Exact Template Matching using Graphs

by

Md. Mazder Rahman, Gerhard W. Dueck, and Joe Horton

TR 13-224, April, 2013

Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3
Canada

Phone: (506) 453-4566
Fax: (506) 453-3566
Email: fcs@unb.ca
http://www.cs.unb.ca

Abstract. Reversible circuits can be represented as cascades of generalized Toffoli gates. Template matching has been successfully used in the optimization of such cascades. One of the difficulties in finding matchings of templates in circuits is due to the mobility of the gates within the circuit. Thus far, template matching procedures have employed heuristics to reduce the matching complexity. A graph structure with the corresponding algorithms is presented for exact template matching. This facilitates the discovery of new templates as well as exact minimization of circuits under very strict conditions.

Keywords: Logic Synthesis, Reversible Logic, Reversible Circuit, Reversible Template.

1 Introduction

The promise of exponential speed up in quantum computation and the reversibility of quantum computation motivates research in synthesizing reversible logic. For the last decades, significant number of research in synthesis of reversible logic in circuits and its optimization towards finding the minimal circuits have been done. Most of the synthesis and post synthesis methods use heuristics and exhaustive search technique. The Transformation Based Algorithm [1] is widely used to find a circuit for a given reversible function. This algorithm is only practical for functions with at most 20 variables. Additionally, it seldom produces optimal results. Therefore, there is still room to improve the results with post synthesis methods. The post synthesis optimization termed template matching for simplifying circuits originated in [1]. Later, the significance of templates at quantum level [2] was recognized and reconfigurable templates were introduced. An algorithm to find templates [3] as well as a new definition of template with its properties [4] have been presented. However, due to the lack of exact matching, no previous work has achieved optimal circuits.

In this paper, we describe the implementation of an exact template matching algorithm in which both circuits and templates are represented as Hasse Diagrams. The significance of this graph based matching is verified with circuits obtained from the transformation based algorithm and from randomly generated circuits.

In the remainder of the paper, the proposed approach and the significance of template matching are described as the follows: Section 2 provides the basics of synthesis of reversible logic and the post synthesis method known as template matching. Section 3 discusses the fundamental issues in template matching. Section 4 delineates the exact template matching algorithm based on Hasse Diagrams. In Section 5 the basic idea of generating templates from identity circuits is outlined with a set of templates. In Section 6, the experiment shows that the minimality of 3-input circuits can be achieved by template matching. Section 7 concludes the paper.

2 Preliminaries

A multiple-input and multiple-output logic function $f : B^n \rightarrow B^n$ is **reversible** if there is a one-to-one and onto mapping between input and output vectors. A reversible function can be realized by the cascades of reversible gates. Such cascades do not have feedback nor fanout to preserve the property of reversibility. A reversible gate over the inputs $X = \{x_0, x_1, \cdots, x_{n-1}\}$ consists of a (possibly empty) set $C = \{x_{i_0}, x_{i_1}, \cdots, x_{i_k}\}$ of control lines $C \subset X$ and a set $T \subset X - C$ of target lines. Reversible gates such as Toffoli [5], Peres [6] and Fredkin [7] are widely used to synthesize reversible circuits.

In this paper we use generalized **Multiple-Control Toffoli** (MCT) gates, denoted as $TOF_n(C, t)$, where n denotes the number of lines and t is a single target. The target line t is inverted if all values on the control lines in C are equal to 1 or if $C = \emptyset$. All remaining values are passed through unchanged. In the case of $C = \emptyset$ and $|C| = 1$ the gates $TOF_0()$ and $TOF_1(t)$ are known as NOT and $CNOT$ respectively. A reversible circuit is referred to as a MCT circuit if it is realized by MCT gates.

Definition 1. *The* **size** *of a circuit c is defined as the number of its gates and denoted by $|c|$. The realization of a function f, by circuit c is said to be minimal if no realization with fewer gates exists.*

Gates in a reversible circuit can often be reordered without affecting the function of the circuit. The mobility of gates is determined by the **moving rule** that relies on the following property [1].

Property 1. Two adjacent gates g_1 and g_2 with controls C_1 and C_2 and targets t_1 and t_2 in a circuit can be interchanged if $C_1 \cap t_2 = \emptyset$ and $C_2 \cap t_1 = \emptyset$.

If two identical self-inverse gates in a circuit can moved such that they are adjacent, then that pair of gates realizes the identity function and both can be deleted. This is known as the **gate deletion rule**.

It has been shown that circuits realizing the identity function can be used to reduce the cost of circuits [8]. Such identity circuits are called templates since they allow the replacement of part of a circuit with a sequence of gates that has lower cost. The formal template definition from [4] is given below.

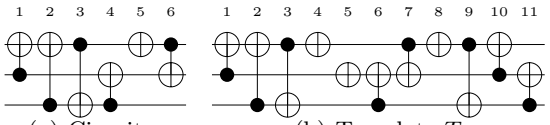
Definition 2. *A* **template** *T is a circuit that realizes the identity function such that there is a sequence of gates with size $|T|/2 + 1$ in T is not reducible by any other template.*

If a sequence of gates in a circuit matches a sequence of gates s in template T, then the matched sequence of gates in the circuit can be replaced with the inverse of the remaining sequence in T. This process is known as **template matching**. Template matching results in circuits with fewer gates if $|s| > |T|/2$. The gate sequences of templates can be matched in circuits in both in forward and backward directions [8, 9].

3 Issues in Template Matching

Some gates in reversible circuits may be moved significant distances. This mobility of gates makes the matching process challenging. The sequence of gates to be matched in a template may not even appear in the same order in the circuit. This problem is illustrated in the following example.

Example 1. The circuit c is to be optimized with template T (both are shown in Fig. 1). The sequences of gates in the template are considered in circular as well as forward and backward directions. The gates sequence $\{1, 2, 3, 4\}$ of template T matches with the gates sequence $\{1, 2, 3, 5\}$ in circuit c. However, for gate 5 in template T, no match is found. Therefore, template matching stops at this point. Since the size of matched sequence $5 < |T|/2 + 1$ the template is not applied. However, it is clear that all gates in circuit can be matched with the gates in template if the sequence of gates in template appears in different ordered such as gate 5 moves to the position 7.

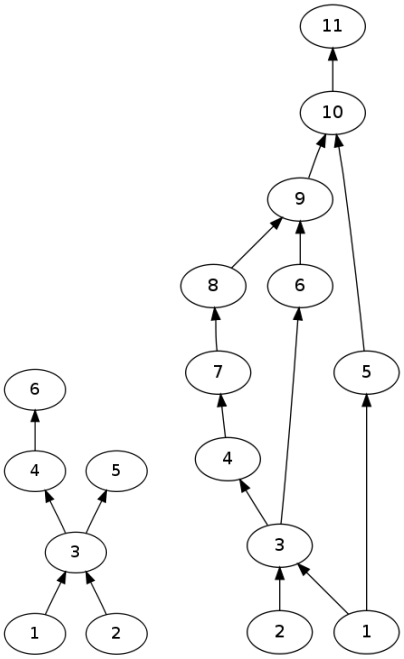


(a) Circuit c. (b) Template T.

Fig. 1. Reversible circuits.

The complexity in exact template matching result from two sources. First, due to mobility of the gates in the template many subsequences must be checked. Second, once a potential sequence in the template has been identified, the matching gates in the circuit may not be adjacent – in fact they may appear in different order. To reduce the matching time, heuristics are employed in most proposed optimization procedures. For the first problem gates are only taken in the order in which they appear in the template (templates are also reversed and viewed as circular [8]). For the second problem, gates are only processed in the order in which they appear in the circuit and the search is limited to a certain distance from the current match [9]. This is justified by the fact that the likelihood of find a matching gate that is at a considerable distance from the current match is small.

The first attempt to systematically find all possible sequences was done in [10], where the partial order of gates in the circuit are captured in a Hasse Diagram. The Hasse Diagram representation of the circuit c in Fig. 1(a) is shown in Fig. 2(a). Not all possible sequences of the template, however, are considered in [10]. Here we propose to represent both the template as well as the circuit with Hasse diagrams, together with a set of algorithms for efficient matching. The Hasse Diagram for template T in Fig. 1(b) is shown in Fig. 2(b).



(a) H_1 (b) H_2

Fig. 2. Hasse Diagram representations of circuits: 2(a) H_1 for c and 2(b) H_2 for T in Fig. 1.

4 Exact Template Matching

In this section we will describe an algorithm that given a template T and a circuit C will find the largest sequence of gates in T that match a sequence of gates in C. If the matched sequence is larger than $|T|/2$ than the sequence will be replaced as indicated before.

4.1 Graph Algorithms

Given any MCT circuit, the Hasse Diagram can be found with the algorithm shown in Fig. 3. As indicated before, a template may contain many valid sequences of gates that can potentially be matched. For example, the template shown in Fig. 1(b) has many sequences of size 6 that must be checked. The first few sequences are $\{123456, 123457, 123467, 123546, \ldots\}$. However, if 123456 has been checked, then 123546 does not need to be considered. Due to the mobility of gates this sequence will be checked in the circuit automatically when 123456 is considered. Our algorithm works incrementally. That is, if no match for a sequence is found, then the sequence is not extended. For example, given the template in the example above, if sequence 123 is not found, then no sequence with this subsequence needs to be checked.

```
(1)  Graph ConstructGraph(C)
(2)    // C is an array of gates
(3)    // let n be the number of gates in C
(4)    // let used be a bit array of length n
(5)    // let G be a graph with n nodes
(6)    for i = 2 to n
(7)      set the first i − 1 bits of used to false
(8)      for j = i − 1 down to 1
(9)        if !used[j]
(10)         if C[i] is blocked by C[j]
(11)           addEdge(G, i, j)
(12)           MarkPred(G, used, j)
(13)     return G
```

Fig. 3. Algorithm to construct the graph for a circuit.

```
(1)  Match FindBestMatch(T, C)
(2)    G = ConstructGraph(C)
(3)    B = null // the best match
(4)    s = ∅ // sequence to be matched
(5)    initialize M, invalid and in_degree
(6)    while s can be extended
(7)      extend s by one gate
(8)      while CheckMatch(G, T, M, s, invalid, in_degree)
(9)        if better match than B is found
(10)         set B
(11)       extend s
(12)   return B
```

Fig. 4. Algorithm to find the best match of template T in circuit C.

```
(1)  bool CheckMatch(G, T, M, s, invalid, in_degree)
(2)    // Check if the match can be extended
(3)    // G — graph of the circuit under consideration
(4)    // T — the template
(5)    // M — the set of matches M = {M_1, M_2, ... M_|s|}
(6)    // where M_i contains the matches with the first i gates in s
(7)    // invalid — bitstring for invalid nodes
(8)    // in_degree — number of in edges for each node
(9)    for each m ∈ M_||S|−1
(10)     if m can be extended to s
(11)       let v be the node corresponding to the matched gate
(12)       MarkNodeMatched(G, invalid, in_degree, v)
(13)       M_||S| = M_||S| ∪ v
(14)   return |M_||S|| > 0
```

Fig. 5. Algorithm to check if sequence s can be matched.

Example 2. According to the proposed algorithm, the template is mapped into the circuit as shown in Fig. 9(c). The resulting optimized circuit would be the inverse sequence of gates $\{5, 8, 9, 10, 11\}$.

```
(1)  MarkNodeMatched(G, invalid, in_degree, v)
(2)    // update the graph G after matching node v
(3)    // invalid — bitstring for invalid nodes
(4)    // in_degree — number of in edges for each node
(5)
(6)    invalid[v] = true
(7)    MarkPred(G, invalid, v)
(8)    for each v1 ∈ {succ(G, v)}
(9)      in_degree[v1] = in_degree[v1] − 1)
(10)       if in_degree[v1] = 0
(11)         invalid[v1] = false
(12)       for each v2 ∈ {succ(G, v1)}
(13)         MarkSucc(G, invalid, v)
```

Fig. 6. Algorithm to update the graph G after matching node v.

```
(1)  MarkSucc(G, invalid, v)
(2)    if !invalid[v]
(3)      invalid[v] = true
(4)      for each v1 ∈ {succ(G, v)}
(5)        MarkSucc(G, invalid, v1)
```

Fig. 7. Algorithm to mark successors of node v as invalid.

```
(1)  MarkPred(G, invalid, v)
(2)    if !invalid[v]
(3)      invalid[v] = true
(4)      for each v1 ∈ {pred(G, v)}
(5)        MarkPred(G, invalid, v1)
```

Fig. 8. Algorithm to mark predecessors of node v as invalid.

5 Generation of Templates

From the properties of templates [4] we have the following. Given a template $T = s_1 s_2$ where $|s_1| >= |T|/2 + 1$ and s_1 can not reduced any other template, then the remaining sequence s_2 where $|s_2| < |T|/2$ is minimal. Based on minimal circuits for the function f, denoted by $S_f = \{C_n^1, C_n^2, \ldots, C_n^m\}$ where n is the size of the circuits, a systematic method to find identities from the sets S_f and $S_{f^{-1}}$ was proposed in [3]. Template matching is used to determine if a given identity is a template. The number of potential identities and generated templates are shown in column II and III in Table 1. It can noticed that the number of identities of size from 13 to 17 are very large. Each identity must be tested via template matching. Currently we tested only the identities of size up to 12. With some fine tuning of the screening process and with the help of a super computer, we expect to complete the tests for all 3-input identities in the near future.

Traditionally, a single template has represented many re-writing rules. For example, the template with 5 gates presented in [8] and shown in Fig. 10(e) actually represents several identities. Any line that has no targets may appear once, not at all, or multiple times. In our prototype implementation we did not use this encoding. Therefore, we have four templates of of size five (shown

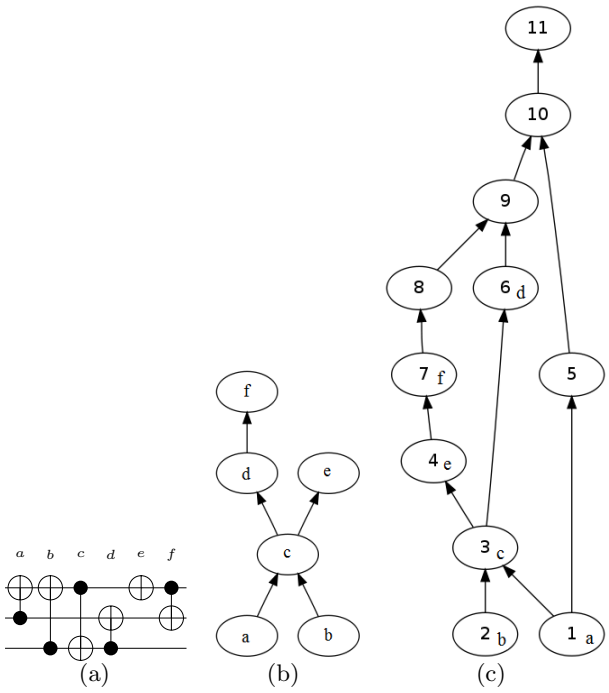


Fig. 9. Template Matching: 9(a) circuit; 9(b) Graph of 9(a), 9(c) Mapping the gates of template into circuit.

in Fig. 10). The encoding of the templates has two advantages. First, fewer templates need to be stored and checked. Second, the matching process should be faster, since fewer templates must be checked. On the other hand, the matching process is more complicated – for this reason it was not used in our prototype. The tradeoffs will be analyzed with further experiments.

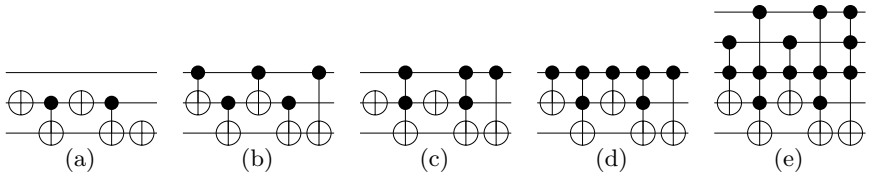


Fig. 10. Encoding of templates.

$Size$	$\#PID$	$\#Templates$	$\#Used(T)$	$\#Applications(T)$
0	0	0	0	0
1	0	0	0	0
2	12	3	0	0
3	0	0	0	0
4	0	0	0	0
5	144	4	4	35,530
6	111	12	11	4,818
7	1,740	5	5	11,601
8	2,874	31	14	4,997
9	24,048	140	125	17,875
10	39,462	221	53	2,291
11	294,864	1061	636	13,476
12	408,891	1196	131	624
13	2,932,674	U	1640	6,716
14	2,328,768	U	109	176
15	15,316,914	U	1126	1,713
16	3,322,881	U	4	4
17	8,902,932	U	6	6
Total	33,576,316	U	3862	99,827

Table 1. Generation of 3-line MCT Templates.

Size: Number of gates in circuits
#PID: Number of potential identities to be tested.
#Templates: Number of Templates.
U: Unknown.
#Used(T): Number of templates applied successfully during optimization.
#Applications(T): Number of times templates are applied in optimization.

6 Experimental Results

To evaluate the proposed algorithms, 3-input circuits obtained from transformation based algorithm [1] are optimized. The average size of the circuits used as inputs is shown in row III in Table 2. In the optimization process, the templates are applied in order of increasing size and the average reductions are shown in Table 2. By applying all templates of size up to 12 we have achieved significant reductions, in particular all minimal circuits of size up to 4 were found. However, at this point it it can be observed that the optimized circuits are very compact (most of them are only one gate away from the minimal circuits) and requires larger templates. Some templates can be strategically generated by using non-minimal and minimal circuits. The significance of such identities of size 13 to 17 as shown in column IV in Table 2 are detected in optimization. The column IV in Table 1 also shows that there are some templates have no significance on this particular benchmarks. The templates of odd size are used more frequently as shown in column V in Table 1.

Additionally, 20 random circuits of size 100 were generated and by applying templates of size up to 12, we achieved 14 minimal circuits with size 5 and 6.

$Size$	0	1	2	3	4	5	6	7	8
$\#Reals$	1	12	102	625	2780	8921	17049	10253	577
$Avg(TB)$	0	1	2.44118	4.1824	6.00719	7.64724	9.01689	9.93924	10.7452

Optimization

Templates	Average Size of Circuits by Applying Templates								
T_2	0	1	2.44118	4.1824	6.00719	7.64724	9.01689	9.93924	10.7452
T_5			2.11765	3.5552	5.33022	6.99697	8.39035	9.32283	10.1127
T_6			2.0	3.3312	5.0777	6.7822	8.18834	9.1383	9.93241
T_7				3.2352	4.85432	6.50286	7.89436	8.85341	9.66898
T_8				3.1216	4.61187	6.23338	7.63247	8.59368	9.37088
T_9				3.0576	4.23813	5.67851	7.10435	8.1502	9.01733
T_{10}				3.0192	4.10504	5.50364	6.94686	7.98469	8.85269
T_{11}				3.0	4.00791	5.07791	6.35662	7.48347	8.37088
T_{12}					4.0	5.01917	6.31016	7.46747	8.36395
T_{13}						5.0	6.02059	7.17488	8.13172
T_{14}							6.00317	7.16824	8.13172
T_{15}							6.0	7.00078	8.0104
T_{16}								7.0	8.0104
T_{17}									8.0

Table 2. Optimization of reversible circuits of 3-qubit.

Size: Number of gates in the minimal circuits.
#Real: Number of realizations.
Avg(TB): Average number of gates in circuits obtained from Transformation Based Algorithm [1].

The significant reduction from size 100 to size 5 or 6 can be observed in the snapshots as shown in Fig. 11.

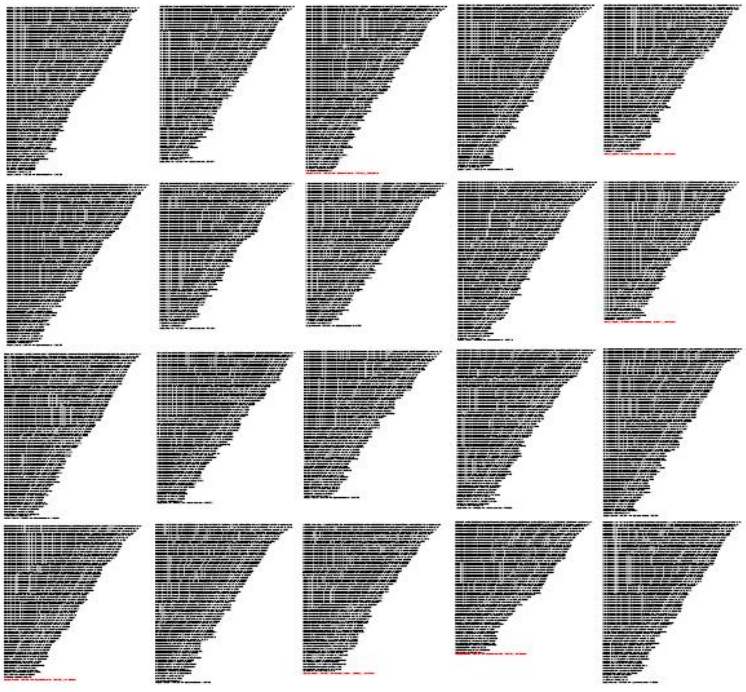
7 Conclusion

We have presented exact template matching that will find all minimal circuits if a complete set of templates is given. The significant reduction in random circuits that were optimized with an incomplete set of templates (only up to size 12) is promising. This prototype implementation for 3-line circuits can be extended to templates as well as circuits with more lines. Empirical results have shown that some templates are applied more often than others. A detailed study is needed to find the set of the most useful templates. Depending on the level of optimization that is required, different set may be applied. The applicability of a given template may depend on the method by which the circuits were generated. For example, circuits obtained by the transformation based synthesis will have a different structure that those obtained from BDD based synthesis [11]. This analysis with the corresponding recommendations will be done in the near future.

References

1. Miller, D.M., Maslov, D., Dueck, G.W.: A transformation based algorithm for reversible logic synthesis. In: Design Automation Conference. (2003)
2. Rahman, M.M., Dueck, G.W., Banerjee, A.: Optimization of reversible circuits using reconfigured templates. In: 3rd Workshop on Reversible Computation. (2011) 143–154
3. Rahman, M.M., Dueck, G.W.: An algorithm to find quantum templates. In: IEEE Congress on Evolutionary Computation. (2012) 623–629
4. Rahman, M.M., Dueck, G.W.: Properties of quantum template. In: 4th Workshop on Reversible Computation. (2012) 171–185
5. Toffoli, T.: Reversible computing. Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci (1980)
6. Peres, A.: Reversible logic and quantum computers. Phys. Rev. A **32** (1985) 3266–3276
7. Fredkin, E., Toffoli, T.: Conservative logic. International Journal of Theoretical Physics **21** (1982) 219–253
8. Maslov, D., Dueck, G.W., Miller, D.M.: Toffoli network synthesis with templates. Transactions on Computer Aided Design **24** (2005) 807–817
9. Maslov, D., Dueck, G.W., Miller, D.M.: Techniques for the synthesis of reversible Toffoli networks. ACM Trans. Des. Autom. Electron. Syst. **12** (2007)
10. Scott, N., Dueck, G.W., Maslov, D.: Improving template matching for minimizing reversible Toffoli cascades. In: 7th International Symposium on Representations and Methodology of Future Computing Technologies. (2005) 4–9
11. Wille, R., Drechsler, R.: BDD-based Synthesis of Reversible Logic for Large Functions. In: Design Automation Conference. (2009) 270–275

(a)



(b)

Fig. 11. Snapshot of random circuits optimized with templates.