversible logic gates with truth tables a) AND gate b) NOT gate   c) C-NOT gate. Both the NOT and C-NOT gates are reversible while the AND gate is not.

gate, shown in Figure 1c, is another important reversible gate. This gate passes the first input, called the *control*, through unchanged and inverts the second, called the *target*, if the control is a one. As its truth table shows, this gate is reversible since it maps each input vector to a unique output vector and all output vectors are achievable.

A *reversible circuit* is an acyclic combinational logic circuit where all gates are reversible and are interconnected without fanout [13]. An example of a reversible circuit consisting of C-NOT gates is shown in Figure 2. Note that, as is the case for reversible gates, the function computed by a reversible circuit is bijective.

We say a circuit or gate, computing the function f, is *linear* if $f(x_1 \oplus x_2) = f(x_1) \oplus f(x_2)$ for all $x_1, x_2 \in \mathbb{F}_2^n$, where $\oplus$ is the bitwise XOR operation. The C-NOT gate is an example of a linear gate:

$$f([0\ 0]) \oplus f(x) = f(x) \qquad f([0\ 1]) \oplus f([1\ 0]) = f([1\ 1])$$
$$f(x) \oplus f(x) = f([0\ 0]) \qquad f([0\ 1]) \oplus f([1\ 1]) = f([1\ 0])$$
$$f([1\ 0]) \oplus f([1\ 1]) = f([0\ 1])$$

The action of any linear reversible circuit on n wires can be represented by a linear transformation over $\mathbb{F}_2$. Specifically, we can represent the action of the circuit as multiplication by a non-singular $n \times n$ matrix A with elements in $\mathbb{F}_2$:

$$Ax = y,$$

where x and y are n-dimensional vectors representing the values of the input and output bits respectively. Specifically, x is a column vector whose ith entry contains the value of the ith bit of the input. Similarly, y is a column vector containing the values of the output bits. The matrix representing a linear reversible circuit can be derived directly from its truth table: its columns are simply the outputs for each set of inputs containing a single non-zero bit. For example, the first column of the matrix is composed of the output values for the inputs $(1\ 0\ 0 \cdots 0)$. Note that these matrices are more compact than the matrices used to represent arbitrary gates in quantum computing.

Using this matrix representation, the action of a C-NOT gate corresponds to multiplication by an *elementary matrix*, which is the identity matrix with one off-diagonal entry set to one. The matrix for a C-NOT gate with control i and target j would be the identity matrix with the entry in the ith column and jth row set to one. Consider the gate G_1 in Figure 2 which has the first wire as its control and the second wire as its target. Its matrix has a one in the entry in the first column and second row.

Multiplication by an elementary matrix performs a *row operation*, the addition of one row of a matrix or vector to another. Applying a series of C-NOT gates corresponds to performing a series of these row operations on the input vector or equivalently to multiplying it by a series of elementary matrices. For example, the linear transform computed by the circuit in Figure 2 is given by

$$A = \overset{G_6}{\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&1&1 \end{bmatrix}} \cdot \overset{G_5}{\begin{bmatrix} 1&1&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}} \cdot \overset{G_4}{\begin{bmatrix} 1&0&0&0 \\ 0&1&1&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}}$$

$$\cdot \overset{G_3}{\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&1&1&0 \\ 0&0&0&1 \end{bmatrix}} \cdot \overset{G_2}{\begin{bmatrix} 1&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&1&1 \end{bmatrix}} \cdot \overset{G_1}{\begin{bmatrix} 1&0&0&0 \\ 1&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}} = \begin{bmatrix} 1&0&1&0 \\ 0&0&1&0 \\ 1&1&1&0 \\ 1&1&0&1 \end{bmatrix}$$

Note that the matrix operations appear in the reverse order of the gates in Figure 2, since the gates are applied left to right while the matrix operations are applied right to left. In the matrix expression above, the matrices would be applied to the input vector in the order G_1, G_2, $\cdots$, G_5 though they appear in the reverse order.

To illustrate the mapping between the circuit and its matrix representation, consider the input $[1000]^t$ to the circuit in Figure 2. After the application of gate G_1 the wires have the values $[1100]^t$, corresponding to multiplying the input vector by matrix G_1. The application of G_2 does not change the values of the wires. The application of gate G_3 changes the wire values to $[1110]^t$. Again, this corresponds to multiplying the previous vector by G_3. Applying the remaining gates, or equivalently, multiplying by the corresponding elementary matrices gives the final output value: $[1011]^t$.

We can use the matrix notation to count the number of different n-input linear reversible transformations. In order for the transformation to be reversible, its matrix must be non-singular, in other words, all nontrivial sum of the rows should be non-zero. There are $2^n - 1$ possible choices for the first row, all vectors except for the all zeros vector. There are $2^n - 2$ possible choices for the second row, since it cannot be the equal to the first row or the all zeros vector. In general, there are $2^n - 2^{i-1}$ possible choices for the ith row, since it cannot be any of the 2^{i-1} linear combinations of the previous $i - 1$ rows (otherwise
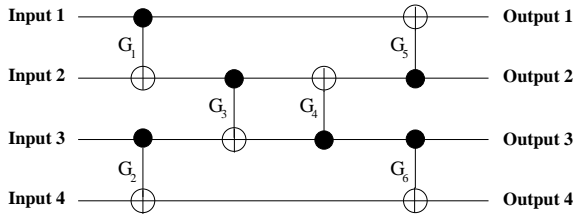
Figure 2: Reversible circuit example.

the matrix would be singular). Therefore there are

$$\prod_{i=0}^{n-1} \left(2^n - 2^i\right)$$

unique n-input linear reversible transformations.

Since any non-singular matrix A can be reduced to the identity matrix using row operations, we can write A as a product of elementary matrices. Therefore, any linear reversible function can be be synthesized from C-NOT gates. Moreover, the problem of C-NOT circuit synthesis is equivalent to the problem of row reduction of a matrix A representing the linear reversible function: any synthesis of the circuit can be written as a product of elementary matrices equal to A and any such product yields a synthesis. The size of the synthesized circuit is given by the number of elementary matrices in the product. Standard Gaussian elimination and LU-decomposition based methods requires $O(n^2)$ gates in the worst-case [6]. However, the best lower bound is only $\Omega\left(n^2/\log n\right)$ gates [13].

Lemma 1 (Lower Bound) *There are n-bit linear reversible transformation that cannot be synthesized using fewer than $\Omega(n^2/\log n)$ C-NOT gates.*

Proof Let d be the maximum number of C-NOT gates needed to synthesize any linear reversible function on n wires. The number of different C-NOT gates which can act on n wires is $n(n-1)$. Therefore the number of unique C-NOT circuit with no more than d gates must be no more than $\left(n^2 - n + 1\right)^d$, where we have included a do-nothing NOP gate in addition to the $n^2 - n$ C-NOT gates to account for circuits with fewer than d gates. Since the number of circuits with no more than d C-NOT gates must be greater than the number of unique linear reversible function on n wires, we have the inequality

$$\left(n^2 - n + 1\right)^d \geq \prod_{i=0}^{n-1}\left(2^n - 2^i\right) \geq 2^{n(n-1)}. \tag{1}$$

Taking the log of both the left and right sides of the equations gives

$$d \geq \frac{n(n-1)\log 2}{\log(n^2 - n + 1)} = \frac{n^2 - n}{\log_2\left(n^2 - n + 1\right)} = \Omega\left(\frac{n^2}{\log n}\right). \tag{2}$$

$\square$

This lemma suggests a synthesis method yielding smaller circuits than standard Gaussian elimination may be possible. The multiplicative constant in this lower bound is $1/2$ (assuming logs are taken base 2).

3 Optimal Synthesis

In this section we present our synthesis algorithm, which achieves the lower bound given in the previous section. In Gaussian elimination, row operations are used to place ones on the diagonal of the matrix and to eliminate any remaining ones. One row operation is required for each entry in the matrix that is targeted. Since there are n^2 matrix entries, $O(n^2)$ row operations are required in the worst case. If instead we group entries together and use single row operations to change these groups, we can reduce the number of row operation required, and therefore the number of gates needed to synthesize the circuit.

The basic idea is as follows. We first partition the columns of the $n \times n$ matrix into sections of no more than m columns each. We call the entries in a particular row and section a *subrow*. For each section we use row operations to eliminate subrow patterns that repeat in that section. This leaves relatively few ($< 2^m$) non-zero sub-rows in the section. These remaining entries are handled using Gaussian elimination. If m is small enough ($< \log_2 n$), most of the row operations result from the first step, which requires a factor of m fewer row operations than full Gaussian elimination. As with the Gaussian elimination based method, our algorithm is applied in two steps; first the matrix is reduced to an upper triangular matrix, the resulting matrix is transposed, and then the process is repeated to reduce it to the identity. Detailed pseudo-code for the proposed algorithm is shown in Algorithm 1.

The following example illustrates our algorithm for a 6-wire linear reversible circuit.

1) Choose $m = 2$ and partition matrix.

2) (Step A - section 1) Eliminate duplicate sub-rows.

3) (Step B - section 1, column 1) One already on diagonal.
4) (Step C - section 1, column 1) Remove remaining ones in column below diagonal.

5→4 2→1 4→2 6→3 5→3 5→4 6→5 | 3→4 4→3 4→6 3→5 2→3 1→2 1→5 1→4

Input 1 — Output 1
Input 2 — Output 2
Input 3 — Output 3
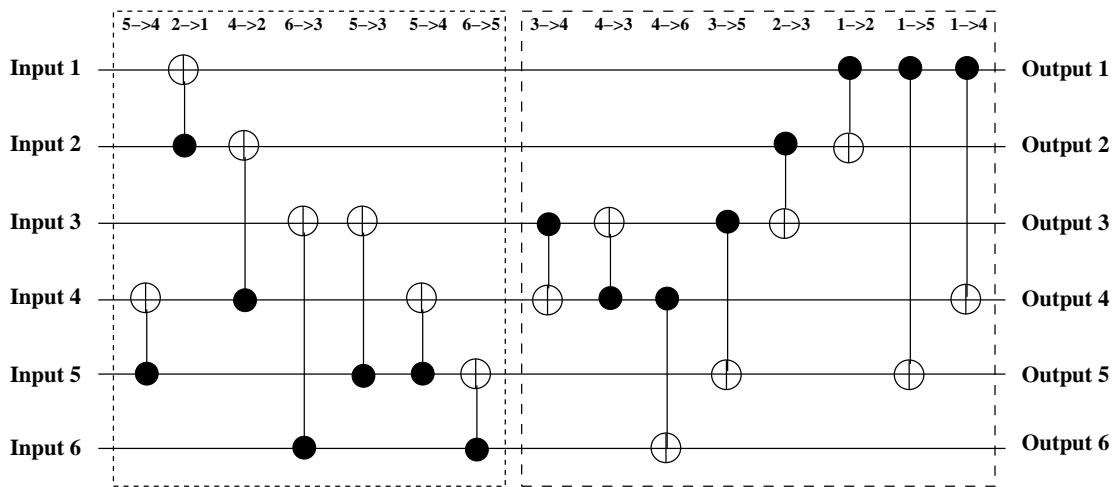Input 4 — Output 4
Input 5 — Output 5
Input 6 — Output 6

Figure 3: Synthesized C-NOT circuit example. The gates in the right and left boxes correspond to row operations before and after the transpose step respectively. Those in the left box are in the order the row operations were applied and their controls and targets are switched. The gates in the right box are in the reverse order that the row operations were applied.

5) (Step B - section 1, column 2) One already on diagonal.

6) (Step C - section 1, column 2) Remove remaining ones in column below diagonal.

$$
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,1 & 0\,0 & 1\,0 \\
0\,0 & 1\,1 & 1\,1 \\
0\,0 & 0\,1 & 1\,1 \\
0\,0 & 1\,1 & 1\,0
\end{bmatrix}
\xRightarrow{2 \to 3}
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 0\,1 & 0\,0 \\
0\,0 & 1\,1 & 1\,1 \\
0\,0 & 0\,1 & 1\,1 \\
0\,0 & 1\,1 & 1\,0
\end{bmatrix}
$$

7) (Step A - section 2) Eliminate duplicate sub-rows below row 2.

$$
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 0\,1 & 0\,0 \\
0\,0 & 1\,1 & 1\,1 \\
0\,0 & 0\,1 & 1\,1 \\
0\,0 & 1\,1 & 1\,0
\end{bmatrix}
\xRightarrow[4 \to 6]{3 \to 5}
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 0\,1 & 0\,0 \\
0\,0 & 1\,1 & 1\,1 \\
0\,0 & 0\,0 & 1\,1 \\
0\,0 & 0\,0 & 0\,1
\end{bmatrix}
$$

8) (Step B - section 2, column 3) Place one on diagonal.

$$
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 0\,1 & 0\,0 \\
0\,0 & 1\,1 & 1\,1 \\
0\,0 & 0\,0 & 1\,1 \\
0\,0 & 0\,0 & 0\,1
\end{bmatrix}
\xRightarrow{4 \to 3}
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 1\,0 & 1\,1 \\
0\,0 & 1\,1 & 1\,1 \\
0\,0 & 0\,0 & 1\,1 \\
0\,0 & 0\,0 & 0\,1
\end{bmatrix}
$$

9) (Step C - section 2, column 3) Remove remaining ones in column below diagonal.

$$
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 1\,0 & 1\,1 \\
0\,0 & 1\,1 & 1\,1 \\
0\,0 & 0\,0 & 1\,1 \\
0\,0 & 0\,0 & 0\,1
\end{bmatrix}
\xRightarrow{3 \to 4}
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 1\,0 & 1\,1 \\
0\,0 & 0\,1 & 0\,0 \\
0\,0 & 0\,0 & 1\,1 \\
0\,0 & 0\,0 & 0\,1
\end{bmatrix}
$$

10) Matrix is now upper triangular. Transpose and continue.

$$
\begin{bmatrix}
1\,1 & 0\,0 & 0\,0 \\
0\,1 & 0\,1 & 1\,0 \\
0\,0 & 1\,0 & 1\,1 \\
0\,0 & 0\,1 & 0\,0 \\
0\,0 & 0\,0 & 1\,1 \\
0\,0 & 0\,0 & 0\,1
\end{bmatrix}
\xRightarrow{\text{transpose}}
\begin{bmatrix}
1\,0 & 0\,0 & 0\,0 \\
1\,1 & 0\,0 & 0\,0 \\
0\,0 & 1\,0 & 0\,0 \\
0\,1 & 0\,1 & 0\,0 \\
0\,1 & 1\,0 & 1\,0 \\
0\,0 & 1\,0 & 1\,1
\end{bmatrix}
$$

11) (Step A - section 1) Eliminate duplicate sub-rows.

$$
\begin{bmatrix}
1\,0 & 0\,0 & 0\,0 \\
1\,1 & 0\,0 & 0\,0 \\
0\,0 & 1\,0 & 0\,0 \\
0\,1 & 0\,1 & 0\,0 \\
0\,1 & 1\,0 & 1\,0 \\
0\,0 & 1\,0 & 1\,1
\end{bmatrix}
\xRightarrow{4 \to 5}
\begin{bmatrix}
1\,0 & 0\,0 & 0\,0 \\
1\,1 & 0\,0 & 0\,0 \\
0\,0 & 1\,0 & 0\,0 \\
0\,1 & 0\,1 & 0\,0 \\
0\,0 & 1\,1 & 1\,0 \\
0\,0 & 1\,0 & 1\,1
\end{bmatrix}
$$

12) (Step B - section 1, column 1) Because matrix is triangular and non-singular there will always be ones on the diagonal.

13) (Step C - section 1, columns 1 and 2) Remove remaining ones in column 1 and then column 2.

$$
\begin{bmatrix}
1\,0 & 0\,0 & 0\,0 \\
1\,1 & 0\,0 & 0\,0 \\
0\,0 & 1\,0 & 0\,0 \\
0\,1 & 0\,1 & 0\,0 \\
0\,0 & 1\,1 & 1\,0 \\
0\,0 & 1\,0 & 1\,1
\end{bmatrix}
\xRightarrow[2 \to 4]{1 \to 2}
\begin{bmatrix}
1\,0 & 0\,0 & 0\,0 \\
0\,1 & 0\,0 & 0\,0 \\
0\,0 & 1\,0 & 0\,0 \\
0\,0 & 0\,1 & 0\,0 \\
0\,0 & 1\,1 & 1\,0 \\
0\,0 & 1\,0 & 1\,1
\end{bmatrix}
$$

Algorithm 1: C-NOT Circuit Synthesis

```
[circuit] = CNOT_Synth(A, n, m)
{
    // synthesize lower/upper triangular part
    [A,circuit_l] = Lwr_CNOT_Synth(A, n, m)
    A = transpose(A);
    [A,circuit_u] = Lwr_CNOT_Synth(A, n, m)

    // combine lower/upper triangular synthesis
    switch control/target of C-NOT gates in circuit_u;
    circuit = [reverse(circuit_u) | circuit_l];
}
```

```
        [A,circuit] = Lwr_CNOT_Synth(A, n, m)
        {
            circuit = [];
            for (sec=1; sec<=ceil(n/m); sec++) // Iterate over column sections
            {
                // remove duplicate sub-rows in section sec
                for (i=0; i<2^m; i++)
                    patt[i] = NOT_FOUND; //marker for first positions of sub-row patterns
                for (row =(sec-1)*m; row <n; row++)
                {
                    sub-row_patt = A[row,(sec-1)*m:sec*m-1];
                    // if first copy of pattern save otherwise remove
                    if (patt[sub-row_patt] == NOT_FOUND)
                        patt[sub-row_patt] = row;
                    else
                        A[row,:]  += A[patt[sub-row_patt],:];
Step A                  circuit = [C-NOT(patt[sub-row_patt],row) | circuit];
                }

                // use Gaussian elimination for remaining entries in column section
                for (col=(sec-1)*m; col<sec*m-1; col++)
                {
                    // check for 1 on diagonal
                    diag_one = 1;
                    if (A[col,col] == 0)
                        diag_one = 0;

                    // remove ones in rows below column col
                    for (row=col+1; row<n; row++)
                    {
                        if (A[row,col] == 1)
                          if (diag_one == 0)
                            A[col,:]  += A[row,:];
Step B                      circuit = [C-NOT(row,col) | circuit];
                            diag_one = 1;
                          A[row,:]  += A[col,:];
Step C                      circuit = [C-NOT(col,row) | circuit];
        } } } }
```

14) (Step A - section 2) Eliminate duplicate sub-rows.

$$\left[\begin{array}{cc|cc|cc} 1\,0 & & 0\,0 & & 0\,0 & \\ 0\,1 & & 0\,0 & & 0\,0 & \\ \hline 0\,0 & & 1\,0 & & 0\,0 & \\ 0\,0 & & 0\,1 & & 0\,0 & \\ \hline 0\,0 & & 1\ \ 1 & & 1\,0 & \\ 0\,0 & & \boxed{1}\ \ \boxed{0} & & 1\,1 & \end{array}\right] \begin{array}{c} 3\to 6 \\ \Longrightarrow \end{array} \left[\begin{array}{cc|cc|cc} 1\,0 & & 0\,0 & & 0\,0 & \\ 0\,1 & & 0\,0 & & 0\,0 & \\ \hline 0\,0 & & 1\,0 & & 0\,0 & \\ 0\,0 & & 0\,1 & & 0\,0 & \\ \hline 0\,0 & & 1\ \ 1 & & 1\,0 & \\ 0\,0 & & 0\ \ 0 & & 1\,1 & \end{array}\right]$$

15) (Step C - section 2, columns 1 and 2) Remove remaining ones in column 1 and then column 2.

$$\left[\begin{array}{cc|cc|cc} 1\,0 & & 0\,0 & & 0\,0 & \\ 0\,1 & & 0\,0 & & 0\,0 & \\ \hline 0\,0 & & 1\,0 & & 0\,0 & \\ 0\,0 & & 0\,1 & & 0\,0 & \\ \hline 0\,0 & & \boxed{1}\ \ \boxed{1} & & 1\,0 & \\ 0\,0 & & 0\ \ 0 & & 1\,1 & \end{array}\right] \begin{array}{c} 3\to 5 \\ 4\to 5 \\ \Longrightarrow \end{array} \left[\begin{array}{cc|cc|cc} 1\,0 & & 0\,0 & & 0\,0 & \\ 0\,1 & & 0\,0 & & 0\,0 & \\ \hline 0\,0 & & 1\,0 & & 0\,0 & \\ 0\,0 & & 0\,1 & & 0\,0 & \\ \hline 0\,0 & & 0\,0 & & 1\,0 & \\ 0\,0 & & 0\,0 & & 1\,1 & \end{array}\right]$$

16) (Step C - section 3, column 1) Remove remaining ones in column.

$$\left[\begin{array}{cc|cc|cc} 1\,0 & & 0\,0 & & 0\,0 & \\ 0\,1 & & 0\,0 & & 0\,0 & \\ \hline 0\,0 & & 1\,0 & & 0\,0 & \\ 0\,0 & & 0\,1 & & 0\,0 & \\ \hline 0\,0 & & 0\,0 & & 1\ \ 0 & \\ 0\,0 & & 0\,0 & & \boxed{1}\ \ 1 & \end{array}\right] \begin{array}{c} 5\to 6 \\ \Longrightarrow \end{array} \left[\begin{array}{cc|cc|cc} 1\,0 & & 0\,0 & & 0\,0 & \\ 0\,1 & & 0\,0 & & 0\,0 & \\ \hline 0\,0 & & 1\,0 & & 0\,0 & \\ 0\,0 & & 0\,1 & & 0\,0 & \\ \hline 0\,0 & & 0\,0 & & 1\,0 & \\ 0\,0 & & 0\,0 & & 0\,1 & \end{array}\right]$$

The synthesized circuit is specified by the row operations and is shown in Figure 3.

In general, the size of the synthesized circuit is given by the number of row operations used in the algorithm. By accounting for the maximum number of row operations in each step, we can calculate an upper bound on the maximum number of gates that could be required in synthesizing an n-wire linear reversible circuit. C-NOT gates are added in the steps marked Step A-C in the algorithm. Step A is used to eliminate the duplicates in the subsections. It is called fewer than $n + m$ times per section (combined for the upper/lower triangular stages of the algorithm), giving a total of no more than $(n + m) \cdot \lceil n/m \rceil$ gates. Step B is used to place ones on the diagonal. It can be called no more than n times. Step C is used to remove the ones remaining after all duplicate sub-rows have been cleared. Since there are only 2^m m-bit words, there can be at most as many non-zero sub-rows below the $m \times m$ sub-matrix on the diagonal. Therefore, Step C is called fewer than $m \cdot (2^m + m)$ times per section, or fewer than $2\lceil n/m \rceil m \cdot (2^m + m)$ times in all. Adding these up we have

$$\begin{aligned} \text{row ops} \quad &\leq \quad (n + m) \cdot \left\lceil \frac{n}{m} \right\rceil + n + 2 \left\lceil \frac{n}{m} \right\rceil m \cdot (2^m + m) \\ &\leq \quad \frac{n^2}{m} + n + n + m + n + 2n2^m + 2nm \\ &\quad\quad + 2m2^m + 2m^2. \end{aligned}$$

If $m = \alpha \log_2 n$,

$$\text{row ops} \quad \leq \quad \frac{n^2}{\alpha \log_2 n} + 3n + \alpha \log_2 n + 2n^{1+\alpha}$$

$$\begin{aligned} &+ 2n\alpha \log_2 n + 2\alpha \log_2 n \cdot n^\alpha \\ &+ 2\left(\alpha \log_2 n\right)^2. \end{aligned} \tag{3}$$

If $\alpha < 1$, the first term dominates as n gets large. Therefore the number of row operations is $O(n^2/\log n)$. Combining this result with Lemma 1, we have the following theorem.

Theorem 1 *The worst-case size of an n-wire C-NOT circuit is* $\Theta(n^2/\log n)$ *gates.*

In Equation 3, α can be chosen to be arbitrarily close to 1. In the limit, the multiplicative constant in the $O(n^2/\log n)$ expression becomes 1 (assuming logs are taken base 2). By contrast, the multiplicative constant in the lower bound in Lemma 1 is $1/2$.

This algorithm, in addition to generating smaller circuits than the standard method, is also asymptotically more efficient in terms of run time. The execution time of the algorithm is dominated by the row operations on the matrix, which are each $O(n)$. Therefore the overall execution time is $O(n^3/\log n)$ compared to $O(n^3)$ for standard Gaussian elimination [12, p. 42].

The result in Theorem 1 has direct implications to the problem of synthesizing an important class of quantum circuits known as stabilizer circuits. One definition of a stabilizer circuit is that it is a quantum circuit consisting of three basic gates (the C-NOT, the phase, and the Hadamard gates) and the measurement operation. An important result by Aaronson and Gottesman [1] shows that any stabilizer circuit can be decomposed into a short sequence of blocks of C-NOT gates, phase gates, and Hadamard gates. Phase and Hadamard gates act on single qubits. These gates can be thought of geometrically as performing $\pi/2$ and π rotations, respectively, in certain planes. This means that four consecutive phase gates or two consecutive Hadamard gates compute the identity function, an operation that leaves the qubit unchanged. Since these gates act on single qubits each section containing only phase gates can be synthesized using at most $3n$ phase gates, where n is the number of qubits in the circuit. Similarly, each Hadamard section can be synthesized using at most n Hadamard gates. The size of the circuit is, therefore, dominated by the C-NOT sections which Theorem 1 shows can be synthesized using $O(n^2/\log n)$ C-NOT gates.

Our algorithm is closely related to Kronrod's Algorithm (also known as "The Four Russians' Algorithm") for construction of the transitive closure of a graph [2]. One important difference between the two is that in their case the goal was a fast algorithm for their application, which is only of secondary concern for our application. Our primary goal is an algorithm that produces small circuits. Generically, our algorithm can be interpreted as producing an efficient elementary matrix decomposition of a binary matrix.

4   Empirical Validation

Though Algorithm 1 is asymptotically optimal, it would be of interest to know how large n must be before the algorithm begins to outperform standard Gaussian elimination. For this purpose we have synthesized linear reversible circuits using both

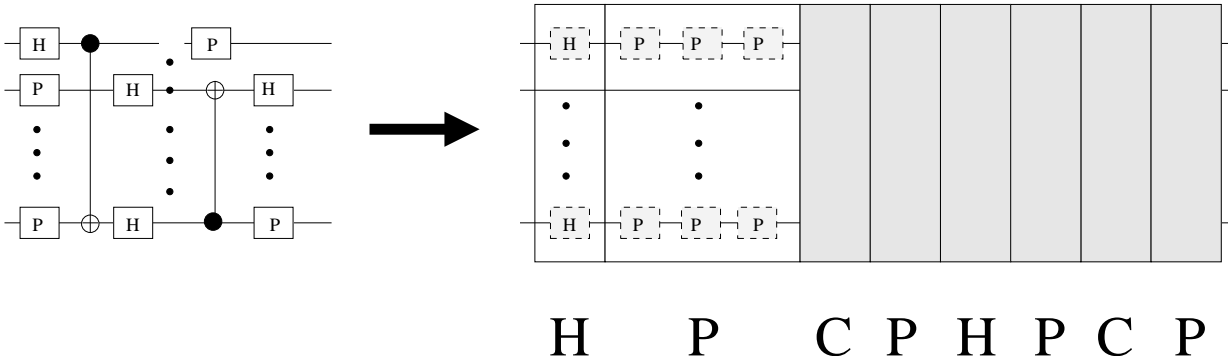$$\text{H} \quad \text{P} \quad \text{C} \; \text{P} \; \text{H} \; \text{P} \; \text{C} \; \text{P}$$

Figure 4: Stabilizer circuit decomposition given by Aaronson and Gottesman. An example of a stabilizer circuit is shown on the left along with its decomposition into phase (P), Hadamard (H), and C-NOT (C) gate blocks on the right. The operation performed by phase and Hadamard gates can be considered to be $\pi/2$ and π rotations, respectively, because four consecutive phase gates or two consecutive Hadamard gates compute the identity function. Therefore, each phase gate block requires at most $3n$ gates and each Hadamard gate block requires at most n gates, where n is the number of input/output wires. If these blocks contained more gates, there would be either four or more consecutive phase gates or two or more consecutive Hadamard gates on a wire. Consequently, the gate count of the decomposed circuit is dominated by the size of the C-NOT blocks which can be synthesized using $O(n^2/\log n)$ gates.
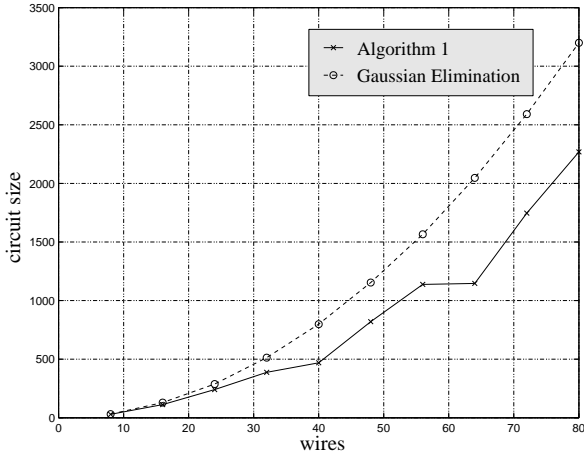

Figure 5: Performance of Algorithm 1 vs. Gaussian elimination on randomly generated linear reversible functions. Each point corresponds to the average size of the circuit generated for 100 randomly generated matrices. The x-axis specifies n, the number of inputs/outputs of the linear reversible circuit, and the y-axis specifies the average number of gates in the circuit synthesis. For Algorithm 1, m was chosen to be round$((\log_2 n)/2)$.

our method and Gaussian elimination for randomly generated non-singular 0-1 matrices. The results are summarized in Figure 5. Our algorithm shows an improvement over Gaussian elimination for n as small as 8. The size of the circuit synthesized by Algorithm 1 is dependent on the choice m, the size of the column sections. Here we have somewhat arbitrarily chosen $m = \text{round}((\log_2 n)/2)$. The performance for some values of n could be significantly improved by optimizing this choice.

This would also smooth out the performance curve in Figure 5 for Algorithm 1.

5 Conclusions and Future Work

We have given an algorithm for linear reversible circuit synthesis that is asymptotically optimal in the worst-case. We show that the algorithm is also asymptotically faster than current methods. While our results are primarily asymptotic, empirical results show that even in the finite case our algorithm outperforms the current synthesis method. Applications of our work include quantum circuit synthesis.

While the primary motivations for the synthesis method we have given here are to provide an asymptotic bound on circuit complexity and a practical method to synthesize small circuits, another application is to bounds on circuit complexity for the finite case. In particular, we can use our method to determine an upper bound on the maximum number of gates required to synthesize any n wire C-NOT circuit. For this application the particular partitioning of the columns can be very important. For example, much better bounds can be determined if the size of the sections are a function of the location of the section in the matrix. Sections to the left have more rows below the diagonal and therefore should be larger than sections towards the right of the matrix which have fewer rows below the diagonal. An ongoing area of work is determining optimal column partitioning methods.

Our algorithm basically yields an efficient decomposition for matrices with elements in $\mathbb{F}_2$, and can be generalized in a straightforward manner for matrices over any finite field. The asymptotic size of the generalized decomposition is $O(n^2/\log_{|F|} n)$, where $|F|$ is the order of the finite field. Our algorithm, particularly in this generalized form, is quite generic and may lend itself to a wide range of other applications. Re-

lated algorithms [2] have applications in finding the transitive closure of a graph, binary matrix multiplication, and pattern matching.

The work of Aaronson and Gottesman [1] shows that our results are directly applicable to the synthesis of stabilizer circuits, an important class of quantum circuits. A major area of future work is extending our results to other classes of reversible circuits, particularly other quantum circuits. Currently, there is an asymptotic gap between the best upper and lower bounds on the worst-case circuit complexity both for general classical reversible circuits and quantum circuits. The gap for classical reversible circuits is the same logarithmic factor that previously existed for linear reversible circuits [13], which suggests it may be possible to extend our methods to this problem.

Our results may also be directly applicable to the general reversible circuit synthesis problem. It has been shown that any reversible circuit can be decomposed into a series of four circuit blocks: a T block composed of only Toffoli gates (a generalized three input/output C-NOT gate), a C block composed of only C-NOT gates, another T block, and finally an N block composed of only NOT gates [13]. Any classical reversible circuit can be synthesized by synthesizing each of the individual blocks. Synthesizing the N block is trivial, and the results here provide asymptotically optimal realizations for the C block. Thus, a synthesis method producing small circuits for the remaining T blocks could yield small overall circuits. However, unlike for the case of stabilizer circuits, here the circuit size is not dominated by the C-NOT sections, but rather by the T sections.

While the focus here has been on reducing the number of gates in the synthesized circuit, in practice the proposed algorithm also typically reduces the circuit depth over Gaussian elimination based methods. However, modifications to the proposed algorithm can reduce the circuit depths further, and are an area of future work.

References

[1] S. Aaronson and D. Gottesman. "Improved simulation of stabilizer circuits." *Manuscript in preparation.*

[2] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev. "On economical construction of the transitive closure of an oriented graph." *Soviet Mathematics Doklady*, pages 1209–10, 1970.

[3] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. Smolin, and H. Weinfurter. "Elementary gates for quantum computation." *Physical Rev. A*, pages 3457–67, 1995.

[4] C. H. Bennett, G. Brassard, C. Crepeau, R. Jozsa, A. Peres, and W. Wootters. "Teleporting an unknown quantum state via dual classical and EPR channels." *Physical Rev. Letters*, pages 1895–1899, 1993.

[5] C. H. Bennett and S. J. Wiesner. "Communication via one- and two-particle operators on Einstein-Podolsky-Rosen states." *Physical Rev. Letters*, pages 2881–2884, 1992.

[6] T. Beth and M. Rötteler. "Quantum algorithms: Applicable algebra and quantum physics." *Quantum Information*, pages 96–150. Springer, 2001.

[7] G. Cybenko. "Reducing quantum computations to elementary unitary operations." *Comp. in Sci. and Engin.*, pages 27–32, March/April 2001.

[8] D. P. DiVincenzo. "Two-bit gates are universal for quantum computation." *Physical Rev. A*, pages 1015–22, 1995.

[9] D. M. Miller, D. Maslov, and G. W. Dueck, "A Transformation Based Algorithm for Reversible Logic Synthesis." DAC, pages 318–323, 2003.

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

[11] M. Perkowski et al., "A general decomposition for reversible logic." *Reed-Muller Workshop*, August 2001.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambrigde University Press, 1992.

[13] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. "Synthesis of reversible logic circuits." *IEEE Trans. on CAD*, pages 710-722, June 2003.