Improved Algorithms for Debugging Problems on Erroneous Reversible Circuits

Yuma Inoue[1](✉) and Shin-ichi Minato[1,2]

[1] Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, Japan
yuma@ist.hokudai.ac.jp

[2] JST ERATO MINATO Discrete Structure Manipulation System Project, Sapporo-shi, Japan

Abstract. Reversible circuits and their synthesis methods have been actively studied in order to realize reversible computation. However, there are few known ways to debug erroneous reversible circuits. In this paper, we propose new algorithms for debugging problems. For single gate error, we improve the theoretical efficiency of previous methods, which use worst case exponential time algorithms such as SAT or decision diagrams. We also propose an algorithm debugging multiple gate error circuits by using πDDs, decision diagrams for permutation sets. We evaluate our algorithms theoretically and experimentally, and confirm significant improvement.

Keywords: Reversible computation · Circuit design · Permutations · Algorithms · Decision diagrams · Dynamic programming

1 Introduction

Reversible computation is a fundamental technology for next generation computation such as quantum computation [11] and optical computing [3]. Computation is reversible if we can determine an input pattern for a given output pattern. This means reversible computation is information-lossless. Therefore, reversible computation is also used for low power design [1,8].

Due to the reversible property, a reversible logic circuit has neither fan-out nor feedback, i.e. formed as a cascade of reversible logic gates. This distinguishes synthesis of reversible circuits from irreversible ones, and attracts many researchers to study synthesis approaches [2,4,9,13,16].

On the other hand, there are few results concerning debugging such circuits, which is another important process to analyze reversible circuits. Wille et al. [17] proposed the first algorithm to debug reversible circuits using SAT formulation and solvers based on debugging techniques for irreversible circuits. Frehse et al. [6] gave a simulation-based approach and combined it with the SAT-based approach. Since their methods consider only a single gate error, Jung et al. [7] proposed an extended approach for multiple gate errors.

J. Krivine and J.-B. Stefani (Eds.): RC 2015, LNCS 9138, pp. 186–199, 2015.
DOI: 10.1007/978-3-319-20860-2_12

Tague et al. [14] provided another approach for a single gate error using πDDs [10], decision diagrams for permutation sets. However, there are two problems to be considered:

– These algorithms use exponential algorithms or data structures, i.e. they are intractable in the worst case.
– These algorithms only detect error positions, i.e. cannot fix errors efficiently.

In this paper, we address these tasks with different approaches for a single error and multiple errors, respectively. For a single error, we propose a theoretically improved debugging algorithm. This algorithm uses the lemma in [17], which states correction is determined by function composition, and valid gate checking algorithms. For multiple errors, we provide a dynamic programming approach using πDDs. Although this algorithm has worst-case complexity similar to the approach for a single error of Tague et al. [14], it can fix multiple errors and debug them.

We evaluate the efficiency of our algorithms using computational experiments. For single error circuits, our algorithm achieves a significant improvement compared with previous approaches. For multiple error circuits, our algorithm succeeds to fix errors with minimal corrections in circuits with few lines.

The remainder of this paper is organized as follows. Section 2 briefly reviews reversible circuits and πDDs, which are used in our algorithm. In Section 3, we define the problem of debugging single error circuits, review previous work, and introduce our algorithm for debugging single error circuits. In Section 4, we extend the debugging problem of single error circuits to multiple errors, and provide our πDD-based debugging method. Experimental results to evaluate the practical performance of our algorithms are in Section 5 and Section 6 concludes this paper.

2 Preliminary

In this section, we review reversible functions and circuits before proceeding to πDDs, which are used in both previous work and our proposed method.

2.1 Reversible Functions and Permutations

A function $f : \{0,1\}^n \to \{0,1\}^n$ is *reversible* if it is bijective, i.e., we can determine an input from the corresponding output. Hence, a function f is considered as a permutation on $\{0, 1, \dots, 2^n - 1\}$.

We define notations of permutations. A permutation on $\{0, 1, \dots, m - 1\}$, m-permutation for short, is written in a one-line form $\pi = \pi(0)\pi(1)\dots\pi(m-1)$. The identity m-permutation is denoted by e_m, which satisfies $e_m(i) = i$ for each $0 \le i \le m-1$. We denote by π^{-1} the inverse permutation of π, which satisfies $\pi * \pi^{-1} = \pi^{-1} * \pi = e_m$, where $*$ means composition of permutations: $p = q * r$ means $p(i) = r(q(i))$ for all i.

f

x_1	x_2	x_3	y_1	y_2	y_3	π_f
0	0	0	0	0	1	1
0	0	1	0	0	0	0
0	1	0	0	1	1	3
0	1	1	0	1	0	2
1	0	0	1	0	1	5
1	0	1	1	1	0	6
1	1	0	1	1	1	7
1	1	1	1	0	0	4

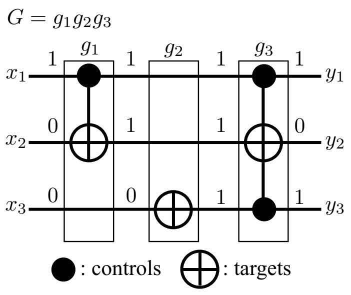


Fig. 1. Truth table of a reversible function f and a reversible circuit G realizing f

We denote by π_f the permutation corresponding to f such that we consider input and output bit vectors as binary representations of integers. For example, π_f corresponding to the reversible function f on the left of Fig. 1 is $\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 3 & 2 & 5 & 6 & 7 & 4 \end{pmatrix}$, which is briefly written as $\pi_f = 10325674$.

2.2 Reversible Circuits and Gates

Reversible circuits realize reversible functions and consist of reversible gates. A reversible circuit for an n-bit Boolean function has n lines as shown on the right of Fig. 1. Reversible circuits have no fan-out or feedback due to their reversible properties. Therefore, a reversible circuit is a cascade of reversible gates. Several reversible gates have been invented to synthesize reversible circuits, such as Toffoli [15], Fredkin [5], and Peres [12] gates. In this paper, we focus on Toffoli gates.

Let $L = \{1, \ldots, n\}$ be a set of lines. *Toffoli gates* have multiple (possibly zero) *control lines* $C = \{c_1, \ldots, c_k\} \subset L$ and one *target line* $t \in L \setminus C$. For example, the Toffoli gate g_3 in Fig. 1 has the control lines $C = \{1, 3\}$ and the target line $t = 2$. A Toffoli gate inverts the target line when all control lines are 1. Let x_i and y_i be the i-th line's input and output of a Toffoli gate, respectively. Then, we formally define Toffoli gates as follows:

$$y_t = x_t \oplus x_{c_1} \cdots x_{c_k},$$
$$y_i = x_i \quad \text{if } i \neq t.$$

Since a Toffoli gate itself represents a reversible function, we can represent the function corresponding to a Toffoli gate as a permutation. We denote by π_g the permutation corresponding to a gate g as well as a reversible function. Then the permutation representation π_G of the function realized by a reversible circuit $G = g_1 \cdots g_d$ equals $\pi_{g_1} * \cdots * \pi_{g_d}$.

2.3 πDD

First, we define a *transposition* $\tau_{i,j}$ as the permutation such that $\tau_{i,j}(i) = j$ and $\tau_{i,j}(j) = i$, but $\tau_{i,j}(k) = k$ for other k. Any n-permutation can be uniquely

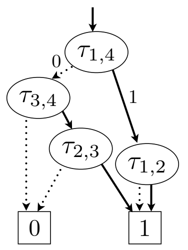


Fig. 2. The πDD representing $\{2431, 4231, 1423\} = \{\tau_{1,2} * \tau_{1,4}, \tau_{1,4}, \tau_{2,3} * \tau_{3,4}\}$

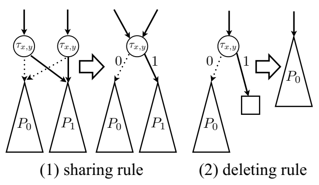


Fig. 3. Two reduction rules of πDDs

decomposed into a composition of at most $n-1$ transpositions by the following algorithm: we start with e_n and repeat swaps to move $\pi(k)$ to the k-th position in its one line form, where k runs from right to left.

A *πDD* is a rooted directed graph representing a set of permutations compactly, and has efficient set operations for permutation sets [10]. πDDs consist of five components: labeled internal nodes, 0-edges, 1-edges, the 0-sink, and the 1-sink. Fig. 2 shows an example of a πDD. Each internal node has exactly two edges, a 0-edge and a 1-edge. Each path to the 1-sink in a πDD represents a permutation in the set represented: if a 1-edge originates from a node with label $\tau_{x,y}$, the decomposition of the permutation contains $\tau_{x,y}$, while a 0-edge means that the decomposition excludes $\tau_{x,y}$.

A πDD becomes a compact and canonical form by fixing its transposition order and applying the following two reduction rules (Fig. 3):

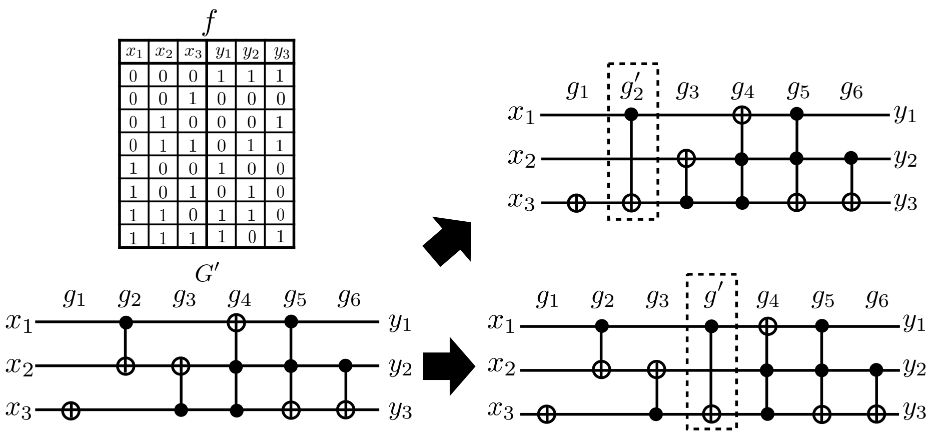
(1) sharing rule: share all nodes which have the same labels and child nodes.
(2) deleting rule: delete all nodes whose 1-edge points to the 0-sink.

In many practical cases, πDDs demonstrate high compression ratio, although πDD size (i.e. the number of nodes) in the worst case is exponential in the length of permutations.

In addition, πDDs support efficient set operations such as union and intersection on permutation sets. In particular, Cartesian product operation $P \times Q$, which returns the union set of compositions of all pairs $p \in P$ and $q \in Q$, is important and useful for our algorithm to be described later. Since the computation time of these operations depends on the size of πDDs, compactness helps to speed-up πDD operations.

3 Debugging Single Error

We define the single error debugging problem of reversible circuits. Let $f : \{0,1\}^n \to \{0,1\}^n$ be a reversible function and $G = g_1 \cdots g_d$ be a reversible circuit with n lines such that $\pi_G = \pi_f$. We define G' to be a single error circuit for f if $\pi_{G'} \neq \pi_f$ and G' has:



x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	1	1	1
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	0	1

Fig. 4. An erroneous circuit G' and two fixed circuits realizing f

- a replaced error: there is a gate $g' \neq g_i$ s.t. $G' = g_1 \cdots g_{i-1} g' g_{i+1} \cdots g_d$,
- an inserted error: there is a gate g' s.t. $G' = g_1 \cdots g_{i-1} g' g_i g_{i+1} \cdots g_d$, and
- a removed error: $G' = g_1 \cdots g_{i-1} g_{i+1} \cdots g_d$.

The goal of the single error debugging problems is to find the position of an error in an erroneous circuit G' and fix it in order to realize f correctly. We note that even if the number of embedded errors is only one, sometimes there are several ways to debug the circuit. For example, Fig. 4 describes an erroneous circuit G' and an objective function f. At this instance, we have the two ways to debug G': replacing g_2 with g_2' or inserting g' between g_3 and g_4. In general, we cannot determine which of them the original error is. Therefore, we set our goal to list all ways to debug G'.

3.1 Related Work

Wille et al. proposed a debugging method using SAT solvers [17]. They used SAT (Boolean satisfiability) formulation for debugging problems and solved it with SAT solvers. This method has three problems to be overcome:

- There are $O(nd)$ variables in SAT formula. Though state-of-the-art SAT solvers work practically fast, solving SAT is believed to require exponential time in the worst case. This is therefore not scalable for a large d.
- Their method can find only error candidates, which may include non-errors.
- Their method can debug only a replaced error.

We also note that this method requires verification preprocess to obtain some counterexamples.

Frehse et al. provided a simulation-based debugging algorithm [6]. Their method eliminates error candidates based on the fact that an error gate must be activated (i.e. all the inputs of control lines are 1) for all counterexamples. This method is fast because it runs in linear time with respect to the number of gates and lines. However, outputs of this method also can contain non-errors, since the activation property is a necessary condition but not a sufficient condition.

Tague et al. gave a debugging method using πDDs for a removed error [14]. They considered a gate as a permutation, and used πDDs to represent the set of gates. They insert a πDD into an erroneous circuit G' as an arbitrary gate, and calculate the compositions by Cartesian product operations. If the compositions contain π_f, it means G' has a removed error. This method also has two problems:

– The size of πDDs for a set of N-permutations is $O(2^{N^2})$, and now $N = 2^n$. It is not scalable for even small n.
– Their method can detect an error but cannot find its position and fix it.

In the next subsection, we provide an algorithm overcoming these problems. More precisely, we propose a worst-case $O(n2^n d)$ time algorithm, which can find and fix all the three types of errors.

3.2 Proposed Method for Single Error

Our method is based on Lemma 3 in [17]:

Theorem 1 (Lemma 3 in [17]). *Let F be an error-free circuit of a reversible function and $G = G_1 g_i G_2$ be an erroneous circuit of F. Then G can be fixed by replacing any gate g_i of G with a cascade of gates $G_i^{fix} = G_1^{-1} F G_2^{-1}$.*

This theorem states that, if G_i^{fix} can be represented by a Toffoli gate, the i-th gate is a replaced error and we can fix it by replacing it with the Toffoli gate corresponding to G_i^{fix}. Hereafter, we assume the objective function f and each gate g_i are represented as permutations, and a cascade of gates means the composition of permutations. Then the single replaced error circuit problem can be solved as follows: checking whether $G_i^{rep} = g_{i-1}^{-1} \cdots g_1^{-1} f g_d^{-1} \cdots g_{i+1}^{-1}$ can be represented as a single Toffoli gate for all $1 \leq i \leq d$. Similarly, debugging problems for other types of errors can be solved too:

– an inserted error: checking whether $G_i^{ins} = g_{i-1}^{-1} \cdots g_1^{-1} f g_d^{-1} \cdots g_{i+1}^{-1}$ can be represented as an identity permutation e_{2^n} for all $1 \leq i \leq d$.
– a removed error: checking whether $G_i^{rem} = g_i^{-1} \cdots g_1^{-1} f g_d^{-1} \cdots g_{i+1}^{-1}$ can be represented as a single Toffoli gate for all $0 \leq i \leq d$.

Note that the position of a removed error is between two gates or two ends. We say a removed error occurs at the 0-th position if the error position is the left g_1, and at the i-th position if the error position is the right of g_i.

We let $N = 2^n$ for brevity. If we had an $O(h(n))$ time algorithm checking whether a given permutation represents a Toffoli gate, we could solve the single error circuit problem in $O(d(Nd + h(n)))$ by calculating the products G_i^{rep}, G_i^{ins}, and G_i^{rem} of $O(d)$ N-permutations and running a checking algorithm for all $0 \leq i \leq d$. We can improve this complexity by using the following properties:

- $G_i^{rep} = G_i^{ins}$,
- $G_i^{rem} = g_i^{-1} G_i^{ins}$,
- $G_i^{ins} = G_{i-1}^{rem} g_i$.

That is, incremental calculation of G_i^x from G_{i-1}^y costs only $O(N)$ time, hence we can solve a single error circuit problem in $O(d(N+h(n)))$. Algorithm 1 gives the entire procedure.

Algorithm 1. Debugging single error circuits

```
 1: procedure DebugSingleError($f, G$)
 2:     $G_0^{rem} \leftarrow f g_d^{-1} g_{d-1}^{-1} \cdots g_1^{-1}$
 3:     if isToffoli($G_0^{rem}$) then
 4:         Report a removed error: the gate $G_0^{rem}$ is removed at the 0-th position.
 5:     end if
 6:     for $i = 1$ to $d$ do
 7:         $G_i^{ins} \leftarrow G_{i-1}^{rem} g_i$
 8:         if $G_i^{ins} = e_N$ then
 9:             Report an inserted error: $g_i$ is an extra gate.
10:         else if isToffoli($G_i^{ins}$) then
11:             Report a replaced error: $g_i$ should be replaced by $G_i^{ins}$.
12:         end if
13:         $G_i^{rem} \leftarrow g_i^{-1} G_i^{ins}$
14:         if isToffoli($G_i^{rem}$) then
15:             Report a removed error: the gate $G_i^{rem}$ is removed at the $i$-th position.
16:         end if
17:     end for
18: end procedure
```

The Toffoli gate checking problem is also solved in $O(nN)$ time by Algorithm 2. A permutation representing a Toffoli gate works as a transposition between integers a and b if a and b differ exactly a target bit and their bits in a control bit set are all 1. Lines 3–22 of Algorithm 2 identify control lines and a target line, eliminating cases not satisfying necessary conditions. Lines 24–31 check whether control lines and a target line work as an expected Toffoli gate. This algorithm works as not only a check but also an identification of the corresponding Toffoli gate. That is, we can directly debug G' using the Toffoli gate. It costs $O(nN)$ time and therefore we can solve the single error circuit problem in $O(nNd)$ time.[1]

We can design checking algorithms for Fredkin gates and Peres gates similarly. Generally speaking, given a set of gates, we can solve the single error circuit problem in $O(d(N+h(n)))$ time if we have an $O(h(n))$ time checking algorithm for the gates. We also can easily adapt to deal with negative control

[1] If we assume w-bit word RAM model, we can improve it to $O(\lfloor \frac{n}{w} \rfloor Nd)$ by adopting bit parallel techniques to manage control lines C.

lines. A Toffoli gate with positive and negative control lines inverts its output of the target line when the inputs of the positive controls are all 1 and the negative controls are all 0.

Algorithm 2. Checking whether a given permutation represents a Toffoli gate.

```
 1: procedure isToffoli($\pi$)
 2:     $C \leftarrow \{1, \ldots, n\}$, $T \leftarrow \phi$
 3:     for $i = 0$ to $N - 1$ do
 4:         if $\pi_{\pi_i} \neq i$ then                ▷ $\pi_i$ is neither $i$ nor swapped with $\pi_{\pi_i}$
 5:             return False
 6:         end if
 7:         if $i$ and $\pi_i$ are swapped then
 8:             if $i$ and $\pi_i$ differ only the $j$-th bit in binary then
 9:                 $T \leftarrow T \cup \{j\}$
10:                 if $|T| > 2$ then    ▷ there are two or more candidates of target lines
11:                     return False
12:                 end if
13:             else                     ▷ there are two or more candidates of target lines
14:                 return False
15:             end if
16:             for $j = 1$ to $n$ do
17:                 if the $j$-th bit of $i$ in binary is 0 then
18:                     $C \leftarrow C \setminus \{j\}$          ▷ eliminate candidates of control lines
19:                 end if
20:             end for
21:         end if
22:     end for
23:     ▷ The Toffoli gate corresponding to $\pi$ must have controls $C$ and a target $t \in T$
24:     for $i = 0$ to $N - 1$ do
25:         if $\forall j \in C$, the $j$-th bit of $\pi_i$ in binary is 1, but $\pi_i = i$ then
26:             return False          ▷ all controls are 1 but the target is not inverted
27:         end if
28:         if $\exists j \in C$, the $j$-th bit of $\pi_i$ in binary is 0, but $\pi_i \neq i$ then
29:             return False          ▷ some controls are 0 but the target is inverted
30:         end if
31:     end for
32:     return True
33: end procedure
```

4 Debugging Multiple Errors

We extend single error circuit problems to multiple error circuit problems. We define that k-error circuits are circuits including k errors. Note that k errors can consist of different kinds of errors; replaced errors, inserted errors, and removed errors can be mixed together. We also note that k-error circuits may be debugged by less than k corrections. For example, two inserted errors of a same Toffoli gate at adjacent positions need not to be debugged, in other words these can be debugged by 0 corrections. In multiple error circuit problems, we set our goal to find minimum corrections.

4.1 Related Work

Jung et al. [7] proposed a SAT based debugging algorithm for multiple errors, which is an extension of [6]. They used pruning based on hitting set problems and encoded it into SAT formulation. Although their method can process large circuits, it has two problems to be considered:

- Their method can debug only replaced errors.
- Their method can detect only error candidates, which includes non-errors and cannot fix them directly.

In this section, we try to overcome these problems.

4.2 Naïve Extension of Existing Method

Our proposed method for k-error circuits is derived from Tague's πDD-based approach for single error circuits [14]. For an inserted error, this approach tries to insert a πDD representing usable gates into all possible positions. It can be easily extend to replaced errors and removed errors. If we insert (or replace, remove) k πDDs as sets of usable gates at all possible positions for each, we can detect all error positions and error types. However, there are the following problems:

- The number of all combinations of k positions are $\binom{d}{k} = O(d^k)$. Furthermore, we consider 3 types of errors for each position, i.e. there are 3^k ways of combinations of error types. That is, this algorithm requires $O(3^k d^{k+1})$ πDD operations.
- All error positions are can be detected, but correct gates for replaced errors and removed errors cannot be determined.

We attack these problems with our algorithm proposed in the next subsection.

4.3 Proposed Method for Multiple Errors

We propose a debugging algorithm requiring only $O(dk)$ πDD operations[2] for k-error circuits. Our approach uses dynamic programming calculating $S_{i,j}$, defined as a set of permutations representing functions which can be realized by the first j gates with i errors. The minimum x such that $\pi_f \in S_{x,d}$ is the size of minimum corrections. We can calculate $S_{i,j}$ by the following recurrence relations:

$$S_{0,0} := \phi,$$
$$S_{i,j} := (S_{i,j-1} \times \{g_j\}) \cup (S_{i-1,j-1} \times L) \cup (S_{i-1,j} \times L) \cup S_{i-1,j-1},$$

[2] Note that each πDD operation costs exponential time in N^2 in the worst case.

where L is a set of usable gates, which are Toffoli gates in this paper. The first term represents non-error, the second one represents a replaced error, the third one represents an inserted error, and the last one represents a removed error.

Since each $S_{i,j}$ is a set of permutations, we can use πDDs to represent them. Further, calculation of recurrence relations requires only permutation set algebra such as union and Cartesian product, which are supported by πDDs. Each calculation of $S_{i,j}$ requires at most a constant number (i.e. 6) of operations. Hence this algorithm takes only $O(dk)$ πDD operations. In addition, we can calculate this recurrence relation by incrementing k. This means if the minimum corrections of a given k-error circuit is k', this algorithm only costs $O(dk')$ πDD operations, instead of $O(dk)$.

This algorithm can determine the minimum corrections, but cannot identify error positions and types yet. Error identification can be realized by starting from $S_{k',d}$ with π_f and reversely traversing to $S_{0,0}$. For example, if we now consider $S_{i,j}$ with π_x and $(\{\pi_x\} \times L^{-1}) \cap S_{i-1,j-1} \neq \phi$, an replaced error is detected at position j. Furthermore, let $\pi_y \in (\{\pi_x\} \times L^{-1}) \cap S_{i-1,j-1}$, we identify the original gate is $\pi_x * \pi_y^{-1}$. We then restart reverse traversal from $S_{i-1,j-1}$ with π_y until the first index is not 0.

5 Experiments

We implemented all algorithms in C++[3] and carried out experiments on a 3.20GHz CPU machine with 64GB memory. We randomly generate d Toffoli gates with n lines and concatenate them to make correct reversible circuits G. We prepare objective functions f for each circuit by simulating the circuit. Next, we generate erroneous reversible circuits G' with k errors based on correct circuits: we randomly select a position and replace with a random gate, insert a random gate, or remove a gate k times.

Our implementation uses f and G' as inputs. For single error circuits, our implementation detects all corrections but only outputs the number of ways of corrections in order to reduce I/O time. For multiple error circuits, since the way of minimum corrections can be huge, our implementation detects only one way of minimum corrections and outputs it.

5.1 Experiments for Single Error

Computation time of Algorithm 1 for single error circuits (i.e. k=1) is shown in Table 1. This table shows that our algorithm is linear with the number of gates d and almost exponential with the number of lines n. It agrees with the theoretical complexity of our algorithm analyzed in Section 3.

In [17], the SAT solver-based algorithm takes about 2000 seconds or more for $n \geq 8$ and $d \geq 5000$ circuits. On the other hand, our algorithm takes under

[3] Note that our implementation of Algorithm 2 uses bitwise operations of 64-bit integer (unsigned long long int in C++) to manage control lines C.

Table 1. Computation time (seconds) for single error circuits

		d								
		10	50	100	500	1000	5000	10000	50000	100000
n	2	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.05	0.10
	4	0.00	0.00	0.00	0.00	0.00	0.01	0.02	0.09	0.17
	6	0.00	0.00	0.00	0.00	0.00	0.02	0.02	0.12	0.24
	8	0.00	0.00	0.00	0.00	0.00	0.02	0.04	0.21	0.41
	10	0.00	0.00	0.01	0.01	0.01	0.05	0.11	0.54	1.08
	12	0.00	0.00	0.01	0.03	0.04	0.21	0.40	2.05	3.99
	14	0.01	0.02	0.04	0.20	0.38	1.90	3.78	8.83	17.64
	16	0.03	0.10	0.19	0.89	1.75	8.78	17.61	87.71	149.00
	18	0.16	0.52	1.03	4.81	9.46	48.37	97.47	493.42	987.10
	20	0.60	1.87	3.88	18.28	35.90	187.28	377.66	—	—

1 second for circuits of such scale. Further, in [14], the πDD-based algorithm takes more than 100 seconds for $n \geq 4$ and $d \geq 1000$ cases, while our algorithm takes under 0.01 seconds for these cases. This significant improvement is likely due to the theoretical improvement of our algorithm, and not simply to hardware and test case differences.

The simulation-based approach proposed by Frehse et al. in [6] seems to be faster than or equal to our algorithm: Their method completed simulation to detect error candedates in 20 seconds for the $n = 15$ and $d = 716934$ circuit. However, their method output over 30000 error candidates, which is impractical to check manually. In contrast, our algorithm returned only one correction for the $n = 16$ and $d = 100000$ erroneous circuit embedded a replaced error.

5.2 Experiments for Multiple Errors

We also carried out experiments for multiple error circuits. We randomly embedded k errors in circuits consisting of d gates with n lines. Figs. 5–8 show experimental results for 1-, 2-, 3-, and 4-error circuits, respectively.

For 1-error circuits in Fig. 5, i.e. single error circuits, our πDD-based algorithm can perform in 1000 seconds for $n = 5$ and $d = 600$ circuits. For 2-error circuits in Fig. 6, however, all cases with $n = 5$ are time-outs even at $d = 50$. Almost all $n = 4$ cases also time-out; the algorithm can debug up to 100-gate circuits. Results in [7] show that the SAT based method is more scalable: e.g. this method can process $n = 8$ and $d = 637$ circuits in about 300 seconds. However, outputs of this method can include non-errors, and cannot fix them automatically. On the other hand, our method can fix them. For sufficiently small circuits, our method can provide richer debugging information.

Results of 3- and 4-error circuits in Figs. 7 and 8. Our algorithm seems to be enough scalable for the circuits with $n \leq 3$. Debugging time for 3-errors and one of 4-errors seems similar. This is because in random circuits we prepared, the minimum correction of $n = 2$ circuits is usually 1, and for $n = 3$ circuits is

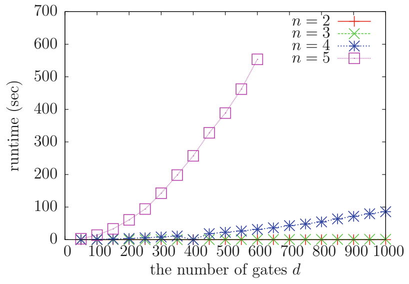


Fig. 5. Runtime for debugging 1-error circuits

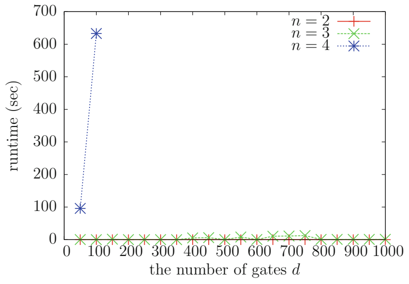


Fig. 6. Runtime for debugging 2-error circuits

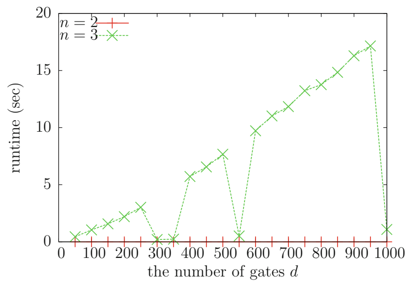


Fig. 7. Runtime for debugging 3-error circuits

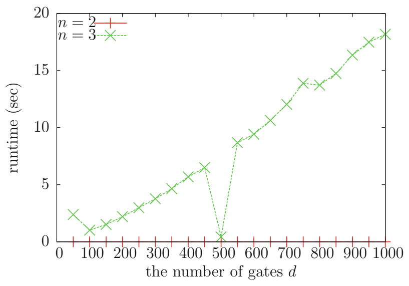


Fig. 8. Runtime for debugging 4-error circuits

usually 2, regardless of the number of embedded errors. In Fig. 8, $d = 50$ and $d = 500$ in $n = 3$ cases seem to be somehow outliers. It is true; the minimum correction size of the $d = 50$ circuit is 3, and for $d = 500$ circuit it is 1.

These results indicate that the minimum correction and the number of lines exponentially affect computation time. On the other hand, the number of gates seems to affect linearly for small gates ($n = 2, 3$), but affect quadratically or exponentially for slightly larger gates ($n = 4, 5$).

6 Concluding Remarks

For debugging erroneous reversible circuits, we propose two kinds of algorithms. The first one is an efficient method for circuits having at most one error. This method uses permutation properties of reversible gates and gate checking algorithms. This method can handle more general gate library by designing gate checking algorithms. The efficient performance of this method is shown theoretically and experimentally, comparing with existing methods. The second algorithm can debug multiple error circuits based on a dynamic programming approach and πDDs. Although the scalability of this algorithm is exponentially worse than the first one, the algorithm enables us to debug more general erroneous reversible circuits.

For future work, we would like to modify the first algorithm to handle circuits with garbage output lines. Garbage lines can output arbitrary values, i.e. multiple permutations can realize desired behavior. This means that multiple G_i's should be considered. Of course πDDs can handle this, but such an algorithm will lose the efficiency of our first approach.

For multiple errors, more scalable algorithms are desirable. We are also interested in expected sizes of minimum corrections for circuits with n lines, d gates, and k randomly-embedded errors. From experimental results, we guess that minimum correction tend to become relatively small with the number of embedded errors. If we show that the size is sufficiently small with high probability, perhaps we need not to consider debugging circuits with a large number of errors.

Acknowledgments. We would like to thank Dr. Mathias Soeken and Dr. Robert Wille for valuable discussion and feedback to write this paper.

References

1. Bérut, A., Arakelyan, A., Petrosyan, A., Ciliberto, S., Dillenschneider, R., Lutz, E.: Experimental verification of Landauer's principle linking information and thermodynamics. Nature **483**(7388), 187–189 (2012)
2. Chattopadhyay, A., Majumder, S., Chandak, C., Chowdhury, N.: Constructive reversible logic synthesis for boolean functions with special properties. In: Yamashita, S., Minato, S. (eds.) RC 2014. LNCS, vol. 8507, pp. 95–110. Springer, Heidelberg (2014)
3. Cuykendall, R., Andersen, D.R.: Reversible optical computing circuits. Optics Letters **12**(7), 542–544 (1987)
4. Donald, J., Jha, N.K.: Reversible logic synthesis with Fredkin and Peres gates. ACM Journal on Emerging Technologies in Computing Systems (JETC) **4**(1), 2 (2008)
5. Fredkin, E., Toffoli, T.: Conservative logic. International Journal of Theoretical Physics 219–253 (1982)
6. Frehse, S., Wille, R., Drechsler, R.: Efficient simulation-based debugging of reversible logic. In: the 40th IEEE International Symposium on Multiple-Valued Logic (ISMVL), pp. 156–161 (2010)
7. Jung, J.C., Frehse, S., Wille, R., Drechsler, R.: Enhancing debugging of multiple missing control errors in reversible logic. In: the 20th symposium on Great Lakes symposium on VLSI (GLVLSI), pp. 465–470. ACM (2010)
8. Landauer, R.: Irreversibility and heat generation in the computing process. IBM Journal of Research and Development **5**(3), 183–191 (1961)
9. Maslov, D., Dueck, G.W., Miller, D.M.: Techniques for the synthesis of reversible toffoli networks. ACM Transactions on Design Automation of Electronic Systems (TODAES) **12**(4), 42 (2007)
10. Minato, S.: πDD: a new decision diagram for efficient problem solving in permutation space. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 90–104. Springer, Heidelberg (2011)

11. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2010)
12. Peres, A.: Reversible logic and quantum computers. Physical Review A **32**(6), 3266 (1985)
13. Rahman, M.Z., Rice, J.E.: Templates for positive and negative control toffoli networks. In: Yamashita, S., Minato, S. (eds.) RC 2014. LNCS, vol. 8507, pp. 125–136. Springer, Heidelberg (2014)
14. Tague, L., Soeken, M., Minato, S., Drechsler, R.: Debugging of reversible circuits using πDDs. In: the 43rd IEEE International Symposium on Multiple-Valued Logic (ISMVL), pp. 316–321. IEEE (2013)
15. Toffoli, T.: Reversible Computing. Springer (1980)
16. Wille, R., Große, D., Dueck, G.W., Drechsler, R.: Reversible logic synthesis with output permutation. In: the 22nd International Conference on VLSI Design, pp. 189–194. IEEE (2009)
17. Wille, R., Große, D., Frehse, S., Dueck, G.W., Drechsler, R.: Debugging of Toffoli networks. In: The Conference on Design. Automation and Test in Europe (DATE), pp. 1284–1289. European Design and Automation Association, IEEE (2009)