Variable Reordering and Sifting for QMDD

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

David Y. Feinstein and Mitchell A. Thornton
Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX, USA
dfeinste,mitch@engr.smu.edu

Abstract

This paper considers variable reordering for quantum multiple-valued decision diagrams (QMDD) used to represent the matrices describing reversible and quantum gates and circuits. An efficient method for adjacent variable interchange is presented and this method is employed to implement sifting of QMDDs. Experimental results are presented showing the effectiveness of the proposed techniques.

1. Introduction

A reversible / quantum circuit is a cascade of reversible / quantum gates. The behaviour of each such gate can be described as a matrix and the function performed by the circuit is described by the product of the individual gate matrices. The computational problem is that an *n*-line gate in *r*-valued logic has dimension $r^n \times r^n$ so the matrices quickly become very unwieldy.

The quantum multiple-valued decision diagram (QMDD) data structure presented in [9][10] was specifically designed to address this problem. A QMDD represents the matrix corresponding to a gate or circuit as a directed acyclic graph. Efficient methods for constructing QMDDs for individual gates and for performing matrix multiplication directly with QMDDs is presented in [9].

As is the case for other decision diagram representations such as the ordered binary decision diagram [2], the number of vertices in a QMDD depends on the variable ordering selected. This paper addresses variable ordering for QMDDs. We present methods for adjacent variable interchange and a heuristic vertex reduction algorithm based on Rudell's 'sifting' technique [7]. We also describe enhancements to the QMDD structure and its implementation required to make variable reordering efficient and effective.

Section 2 presents the basic concepts of binary and MVL reversible and quantum gates and circuits with particular emphasis on the matrix representation. Section 3 addresses adjacent variable interchange for QMDD. Section 4 shows how the heuristic variable ordering technique known as 'sifting' can be applied to QMDDs. Experimental results are presented in section 5 and the paper concludes with observations and suggestions for further research.

2. Preliminaries

2.1 Reversible Logic and Quantum Circuits

We present the basic concepts of reversible and quantum circuits necessary for this paper. More extensive background is available in the literature (*e.g.* [6]).

Definition 1: A gate / circuit is logically *reversible* if it maps each input pattern to a unique output pattern.

Binary reversible gates and circuits have seen considerable interest due to Landauer's principle which states that the erasure of information dissipates energy. Bennet [1] showed that for a binary circuit to not consume energy, it must be composed of reversible gates. The concept of reversibility has been extended to MVL circuits [5]. Quantum logic gates and circuits are inherently both logically and physically reversible [6]. In general, the behaviour of reversible and quantum gates and circuits can be described by complex-valued matrices and are modeled as bijective functions.

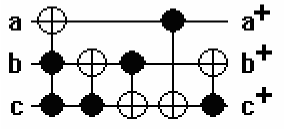

Figure 1: A binary reversible circuit

Fig. 1 shows a binary reversible circuit with 3 lines and 5 gates. The symbol ⊕ denotes the NOT operation. For each gate, the NOT operates on the *target* line if every *control* line (lines with a black circle) has the value 1. Otherwise the target line is unchanged. Control and unconnected lines pass through the gate unchanged. A gate with no controls is a conventional *NOT* gate. One with a single control is termed a *controlled-NOT* and gates with more than one control are *Toffoli* gates [6].

Multiple-valued reversible circuits have been considered in [5]. The structure illustrated in Fig. 1 is generalized so that the target line is operated on by a negation or cycle operation depending on the values of the control lines. The non-zero values indicated in the control line connections specify the nonzero value required in order to trigger the operation. Fig. 2 shows a reversible circuit from [5] that operates as described and realizes a ternary full adder.

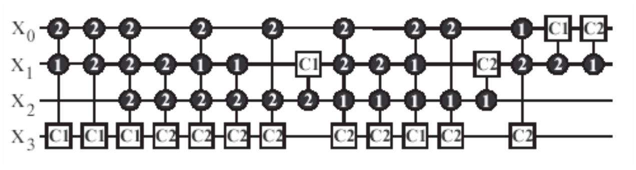

Figure 2: A ternary reversible full adder

Quantum logic gates [6] operate in a similar fashion with the values on designated control lines determining if a particular quantum logic transformation is to be applied to the target line.

2.2 Matrix Representation of Reversible and Quantum Gates and Circuits

The operations performed on the target line for the gates considered in this paper are given by the $r \times r$ matrices in Table 1. NOT is the normal binary complement shown as $\oplus$ in Fig. 1. V and V^+ are quantum operations. Note that $V^+ = V^{-1}$. NEG is ternary negation and C1 and C2 are the two ternary unary cycle operations [5].

Table 1: Gate operation matrices

Binary (r=2)Matrices		Ternary (r=3)Matrices	
NOT	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	NEG	$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$
V	$\begin{pmatrix} \frac{1+i}{2} & \frac{1-i}{2} \\ \frac{1-i}{2} & \frac{1+i}{2} \end{pmatrix}$	C1	$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$
V^+	$\begin{pmatrix} \frac{1-i}{2} & \frac{1+i}{2} \\ \frac{1+i}{2} & \frac{1-i}{2} \end{pmatrix}$	C2	$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

The matrices in Table 1 define the operation on the target line. The matrix definition for an r-valued gate in an n-line reversible or quantum circuit has dimension $r^n \times r^n$ taking into account the operation on the target line and the control and unconnected lines.

For example, the leftmost gate in Fig. 1 has the specification and matrix definition given in Fig. 3.

c	b	a	c^+	b^+	a^+
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	1
1	0	0	1	0	0
1	0	1	1	0	1
1	1	0	1	1	1
1	1	1	1	1	0

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Figure 3: Specification and matrix representation for the Toffoli gate T(a;b,c) in a 3-line circuit

Fig. 4 shows the matrix specification for a V type quantum gate which has complex valued entries.

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1+i}{2} & \frac{1-i}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & \frac{1-i}{2} & \frac{1+i}{2} \end{bmatrix}$$

Figure 4: Matrix representation for the quantum gate V(a;c) in a circuit with lines a,b,c

A reversible / quantum circuit is a cascade (from input to output) of gates $g_0, g_1, g_2, \ldots$ Each gate g_i has a matrix representation M_i where the dimension depends on the radix and the number of lines in the circuit. The matrix defining the transformation performed by the overall circuit is given by $\ldots M_2 \times M_1 \times M_0$. The challenge is that the size of these matrices and the computation required for matrix multiplication by traditional techniques is prohibitive for all but a small number of circuit lines. For example, for a ternary circuit with 10 lines each matrix has dimension 59049 by 59049.

2.3 Quantum Multiple-valued Decision Diagrams

Quantum multiple-valued decision diagrams (QMDD) were introduced in [9][10] as a means to represent and manipulate the matrices required for reversible and quantum gates and circuits. Here we present a brief description of QMDDs and assume the reader is familiar with the fundamentals of decision diagram techniques [12].

As noted above, a reversible / quantum circuit with n lines has a transformation matrix of dimension $r^n \times r^n$ where r is the radix. Such transformation matrices quickly explode in size. However, they do exhibit a great degree of regularity.

A matrix of dimension $r^n \times r^n$ can be partitioned as:

$$M = \begin{bmatrix} M_0 & M_1 & \cdots & M_{r-1} \\ M_r & M_{r+1} & \cdots & M_{2r-1} \\ \vdots & \vdots & \ddots & \vdots \\ M_{r^2-r} & M_{r^2-r+1} & \cdots & M_{r^2-1} \end{bmatrix}$$

where each M_i element is a matrix of dimension $r^{n-1} \times r^{n-1}$. Each of the M_i can be similarly partitioned and the process repeated until scalars are reached. This repeated partitioning leads to the fundamental QMDD structure.

Definition 2: A *quantum multiple-valued decision diagram* (QMDD) [9] is a directed acyclic graph with the following properties:

- There is a single *terminal* vertex with associated value 1. The terminal vertex has no outgoing edges.
- There are some number of *non-terminal* vertices each labeled by an r^2-valued selection variable. Each non-terminal vertex has r^2 outgoing edges designated $e_0, e_1, ..., e_{r^2-1}$.
- One vertex is the *start* vertex and has a single incoming edge that itself has no source vertex.
- Every edge in the QMDD, including the one leading to the start vertex, has an associated complex-valued *weight*. An edge with weight of 0 must point to the terminal vertex. This is required to ensure uniqueness of the representation of each matrix.
- The selection variables are *ordered* (assume with no loss of generality the ordering $x_0 \prec x_1 \prec ... \prec x_{n-1}$) 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 non-terminal vertex labeled x_i points to a non-terminal vertex labeled x_j, $j < i$ or to the terminal vertex. Hence x_0 is closest to the terminal and x_{n-1} labels the start vertex.
- No non-terminal vertex is *redundant*, *i.e.* no non-terminal vertex has its r^2 outgoing edges all with the same weights and pointing to a common vertex.
- Each non terminal vertex is normalized (see details in next subsection).
- Non-terminal vertices are *unique*, *i.e.* no two non-terminal vertices labeled by the same x_i can have the same set of outgoing edges (destinations and weights).

As is common for decision diagram representations, a key property of QMDDs is that the representation for any given matrix is unique. A proof is available from the authors. A key feature of this proof is the normalization process that is applied during the construction of a QMDD.

2.4 Vertex Normalization

The initial definition of QMDD [9][10] used the following normalization rule:

Definition 3: A QMDD vertex is *normalized* if its outgoing edges are such that there is a j such that $e_j, 0 \le j \le r^2 - 1,$ has weight 1 and $e_i, \forall i, 0 \le i < j.$ has weight 0.

When this rule is used, each vertex is normalized when it is constructed by finding the nonzero weight on the lowest index edge (one must exist or the vertex is redundant), dividing all edge weights by the weight identified, and attaching the identified weight to the edge leading to the vertex.

We have found that vertex normalization as given in Definition 3 does allow for adjacent variable interchange to be performed as an operation local to the variables being interchanged for QMDDs representing reversible binary and multiple-valued circuits. However, it does not for QMDDs representing some quantum circuits. Space does not permit us to provide a full explanation here but basically the problem arises since variable reordering changes the order in which the elements of the matrix are considered and normalization as defined in Definition 3 can identify different normalization divisors depending on the variable order.

The solution is to use the following alternative normalization definition:

Definition 4: A QMDD vertex is *normalized* if its outgoing edges are such that the largest weight on any edge out of the vertex is 1.

When this rule is used, the process for normalizing a vertex is as described above except the maximum weight (i.e. largest magnitude among the complex weights) on the edges from the vertex is used as the divisor and incoming edge weight. This approach is independent of variable order, and allows for adjacent variable interchange as a local operation since the maximum value in a matrix is independent of which order one traverses the matrix.

2.5 Skipped Variables

Definition 5: Given the ordering $x_0 \prec x_1 \prec ... \prec x_{n-1}$ an edge from a vertex labeled $x_i, i > 0$, *skips* a variable if it points to the terminal vertex or it points to a vertex labeled $x_j, j < i-1$.

Theorem 1: A QMDD for a matrix representing a binary or multiple-valued reversible circuit has no edges that skip variables except for edges that point to the terminal vertex and have weight 0.

Proof: It is clear from the definition of QMDD, that an edge with nonzero weight that skips a variable means the corresponding matrix has a sub-matrix of adjacent non-zero entries of dimension $r^k \times r^k$ for some $k > 0$. However, the matrix representing a binary or multiple-valued reversible function is a permutation matrix since reversible functions are bijections and thus has only 0 and 1 entries with a single 1 in each row and column. It follows that the QMDD for such a circuit can have no edges with nonzero weight that skip variables. □

Conjecture 1: Theorem 1 also holds for a QMDD corresponding to a circuit composed of quantum gates.

Empirical evidence shows conjecture 1 is likely true. We are currently working on a formal proof.

It is straightforward to show this conjecture is true for the matrices describing individual reversible and quantum gates by construction, and also for the matrix representing a reversible binary or MVL circuit since these matrices are permutation matrices and the only constant square blocks in a permutation matrix have value 0. We have not yet been able to show it is true for arbitrary cascades of quantum gates.

3. Interchanging Adjacent Variables

We consider the case of interchanging variables α and β where the former is immediately above the latter (closest to the initial vertex) in the QMDD. Note that each vertex has r^2 outgoing edges. The key is, as mentioned above, to perform the interchange as a local transformation. The technique presented is based on the technique developed by Miller and Drechsler [4] for multiple-valued decision diagrams.

Consider a vertex γ labelled by variable α. We construct a square matrix $\mathbf{T}$ of dimension r^2. For i=0,1,…, r^2-1,

(a) if the i-edge from γ leads to a vertex δ labelled by variable β, then for j=0,1,…, r^2-1, $\mathbf{T}_{ij}$ is set to point to the vertex pointed to by the j-edge of δ with the edge weight being the product of the edge weights on the i-th edge from γ and the j-th edge from δ;

(b) if the i-edge from γ leads to a vertex δ not labelled β, then $\mathbf{T}_{ij}$ is set to the i-edge from γ for j=0,1,…, r^2-1.

Once $\mathbf{T}$ is constructed as above, the level interchange is made by *relabelling* γ with β, and setting each j-edge from γ, j=0,1,…, ~~p~~r^2-1 to point to a vertex labelled α whose i-th edge, i=0,1,…, ~~q~~r^2-1, points to the vertex pointed to by $\mathbf{T}_{ij}$. During this construction,~~ if β denotes a variable, the edge operations~~ **the vertices** are normalised as described in ~~the previous subsection~~ **Definition 4**. It is easily seen that following this construction, vertex γ, now labelled β, is the top of a decision diagram representing the same matrix it did when originally labelled α.

The complete level interchange is accomplished by performing the above for all vertices originally labelled α. These are readily identified as we use a separate unique table [12] for each variable. The idea of relabelling these vertices, as opposed to creating new vertices, is critical as it means that edges leading to them, and the vertices from whence those edges originate, are unaffected by the level interchange. When a vertex is relabelled it must be removed from one unique table and entered into the unique table corresponding to its new variable label but this is a relatively simple operation given the data structures used for QMDD [10].

The vertices originally labelled β are affected as edges to them are removed. The use of reference count garbage collection [10] accounts for when a vertex can be deleted (actually reused) or must be retained.

It is critical to note that no vertex above or below the two levels being interchanged is affected except for changing the reference counts of vertices immediately below. The result is that adjacent variable interchange is a local operation affecting only the two levels being interchanged and reference counts for vertices immediately below those levels.

4. Sifting QMDD

Given the above method for adjacent variable interchange, variable reordering for QMDDs is readily implemented using an approach based on Rudell's sifting approach [7] developed for BDDs.

In general terms, our sifting method proceeds as follows:

QMDD Sifting Procedure:

i) Select a variable α that labels the most vertices in the QMDD. In the event of a tie, choose the variable closest to the terminal vertex.
ii) Sift α to the bottom (closest to the terminal vertex) of the QMDD by a sequence of adjacent variable interchanges.
iii) Sift α to the top of the QMDD by a sequence of adjacent variable interchanges.
iv) During steps (ii) and (iii) a record is kept of the position of α that yields the smallest vertex count in the QMDD, so now sift α back down to that position.
v) Repeat steps (i) to (iv) until each variable has been sifted into its *best* position noting that once a variable is selected for sifting, it is not selected a second time.

Note that the size of the QMDD after each variable interchange required in step (iv) is determined by checking reference counts for the vertices for the two variables being interchanged. It is not necessary to traverse the entire QMDD. There are $n!$ possible orderings of n variables. The sifting method examines on the order of n^2 orderings, and determines the ordering among this subset that result in the smallest QMDD.

5. Experimental Results

The QMDD package is implemented in C. The results reported here were run on a laptop computer with a 1.73 GHz Intel Pentium M processor and 1GB RAM running LINUX on a 256MB virtual machine under VMware 5.5. We used LINUX in order to compare our implementation to QuIDDPro 3.0(beta) [11] which is available as an executable only. We used the gcc 4.0.0 C compiler with level 4 optimization to compile the QMDD package.

5.1 Binary Examples

Results for a number of binary functions from Maslov's [3] benchmark web site are reported in Table 2. For each circuit, we give the following information:

a) type – nct: circuit uses not, controlled-not and Toffoli gates; qc: circuit uses controlled-not, V and V^+ gates.
b) lines – number of lines in the circuit,
c) gates – number of gates in the circuit,
d) number of vertices before sifting,
e) time to build the QMDD – CPU msec. using the standard library time.h routines,
f) number of vertices after sifting,
g) time to sift QMDD – CPU msec. using the standard library time.h routines,
h) percentage vertex count reduction by sifting,
i) maximum number of vertices encountered during sifting – this is an indicator of how large the QMDD might be but is not necessarily the maximum since sifting does not consider all variable orderings.

The results show that the effect of sifting varies significantly from example to example. A low improvement can be a result of having started from what is already a good ordering, the fact that the sifting heuristic does not visit all possible variable orderings, or, the function's QMDD representation is insensitive to variable ordering.

The results for the "hidden-weight-bit" problems hbw4 – hbw12 are interesting. They show the size of the QMDD can grow exponentially with the number of lines in the circuit. The benefit gained by sifting also increases with the number of lines.

Table 2 also shows the results of using QuIDDPro Version 3.0(beta) on the same computer. On average for the circuits shown, the number of vertices for the QuIDDPro representation is 2.06 times the number for the QMDD representation prior to sifting. This is as expected since a nonterminal QuIDDPro vertex has two outgoing edges while a nonterminal QMDD vertex has four outgoing edges for binary functions. What is interesting is how much the ratio can differ from 2. The largest sized decision diagrams for the circuits shown is for cyc17_3 where the ratio is 2.47. QuIDDPro uses the highly efficient CUDD decision diagram package and also offers considerably more functionality than the current QMDD implementation. QuIDDPro is designed for binary reversible and quantum gates and circuits.

5.2 Ternary Examples

There are as yet no established benchmarks for multiple-valued reversible and quantum circuits available in the literature. This is largely because CAD tools for designing and simulating such circuits are not generally well developed. Indeed it is hoped that QMDD will be helpful in this regard.

Table 3 contains some ternary examples. The first is the reversible ternary adder from [9] shown in Fig. 2. The initial QMDD is relatively small (23 vertices) but even in this case sifting results in notable reduction.

The S circuits are highly regular. An S circuit with *n* lines has *n*-1 gates where gate g_i is a C1 gate with target x_i and a single 1-control x_{i+1}. As expected, given this regular and quite simple structure, the QMDDs are small (the number of vertices is twice the number of lines in the circuit) and can be shown to have a very regular structure. Sifting results in no improvement, but, and this is a disadvantage of a heuristic, considerable computation is required.

Each R*n*-*m* circuit has *n* lines and *m* pseudo-randomly generated gates. Each gate is randomly chosen to be C1 or C2 with a randomly chosen target and a single randomly chosen control. The control is randomly chosen to be a 1 or 2-control. The improvement by sifting is as expected quite variable.

These examples indicate that QMDD construction and sifting are reasonably practical for quite large binary and ternary problems. It is a concern that the cost of sifting seems quite high for large ternary examples. We are looking for ways to improve the implementation, but it may well be that we are dealing with representations of very large and very complex matrices in those cases.

The QMDD package is applicable for higher radix problems as well. To put this in better context, we note that constructing a QMDD for an *r*-valued, *n*-line circuit with *m* gates is equivalent to constructing *m* $r^n \times r^n$ matrices and performing *m*-1 matrix multiplications but does so in a tractable manner.

6. Conclusions and Future Work

This paper has considered variable reordering in QMDD and has described a sifting technique for determining good variable orderings. The experimental results show the method can be quite effective but also that it can be computationally expensive sometimes with little benefit.

We are working on criteria that may help in determining when it is likely to be useful to apply sifting to a QMDD. We are studying the special structure of QMDDs not evident in general decision diagrams such as the frequency of edges of weight 0 pointing to the terminal vertex, the regular structure of a QMDD resulting from the regular structure of the matrices being represented and our conjecture regarding skipped variables. We are also exploring how sifting can be used to transform QMDD to variable orderings that will better illuminate the structure of the matrices for synthesis procedures.

7. References

[1] C.H. Bennett, "Logical Reversibility of Computation," *IBM, J. Res. Dev.*, Vol. 17, No. 6, pp. 525-532, 1973.

[2] R.E. Bryant. "*Graph-Based Algorithms for Boolean Function Manipulation*". IEEE Transactions on Computers, Vol. C-35 Issue 8, Aug. 1986, pp. 677-691.

[3] D. Maslov. Reversible logic synthesis benchmarks page. http://www.cs.uvic.ca/~dmaslov/, Feb. 15, 2007.

[4] D.M. Miller and R. Drechsler, "Augmented sifting of multiple-valued decision diagrams," *Proc. 2003 Int. Symposium on Multiple-Valued Logic*, Tokyo, Japan, May 2003, pp. 275-382.

[5] D.M. Miller, G. Dueck, and D. Maslov, "A Synthesis Method for MVL Reversible Logic," *Proc. 2004 Int. Symposium on Multiple-Valued Logic*, Toronto, Canada, May 2004, pp. 74-80.

[6] M.A. Nielsen and I.L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, 2000.

[7] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams", In *Proceedings of the International Conference on Computer-Aided Design*, Santa Clara, CA, Nov. 1993, pp. 42-47.

[8] F. Somenzi, "*The CUDD Package*", University of Colorado at Boulder, 1995. Version 2.4.0 available at: http://vlsi.colorado.edu/~fabio/.

[9] D.M. Miller and M.A. Thornton, "QMDD: A Decision Diagram Structure for Reversible and Quantum Circuits", *Proc. IEEE International Symposium on Multiple-Valued Logic,* on CD, May 17-20, 2006.

[10] D.M. Miller, M.A. Thornton, and D. Goodman, "A Decision Diagram Package for Reversible and Quantum Circuits", *Proc. IEEE World Congress on Computational Intelligence*, on CD, July 2006.

[11] G.F. Viamontes, I.L. Markov, and J.P. Hayes, "QuIDDPro: High-Performance Quantum Circuit Simulation", vlsicad.eecs.umich.edu/Quantum/qp/, Oct. 20, 2006.

[12] S.N. Yanushkevitch, D.M. Miller, V.P. Shmerko and R.S. Stankovic, *Decision Diagram Techniques for Micro- and Nanoelectronic Design*, CRC Taylor and Francis, 2006.

Acknowledgements This work was supported by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada. Tony Xiao assisted with the use of QuIDDPro and with the collection of the experimental data.

Table 2: Experimental Results – binary circuits

name	type	lines	gates	vertices before sifting	time to build QMDD (msec)	vertices after sifting	time to sift QMDD (msec)	vertex count reduction by sifting	max. vertices during sifting	QuIDD Pro vertices	time to build BDD (msec)
5mod5	nct	6	17	28	0	16	8	43.9%	28	45	77
6symd2	nct	10	20	247	7	170	42	31.2%	478	299	117
9symd2	nct	12	28	229	7	184	50	19.7%	558	445	194
c2	nct	35	116	150	30	136	289	9.3%	504	348	1546
c2	qc	35	305	150	220	136	241	9.3%	504	348	10245
c3-17	nct	3	6	10	0	10	4	0.0%	11	21	22
c410184	nct	14	46	39	0	33	43	15.4%	86	86	274
c410184	qc	14	74	39	0	33	52	15.4%	86	86	492
cyc17-3	nct	20	48	236	10	42	131	82.2%	418	584	880
ham3	nct	3	5	10	0	10	0	0.0%	10	21	21
ham15	nct	15	132	4522	140	2638	488	42.7%	13878	7547	2051
hwb4	nct	4	11	22	0	20	4	9.1%	22	45	43
hwb4	qc	4	21	22	0	20	8	9.1%	22	45	83
hwb7	nct	7	289	179	30	155	16	13.4%	181	351	2117
hwb8	nct	8	614	343	120	280	28	18.4%	351	684	6254
hwb9	nct	9	1541	683	640	520	40	23.9%	690	1349	24566
hwb10	nct	10	3595	1331	2920	960	59	27.9%	1347	2650	98704
hwb11	nct	11	9314	2639	14290	1730	130	34.4%	2685	5223	478555
hwb12	nct	12	18393	5167	53350	3185	265	38.4%	5254	10283	1722050
rd84d1	nct	15	28	3588	27	396	117	89.0%	4415	1252	255

Table 3: Experimental Results – ternary circuits

name	lines	gates	vertice s before sifting	time to build QMDD (msec)	vertices after sifting	time to sift QMDD (msec)	vertex count reduction by sifting	max. vertices during sifting
adder	4	16	23	0	15	4	34.8%	39
S25	25	24	50	11	50	312	0.0%	184
S50	50	49	100	58	100	1370	0.0%	384
S75	75	74	150	168	150	3112	0.0%	584
S100	100	99	200	370	200	5550	0.0%	784
R5-25	5	25	132	11	124	15	6.1%	185
R5-50	5	50	204	35	200	23	2.0%	205
R5-75	5	75	196	58	193	19	1.5%	208
R5-100	5	100	203	85	198	23	2.5%	213
R10-25	10	25	1308	74	442	105	66.2%	1308
R10-50	10	50	9670	687	6839	1081	29.3%	12151
R10-75	10	75	41170	6776	34991	7901	15.0%	45826
R10-100	10	100	51133	12361	46556	10646	9.0%	52785