Exact Synthesis of Elementary Quantum Gate Circuits for Reversible Functions with Don't Cares

Daniel Große[1] Robert Wille[1] Gerhard W. Dueck[2*] Rolf Drechsler[1]

[1]*Institute of Computer Science, University of Bremen, 28359 Bremen, Germany*
{grosse,rwille,drechsle}@informatik.uni-bremen.de
[2]*Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada*
gdueck@unb.ca

Abstract

Compact realizations of reversible logic functions are of interest in the design of quantum computers. In this paper we present an exact synthesis algorithm, based on Boolean Satisfiability (SAT), that finds the minimal elementary quantum gate realization for a given reversible function. Since these gates work in terms of qubits, a multi-valued encoding is proposed.

Don't care conditions appear naturally in many reversible functions. Constant inputs are often required when a function is embedded into a reversible one. The proposed algorithm takes full advantage of don't care conditions and automatically sets the constant inputs to their optimal values. The effectiveness of the algorithm is shown on a set of benchmark functions.

1. Introduction

Research in quantum circuit synthesis is motivated by the growing interest in quantum computation [17]. Synthesis for quantum circuits can be approached from two different angles. Reversible logic circuits can be designed with generalized Toffoli gates [20], that are later mapped into quantum gates. The alternative is to target elementary quantum gates directly during the synthesis process.

The minimization of generalized Toffoli networks has been studied extensively. It is possible to construct optimal circuits for all $2^3! = 40320$ three-input functions [18] and store the resulting circuits in a table. Once such a table has been constructed, the optimal circuit for any three-input functions can be found in constant time. For the synthesis of larger reversible functions methods based on Boolean Satisfiability (SAT) [7, 22] and Quantified Boolean Formula (QBF) Satisfiability [24] have been proposed. These methods guarantee to find an optimal solution. However, too complex functions have to be synthesized with heuristic methods (see e.g. [12]). Once the circuit is obtained, the generalized Toffoli gates are then translated into elementary quantum gates [2]. During the linear translation the interaction between the gates is ignored, and the resulting circuit may be suboptimal. Techniques such as template matching [14] can be used to reduce the gate count in the resulting quantum circuit.

In order to avoid the mapping of Toffoli gates into quantum gates, elementary quantum gates may be targeted directly. For this case, heuristic as well as exact methods have been proposed. While the former ones (e.g. [1]) can be employed with no guarantee of minimality the exact approaches are limited to small functions. The results of a breath-first search that determines all optimal quantum circuits with three input variables are described in [13]. Another exact approach uses symbolic reachability analysis to find optimal quantum circuits [9].

In this paper we present an exact synthesis algorithm based on Boolean Satisfiability (SAT). The synthesis problem is expressed as a Boolean function such that it is satisfiable iff there exists a network of quantum gates with a given depth. Thereby a multi-valued encoding is used for representing the respective qubits used by the quantum gates. Our method can be applied to synthesize completely specified function, as well as those with don't care conditions. All quantum circuits are by necessity reversible. To realize a non-reversible function it must be embedded into a reversible one. This results in garbage outputs and constant inputs. We refer to [11] for a detailed treatment on garbage in reversible networks. Garbage outputs are by definition don't cares. Additionally, when there are constant inputs, these can be set arbitrarily. The proposed synthesis procedure assigns the optimal values to the constant inputs during this process. Our implementations shows significant speedups over the algorithm described in [9], i.e. in the best case quantum circuits are synthesized up to a factor of 45 faster.

The remaining paper is structured as follows. Section 2 describes the elementary quantum gates used in our target networks and briefly reviews SAT. The exact synthesis algorithm is presented in Section 3. Section 4 introduces the improvements for incompletely specified functions. Experimental results are discussed in Section 5. Finally, the paper is summarized and conclusions are given.

2. Background

2.1. Elementary Quantum Gates

The basic building block for a quantum computer is the qubit. A qubit is a two level quantum system, described by

*This work was done while Gerhard W. Dueck was visiting the University of Bremen.

Gate	Notation	Matrix
NOT		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
V		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$
V+		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \\ 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \end{pmatrix}$

Figure 1. Elementary quantum gates

a two dimensional complex Hilbert space. The two orthogonal quantum states $|0\rangle \equiv \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$ and $|1\rangle \equiv \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$ are used to represent the values 0 and 1. Any state of a qubit may be written as $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers with the following condition $|\alpha|^2 + |\beta|^2 = 1$. The quantum state of a single qubit is denoted by the vector $\binom{\alpha}{\beta}$. The state of a quantum system with $n > 1$ qubits is given by an element of the tensor product of the single state spaces and can be represented as a normalized vector of length 2^n, called the state vector. The state vector is changed through multiplication of appropriate $2^n \times 2^n$ unitary matrices [2].

Elementary gates are the building blocks for quantum computation. The following elementary gates are used in this paper (the notation for the gates with the corresponding matrices are shown in Figure 1):

- Inverter (NOT): A single qubit is inverted.
- Controlled inverter (CNOT): The target qubit is inverted if the control qubit is 1.
- Controlled V gate: The V operation is also know as the square root of NOT, since two consecutive V operation are equivalent to an inversion.
- Controlled V+ gate: The V+ gate performs the inverse operation of the V gate, i.e. $V+ \equiv V^{-1}$.

All elementary gates are assumed to have unit cost [2]. Additionally, when a CNOT and a V (or V+) gate are applied to the same two qubits, the cost of the pair can be considered as a unit as well [19]. The possible pairs (also known as *double gates*) are shown in Figure 2. Using this cost metric allows us a direct comparison to [9]. Note that our approach is not limited to these metrics – other metrics can be applied as well.

In this paper, the input to the circuit and each control line of a gate is restricted to 0 and 1. This has the effect that the value of each qubit is restricted to one of $\{0, 1, V_0, V_1\}$. That is we are dealing with 4-valued logic [9], where $V_0 = \frac{1+i}{2}\binom{1}{-i}$ and $V_1 = \frac{1+i}{2}\binom{-i}{1}$.

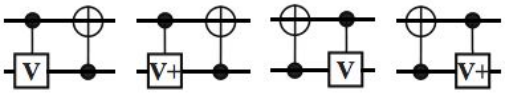

Figure 2. Pairs of gates with unit cost

2.2. Boolean Satisfiability

The *Boolean Satisfiability problem* (SAT problem) is defined as follows:

Definition 1 *Let h be a Boolean function in* Conjunctive Normal Form *(CNF), i.e. a product-of-sum representation. Then the SAT problem is to determine an assignment to the variables of h such that h evaluates to true or to prove that no such assignment exists.*

The CNF is a conjunction of clauses. A clause is a disjunction of literals and each literal is a propositional variable or its negation.

In the past several algorithms (so called *SAT solvers*) were proposed [3, 10, 16, 6]. Most of them are based on three essential procedures: (1) The decision heuristic assigns values to free variables, (2) the propagation procedure determines implications due to the last assignment(s) and (3) the conflict analysis tries to resolve conflicts by backtracking. Advanced techniques like e.g. *efficient Boolean constraint propagation* [16] or *conflict analysis* [10] are common in state-of-the-art SAT solvers. Today, SAT is not only used in formal verification, but in many application domains like automatic test pattern generation or logic synthesis.

3. Exact Synthesis Approach

In this section the exact synthesis algorithm is presented. The basic idea is to check if there exists a quantum gate network representation for the reversible function f with d gates, where d is increased in the next iteration if no realization is found. The iterative checks are performed by (1) encoding the synthesis problem as a SAT instance and (2) testing the SAT instance for satisfiability using an SAT solver. In the following the Boolean encoding of the synthesis problem is presented.

3.1. Boolean Encoding

For a reversible function the synthesis problem is expressed as a Boolean function S_d that is satisfiable iff there exists a quantum gate network representation of size d. Therefore, the following definitions are given:

Definition 2 *Let $f : B^n \rightarrow B^n$ be a reversible function. Then, $\vec{x}_i^k = (x_{in}^k x_{i(n-1)}^k \dots x_{i1}^k)$ with $(1 \leq i \leq 2^n)$ and $(0 \leq k \leq d)$ defines a vector representing the input, temporary, or output variables at depth k for line i of the truth table of f. The left side of the truth table corresponds to the vector $\vec{x}_i^0$, the right side to the vector $\vec{x}_i^d$, respectively.*

Since the temporary variables in $\vec{x}_i^k$ can assume non-Boolean values, i.e. besides 0 and 1 also V_0 and V_1 are possible, each x_{ij}^k with $(1 \leq j \leq n)$ represents a multi-valued

variable. For the transformation into CNF x_{ij}^k is encoded by using two Boolean variables y_{ij}^k and z_{ij}^k:

y_{ij}^k	z_{ij}^k	x_{ij}^k
0	0	0
0	1	V_0
1	0	1
1	1	V_1

That is, if z_{ij}^k is 0 the Boolean domain is considered, otherwise the non-Boolean quantum states V_0 and V_1 are selected.

Definition 3 *Let $f : B^n \rightarrow B^n$ be a reversible function. Then, $\vec{w}^k = (w^k_{\lceil log_2(g) \rceil} \dots w^k_1)$ with $(0 \leq k \leq d-1)$ defines a Boolean vector representing the chosen quantum gate at depth k. Here g denotes the number of different quantum gates in n variables.*

Lemma 1 *For a reversible function with n variables there exists $3n(n-1)+n$ different elementary quantum gates.*

Proof 1 *A* CNOT*,* V *and* V+ *gate has exactly one target line and one control line. With n variables there are $n(n-1)$ possible choices for the target-control pair. Additionally there are n possible* NOT *gates (one for each line). In total we get $3n(n-1)+n$ different elementary quantum gates.* ■

Lemma 2 *For a reversible function with n variables there exists $4n(n-1)$ double gates.*

The proof is similar to Lemma 1. In the following elementary quantum gates and double gates are called *quantum gates*. Thus, there are $7n(n-1)+n$ quantum gates that have be considered at each depth.

The application of the definitions and lemmas are illustrated with the following example.

Example 1 *Figure 3 shows the variables needed to formulate the constraints for a function with $n = 3$ inputs/outputs and depth $d = 2$. The possible positions for the quantum gates are marked with dashed rectangles. For each of the $2^3 = 8$ lines in the truth table $n = 3$ lines in the network with the respective vectors for input, temporary, and output variables are used (i.e. overall $3 \cdot 8 = 24$ lines are considered). As mentioned above the multi-valued variables of the vector $\vec{x}_i^k$ are represented by y_{ij}^k and z_{ij}^k. For each depth $3 \cdot (3-1) \cdot 7 + 3 = 45$ different quantum gates are possible. They can be defined by the $\lceil log_2(45) \rceil = 6$-bit vectors $\vec{w^0}$ and $\vec{w^1}$.*

The defined variables are used to build the instance. The Boolean formulation S_d for the synthesis of the reversible function f consists of the conjunction of the following three constraints:

1. The *input/output constraints* set the input/output pair of each line of the truth table given by the reversible function f:

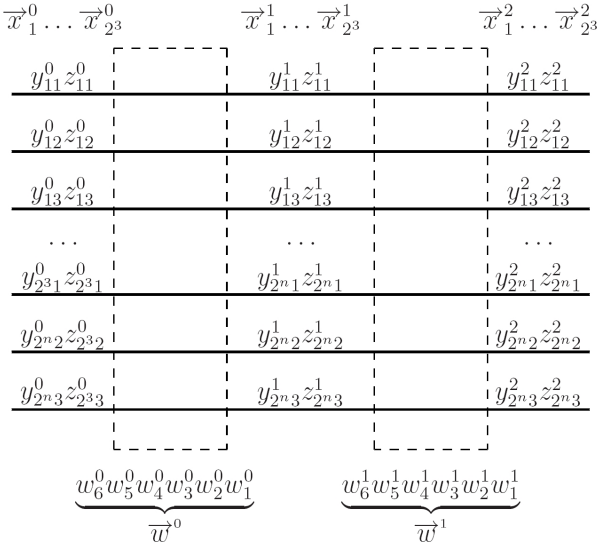

Figure 3. Variables in S_2 with $n = 3$ and $d = 2$

$$\bigwedge_{i=1}^{2^n} [\vec{x}_i^0]_2 = i \wedge [\vec{x}_i^d]_2 = f(i)$$

Here $[\vec{x}_i^0]_2$ ($[\vec{x}_i^d]_2$) represents a natural number derived by its corresponding binary encoding. Since the inputs (outputs) are Boolean, no V_0 and V_1 are allowed for the variables x_{ij}^0 in $\vec{x}_i^0$ and x_{ij}^d in $\vec{x}_i^d$, respectively ($1 \leq j \leq n$). Thus, each z_{ij}^0 (z_{ij}^d) is assigned to 0, while each y_{ij}^0 (y_{ij}^d) is assigned to 1 or 0 according to the truth table line (the specification of function f).

2. The *functional constraints* for the possible quantum gates that are chosen by an assignment to $\vec{w}^k$ are:

$$\bigwedge_{k=0}^{d-1} \bigwedge_{i=1}^{2^n} \vec{w}^k \rightarrow (\vec{x}_i^{k+1} = q(\vec{x}_i^k, \vec{w}^k))$$

These constraints state that if in line i at depth k a concrete quantum gate is selected (i.e. $\vec{w}^k$ is completely assigned), the variables in the next depth $k+1$ are computed from the variables at depth k with the quantum gate function $q(\vec{x}_i^k, \vec{w}^k)$. The function $q(\vec{x}_i^k, \vec{w}^k)$ covers the functionality of a quantum gate represented by $\vec{w}^k$.

3. Illegal assignments to $\vec{w}^k$ are excluded in this constraint, since not all values of $\vec{w}^k$ are legal to enumerate all possible quantum gates:

$$\bigwedge_{k=0}^{d-1} [\vec{w}^k]_2 < g$$

We now have a Boolean formulation for the synthesis problem which uses multi-valued logic for encoding the quantum signals. It is well known that with the introduction of new variables the CNF form for any Boolean formula can be produced in time and space linear in the size of the original Boolean formula [21]. Therefore, we first define methods for the simple logic functions like *AND*, *OR*,

etc. that generate the corresponding clauses. Then, we extended this scheme for more complex logic. This allows a mapping of our formulation including the quantum gate function $q(\vec{x}_i^k, \vec{w}^k)$ to CNF. Furthermore, the assignments of the input/output constraints are applied by using unit clauses. Illegal assignments to $\vec{w}^k$ are expressed by explicitly enumerating all values that are not allowed in form of a blocking clause [15].

This results in a CNF representing the synthesis problem of a reversible function f with d quantum gates that can be given to a SAT solver. The instance is satisfiable iff there exists a network realization for f with exactly d gates. In this case, the resulting quantum gate circuit can be extracted from the assignments to $\vec{w}^k$ for each depth.

4. Incompletely Specified Functions

It is well known that many practical logic functions contain *don't care* conditions. That is, the output for certain input combinations is irrelevant. Furthermore, in order to make a function reversible, it is often necessary to add constant inputs and garbage outputs [11]. Garbage outputs are by definition don't cares. In our SAT based procedure the don't care outputs can be left unspecified (the reversibility is still guaranteed since only reversible gates are part of the solution). The constants have to be assigned with a fixed value.

However, to synthesize a quantum gate circuit with minimal costs for a function with constant inputs, both values 0 and 1 for each constant input have to be checked. For example, while a minimal quantum gate circuit is found for a half adder at depth 5 when the constant input of this function is set to 1, a cheaper network representation with costs 4 can be found if the constant is set to 0 (as shown later in the experiments). For synthesizing the function *Decod24* [8] we can chose the values for the constant inputs from $\{00, 01, 10, 11\}$, i.e. more than one constant input is needed and thus four independent runs of the synthesis algorithm.

In this section we present two improvements for the initial exact synthesis formulation of functions containing garbage outputs and constant inputs. First we show, how a SAT instance representing the synthesis problem can be reduced by removing constraints that only define garbage outputs. Then, a modification of the encoding is introduced, that includes all possible combinations of constant values and thus allows to find a minimal quantum gate circuit by solving only one instance.

4.1. Removal of Constrains

As explained in Section 3 constraints for each line i encode the input/output specification and the quantum gate functionality. The latter constraints ensure, that based on the selected gates (by assignments to $\vec{w}^k$ for each depth k) the inputs of each truth table line are mapped to the corresponding output. However, if the right side of a truth table line only consists of garbage outputs (i.e. don't cares), then all constraints created for this line can be removed.

Table 1. Embedding of function $f = a \cdot b$

c	a	b	f	g_1	g_2
0	0	0	0	-	-
0	0	1	0	-	-
0	1	0	0	-	-
0	1	1	1	-	-
1	0	0	-	-	-
1	0	1	-	-	-
1	1	0	-	-	-
1	1	1	-	-	-

Example 2 *Consider the function $f = a \cdot b$ that was transformed into a reversible one by adding a constant input and two garbage outputs as shown in Table 1. All outputs of the last four lines of the truth table are garbage outputs. Thus, the values of all variables in $\vec{x}_5^k, \vec{x}_6^k, \vec{x}_7^k$ and $\vec{x}_8^k$ (i.e. all variables representing input, temporary or output variables at the last four lines of the truth table at each depth k) are not relevant; the corresponding constraints can be omitted.*

By removing the useless constraints the number of clauses is significantly reduced. Thus, a speed-up of the overall synthesis time can be expected.

4.2. Constraining Constant Inputs

If constant inputs are used, different options exist on how to assign these constants. In Example 2 the constant input was assigned to 0. But the resulting quantum network is not necessarily minimal. A smaller network may be possible if the constant input is set to 1. Thus, both values for each constant have to be checked. For example as already mentioned for function *Decod24* we can chose the values for the constant inputs from $\{00, 01, 10, 11\}$. All these combinations are encoded as single instances and separately solved by the SAT solver. Our new approach uses an encoding that only needs one instance for each depth. Here, all variables in $\vec{x}_i^0$, that represent a constant input will not be initially set to 0 or 1 by the input/output constraints. Instead constraints are added, which ensure that all variables representing the same constant are equal. This way, the SAT solver determines the values for the constants. Since the considered constant inputs are now modeled symbolically (the value of each constant input is not fixed to 0 or 1) only 2^{n-c} truth table lines have to be considered (where c is the number of constants).

Example 3 *Again the function $f = a \cdot b$ of Example 2 is considered. Here, the first input is a constant. (In the SAT encoding x_{i1}^0 is represented by y_{i1}^0 and z_{i1}^0 for each truth table line i.) Instead of solving two SAT instances (the first with $x_{i1}^0 = 0$ and the second with $x_{i1}^0 = 1$) only one has to be solved just by not specifying the value of x_{i1}^0 and additionally constraining $x_{11}^0 = x_{21}^0 = \ldots = x_{2^21}^0$. Then, the SAT solver determines the value of the constant.*

In total the number of SAT instances can be reduced significantly and thus speed-ups in the overall synthesis time are achieved as shown in the experiments.

Table 2. Incompletely specified functions

	SAT Encoding		+Removal of Constr.		+Constraining Const. Inputs	
	D	Time	D	Time	D	Time
Half-adder						
1	4	1.47	4	0.76	4	0.85
2	5	2.65	5	2.17	–	–
total	4	4.12	4	2.93	4	0.85
Half-adder2						
1	4	1.28	4	0.63	4	0.76
2	5	3.53	5	1.41	–	–
total	4	4.81	4	2.04	4	0.76
Full-adder						
1	6	297.56	6	36.79	6	209.95
2	7	1123.55	7	2321.33	–	–
total	6	1421.11	6	2358.12	6	209.95
Low-High						
1	7	10636.95	7	10458.71	6	2180.66
2	6	1444.26	6	1323.42	–	–
total	6	12081.21	6	11782.13	6	2180.66
Zero-One-Two						
1	7	430.30	7	398.17	6	132.46
2	6	91.23	6	38.34	–	–
3	6	204.46	6	122.21	–	–
4	6	477.84	6	181.6	–	–
total	6	1203.83	6	740.32	6	132.46
Decod24						
1	8	2162.51	8	1010.02	8	5110.26
2	8	4391.96	8	2120.46	–	–
3	8	6158.05	8	2029.81	–	–
4	8	6063.92	8	2263.88	–	–
total	8	18776.44	8	7424.17	8	5110.26

5. Experimental Results

We implemented the presented synthesis algorithm in C++. As a SAT solver we use MiniSat v2 [6] including simplification [5]. Unless mentioned otherwise, experiments have been carried out on an AMD Athlon 3500+ with 1 GB of main memory. All benchmarks have been taken from [23].

5.1. Incompletely Specified Functions

We tested our approach for a set of benchmarks that are embedded in reversible functions. The results are shown in Table 2. For each function there are 2^c entries, where c is the number of constant inputs. Each line below the function name corresponds to one assignment of the constant inputs. Columns labeled D show the number of gates in the optimal circuit and columns labeled Time list the corresponding run-time in CPU seconds. Removal of constrains as introduced in Section 4.1 lead to better run-times. In the case of *Decod24*, the run-time is improved by more than a factor of two. Only for the full adder more run-time is required. This can be explained by a different decision order of the SAT solver due to the structure of the respective instance.

However, the combination of both improvements, i.e. removing useless constraints as well as constraining the constant inputs, offers a significant speed-up for all examples. The reduction of run-times is between 70% and 95%. Reductions are more substantial if at least one of the assignments has a solution with more gates than the optimal constant input assignment. This can be observed for all functions in Table 2 except *Decod24*. It should be noted that the constraining of constant input variables requires some computation time (i.e. run-times may be higher than those for solving the function with a fixed constant input assignment). However, this overhead is easily compensated by the fact that only one instance needs to be solved.

Table 3. Effect of double gates

Function	Elem. Gates		Qua. Gates	
	D	Time	D	Time
Completely specified functions				
3_17	10	1641.49	8	280.98
Miller	8	15.49	6	11.60
Fredkin	7	7.04	5	3.28
Peres	4	0.33	4	1.21
Toffoli	5	0.71	5	2.38
Peres-double	6	11.32	6	175.86
Toffoli-double	7	86.75	7	1121.68
graycode6	5	66.50	5	608.11
q4example	6	9.08	5	24.83
Incompletely specified functions				
Half-adder	5	0.40	4	0.85
Half-adder2	4	0.19	4	0.76
Full-adder	7	145.07	6	209.95
rd32	6	37.75	6	436.11
Low-High	7	2245.47	7	2180.60
Zero-One-Two	7	32.05	6	132.46
Decod24	9	5660.77	8	5110.26

5.2. Effect of Double Gates

Double gates (as shown in Figure 2) may not be available at unit cost in the target implementation [2]. In fact, most minimization methods (for example, [1, 14]) consider the cost of a double gate to be two, since they are composed of two elementary gates. To the best of our knowledge [9] describes the only minimization procedure that assigns a single cost unit to double gates. Hence, there are compelling reasons to consider minimizations that rely only on elementary gates.

In one experiment we ignored double gates. This reduces the number of possible gates at each level from $7n(n-1)+n$ to $3n(n-1)+n$ (according to Lemma 1 and 2). The results are summarized in Table 3. In the first column the name of the function is given. The next columns provide the depth and the run-time in CPU seconds for the elementary gates and the quantum gates (i.e. elementary gates and double gates), respectively.

In general, it is expected that more choices of possible gates at each level will increase the time to find a correct solution. This is clearly seen for the benchmark functions where the inclusion of double gates offers no advantage (i.e. both values for D are the same). For example, the run-time for *graycode6* increases by one order of magni-

Table 4. Comparison to exact synthesis in [9]

FUNCTION	RA [9] TIME	SAT (PIII) TIME	SAT (PIII) IMPR
Completely specified functions			
Miller	318.29	34.53	> 9.2
Fredkin	78.02	10.96	> 7.1
Peres	35.18	4.43	> 7.9
Toffoli	122.52	8.45	> 14.5
Incompletely specified functions			
Half-adder	6.77	2.99	> 2.3
Half-adder2	26.25	2.70	> 9.7
Full-adder	25200.00	551.92	> 45.7

tude when double gates are considered – even though the results are identical with respect to the costs. On the other hand for functions where the inclusion of double gates leads to smaller circuits (e.g. *3_17*), the run-time can be reduced since fewer instances have to be solved.

5.3. Comparison with Previous Work

To compare our approach with the results obtained in [9] we used a 733 MHz Pentium III with 512 MB of main memory that is significantly slower than a 850 MHz Pentium III processor, the system used there. The outcome is shown in Table 4. RA denotes the run-times of the reachability analysis obtained from [9] and SAT the run-times of our approach in CPU seconds (using both elementary and double gates). IMPR gives the run-time improvement, i.e. the run-time of RA divided by the run-time of SAT.

We were able to synthesize all functions from [9] that were considered there for exact synthesis in significant shorter run-time – even on a slower processor. Improvements of at least a factor of 2 have been achieved. In the best case an improvement of a factor of 45 is observed.

Furthermore, we compared the synthesized results for the functions *q4-example*, *Peres-double* and *Toffoli-double*. Here, the authors of [9] constrained the search space for these functions. In fact they restrict the target-line of the V and V+ gates to be the forth line. Therefore they cannot guarantee optimal solutions.

The comparison in Table 5 shows, that our approach is able to find the optimal results for these functions with a low run-time increase (the results of the heuristic approach of [9] are denoted by RA_{heu}). Thereby, we prove that for *Peres-double* and *Toffoli-double* the minimal quantum gate networks have been found in [9]. Additionally, in case of *q4-example* our approach synthesizes an optimal quantum gate representation with costs 5 instead of the sub-optimal circuit of size 6 obtained in [9]. Note, again all these benchmarks are carried out on a slower system than the one used in [9]. For absolute run-times on a fast computer see the rightmost column of Table 3.

Table 5. Comparison to heur. synthesis in [9]

FUNCTION	RA_{heu} [9] D	RA_{heu} [9] TIME	SAT (PIII) TIME	SAT (PIII) D
Peres-double	6	171.27	481.71	6
Toffoli-double	7	853.78	2985.88	7
q4-example	6	34.78	78.09	5

6. Conclusions

In this paper we presented an exact synthesis algorithm that finds the minimal realization for a given reversible function. We have demonstrated our synthesis algorithm on a set of benchmarks. The resulting circuits consist of elementary quantum gates. We presented significant improvements of our approach for incompletely specified functions. Furthermore, the algorithm can be modified to accommodate different gate libraries. We have shown this by including/excluding double gates. As in case of optimal logic synthesis of Boolean functions [4], large libraries have a negative effect on the run-time of the synthesis procedure, but may lead to smaller circuits. In comparison to the previous exact synthesis method, in the best case we synthesized the functions up to a factor of 45 faster.

References

[1] A. Abdollahi and M. Pedram. Analysis and synthesis of quantum circuits by using quantum decision diagrams. In *Design, Automation and Test in Europe*, pages 317–322, 2006.

[2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52(5):3457–3467, 1995.

[3] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

[4] R. Drechsler and W. Günther. *Towards One-Path Synthesis*. Kluwer Academic Publishers, 2002.

[5] N. Eén and A. Biere. Effective preprocessing in SAT through variable and clause elimination. In *SAT*, pages 61–75, 2005.

[6] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT*, volume 2919 of *LNCS*, pages 502–518, 2004.

[7] D. Große, X. Chen, G. W. Dueck, and R. Drechsler. Exact SAT-based Toffoli network synthesis. In *Great Lakes Symp. VLSI*, pages 96–101, 2007.

[8] P. Gupta, A. Agrawal, and N. Jha. An algorithm for synthesis of reversible logic circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 25(11):2317–2330, 2006.

[9] W. N. N. Hung, X. Song, G. Yang, J. Yang, and M. A. Perkowski. Optimal synthesis of multiple output boolean functions using a set of quantum gates by symbolic reachability analysis. *IEEE Trans. on CAD*, 25(9):1652–1663, 2006.

[10] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[11] D. Maslov and G. W. Dueck. Reversible cascades with minimal garbage. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 23(11):1497–1509, 2004.

[12] D. Maslov, G. W. Dueck, and D. M. Miller. Toffoli network synthesis with templates. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 24(6):807–817, 2005.

[13] D. Maslov and D. M. Miller. Comparison of the Cost Metrics for Reversible and Quantum Logic Synthesis. *ArXiv Quantum Physics e-prints*, 2005.

[14] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck. Quantum circuit simplification using templates. In *Design, Automation and Test in Europe*, pages 1208–1213, 2005.

[15] K. L. McMillan. Applying SAT methods in unbounded symbolic model checking. In *Computer Aided Verification*, volume 2404 of *LNCS*, pages 250–264, 2002.

[16] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

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

[18] V. V. Shende, A. K. Prasad, I. L. Markov, and J. P. Hayes. Synthesis of reversible logic circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 22(6):710–722, 2003.

[19] J. A. Smolin and D. P. DiVincenzo. Five two-bit quantum gates are sufficient to implement the quantum fredkin gate. *Phys. Rev. A*, 53(4):2855–2856, 1996.

[20] T. Toffoli. Reversible computing. In *ICALP*, pages 632–644, 1980.

[21] G. S. Tseytin. On the complexity of derivation in propositional calculus. In A. O. Slisenko, editor, *Studies in Constructive Mathematics and Mathematical Logic II*, volume 8 of *Seminars in Mathematics: Steklov Mathem. Inst.*, pages 115–125. 1970.

[22] R. Wille and D. Große. Fast exact Toffoli network synthesis of reversible logic. In *Int'l Conf. on CAD*, pages 60–64, 2007.

[23] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: an online resource for reversible functions and reversible circuits. In *Int'l Symp. on Multi-Valued Logic*, 2008. RebLiv is available at http://www.revlib.org.

[24] R. Wille, H. M. Le, G. W. Dueck, and D. Große. Quantified synthesis of reversible logic. In *Design, Automation and Test in Europe*, 2008.