RevLib: An Online Resource for Reversible Functions and Reversible Circuits

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

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

Abstract

Synthesis of reversible logic has become an active research area in the last years. But many proposed algorithms are evaluated with a small set of benchmarks only. Furthermore, results are often documented only in terms of gate counts or quantum costs, rather than presenting the specific circuit. In this paper RevLib *(*`www.revlib.org`*) is introduced, an online resource for reversible functions and reversible circuits. RevLib provides a large database of functions with respective circuit realizations. RevLib is designed to ease the evaluation of new methods and facilitate the comparison of results. In addition, tools are introduced to support researchers in evaluating their algorithms and documenting their results.*

1 Introduction

The number of elements integrated within digital circuits grows exponentially, leading to enormous challenges in *Computer Aided Design* (CAD). Due to this exponential growth physical boundaries will be reached in the near future. Furthermore, power consumption of circuits becomes a major issue. To face these problems the research in the area of reversible logic and its applications in low-power design and quantum computing has become an intensely studied topic.

In the past many researchers have focused on synthesis of reversible logic (e.g. [16, 13, 1, 10, 8, 18, 19, 7]) using the different gate libraries including (multiple control) Toffoli gates [17], Fredkin gates [5], Peres gates [14], and elementary quantum gates [3]. Several synthesis methods – heuristic as well as exact ones – have been proposed.

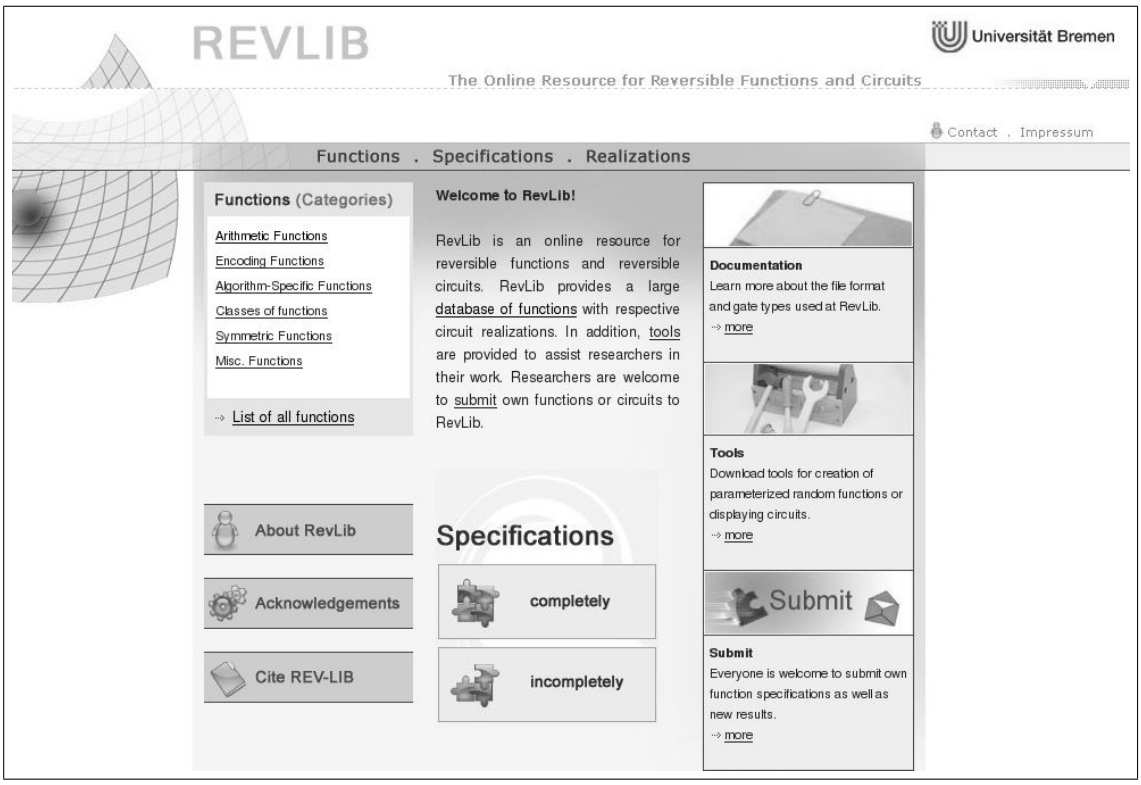
However, most of these approaches are evaluated with limited sets of benchmarks. Due to page limitations many synthesis results are presented only by listing the respective costs but not the concrete circuit realization. This becomes a significant issue, since some authors use their own cost metrics, which makes it hard to decide which realization is better than others. Thus, a complementary comparison of different algorithms with respect to a large set of benchmarks is often not possible.

In this paper *RevLib* (`www.revlib.org`) is introduced, an online resource for reversible functions and reversible circuits. The motivation behind RevLib is to ease empirical studies in order to improve the evaluation of new approaches by providing an easy access to a large and complementary benchmark database. The benefits of widely used benchmark collections have already been seen e.g. in the circuit domain with ISCAS benchmarks [4], in the area of Boolean satisfiability by SATLIB [9] and by TSPLIB [15] for the traveling salesman problem.

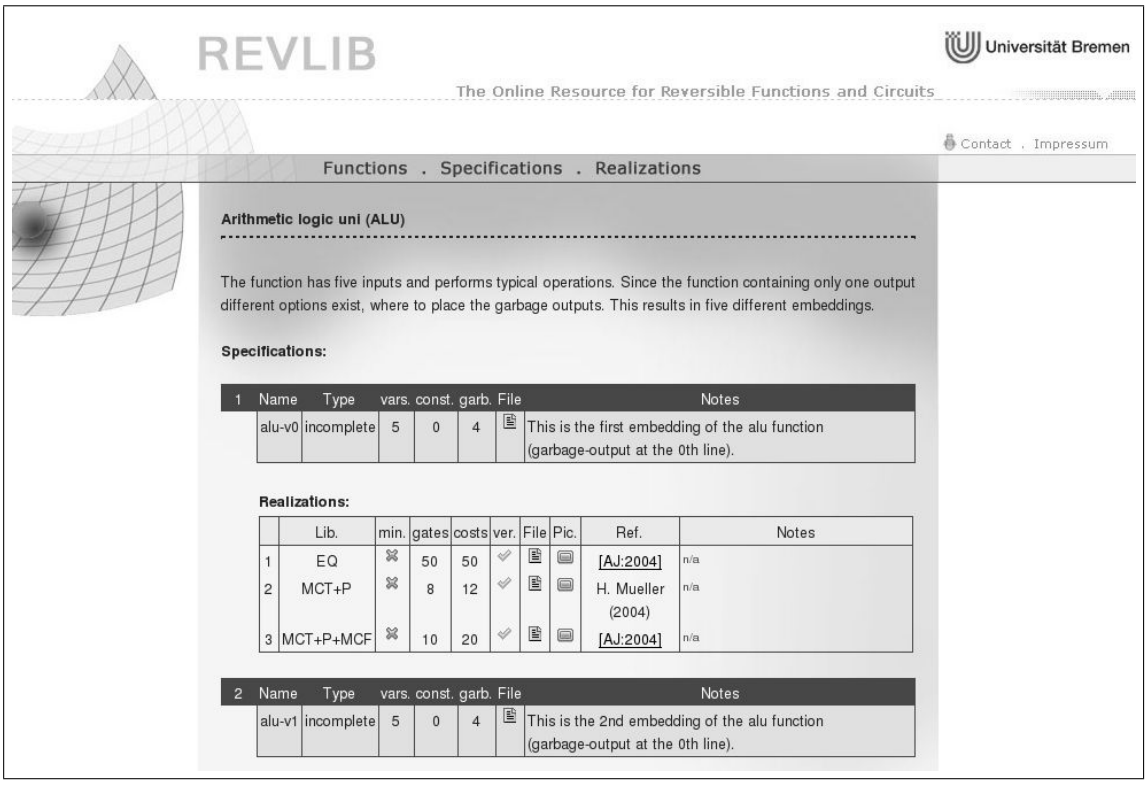
RevLib provides a benchmark suite of reversible functions. For irreversible functions several embeddings (containing constant inputs and gabarge outputs [11]) are available[1]. The benchmark suite contains a variety of different types of problem instances. Furthermore, for each function specification at least one reversible circuit realization is given using different gate libraries. Thereby, reversible circuits obtained from exact approaches as well as heuristic approaches are provided. This allows to compare (new) algorithms with respect to the achieved quality. Figure 1 provides some screenshots of RevLib.

To fill the database any researcher can submit functions and/or circuit realizations to RevLib. Since no widely accepted format for the specification of reversible functions and reversible circuits exists, a standardized file format is proposed in this paper. Additionally, several tools are provided supporting researchers in evaluating their approaches

[1]Pure reversible functions are also called completely specified functions, while embedded irreversible function are called incompletely specified functions. Both, completely specified and incompletely specified functions will be defined in so called *specifications*.



(a) Frontpage



(b) Function page with different specifications and realizations

Figure 1. Screenshots from RevLib

and documenting their results, i.e. a *Benchmark Generator* for creating random reversible functions with user-defined properties and a *Quantum Viewer* (Quiver) for automatic visualization of reversible circuits given in the proposed format.

The rest of the paper is structured as follows. Section 2 describes the benchmark suite provided by RevLib. In Section 3 the standardized file formats are specified; Section 4 introduces the tools currently available at RevLib. Some examples illustrating the advantages of a large benchmark database are discussed in Section 5. Finally, Section 6 concludes the paper.

2 Benchmark Suite

Generally, benchmark collections should provide a large variety of problem instances allowing evaluation of different types of algorithms. The benchmark suite at RevLib offers several functions from different domains, such as arithmetic functions (e.g. adders, divisibility checkers, comparators, *etc.*), encoding functions (e.g. hamming code, graycode, *etc.*), classes of functions (e.g. symmetric functions, linear functions, *etc.*) and algorithm-specific functions (e.g. worst-case functions for heuristic approaches, hidden weighted bit function, *etc.*).

In the absence of suitable "real life" functions, it is desirable to generate pseudo random function. In Section 4.1 a tool for generating randomized benchmarks with user-defined properties is introduced. This tool has the capability to reproduce the same set of functions, given the same parameters. Therefore, only the parameters need to be reported, not the specifications for the functions.

3 File Formats

This section describes the standardized file format used at RevLib for the specification of reversible functions and circuits, respectively. Both formats will be defined in ASCII files consisting of two parts: the header and the specification.

3.1 Header

The header contains information about the characteristics of the instance, i.e.:

- *Version:* In order to support further enhancements, any changes will be associated with a version number. The version of a respective file is given by the line starting with `.version`.
- *Comments:* Comments provide human-readable information about the respective instance, e.g. a short description, authors, or similar. A comment line starts with a # character. All comment lines will be ignored by programs.
- *Number of Variables:* The line starting with `.numvars` signifies the number n of variables (lines) of the respective function (circuit). The number has to be a positive integer.
- *Variables:* The line starting with `.variables` signifies the ordered list of identifiers for the respective variables (lines). An identifier has to be a sequence of letters, digits and underscores. In total, n different identifiers have to be defined in a `.variables` line (separated by spaces).
- *Inputs/Outputs:* The line starting with `.inputs` (`.outputs`) signifies the ordered list of names of the respective inputs (outputs). A name has to be a sequence of letters, digits and underscores. In total, n different names have to be defined in a `.inputs` (`.outputs`) line (separated by spaces). Defining the inputs (outputs) is optional. By default the names of inputs (outputs) is equal to the respective identifier of the variable.
- *Constant Inputs:* Constant inputs can be definied by a line starting with `.constants` followed by a string containing the values for the input constants ('`1`' for constant one, '`0`' for constant zero and '`-`' if the respective input line is not constant) in the order they appear without any spaces. Defining constant inputs is optional. By default all inputs are not constants (i.e. for each input the value is '`-`').
- *Garbage Outputs:* Garbage outputs can be definied by a line starting with `.garbage` followed by a string indicating whether an output is garbage or not ('`1`' if the output is garbage and '`-`' if the output is not garbage) in the order they appear without any spaces. Defining garbage outputs is optional. By default all outputs are not garbage (i.e. for each output the value is '`-`').

Following the header either the function or the circuit have to be specified. Both specifications start with a line containing the string `.begin` and end with a line containing the string `.end`.

3.2 Function Specification

A reversible function is specified by all its truth table entries. The ith line ($0 \leq i < 2^n$) of the specification gives the respective output values (`1` for one, `0` for zero and – for don't care) of the ith line of the truth table by a string in the order they appear without any spaces.

Example 1 *Figure 2 (a) shows a reversible function including one constant input as well as garbage outputs and don't*

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

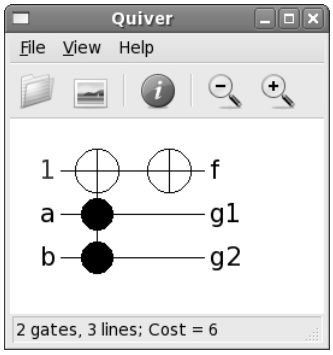
(a) Function

```
1  # Embedded AND function
2  .version 1.0
3  .numvars 3
4  .variables x y z
5  .inputs 1 a b
6  .outputs f g1 g2
7  .constants 1--
8  .garbage -11
9  .begin
10  ---
11 ---
12 ---
13 ---
14 ---
15 0--
16 0--
17 0--
18 1--
18 .end
```

(b) Function format

```
1  # AND circuit
2  .version 1.0
3  .numvars 3
4  .variables x y z
5  .inputs 1 a b
6  .outputs f g1 g2
7  .constants 1--
8  .garbage -11
9  .begin
10  t3 y z x
11 t1 x
12 .end
```

(c) Circuit format



(d) Circuit

Figure 2. File format example

care outputs given as truth table. The respective specification of this function in the proposed format is shown in Figure 2 (b).

3.3 Circuit Specification

A reversible circuit is specified by all its gates. The specification lists the gates in the order they appear in the design. In the current version we distinguish five different gates, whose specifications are given in Table 3.

Example 2 *Figure 2 (d) shows a Toffoli gate network realization of the function considered in Example 1 (drawn by Quiver introduced in Section 4.2). The respective specification of this circuit in the proposed format is shown in Figure 2 (c).*

4 Tools

In order to support researchers in evaluating their algorithms and documenting their results RevLib currently provides two tools: the *Benchmark Generator* and the *Quantum Viewer* (*Quiver*).

4.1 Benchmark Generator

The *Benchmark Generator* automatically creates random reversible functions with respect to the following parameters, which have to be given as input:

- the number of variables in the resulting function
- the number of constant inputs in the resulting function
- the number of garbage outputs
- the percentages of the number of don't care outputs in the resulting function

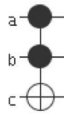
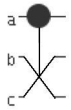
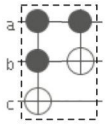
Gate	Specification
Toffoli 	A (multiple control) Toffoli gate is signified by the character `t` and an integer indicating the size of the gate followed by a list of identifiers for the respective lines such that the target line is at the end of the list. Note that Toffoli gates include (controlled) NOT gates as well. Example: `t3 a b c`
Fredkin 	A (multiple control) Fredkin gate is signified by the character `f` and an integer indicating the size of the gate followed by a list of identifiers for the respective lines such that the targets of the gates are at the end of the list. Example: `f3 a b c`
Peres 	A Peres gate is signified by the character `p` and an integer indicating the size of the gate followed by a list of identifiers for the respective lines such that the targets of the gates are at the end of the list. Example: `p3 a b c`
V gate 	A V gate is signified by the character `v` and an integer indicating the size of the gate (i.e. 2) followed by a list of identifiers for the respective lines such that the target line is at the end of the list. Example: `v a b`
V+ gate 	A V+ gate is signified by the characters `v+` and an integer indicating the size of the gate (i.e. 2) followed by a list of identifiers for the respective lines such that the target line is at the end of the list. Example: `v+ a b`

Figure 3. Specification of gates

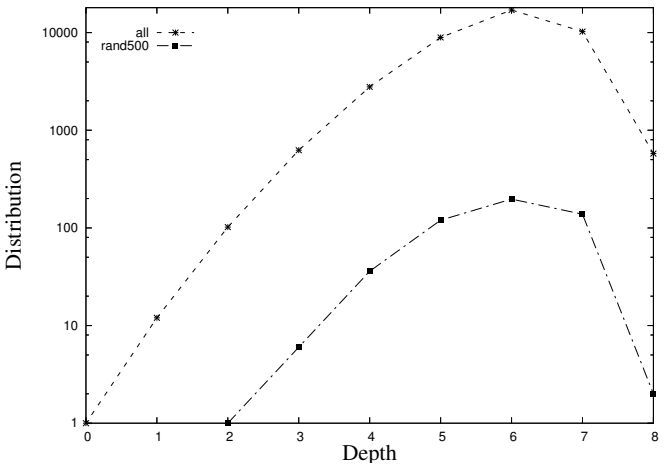


Figure 4. Distribution of random functions

- the number of instances to be generated

The Benchmark Generator is implemented in C++ and executable versions for Linux, Windows, and Mac OS are available. The functions generated with this tool are representative with respect to all functions with the same properties. This was verified by a comparison of all completely specified functions containing 3 variables (40320 in total) with 500 random generated functions applying the same properties. The result is shown in Figure 4. Here, the distribution of the functions with respect to the number of gates in the resulting minimal Toffoli networks is compared. As can be seen from the diagram the randomly generated functions show the same distribution as the complete set of 3-variable functions does.

4.2 Quantum Viewer

The *Quantum Viewer* (*Quiver*) is an application to aid the visualization of reversible quantum circuits. It displays an image of a circuit (given in the proposed specification) and saves this image as a bitmap file. A screenshot of Quiver is given in Figure 2 (d). Currently Toffoli, Fredkin, Peres, V, and V+ gates are supported. Further gate types (or sequences of gates listed above) can be added easily.

Features of the *Quiver* are:

- displaying the gate count
- calculating the quantum cost of the circuit based on the decomposition proposed in [2]
- viewing the output of a circuit in truth-table form
- verifying the output of a circuit against a reversible function specification or a function specification in pla

Quiver is implemented in C++ and executable version for Linux, Windows, and Mac OS are available.

Table 1. Gate counts of synthesis approaches

BENCHMARK	HEUR		EXACT	
	D	SRC.	D	SRC.
alu-v0	18	[8]	7	[6]
decod24-v0	11	[8]	6	[6]
3_17	6	[8]	6	[6]
hwb4	17	[12]	11	[6]
4_49	16	[8]	12	[6]

5 Discussion

In this section some examples are given, which illustrate why a collection of reversible functions and there respective realizations is useful for research in the area of reversible logic. Thereby, we only show the general idea with a small set of benchmarks.

First, the results of several approaches – heuristic as well as exact ones – for synthesis of networks containing multiple control Toffoli gates [12, 8, 6, 18] are considered. Some outcomes are given in Table 1. HEUR denotes the number of gates a network, obtained by heuristic approaches, contains while EXACT denotes the respective number for the exact realization. The sources of the results are given in column SRC.

As expected, the heuristic approaches are not able to determine the exact network for a given function in each case. However, the direct comparison to the minimal realization allows a better evaluation of the algorithms. For example, the function *3_17* is handled well by the heuristic approaches. In contrast the network obtained for *alu* is more than a factor of two larger than the minimal one.

Another point is that – using RevLib – the concrete networks generated by the approaches are available (this is often not possible in the respective publications due to page limitations). Analyzing these networks may lead to further reductions. As an example Figure 5 shows a circuit representing the function *alu* obtained by the heuristic approach proposed in [8]. Here, only output f is of interest since all remaining ones are garbage outputs. If the concrete realization is available an easy reduction is possible, because the last eleven Toffoli gates (marked by a dashed rectangle) only influence the garbage outputs and thus can be removed.

Finally, the effect of applying different gate libraries is considered. Some result are shown in Table 2. Column T denotes the gate count of the exact networks containing multiple control Toffoli networks only. The next columns list the gate counts when multiple control Toffoli and Fredkin gates (T+F), multiple control Toffoli and Peres gates (T+P) and multiple control Toffoli, Fredkin and Peres gates (T+F+P) are used, respectively. Obviously, the use of different libraries results in different minimal circuits. With

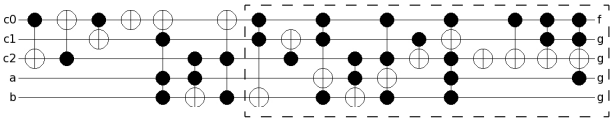


Figure 5. Realization of alu

the help of a large set of realizations – as RevLib provides – such effects can be observed more easily.

6 Conclusion

In this paper we introduced RevLib (`www.revlib.org`), an online resource for reversible functions and reversible circuits. RevLib provides a wide range of reversible functions and their realizations. Therefore, a standardized format has been introduced for specifying the respective functions and circuits. Furthermore, tools have been proposed helping researchers in evaluating their approaches and documenting their results. Some of the benefits of RevLib have been discussed. Researchers are welcome to submit their own functions and circuits to RevLib.

Acknowledgements

We would like to thank Dmitri Maslov for his suggestions. Furthermore, we would like to thank Nathan Scott for his help in implementing the Quiver.

This work was supported by the German Academic Exchange Service (DAAD), which enabled the close contact between the authors.

References

[1] A. Agrawal and N. K. Jha. Synthesis of reversible logic. In *Design, Automation and Test in Europe*, pages 1384–1385, 2004.

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

[3] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVinchenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. Elementary gates for quantum computation. *APS Physical Review A*, 52:3457–3467, 1995.

[4] F. Brglez and R. Drechsler. Design of experiments in CAD: Context and new data sets for ISCAS'99. In *Int'l Symp. Circ. and Systems*, volume VI, pages 424–427, 1999.

[5] E. F. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3/4):219–253, 1982.

Table 2. Gate counts for different libraries

Benchmark	T	T+F	T+P	T+F+P
3_17	6	5	5	5
mod5d2	8	8	6	6
hwb4	11	9	8	8
rd32-v0	4	4	2	2
alu-v0	6	4	6	4

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

[7] D. Große, R. Wille, G. W. Dueck, and R. Drechsler. Exact synthesis of elementary quantum gate circuits for reversible functions with don't cares. In *Int'l Symp. on Multi-Valued Logic*, 2008.

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

[9] H. H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In *I. P. Gent, H. v. Maaren, T. Walsh, editors, SAT 2000*, pages 283–292, 2000. SATLIB is available online at www.satlib.org.

[10] W. Hung, X. Song, G. Yang, J. Yang, and M. 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.

[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*, 24(6):807–817, 2005.

[13] D. M. Miller, D. Maslov, and G. W. Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Automation Conf.*, pages 318–323, 2003.

[14] A. Peres. Reversible logic and quantum computers. *APS Physical Review A*, (32):3266–3276, 1985.

[15] G. Reinelt. TSPLIB - A Traveling Salesman Problem Library. In *ORSA Journal on Computing*, volume 3, pages 376–384, 1991. TSPLIB is available online at www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/.

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

[17] T. Toffoli. Reversible computing. In W. de Bakker and J. van Leeuwen, editors, *Automata, Languages and Programming*, page 632. Springer, 1980. Technical Memo MIT/LCS/TM-151, MIT Lab. for Comput. Sci.

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

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