QMDD and Spectral Transformation of Binary and Multiple-Valued Functions*

D. Michael Miller
University of Victoria
Victoria, BC, Canada
mmiller@uvic.ca

Mitchell A. Thornton
Southern Methodist University
Dallas, TX, USA
mitch@engr.smu.edu

Abstract

The use of decision diagrams (DD) for the computation and representation of binary function spectra has been well studied [2,3,5]. Computations and representation of spectra for multiple-valued logic (MVL) functions have also been considered [6]. For binary functions, this approach can be implemented using one of a number of the highly efficient publicly available binary decision diagram (BDD) packages, *e.g.* CUDD [13]. Work with MVL functions requires a package suited to the MVL case, e.g. [7,8].

Quantum multiple-valued decision diagrams (QMDD) were introduced in [9-12] as a means to represent and manipulate the matrices required for binary or multiple-valued reversible and quantum gates and circuits. In this paper, we show how QMDD can also be applied to the computation of spectral transformations of binary and multiple-valued functions. A major motivation for this work is that it introduces an approach to the spectral analysis of reversible and quantum circuits in a representation that is applicable to simulation and synthesis of such circuits. It is also of interest in that it is a consistent approach for a variety of transformations of binary and multiple-valued functions.

1. Spectral Transformation

We only consider issues here that are involved in computing spectral transformations and not the reasons for doing so or how spectral representations assist in analysis and synthesis problems. The reader interested in those matters should consult the literature, *e.g.* [4,14]. It is also important to note that we only show three representative spectral transformations. The approach is directly applicable to many other transformations found in the literature.

Consider a binary or MVL n-variable function represented by a truth column vector F. We are interested in spectral transformations defined as in eqn. 1 where T^n is a $r^n \times r^n$ transformation matrix and r is the radix of the function. The transformation matrices of interest are defined by the Kronecker product shown in eqn. 2 where T^1 is a $r \times r$ 'base' matrix defining the transformation.

$$S = T^n F \tag{1}$$

$$T^n = \bigotimes_{i=1}^{n} T^1 \tag{2}$$

Rademacher-Walsh: $$T^1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3a}$$

Reed-Muller: $$T^1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{3b}$$

Chrestenson: $$T^1 = \begin{bmatrix} 1 & 1 & 1 \\ 1 & a^2 & a \\ 1 & a & a^2 \end{bmatrix},\ a = e^{\frac{-2\pi i}{r}},\ r = 3 \tag{3c}$$

* *This research was supported in part by a Discovery Grant from the Natural Sciences and Engineering Research Council of Canada*.

Eqn. 3 shows three well studied transformations. The Rademacher-Walsh transformation is computed over the integers; the Reed-Muller transformation is computed over GF(2); and the Chrestenson transformation is computed over the complex numbers. The Chrestenson transform shown in eqn. 3c is for the case r=3. It is readily extended to higher values of r and reduces to the Rademacher-Walsh case for r=2.

For the Rademacher-Walsh case, the function vector F may be in (0,1) or (+1,-1) coding [4]. It is always (0,1) for the Reed-Muller case where the computation is done over $GF(2)$. In the Chrestenson case, F is coded with logic value $p, 0 \le p < r,$ represented by a^p.

2. Spectral Transformation Computation

Computation of a spectral transformation by matrix multiplication as per eqn. 1 requires $r^n \times r^n$ multiplications and $r^n \times (r^n - 1)$ additions so the complexity is $O(r^{2^n})$. This becomes impractical for n of any significant size.

Fast transform techniques analogous to the fast discrete Fourier transform are well known [14] and are much more efficient than direct matrix multiplication. For example, Fig. 1 shows a C program implementing a fast Rademacher-Walsh transform for binary functions. The complexity can be seen from this code to be $O(n2^n)$ where n is the number of function variables.

```
void fht(int f[], int n)
{
  int i,j,k,t,m,p;
  for (m=1;m<(1<<n);m=m<<1)
  {
    for (i=0;i<(1<<n);i+=m<<1)
    {
      for (j=i,p=k=i+m;j<p;j++,k++)
      {
        t=f[j];             // line a
        f[j]=(f[j]+f[k]);   // line b
        f[k]=(t-f[k]);      // line c
      }
    }
  }
}
```

Figure 1: Fast binary Rademacher-Walsh Transform.

The approach illustrated in Fig. 1 can be adapted to other transforms. For example, the Reed-Muller transform is implemented by replacing lines a, b and c by the single assignment `f[k]=(f[j]+f[k])&1` where `&1` is implementing the mod-2 operation (this is more efficient than `%2`).

While fast transform techniques are much more efficient than direct matrix multiplication, they share a major drawback in that the initial truth vector and the resulting spectrum are vectors of length r^n. Not only can such a vector be of prohibitive size, a vector representation does not identify or exploit any structure that might be inherent in the spectrum.

The use of DD techniques for computing spectral transformations has been studied for both binary and MVL functions. As noted above, this work has been somewhat separate since different DD packages have been used for the binary and MVL situations. The approach described below is unified in that it employs the QMDD package as described in [9-12] for all cases. Relatively minor extensions are required including the use of modular arithmetic for the Reed-Muller transform.

The advantages of DD approaches, including the approach described below, lie in the fact that the size of the DD representing the original function and the spectrum is not fixed and in fact exploits the structure that might be present in each. This has space implications and can directly affect the complexity of computing a transformation. Just as important, the DD representing a spectrum often makes the underlying structure of the spectrum apparent.

3. Quantum Multiple-valued Decision Diagrams

We assume the reader is familiar with the fundamentals of DD techniques [1,14].

As noted above, we are in general concerned with transformation matrices of dimension $r^n \times r^n$ where r is the radix and n is the number of variables. Such a matrix can be partitioned as shown in the following equation:

$$M = \begin{bmatrix} M_0 & M_1 & \cdots & M_{r-1} \\ M_r & M_{r+1} & \cdots & M_{2r-1} \\ \vdots & \vdots & \ddots & \vdots \\ M_{r^2-r} & M_{r^2-r+1} & \cdots & M_{r^2-1} \end{bmatrix} \quad (4)$$

where each M_i 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 1: A *quantum multiple-valued decision diagram* (QMDD) is a directed acyclic graph with the following properties:

1. There is a single *terminal vertex* with associated value 1 and no outgoing edges.
2. 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 $e_0, e_1, ..., e_{r^2-1}$.
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*. An edge with weight of 0 must point to the terminal vertex.
5. The selection variables are *ordered* (assume with no loss of generality the ordering $x_0 \prec x_1 \prec \ldots \prec x_{n-1}$) and the QMDD satisfies the following two rules:
   i) Each selection variable appears at most once on each path from the start vertex to the terminal vertex.
   ii) 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.
6. No non-terminal vertex is *redundant*, *i.e.* no non-terminal 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 for some j, $e_j, 0 \le j \le r^2 - 1$, has weight 1 and all $e_i, i < j$, have weight 0. Note, such a j must exist or the vertex would be redundant (all weights 0).
8. 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 DD representations, a key property of QMDD 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. The normalization rule used here is as introduced in [9]. This normalization procedure extracts common multiplicative factors as each vertex is created and thus extracts common factors for each subgraph of the QMDD and the QMDD itself.

Skipped variables (see Defn. 2) must be considered in DD based algorithms.

Definition 2: Given the ordering $x_0 \prec x_1 \prec \ldots \prec x_{n-1}$ an edge from a vertex labeled $x_i, i > 0$, *skips* a variable, or variables, if it points to the terminal vertex or it points to a vertex labeled $x_j, j < i - 1$.

For reversible circuits, skipped variables only appear when an edge has weight 0 and points to the terminal vertex. However, this is not the case for circuits composed of general quantum gates. Skipped variables also occur when QMDD are used for spectral transformation and must be taken into account as described in the next section.

4. QMDD-Based Spectral Transformation

Fig. 2 shows the QMDD representation for the 3-variable Rademacher-Walsh transform matrix. Note that the numbers on the edges are multiplicative weights and do not indicate variable value identifications as is usually the case for DD figures. The edges values represent transformation matrix quadrants 0, 1, 2, and 3 from left to right out of each vertex. The QMDD has a single vertex for each variable and thus grows linearly as n increases. This is an important property which is true for any transformation matrix defined as shown in eqn. 2 and is particularly important with regard to efficiency in spectral computations.

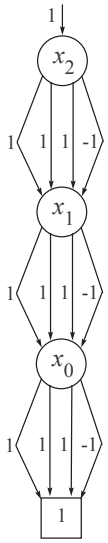

Figure 2: QMDD for 3-variable Rademacher-Walsh transform matrix.

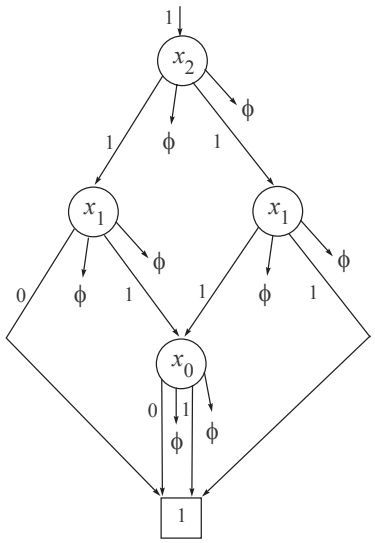

Figure 3: QMDD for 3-variable majority function.

QMDD were developed for representing and manipulating $r^n \times r^n$ matrices. To apply QMDD to the spectral transform computation problem requires a representation for column vectors. Rather than introducing a separate structure, we adapt the QMDD structure. In particular, a column vector can be considered to be a matrix with 'empty' submatrices which can be represented by null pointers in the QMDD structures. This is illustrated in Eqn. 5 where ϕ denotes an empty submatrix.

$$V = \begin{bmatrix} V_0 & \phi & \cdots & \phi \\ V_r & \phi & \cdots & \phi \\ \vdots & \vdots & \ddots & \vdots \\ V_{r^2-r} & \phi & \cdots & \phi \end{bmatrix} \tag{5}$$

This structure is repeatedly applied as the QMDD is formed. For example, the QMDD in Fig. 3, represents the column vector for the 3-variable majority function.

The QMDD structure is similar to many DD structures the reader will be familiar with. However, we note each node has multiple outgoing edges (not two as in a BDD), that the numbers on the edges are multipliers, and there is a single terminal vertex. The diagram in Fig. 3 is in effect a BDD but again note that the use of weighted edges is different from BDD.

Our QMDD-based spectral transformation method involves implementing the multiplication of a matrix and column vector represented as a QMDD resulting in a column vector. The implementation is modeled upon Bryant's *APPLY* algorithm for BDD [1], and is based on the fact that matrix multiplication can be decomposed as shown in eqn. 6. The required QMDD matrix addition is also discussed in [9] and is again based on Bryant's *APPLY*.

$$\begin{bmatrix} A_0 & A_1 & \cdots & A_{r-1} \\ A_r & A_{r+1} & \cdots & A_{2r-1} \\ \vdots & \vdots & \ddots & \vdots \\ A_{r^2-r} & A_{r^2-r+1} & \cdots & A_{r^2-1} \end{bmatrix} \times \begin{bmatrix} B_0 & B_1 & \cdots & B_{r-1} \\ B_r & B_{r+1} & \cdots & B_{2r-1} \\ \vdots & \vdots & \ddots & \vdots \\ B_{r^2-r} & B_{r^2-r+1} & \cdots & B_{r^2-1} \end{bmatrix}$$

$$= \begin{bmatrix} A_0B_0 + A_1B_r + \ldots + A_{r-1}B_{r^2-r} & A_0B_1 + A_1B_{r+1} + \ldots + A_{r-1}B_{r^2-r+1} & \cdots & \vdots \\ A_rB_0 + A_{r+1}B_r + \ldots + A_{2r-1}B_{r^2-r} & \vdots & \cdots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ \cdots & \cdots & \cdots & A_{r^2-r}B_{r-1} + A_{r^2-r+1}B_{2r-1} + \ldots + A_{r^2-1}B_{r^2-1} \end{bmatrix} \tag{6}$$

Three issues must be considered when manipulating column vectors represented as QMDD. First, null pointers must be taken into account. In multiplication, one of the arguments being a null pointer results in a null pointer in the product. Addition when an argument is a null pointer results in an answer equal to the other argument.

Second, QMDD are defined with complex-valued edge weights and the package thus supports arithmetic operations on complex values. The Rademacher-Walsh spectrum is computed over the integers which is not an issue since the integers are a subset of the complex numbers. Reed-Muller transforms are computed over GF(2) which requires a minor modification to the complex value arithmetic packages originally implemented for QMDD.

The third, and most involved issue, is the handling of skipped variables. For a matrix, a skipped variable means the matrix is such that all the submatrices in its decomposition are identical so the selection variable can be skipped. For a column vector, the interpretation is the same but only for the left-most submatrices – the others must be understood to be empty. Both situations are readily taken into account in the matrix multiplication algorithm. The minor complication is that a different interpretation is required depending on whether the QMDD represents a matrix or a column vector.

The interested reader can find details on the implementation of the QMDD package in [10,11] including a variety of standard BDD and QMDD specific techniques used to enhance the speed of the implementation. Much of the speed comes from using standard DD compute table techniques as well as computation tables for caching the results of complex number operations.

5. Experimental Results

The QMDD package is implemented in C. Extending it to compute spectral transforms was relatively straightforward and involved:

1. Implementing a routine to build a QMDD given the truth vector for a function. This is a simple recursive construction.
2. Implementing a routine to build the QMDD for the required transformation matrix. This is a simple iterative routine as a result of the compact linear structure illustrated in Fig. 2.
3. Modifications to the matrix multiplication and addition routines to handle null pointers and general skipped variables separately for matrices and column vectors.
4. Modification to the complex value operation routines to do computation over GF(2) when required for the transformation.

The results reported are for experiments run on a Toshiba Protégé laptop with a 750 MHz Pentium III and 512 MB Ram. The Metrowerks compiler V.4 was used with the highest level of global optimization.

5.1 Binary Examples

Table 1 shows results for the AND of an increasing number of variables. The results show that the QMDD method becomes increasingly better in terms of number of operations as n increases, although it should be noted that the operations for the QMDD approach are considerably more complex. The size of the QMDD for the spectrum grows linearly. The space required for the QMDD is initially much greater than for the FT spectrum but the relative size decreases as n increases and will continue to do so for higher n since while the QMDD is growing linearly, the size of the FT vector grows exponentially. Note that each QMDD node occupies 44 bytes while each FT value is 4 bytes (a simple integer which is sufficient up to n=30).

Tables 2 and 3 show similar performance for the OR and the XOR function. This is not surprising since these are highly regular functions.

Table 4 is similar to the earlier tables but for randomly generated functions. This shows that the QMDD technique is not very effective for random functions and generally requires more storage than the fast transform approach. This is not unexpected since QMDD exploit structure.

Table 5 shows a number of situations comparing Rademacher-Walsh (+1,-1) coding (S), Rademacher-Walsh (0,1) coding (R), and Reed-Muller (RM) transform results. The data verifies that, as expected, the RM spectrum is most efficient in terms of QMDD size. For the 12 variable random function, the user timing routine provided in C showed an execution time of 1.001, 0.030 and 1.201 sec. for the S, RM and R spectra computations. The timings for all other examples were negligible (less than 10 sec.) It is noteworthy that the performance for the structured functions is much better in terms of time and space for this higher (15) value of n. The performance for random functions is improved in terms of number of operations but still requires more space than the FT approach.

The results given must be considered in the context that the QMDD structure can illuminate the structure of a function spectrum, whereas the FT representation is by definition flat and indicates nothing directly about structure.

5.2 Ternary Examples

Tables 6, 7 and 8 show the results for computing the Chrestenson spectra for the MIN, MAX and MOD-SUM functions for r=3 for n=2,…,10. The data shown is labeled as in the binary case. Note that in this case each QMDD vertex occupies 84 bytes while each FT value, which in this case is a complex value occupies 16 bytes.

The performance for these structured functions is as expected. The QMDD representation gains efficiency on the FT approach more quickly in the ternary case as compared to the binary case. And, once again, it should be emphasized that the QMDD method better represents the underlying transformation matrix structure. The CPU timings were again negligible.

6. Concluding Remarks

We emphasize that while this paper has used the Rademacher-Walsh, Reed-Muller, and Chrestenson spectral transformations for illustration, the approach is directly applicable to any transformation that can be expressed as shown in eqn. 1 with arithmetic over the complex numbers or some subset thereof. It is most efficient for transformations defined as the iterative Kroneker product of an $r \times r$ matrix as in Eqn. 2 since that leads to a compact QMDD for the transformation matrix.

We have implemented a QMDD-based implementation of explicit matrix multiplication. It is also possible to implement a form of fast transform on a DD but we do not anticipate that this will lead to a significantly faster implementation since the QMDD for the transformation matrices are linear and essentially direct the computation in effectively the same way a fast implementation would. The use of a compute table avoids duplicate computation. We will however examine this in depth to ensure our intuition is correct. Formal analysis of the computational complexity and memory use of our approach is ongoing particularly with respect to BDD approaches for the binary case. The analysis is complicated by the use of compute and computation table techniques which are highly function dependent.

The examples presented here are preliminary. Our ongoing work will involve testing the approach on appropriate benchmark functions. Also, we have yet to consider the effect of variable ordering on QMDD size in the representation of spectra. It is possible, for example, that the randomly generated results here are very pessimistic and could be improved through variable reordering of the QMDD.

References

[1] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8): 677-691, 1986.

[2] E.M. 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: An International Journal*, 10(2-3):137–148, 1997.

[3] M. Fujita, P.C. McGeer, and J.C.-Y. Yang. Multi-terminal Binary Decision Diagrams: An Efficient Data Structure for Matrix Representation. *Formal Methods in System Design: An International Journal*, 10(2-3):149–169, 1997.

[4] S. L. Hurst, D.M. Miller and J.C. Muzio. *Spectral Techniques in Digital Logic*. Academic Press, New York & London, 1985.

[5] D.M. Miller. Graph Algorithms for the Manipulation of Boolean Functions and their Spectra. *Congressus Numerantium*, 57:177-199, 1987.

[6] D.M. Miller. Spectral Transformation of Multiple-valued Decision Diagrams. *Proc. IEEE International Symposium on Multiple-Valued Logic*, pp. 89-96, May 1994.

[7] D.M. Miller and R. Drechsler. Implementing a Multiple-valued Decision Diagram Package. *Proc. IEEE International Symposium on Multiple-Valued Logic*, pp. 52-57, 1998.

[8] D.M. Miller and R. Drechsler. Augmented Sifting of Multiple-valued Decision Diagram. *Proc. IEEE International Symposium on Multiple-Valued Logic*, pages 275–282, 2003.

[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*, 6 p. on CD, 2006.

[10] D.M. Miller, M.A. Thornton, and D. Goodman. A Decision Diagram Package for Reversible and Quantum Circuit Simulation. *Proc. IEEE World Congress on Computational Intelligence*, 6 p. on CD, July 2006.

[11] D.M. Miller, D.Y. Feinstein and M.A. Thornton. Variable Reordering and Sifting for QMDD. *Proc. IEEE International Symposium on Multiple-Valued Logic*, 7 p. on CD, May 2007.

[12] D. Michael Miller and Mitchell A. Thornton. *Multiple-Valued Logic: Concepts and Representations*. Morgan&Claypool Publishers, 2008.

[13] F. Somenzi. CUDD: CU Decision Diagram Package 2.4.1. vlsi.colorado.edu/~fabio/CUDD/cuddIntro.html.

[14] 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.

Table 1: Results for AND function – Rademacher-Walsh Transform (+1,-1) coding.

n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	Total / FT ops	(a)QMDD size	(b)FT size	(a) / (b)
2	3	3	5	12	17	29	8	362.50%	212	16	1325.00%
3	4	7	8	32	33	123	24	512.50%	436	32	1362.50%
4	5	10	14	60	49	181	64	282.81%	664	64	1037.50%
5	6	13	20	96	65	181	160	113.13%	892	128	696.88%
6	7	16	26	142	85	241	384	62.76%	1120	256	437.50%
7	8	19	32	192	97	315	896	35.16%	1348	512	263.28%
8	9	22	38	252	113	397	2048	19.38%	1576	1024	153.91%

Table 2: Results for OR function – Rademacher-Walsh Transform (+1,-1) coding.

n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	Total / FT ops	(a)QMDD size	(b)FT size	(a) / (b)
2	3	3	5	12	17	29	8	362.50%	212	16	1325.00%
3	4	4	9	26	29	55	24	229.17%	320	32	1000.00%
4	5	5	13	44	41	85	64	132.81%	428	64	668.75%
5	6	6	17	66	53	119	160	74.38%	536	128	418.75%
6	7	7	21	92	65	157	384	40.89%	644	256	251.56%
7	8	8	25	122	77	199	896	22.21%	752	512	146.88%
8	9	9	29	156	89	245	2048	11.96%	860	1024	83.98%

Table 3: Results for XOR function – Rademacher-Walsh Transform (+1,-1) coding.

n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	Total / FT ops	(a)QMDD size	(b)FT size	(a) / (b)
2	4	3	6	8	17	25	8	312.50%	228	16	1425.00%
3	6	4	8	16	33	49	24	204.17%	304	32	950.00%
4	8	5	10	24	49	73	64	114.06%	380	64	593.75%
5	10	6	12	32	65	97	160	60.63%	456	128	356.25%
6	12	7	14	40	81	121	384	31.51%	532	256	207.81%
7	14	8	16	48	97	145	896	16.18%	608	512	118.75%
8	16	9	18	56	113	169	2048	8.25%	684	1024	66.80%

Table 4: Results for random functions – Rademacher-Walsh Transform (+1,-1) coding.

n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	Total / FT ops	(a)QMDD size	(b)FT size	(a) / (b)
2	4	3	6	8	17	25	8	312.50%	228	16	1425.00%
3	6	7	8	34	37	71	24	295.83%	436	32	1362.50%
4	9	11	13	74	77	151	64	235.94%	692	64	1081.25%
5	15	21	17	258	161	419	160	261.88%	1196	128	934.38%
6	25	47	30	720	413	1133	384	295.05%	2548	256	995.31%
7	43	96	76	1794	773	2567	896	286.50%	5440	512	1062.50%
8	74	196	130	4458	1385	5843	2048	285.30%	10704	1024	1045.31%

Table Key

n – number of variables
F size – number of vertices in QMDD for the function
S size – number of vertices in QMDD for the spectrum
Edge Weights – number of unique edge weights
Add – number of add operations in QMDD transform
Mult – number of multiplication operations in QMDD transform
Total – total operations in QMDD transform
FT ops – operations in fast transform
QMDD size – size of QMDD for spectrum (bytes)
FT size – size of spectrum vector for fast transform (bytes)

Table 5: Comparisons for various functions and transformations.

		n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	(a)QMDD size	((b)FT size	(a) / (b)
S	AND	15	16	43	80	916	261	2.15E+08	0.00%	3172	131072	2.42%
RM	AND	15	16	16	2	30	61	2.15E+08	0.00%	736	131072	0.56%
R	AND	15	16	30	3	58	117	2.15E+08	0.00%	1368	131072	1.04%
S	OR	15	16	16	57	506	173	2.15E+08	0.00%	1616	131072	1.23%
Rm	OR	15	16	16	2	268	117	2.15E+08	0.00%	736	131072	0.56%
R	OR	15	16	16	59	506	173	2.15E+08	0.00%	1648	131072	1.26%
S	XOR	15	30	16	32	112	225	2.15E+08	0.00%	1216	131072	0.93%
RM	XOR	15	30	30	2	450	117	2.15E+08	0.00%	1352	131072	1.03%
R	XOR	15	30	30	31	320	225	2.15E+08	0.00%	1816	131072	1.39%
S	random	12	736	2596	1084	283446	50161	6377292	5.23%	131568	16384	803.03%
RM	random	12	736	724	2	17248	3293	6377292	0.32%	31888	16384	194.63%
R	random	12	736	2597	1442	299568	47633	6377292	5.44%	137340	16384	767.85%

Table 6: Results for MIN function r=3 – Chrestenson Spectrum.

n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	Total / FT ops	(a)QMDD size	(b)FT size	(a) / (b)
2	4	5	27	66	73	139	18	772.22%	852	144	591.67%
3	6	11	66	237	154	391	81	482.72%	1980	432	458.33%
4	8	20	122	681	397	1078	324	332.72%	3632	1296	280.25%
5	10	32	191	1152	316	1468	1215	120.82%	5744	3888	147.74%
6	12	47	257	1980	397	2377	4374	54.34%	8060	11664	69.10%
7	14	65	323	3120	559	3679	15309	24.03%	10628	34992	30.37%
8	16	86	389	4440	559	4999	52488	9.52%	13448	104976	12.81%
9	18	110	455	6138	640	6778	177147	3.83%	16520	314928	5.25%
10	20	137	521	8214	874	9088	590490	1.54%	19844	944784	2.10%

Table 7: Results for MAX function r=3 – Chrestenson Spectrum.

n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	Total / FT ops	(a)QMDD size	(b)FT size	(a) / (b)
2	4	5	28	66	73	139	18	772.22%	868	144	602.78%
3	6	11	64	192	136	328	81	404.94%	1948	432	450.93%
4	8	20	118	387	199	586	324	180.86%	3568	1296	275.31%
5	10	32	174	672	262	934	1215	76.87%	5472	3888	140.74%
6	12	47	230	1047	325	1372	4374	31.37%	7628	11664	65.40%
7	14	65	286	1521	388	1909	15309	12.47%	10036	34992	28.68%
8	16	86	341	2076	451	2527	52488	4.81%	12680	104976	12.08%
9	18	110	398	2736	514	3250	177147	1.83%	15608	314928	4.96%
10	20	137	454	3498	577	4075	590490	0.69%	18772	944784	1.99%

Table 8: Results for MOD-SUM function r=3 – Chrestenson Spectrum.

n	F size	S size	Edge Weights	Add	Mult	Total	FT ops	Total / FT ops	(a)QMDD size	(b)FT size	(a) / (b)
2	5	3	20	48	73	121	18	672.22%	572	144	397.22%
3	8	4	29	102	154	256	81	316.05%	800	432	185.19%
4	11	5	38	156	235	391	324	120.68%	1028	1296	79.32%
5	14	6	47	210	316	526	1215	43.29%	1256	3888	32.30%
6	17	7	56	264	397	661	4374	15.11%	1484	11664	12.72%
7	20	8	65	318	478	796	15309	5.20%	1712	34992	4.89%
8	23	9	74	372	559	931	52488	1.77%	1940	104976	1.85%