QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits

D. Michael Miller
Dept. of Computer Science
University of Victoria
Victoria, BC, Canada
mmiller@cs.uvic.ca

Mitchell A. Thornton
Dept. of Computer Science and Engineering
Southern Methodist University
Dallas, TX, USA
mitch@engr.smu.edu

Abstract

In this paper, we present a novel structure, Quantum Multiple-valued Decision Diagrams (QMDD), specifically designed to represent and manipulate the matrices encountered in the specification of reversible and quantum gates and circuits, both binary and multiple-valued. QMDD use many common decision diagram techniques, ideas introduced in QuIDDPro and novel techniques introduced here. Building the QMDD for the matrices for individual gates and the subsequent construction of the QMDD for the matrix describing a circuit are discussed. A prototype C implementation is described and experimental results are given that show the new structure is a promising and compact representation for reversible and quantum logic circuits.

1. Introduction

The functional behaviour of a primitive reversible (including quantum) gate can be represented as a transformation matrix and the functional behaviour of a circuit(cascade) composed of such gates is given by the product of those matrices. The matrices are complex-valued for quantum gates. For an r-valued circuit with n lines, each matrix has dimension $r^n \times r^n$ so in order to handle large circuits it is necessary to develop efficient methods for the storage and manipulation of the required matrices.

The use of decision diagrams for storing and manipulating matrices has been previously considered [3, 5]. Recently, Viamontes, Markov and Hayes [12, 13, 14] have presented the QuIDDPro package for handling the matrices encountered in specifying and simulating binary quantum circuits. QuIDDPro uses CUDD, a robust and highly optimized decision diagram package developed by Somenzi [10]. In particular, QuIDDPro uses algebraic decision diagrams [1] as implemented in CUDD, with some extensions. A major advantage of this approach is it makes use of the extensive optimization in CUDD.

In this work, we introduce an alternative approach which we call Quantum Multiple-valued Decision Diagrams (QMDD). The goal in developing the QMDD representation and package is to effectively represent and manipulate the complex matrices encountered for binary and multiple-valued reversible / quantum circuits.

2. Reversible and Quantum Circuits

Definition 1. *An n-input, n-output, (written $n \times n$) totally-specified MVL function is reversible if it maps each input assignment to a unique output assignment.*

A reversible function defines a permutation of the input patterns and there are thus $r^n!$ r-valued, $n \times n$ reversible functions. Reversible functions are realized by circuits composed of reversible gates.

Definition 2. *An m-input, m-output gate is reversible if it realizes a reversible function.*

A variety of binary reversible gates have been considered. The family of Toffoli gates [11] is defined as follows:

Definition 3. *An $m \times m$ Toffoli gate passes the first $m-1$ lines (controls) through unchanged, and inverts the m^{th} line (target) if the control lines are all 1.*

The 2×2 Toffoli gate and the 3×3 Toffoli gate have been named the *controlled-NOT* and *controlled-controlled-NOT* gates, respectively. A Toffoli gate with no controls is a *NOT* gate.

De Vos *et al.* [4] have considered the C and N gates in Table 1 and the controlled versions of those gates, denoted CC (see Table 2) and CN as generators of the group of all 2×2 3-valued reversible logic functions.

x	C	C2	N	D	E
0	1	2	2	1	0
1	2	0	1	0	2
2	0	1	0	2	1

Table 1. 3-valued permutation operators

x	y	x	$CC(x,y)$
0	0	0	0
0	1	0	1
0	2	0	2
1	0	1	0
1	1	1	1
1	2	1	2
2	0	2	1
2	1	2	2
2	2	2	0

Table 2. 3-valued controlled-cycle

Extensions to the above (including C2, D and E in Table 1) and synthesis using these gates were considered in [8]. Negation extends to any r-valued logic as $x' = (r-1) - x$. There are $r-1$ cycle inversions for r-valued logic defined in the obvious way. Controlled cycle is extended to the $m \times m$ case for r-valued logic as follows:

Definition 4. *An $m \times m$ r-valued controlled cycle gate passes the first $m-1$ lines (control) through unchanged, and cycles the m^{th} line (target) by a specified amount (1 to $r-1$) if the control lines each assume their specified active value $0...r-1$.*

Controlled-cycle is sufficient for illustrative purposes in this paper. The application of the methods described below to the many other possible multiple-valued gates is straightforward.

Many quantum gates have been defined and studied in the literature [9]. We here restrict our attention to quantum circuits composed of:

- NOT gates;
- Controlled-NOT gates;
- The 2-line controlled-V gate that changes the target line using the transformation given by the matrix $\mathbf{V} = \frac{1+i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$ if the control line has the value 1;
- The 2-line controlled-V^+ that changes the target line using the transformation $\mathbf{V}^+ = \mathbf{V}^{-1} = \frac{1-i}{2}\begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix}$ if the control line has the value 1.

Gates V and V^+ are often referred to as *controlled-square-root-of*-NOT gates since $V^2 = (V^+)^2 = NOT$.

Once again, it is important to note that while we only use a small set of gates for illustration in this paper, the QMDD methods are directly applicable to other common quantum gates such as the Hadamard gate, Pauli gates and rotation gates [9], and many that may be conceived of for multiple-valued quantum logic. The only requirement is that the behaviour of the gate be describable as a matrix in the manner outlined in the next Section.

3. Matrix Representation

The common feature of all the gates noted above (and other reversible and quantum gates) is that the transformation performed by a gate, and the transformation performed by a cascade of gates, can be represented as a matrix of dimension $r^n \times r^n$ for an n-line circuit and r-valued logic. For example for $n = 3$ and $r = 2$, the Toffoli gate $T(x_0, x_2; x_1)$, where x_0 and x_2 are control variables and x_1 is the target, has the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (1)$$

Note that the rows and columns of this matrix are identified by 3-bit indices in natural order corresponding to $x_2x_1x_0$.

The controlled-cycle gate defined in Table 2 ($n = 2, r = 3$) has the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad (2)$$

In this case, the ordering is as indicated above with both the variables and indices 3-valued.

The controlled-V gate $V(x2; x1)$, where x_2 is the control and x_1 is the target, with $n = 3, r = 2$ has the transformation matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1+i}{2} & 0 & \frac{1-i}{2} \\ 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1-i}{2} & 0 & \frac{1+i}{2} \end{bmatrix} \quad (3)$$

The row and column indices and variable ordering are the same as for the matrix in (1). This matrix has complex-valued entries and is not a permutation matrix.

4. Quantum Multiple-valued Decision Diagrams

The representation of a matrix by a decision diagrams is discussed in [3, 5]. The QMDD structure presented below extends the earlier approach.

Definition 5. *A* **QMDD** *is a directed acyclic graph with the following properties:*

1. *There is a single* **terminal vertex** *with associated value 1. The terminal vertex has no outgoing edges.*
2. *There are some number of* **non-terminal vertices** *each labeled by an r^2-valued selection variable. Each nonterminal vertex has r^2 outgoing edges designated $e_0, e_1, ..., e_{r^2-1}$.*
3. *One vertex is the* **start vertex** *and has a single incoming edge that itself has no source vertex.*
4. *Every edge in the QMDD (including the one leading to the start vertex) has an associated complex-valued* **weight**.
5. *The selection variables are* **ordered**, *assume with no loss of generality $x_0 \prec x_1 \prec ... \prec x_{n-1}$, and the QMDD satisfies the following two rules:*
   - *Each selection variable appears at most once on each path from the start vertex to the terminal vertex.*
   - *An edge from a nonterminal vertex labeled x_i points to a nonterminal vertex labeled $x_j, j < i$ or to the terminal vertex. Hence x_0 is closest to the terminal vertex and x_k for the highest k present labels the start vertex.*
6. *No nonterminal vertex is* **redundant**, i.e. *no nonterminal vertex has its r^2 outgoing edges all with the same weight and pointing to a common vertex.*
7. *Each nonterminal node is* **normalized** *such that $e_j, 0 \leq j \leq r^2 - 1$ has weight 1 and $\forall e_i, i < j$ have weight 0. Note, such an e_j must exist or the vertex would be redundant (all weights 0).*
8. *Nonterminal vertices are* **unique**, i.e. *no two nonterminal vertices labeled by the same x_i can have the same set of outgoing edges (destinations and weights).*

Note that the start vertex is a nonterminal vertex except in the case where the QMDD has no nonterminal vertices in which case the QMDD consists of just the terminal vertex which is also the start vertex. This extreme case corresponds to a constant valued matrix.

Figure 1 illustrates the QMDD representation for a quantum V gate. The 0,1,2,3 edges from each vertex are from left to right. The weights are shown next to each line.

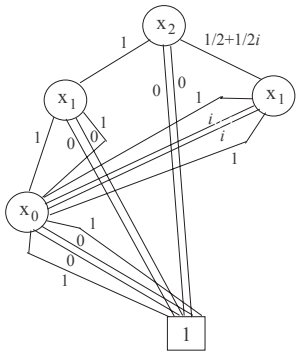

Figure 1. QMDD for $V(x_2; x_1)$ with $n = 3$ (see eqn. 3).

Definition 6. *To* **evaluate a QMDD** *for a particular selection variable assignment, find the path from the start vertex to the terminal vertex (including the edge leading to the start vertex) determined by the assignment. In particular, from a vertex labeled x_i follow edge e_j where j is the value assigned to x_i. Note that some variables may not appear on the designated path. The value associated with the selection variable assignment is the product of the weights on the edges in the corresponding path.*

Theorem 1. *An $r^n \times r^n$ complex valued matrix M has a unique (up to variable reordering or relabeling) QMDD representation.*

Proof: The proof is omitted due to space limitations and is available from the authors.

Three matrix operations are required and we here describe how they are computed directly on QMDD. Given an edge e, we use: $w(e)$ to denote the weight on e; $v(e)$ to denote the vertex e points to; $x(e)$ to denote the variable that labels the vertex e points to (not defined for the terminal vertex); $E_i(e)$ to denote the i^{th} edge out of the vertex that e points to; and $T(e)$ to denote a Boolean test that is true if e points to the terminal vertex. The variables adhere to the same order in all QMDD and we shall use $\prec$ to denote the fact one variable precedes another and hence appears closer to the terminal vertex in the QMDD. r is the radix of the logic system being considered so every nonterminal vertex has r^2 outgoing edges.

Matrix Addition: Let e_0 and e_1 be two edges pointing to two QMDD (matrices) to be added. The procedure is recursive and involves the following steps:

1. If $T(e_1)$, swap e_0 and e_1.
2. If $T(e_0)$,
   (a) if $w(e_0) = 0$, the result is e_1;
   (b) if $T(e_1)$, the result is an edge pointing to the terminal vertex with weight $w(e_0) + w(e_1)$.
3. If $T(e_0)$ or $x(e_0) \prec x(e_1)$, swap e_0 and e_1.
4. For $i = 0, 1, ..., r^2 - 1$
   (a) Create an edge p pointing to $v(E_i(e_0))$ with weight $w(e_0) \times w(E_i(e_0))$.
   (b) If $x(e_0) = x(e_1)$, create an edge q pointing to $v(E_i(e_1))$ with weight $w(e_1) \times w(E_i(e_1))$, else set $q = y$.
   (c) Recursively invoke this procedure to add p and q giving z_i.
5. The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, ...r^2 - 1$. This vertex and the edge pointing to it are normalized as described above.

Matrix Multiplication: Let e_0 and e_1 be two edges pointing to two QMDD (matrices) to be multiplied. The procedure is recursive and involves the following steps:

1. If $T(e_1)$, swap e_0 and e_1.
2. If $T(e_0)$ then
   (a) if $w(e_0) = 0$, the result is e_0;
   (b) if $w(e_0) = 1$, the result is e_1;
   (c) otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \times w(e_1)$.
3. If $x(e_0) \prec x(e_1)$, swap e_0 and e_1.
4. For $i = 0, r, 2r, ..., (r-1)r$
   For $j = 0, 1, ..., r - 1$
   Set z_{i+j} to be an edge with weight 0 pointing to the terminal vertex.
   For $k = 0, 1, ..., r - 1$
   (i) Create an edge p pointing to $v(E_{i+k}(e_0))$ with weight $w(e_0) \times w(E_{i+k}(e_0))$.
   (ii) If $x(e_0) = x(e_1)$, create an edge q pointing to $v(E_{j+r\times k}(e_1))$ with weight $w(e_1) \times w(E_{j+r\times k}(e_1))$, else set $q = y$.
   (iii) Recursively invoke this procedure to multiply the QMDD pointed to by p and q and then use the procedure above to add the result to the QMDD pointed to by z_{i+j}. The result of the addition replaces z_{i+j}.
5. The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, ...r^2 - 1$. This vertex and the edge pointing to it are normalized as described above.

Kronecker Product: Let e_0 and e_1 be two edges pointing to two QMDD (matrices) for which we want to compute the Kronecker product $A \bigotimes B$ (note that this operation is not commutative). For the application considered here, the selection variables for B precede the selection variables for A. This greatly reduces the complexity of the algorithm for computing the Kronecker product of two QMDD. The procedure is recursive and involves the following steps:

1. If $T(e_0)$ then
   (a) if $w(e_0) = 0$, the result is e_0;
   (b) if $w(e_0) = 1$, the result is e_1;
   (c) otherwise, the result is an edge pointing to $v(e_1)$ with weight $w(e_0) \times w(e_1)$.
2. For $i = 0, 1, ..., r^2 - 1$
   Recursively invoke this procedure to find the Kronecker product of $E_i(e_0)$ and e_1 setting z_i to the result.
3. The result is an edge pointing to a vertex labeled $x(e_0)$ with outgoing edges $z_i, i = 0, 1, ...r^2 - 1$. This vertex and the edge pointing to it are normalized as described above.

Gate QMDD Construction: Given these basic operations, an important procedure we have developed builds the QMDD representation of the matrix defining a gate transformation. Assume, as above, the variable order $x_0, x_1, ..., x_{n-1}$ from the terminal vertex towards the start vertex. A gate G is specified by the $r \times r$ base transition matrix M, the target line x_t and a possible empty set of control lines C. Space here only allows us to outline the procedure. A detailed understanding is best derived from the C code implementation available from the authors.

The procedure builds the QMDD from the terminal vertex towards the start vertex. It has three phases:

1. **Variables below the target**: For these variables, r^2 separate QMDD are constructed. The $(i\times r+j)^{th}$ QMDD has a path following the active values (the values required to activate the controls) for the control variables to the terminal value $M_{i,j}$ with all other paths leading to 0. Non-control variables (unconnected lines) are taken into account through certain identity matrices combined with the active part of the QMDD using Kronecker products.
2. **The target variable**: A single QMDD is formed with the start vertex controlled by the target vari-

able and the edges from that vertex leading to the QMDDs constructed in 1.

3. **Variables above the target**: The upper part of the QMDD has a path following the active values for the control variables with all other paths leading to 0. Again, non-control variables are taken into account through appropriate identity matrices and Kronecker products.

The most important aspect of this procedure is that it builds the QMDD corresponding to a gate from terminal to start vertex in a single pass by iterating through x_0 to x_{n-1} with no backtracking or recursion. It is thus quite efficient and to a large extent the time required is independent of the complexity of the gate under consideration.

The above outlines how the QMDD giving the matrix tranformation defining a single gate is constructed. For a cascade of gates (circuit) $G_0G_1...G_{t-1}$ where transformation for gate G_i is defined by matrix M_i, the transformation for the complete circuit is $M_{t-1}\times M_{t-2}\times ...\times M_0$.

5. Implementation Notes and Efficiency

We have implemented a QMDD package in C (available at www.cs.uvic.ca/~mmiller/QMDD). Because of its unique approach, this is a stand-alone program which is not built on top of an existing package such as CUDD.

We use standard dynamic memory allocation techniques and a unique table as introduced in [2] for creation and storing vertices. The unique table ensures the uniqueness of the representation.

A computed table [2] is used to reduce duplicate computations. When an operation $AopB$ is to be performed where A and B are two matrices represented as QMDD and *op* denotes one of matrix addition, matrix multiplication or Kronecker product, the computed table is checked to see if the result of the computation is available there before performing the full computation. Since the computations are themselves recursive, this check is performed at each recursive step so that results known for subcomputations can also be employed.

To further reduce computation time for matrix multiplication and the Kronecker product, a flag is associated with each vertex to identify whether the QMDD it and its descendants comprise represents an identity matrix. The terminal vertex, which has value 1, represents the $r^0 \times r^0$ identity matrix. Each nonterminal vertex represents a block matrix and is an identity matrix only if the block matrices on the diagonal are identity matrices involving one less variable and all off diagonal entries are 0 matrices. These conditions are readily checked by examining the edges from the vertex in question and the vertices those edges point to. Note that the check only goes to depth 1 because it is known whether each immediate descendant is an identity matrix or represents a 0 matrix (an edge with weight 0 pointing to the terminal vertex).

Benchmark	Lines	Gates	QMDD Vertices
ham3tc.tfc	3	3	10
3_17tc.tfc	3	6	10
hwb4tc.tfc	4	17	22
xor5d1.tfc	5	4	10
mod5d1.tfc	5	8	13
5mod5tc.tfc	5	17	18
hwb5tc.tfc	5	55	47

Table 3. Selected Reversible Benchmarks.

6. Experimental Results

We have used our package to build QMDDs for a selection of benchmarks available on the reversible logic site maintained by D. Maslov [6]. The results are shown in Table 3. The circuits shown in this table are quite small for a reason. In order to verify our work, we built the QMDD for each specification from the circuit and also by recursive partitioning of the permutation defining the reversible function. Testing for equivalence only requires we check equality of the start edges for the resulting pair of QMDD. Although limited to fairly small problems, the above was quite useful in building confidence in the correctness of our implementation. The execution time for all the examples is negligible.

As a larger scale example we consider a circuit known as *cycle17_3* [6]. The circuit has 20 lines and 48 Toffoli gates with from 2 to 18 inputs. The matrix representations for each gate and the circuit itself are $2^{20}\times 2^{20}$ permutation matrices. Our prototype package reads the circuit specification, builds the QMDD for each gate, and finds the QMDD for the circuit (by multiplying the 48 gate QMDD) in negligible time (reported as 0.0 user seconds by the UNIX *time* function on a Sun server). The QMDD for the circuit has 236 vertices with a total of 3,651 vertex spaces being allocated.

A total of 130,920 computed table lookups were performed with 56,537 of 65,424 add lookups hits, 38,614 of 64,847 multiplication lookups hits and 160 of 649 Kronecker product lookups hits, where a hit means the result was found in the computed table thereby avoiding recomputation. This experiment was run with a unique table with 8,192 buckets and a computed table with 4,096 slots. Note that if no computed table is used this problem takes 72 user seconds on the same machine. This illustrates the critical importance of the computed table for larger problems.

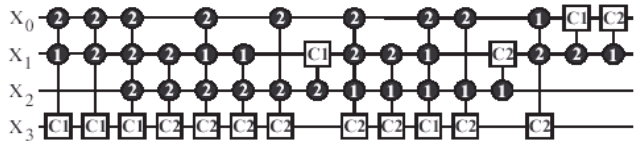

Figure 2. A reversible ternary full adder [8].

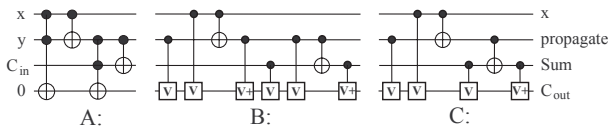

Figure 3. Three implementations of a binary full adder [7].

Figure 2 shows a circuit for a ternary full adder as presented in [8]. The black circles indicate controls with each one labeled by the active control value for that connection. C1 and C2 represent cycle-by-1 and cycle-by-2 operations respectively. The QMDD for the circuit has 23 vertices with 180 vertex spaces in total being used during its construction. There were 12,738 computed table references with a hit rate of 80.8%. Again the computation time is negligible.

As a final example, we consider the verification of three implementations of a binary 3-bit full adder as shown in Figure 3; one Toffoli gate circuit and two quantum gate circuits are given. In building the QMDD for circuit A, space for 38 vertices was allocated. Building the QMDD for circuit B required an additional 69 vertices while building the QMDD for circuit C required only a further 4. The QMDD for each circuit is, as expected, the same and requires 9 vertices. The key thing to note is that having built QMDD for two circuits, verifying that the two circuits have the same functionality and hence identical QMDD only requires that we verify that the edges, including the weights, leading to the start vertices for the two QMDD are the same. One does not have to compare the complete QMDD. This is a result of using a common unique table.

7. Conclusion

We have presented QMDD, a new decision diagram representation targeted to the matrices encountered in specifying reversible and quantum gates and circuits, and the results for a prototype implementation that show it is efficient for circuits of quite reasonable size.

Our ongoing work involves further development of the package including a QuIDDpro-like interface, and examining the possibility of QMDD as a representation for use in a circuit synthesis tool.

Acknowledgements

This work was supported by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada. David Goodman assisted in gathering some of the experimental results.

References

[1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebric decision diagrams and their applications. *Form. Methods Syst. Des.*, 10(2-3):171–206, 1997.

[2] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a bdd package. In *27th Design Automation Conference*, June 1990.

[3] E. Clarke, K. Mcmillan, X. Zhao, M. Fujita, and J. Yang. Spectral transforms for large boolean functions with applications to technology mapping. *Formal Methods in System Design*, 10(2-3):137–148, 1997.

[4] A. De Vos, B. Raa, and L. Storme. Generating the group of reversible logic gates. *J. of Physics A: Mathematical and General*, 35:7063–7078, 2002.

[5] M. Fujita, P. C. McGeer, and J. C.-Y. Yang. Multiterminal binary decision diagrams: An efficient data structure for matrix representation. *Form. Methods Syst. Des.*, 10(2-3):149–169, 1997.

[6] D. Maslov. Reversible logic synthesis benchmarks page. http://www.cs.uvic.ca/~dmaslov/, 2005.

[7] D. Maslov, C. Young, D. M. Miller, and G. W. Dueck. Quantum circuit simplification using templates. In *Design and Test Europe (DATE)*, pages 1208–1213, Munich, Germany, March 2005.

[8] M. Miller, G. Dueck, and D. Maslov. A synthesis method for MVL reversible logic. In *International Symposium on Multiple-Valued Logic*, pages 74–80, Toronto, Canada, May 2004.

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

[10] F. Somenzi. CUDD: CU Decision Diagram package - release 2.4.1. http://vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html, 2005.

[11] T. Toffoli. Reversible computing. *Tech memo MIT/LCS/TM-151, MIT Lab for Comp. Sci*, 1980.

[12] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based simulation of quantum computation in the state-vector and density-matrix representation. In *Proceedings of SPIE*, volume 5436, April 2004.

[13] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Graph-based simulation of quantum computation in the density matrix representation. *Quantum Information & Computation*, 5(2):113–130, 2005.

[14] G. F. Viamontes, I. L. Markov, and J. P. Hayes. Quiddpro: High-performance quantum circuit simulation. http://vlsicad.eecs.umich.edu/Quantum/qp/, 2005.