On the Optimisation of Reed-Muller Expressions

K.J. Adams J. McGregor

Intelligent Systems Engineering Laboratory
Faculty of Engineering, University of Ulster
Magee College, Londonderry BT48 7JL
United Kingdom

rj.mcgregor@ulst.ac.uk kj.adams@ulst.ac.uk

Abstract

The choice of a set of basis functions with which to represent Reed-Muller canonical forms makes less and less difference, on average, to the efficiency, as measured by the number of non-zero terms, in which they can be expressed, as the number of variables in the function increases. This is because the possible efficiency gain itself declines exponentially in the general case, independently of the basis used. We explain why this is so, and using an integrated set of software tools, including a genetic algorithm, we provide supporting evidence of the phenomenon.

1. Introduction

A combination of sum and product type operations can be used to obtain a representation for multiple-valued logic functions. The minimisation of these expressions in terms of the complexity of their representation is important, and attracts interest, because of the need to optimise their implementation on Programmable Logic Array architecture, for example [Utsumi et al 1997], [Tirumalai et al 1991].

Early attempts to compare the complexities of different algebras statistically using computer-generated data have encountered the problem of excessively long computation times to process samples of functions of reasonable size. [Stankovic et al 1998] compared two algebras for 2 and 3-variable functions for a sample size of 20,000 and [Adams et al 2002] compared five algebras using sample sizes of 20,000 2-variable functions. In [Adams et al 2003], we compared six algebras with functions of up to 5 variables with a sample size of 1,000, and found evidence that the optimisation procedures deliver less and less reduction in the number of non-zero terms, and that the bases perform similarly and converge in performance, as the number of variables increases. Recent improvements in the algorithm for exact optimisation [Stankovic et al, 2003], and the use of a genetic algorithm (in this paper), has enabled us to revisit this question and extend the function space to 6,7,8, and 9 variable functions. By examining samples from these higher variable function spaces we have found further evidence that the phenomenon of convergence is real. In this paper we present the evidence and provide an explanation as to why it occurs.

The basis functions and the corresponding transform matrices for the algebras used are given in Table 1. These are the Reed-Muller-Fourier of [Stankovic et al 1998], the algebra of [Dubrova et al 1996], Level Detector algebra described in [Adams et al 2002], and an extension of the algebra proposed by [Calingairt 1961], described in [Adams et al 1999]. These all use MOD 4 arithmetic. The fifth is the finite field arithmetic of [Green et al 1974] and lastly we include the McGregor/Adams basis introduced in [Adams et al 2002] . These latter two use GF(4) arithmetic. The basis functions are represented as matrix columns and the corresponding transform matrix is the inverse, using the appropriate arithmetic.

The structure of the remainder of this paper is as follows. Section 2 explains the theoretical background and why convergence occurs. Section 3 describes the procedure carried out and the software used. Section 4 presents the experimental results. Finally Section 5 draws conclusions.

2. Theoretical background

Let $\mathbb{Q}$ be the set {0,1,2,3}; then a totally defined function, $f : {}_{i=1}\mathbb{X}^{i=n}\, \mathbb{Q} \rightarrow \mathbb{Q}$ where $\mathbb{X}$ is the cross product, is a quaternary multi-valued *n*-variable function. Let $\mathbb{F}$ be the set of all such functions. An element of $\mathbb{F}$ is $f(x_1, x_2, x_3 \ldots x_i \ldots x_n) = [f_1, f_2, f_3 \ldots f_j \ldots f_m]^T$, which is the truth table of the function, where $x_i, f_j \in \mathbb{Q}$, and m =

Proceedings of the 34th International Symposium on Multiple-Valued Logic (ISMVL'04)
0195-623X/04 $20.00 © 2004 **IEEE**

IEEE COMPUTER SOCIETY

4^n. We can transform this function f into a function g using a 4x4 basis matrix **B**, as follows: $g = (_{i=1}\otimes^{i=n}\mathbf{B}^{-1})\ f$, where $\otimes$ represents the Kronecker product, the elements of **B** are quaternary digits, and the algebra used is any arbitrarily defined sum-of-product algebra. Bases defined in the literature and used in this paper are given in Table 1. We can extend this concept of function transformation by considering also the use of polarities. For our purposes here we define a polarity change as a permutation of the rows of the basis matrix **B**, and usually four of these are considered, corresponding to the effect of using the additive complements to the elements of Q, as defined by the associated algebra. The 4x4 permutation matrices **P** for the two algebras used (GF(4) and MOD 4) are given in Table 2. Thus a function f can be transformed into a function $g = (_{i=1}\otimes^{i=n}(\mathbf{P}^{<p_i>}\mathbf{B})^{-1})\ f$, with $p_i = 0,1,2$, or 3. Where $p_i = 0$, **P** is the identity matrix, and the function g is referred to as the zero-polarity transform of f.

If we denote by <p> the polarity vector $<p_1, p_2, p_3 \ldots p_i \ldots p_n>$, we now define $\mathbf{X}^{<p>} = {}_{i=1}\otimes^{i=n}(\mathbf{P}^{<p_i>}\mathbf{B})^{-1}$, so that $g = \mathbf{X}^{<p>}f$, and we let $\mathsf{G}(f) \subseteq \mathsf{F}$ be the set of functions for which there exists a polarity vector <p> such that $g = \mathbf{X}^{<p>}f$. This defines a partition of F where each equivalence class is that set of functions which can be transformed into one another using the polarities of the basis. We note that for any f, the maximum size of this set, $|\mathsf{G}(f)|$, is 4^n.

We can now define the optimisation procedure as a function $op : \mathsf{F} \rightarrow \mathsf{F}$ where $op(f) \in \mathsf{G}(f)$ and is a function with the minimum possible number of non-zero values. This is an injection from F to F. Since $|\mathsf{F}| = 4^m$ and the maximum size of $\mathsf{G}(f)$ is 4^n, the minimum size of the image of op, Im(op), is $4^m/4^n$, or 4^{m-n}. The theoretically ideal basis **B** is the one for which Im(op) achieves this minimum, and contains no function which has more non-zero values than a function not in Im(op).

For a given number of variables, n, there are $m = 4^n$ possible terms in a function, and 4^m possible functions. The number of these that have t terms is ${}^mC_t * 3^t$ since there are ${}^mC_t$ ways of choosing the terms, and each of these terms can have one of three possible coefficients (1,2, or 3). This is the Binomial distribution, centred around a mean, or expected value E_n, of $(¾)*4^n$. A illustrative graph of this distribution, for the case of 4-variable functions is shown in Figure 1. The formula for the mean is derived from elementary statistical theory, but we can see it intuitively by noting that since the 4 possible coefficients 0,1,2, and 3 occur equally often, and that three of them are non-zero, then we can expect, on average, ¾ of the maximum number of terms, 4^n. By examining the frequency distribution of the numbers of functions versus the number of terms in the function (t), we can calculate the average number of terms for |Im(op)| functions, starting with the those with the lowest number of terms, and continuing with increasing numbers of terms until we have 4^{m-n} = |Im(op)| functions. Table 3 shows the values obtained by this process. In Table 3 we can see that for 8 variables the possible minimum number of terms (on average) rises to 99% of the expected number of terms, so that, on average, only a 1% reduction in the number of terms is available for a polarity procedure where there are four polarities per variable. A graph of this reduction is shown in Figure 2.

Thus there is an inherent upper bound to the available savings in numbers of non-zero terms available by employing a polarity procedure for optimising functions. This bound is low, and declines exponentially as the number of variables increases.

3. Experiments and software

3.1. Exact optimisation

In [Jankovic et al 2003] an optimisation technique making use of the dual polarity property is described. Essentially it exploits the fact that when the number of terms needed to represent a function is known for a given polarity vector for the variables, then the number of terms needed for a polarity vector which differs by only one value can be efficiently calculated. We have implemented an improved variation on this by starting with the zero-polarity representation and stepping through the polarity vectors in a Gray code order to search for the vector corresponding to the representation with the minimum number of terms. In fact the difference in the numbers of terms as we move from one set of polarities to the next does not depend on the function and can be pre-calculated and stored in a look-up table.

An integrated suite of C programs used to optimise functions expressed in an arbitrary basis and algebra is described in [Adams et al 2002] and [Adams et al 2003]. We implemented the above technique and incorporated it into our system. Using C running under LINUX on a

IEEE COMPUTER SOCIETY

1GHz PC with 128Mb of memory the times taken to optimise functions are shown in the first column of Table 4. The times are independent of the number of terms in the function, and independent of the algebra and basis used. We found, for example, that the time to optimise exactly any 7-variable function is 43 seconds as opposed to figures ranging from 251.32 to 299.28 seconds as given in [Stankovic et al, 2003] running on a similar hardware configuration.

At 9 variables the time taken to optimise a function (about 5 hours) becomes prohibitive, but in addition to that, a space limitation, limiting the number of variables, becomes apparent. In the case of 10 variables, a fully defined function in a 10 dimensional array is needed, implying 4^{10} entries, and this is infeasible without a strategy which would involve prohibitive amounts of Disk I/O which is very time consuming. Accurate and comfortable exploration of function spaces beyond 9 variables is therefore problematic for fully defined functions both for space *and* time considerations, and would require a different approach.

The computer code is available at **http://spring.infm.ulst.ac.uk/mvl/**, which also has other supporting material.

3.2. Optimisation using a genetic algorithm

The Genetic Algorithm used is a basic model. For an *n*-variable function, a gene is an *n*-length vector of numbers ranging from 0 to 3, which represent the polarities of the basis functions in which the variables are expressed, $<p_1, p_2, p_3 \ldots p_n>$. The fitness of the gene is the number of zero coefficients in the resulting canonical expression of the function using these polarities. We form an initial pool of *poolSize* randomly constructed genes and calculate their fitness as the number of zero coefficients found by searching through all the polarities in the manner described above. Using a roulette wheel selection method we randomly choose a pair of genes and select (randomly) a chromosome at which to split them. We top and tail the genes at this chromosome to produce two children, and add them to a subsidiary pool. We repeat this process *nCrossover* times. We then randomly choose a child gene from this subsidiary pool and make a random change in one of its (randomly chosen) chromosomes. We do this *nMutation* times. We note in passing that a child could remain unchanged, and that a child could also be changed more than once. We then calculate the fitness of the resulting children and pick from the original pool and the subsidiary pool the fittest *poolSize* genes. We repeat this process *nGeneration* times.

The parameters available for variation are the size of the gene pool (*poolSize*), the number of crossovers to be used (*nCrossover*) and the number of mutations where a random chromosome from a randomly chosen gene is randomly changed (*nMutation*). During the testing phase the algorithm was run until the known or estimated minimum was found; subsequently the GA was run for a suitable fixed number of generations.

4. Experimental Results

4.1. Exact optimisation for 5-variable functions

Using the exact optimisation technique described above, we constructed 20,000 random 5-variable functions and found the minimum number of coefficients needed to represent them using both the Green/Taylor basis (GT) and the Level Detector basis (LD). The results were 723 for GT and 726 for LD. The expected mean (E_5) over the entire function space is 768 and this proved to be the average number of terms in the zero-polarity transform of the initial functions. Thus the 'savings' as a proportion of E_5 are 5.9% and 5.5% respectively, and this is just below the theoretical average maximum percentage available of 6% (see Table 3). This provides further evidence that there is a low upper bound to the savings available, and that consequently the reduction achieved by different bases is approximately the same.

4.2. Genetic algorithm performance vs exact method

4.2.1. Speed of the genetic algorithm

The time for exact optimisation of functions increases exponentially as the number of variables increases. A procedure such as a genetic algorithm is needed if any reasonable numbers of functions are to be examined. In order to demonstrate the efficiency in speed terms we first optimised numbers of random functions exactly (for the Green/Taylor basis), and then used the genetic algorithm to try to find this exact optimum. In all the cases that we tried the genetic algorithm was successful, and the average CPU seconds required over the various

IEEE COMPUTER SOCIETY

samples of functions is given in the latter columns of Table 4. On average, over 2,000 functions, the algorithm was able to optimise a 7-variable function, for example, in 14.2 seconds (as opposed to 43 seconds using the exact technique). Our conclusion is that sufficient speed gains to examine reasonable numbers of functions can be achieved without loss of accuracy by using the genetic algorithm.

4.2.2. 5-variable functions

In order to demonstrate the effectiveness of the genetic algorithm we ran it against 5,000 randomly generated 5-variable functions which we first optimised using the exact method referred to in Section 3. By this means we obtained a selection of functions where the minimum number of non-zero coefficients needed to represent them was known. We then used the genetic algorithm to see if we could arrive back at this minimum number of terms by finding the correct polarity vector. We choose a gene pool size of 24, with a crossover rate of 10 and a mutation rate of 10 per generation, and we ran the algorithm for 250 generations. We did this for two bases, Green/Taylor and Level Detector.

These results are shown in Table 5. We see that 60.1% (GT) and 80.4% (LD) of functions are optimised within 25 generations, with over 99% being optimised within 250 in each case. The average number of generations to optimise a 5-variable function is 33.7 in the case of GT, and 18.9 in the case of the LD. Our conclusion is that the genetic algorithm can optimise 5-variable functions accurately and quickly.

4.2.3. 6, 7, 8, and 9-variable functions

As the number of variables increases the time taken to optimise functions increases also. As alluded in Section 3, for functions of 6 and more variables, exact optimisation of large numbers of functions becomes infeasible. For these cases, random functions of specific numbers of (unique) terms were generated and this therefore provided an upper bound for the minimum number needed to represent the function. Thus we tested the ability of the genetic algorithm to find the number of terms originally used to generate the function, rather than the actual, exact minimum.

The results are shown in Table 6. They indicate, for example, that for a 7-variable function, which is pre-constructed from a fixed number of terms, the genetic algorithm can find that number of terms in 10.2 seconds, on average (calculated over a random sample of 2,000 functions). The algorithm performs better at this task than the results shown for exact optimisation in Table 4, since it is not finding the actual minimum, but an upper bound to that minimum. We conclude that the genetic algorithm can find functions where there is a significant reduction in the number of terms, and can do so efficiently.

4.3. Results found by genetic algorithm

We ran the genetic algorithm to optimise randomly generated sets of 5, 6, 7, and 8-variable functions for the six bases described in Section 1. We used a pool size of 24, and crossover and mutation rates of 10 per generation, and we ran for 40 generations. Excessive computation times (ranging from 5.4 hours for the 5-variable experiment, to 47.7 hours for 8 variables) precluded the use of greater sample sizes, or a larger number of generations. The results are shown in the four parts of Table 7. Each of the four parts can be interpreted similarly and we now discuss the 6-variable case to explain the results.

For a random 6-variable function, the expected (or average) number of terms will be $E_6 = ¾*4^6 = 3{,}072$. From the discussion in Section 2 we have that the theoretical minimum number of non-zero coefficients needed to represent 6-variable functions, on average, is 2,967 (See Table 3). That is to say, *on average*, using an ideal basis and arithmetic, we could expect to reduce the figure of 3,072 terms to a figure of 2,967 terms. This figure is independent of the basis used, and is a consequence of the properties of the Binomial distribution, which corresponds to the distribution of the numbers of non-zero terms in quaternary multiple-valued functions.

In Table 7 we see that the average minima found by the genetic algorithm for the six different bases are (a) close to this figure, ranging from 0.47% to 0.61% above it, and (b) close to each other, ranging from 2981 to 2985. The other parts of Table 7 present a similar picture and can be interpreted in the same way. They provide confirming evidence of our conclusions.

5. Conclusions

The first conclusion to be drawn is that optimisation of functions using polarities, when taken over the entire function space, delivers

IEEE COMPUTER SOCIETY

less and less reduction of the numbers of non-zero coefficients as the number of variables increases, diminishing to an average of 1% with 8-variable functions. This phenomenon is a consequence of the statistical properties of the Binomial distribution, and is confirmed by the experimental evidence here presented. Secondly, it follows that choosing different bases makes no difference, and our evidence supports this conclusion also.

Essentially the proportion of simple functions (those with fewest non-zero coefficients) diminishes exponentially as the number of variables decreases and therefore there is an exponentially diminishing opportunity to significantly reduce the number of terms in a function by searching among the different polarities. The strategy of randomly sampling the entire function space, testing different bases and polarity procedures, is therefore of little value to distinguish between algebras.

It remains to examine, perhaps, classes of functions, such as symmetric, monotone, or threshold functions for example, to see if the picture is any different. As well, a comprehensive set of carefully constructed benchmark functions (which could be quite large in size), incorporating examples of all of the features of interest in multiple-valued functions might be much more useful as a tool to investigate the complexities and features of possible different algebras.

References

Adams, K.J., Campbell, J.G., Maguire L.P., Webb J.A.C. (1999) State assignment techniques in multiple valued logic. *Proc. 29th Int. Symp. on Multiple-Valued Logic, 220-225*, Freiburg. (IEEE)

Adams, K.J., McGregor, J. (2002) Comparison of Different Features of Quaternary Reed-Muller Canonical Forms and Some New Statistical Results. *Proc. 32nd Int. Symp. On Multiple-Valued Logic, 83-88*, Boston. (IEEE)

Adams, K.J., McGregor, J. (2003) New Information on the Effectiveness of Different Reed-Muller Algebras on the Representation of Quaternary Functions. *Proc. 33rd Int. Symp. On Multiple-Valued Logic, 33-39*, Tokyo. (IEEE)

Calingairt P. (1961) Switching Function Canonical forms Based on Commutative and Associative Binary Operations. *Trans. Amer. Inst. Elect. Eng. Vol 79 Jan, 808-814* (Springer)

Dubrova E.V., Muzio, J.C. (1996) Generalized Reed-Muller Canonical Form for Multiple-valued Algebra. *Multiple Valued Logic, An International Journal, 1, 65-84*. (Gordon and Breach, Netherlands)

Green, D.H., Taylor, I.S. (1974) Modular Representation of Multiple-valued Logic Systems. *Proc. Of the IEE, Vol. 121, 424-429*. (IEE)

Jankovic, D., Stankovic, R.S., Moraga, C. (2003) Optimisation of GF(4) Expressions Using the Extended Dual Polarity Property. *Proc. 33rd Int. Symp. On Multiple-Valued Logic, 50-55*, Tokyo. (IEEE)

Spiegel, M.R. (1972) *Theory and Problems of Statistics*. McGraw-Hill.

Stankovic R.S., Jankovic, D., Moraga C. (1998) Reed-Muller-Fourier versus Galois Field Representation of Four-valued Logic Functions. *Proc. 28th Int. Symp. on Multiple-Valued Logic, 186-191*. Fukuoka. (IEEE)

Tirumalai P., Butler J. (1991) Minimisation Algorithms for Multiple-Valued Programmable Logic Arrays. *IEEE Trans. on Comp. Vol 40, No. 2, 167-177*.

Utsumi T., Kamiura N., Hata Y., Yamato K. (1997) Multiple-valued Programmable Logic Array with Universal Literals, *Proc. 27th Int. Symp. Multiple Valued Logic*, 163-169. (IEEE)

(DM) Dubrova/Muzio (MOD 4)								
Basis **B**					Inverse $\mathbf{B}^{-1}$			
1	0	0	0		1	0	0	0
1	1	0	0		3	1	0	0
1	0	1	0		3	0	1	0
1	0	0	1		3	0	0	1
(GT) Green/Taylor (GF(4))								
Basis **B**					Inverse $\mathbf{B}^{-1}$			
1	0	0	0		1	0	0	0
1	1	1	1		0	1	3	2
1	2	3	1		0	1	2	3
1	3	2	1		1	1	1	1
(CQ) Calingairt-Q (MOD 4)								
Basis **B**					Inverse $\mathbf{B}^{-1}$			
1	0	0	0		1	0	0	0
1	1	0	0		3	1	0	0
1	0	1	0		3	0	1	0
1	1	1	1		1	3	3	1
(LD) Level Detector (MOD 4)								
Basis **B**					Inverse $\mathbf{B}^{-1}$			
1	0	0	0		1	0	0	0
1	1	0	0		3	1	0	0
1	1	1	0		0	3	1	0
1	1	1	1		0	0	3	1
(RM) Reed/Muller/Fourier (MOD 4)								
Basis **B**					Inverse $\mathbf{B}^{-1}$			
3	0	0	0		3	0	0	0
3	1	0	0		3	1	0	0
3	2	3	0		3	2	3	0
3	3	1	1		3	3	1	1
(MA) McGregor/Adams (GF(4))								
Basis **B**					Inverse $\mathbf{B}^{-1}$			
0	2	0	3		1	3	2	2
1	3	1	1		2	3	0	1
2	0	1	2		3	2	2	2
3	0	3	2		3	2	0	3

Table 1 Basis functions and their inverses

GF(4)																
$\mathbf{P}^{<0>}$					$\mathbf{P}^{<1>}$					$\mathbf{P}^{<2>}$					$\mathbf{P}^{<3>}$			
1	0	0	0		0	1	0	0		0	0	1	0		0	0	0	1
0	1	0	0		1	0	0	0		0	0	0	1		0	0	1	0
0	0	1	0		0	0	0	1		1	0	0	0		0	1	0	0
0	0	0	1		0	0	1	0		0	1	0	0		1	0	0	0

MOD 4																
$\mathbf{P}^{<0>}$					$\mathbf{P}^{<1>}$					$\mathbf{P}^{<2>}$					$\mathbf{P}^{<3>}$			
1	0	0	0		0	1	0	0		0	0	1	0		0	0	0	1
0	1	0	0		0	0	1	0		0	0	0	1		1	0	0	0
0	0	1	0		0	0	0	1		1	0	0	0		0	1	0	0
0	0	0	1		1	0	0	0		0	1	0	0		0	0	1	0

Table 2 Polarity permutation matrices

IEEE COMPUTER SOCIETY

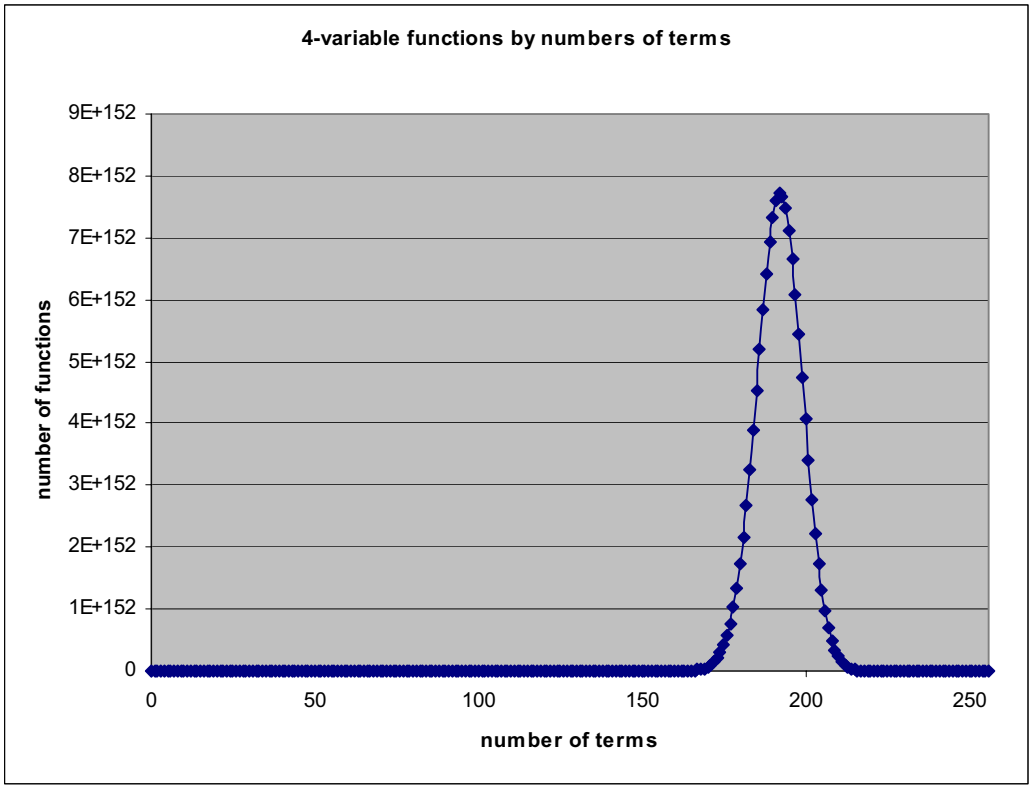

Figure 1 Distribution of 4-variable functions by the number of non-zero terms

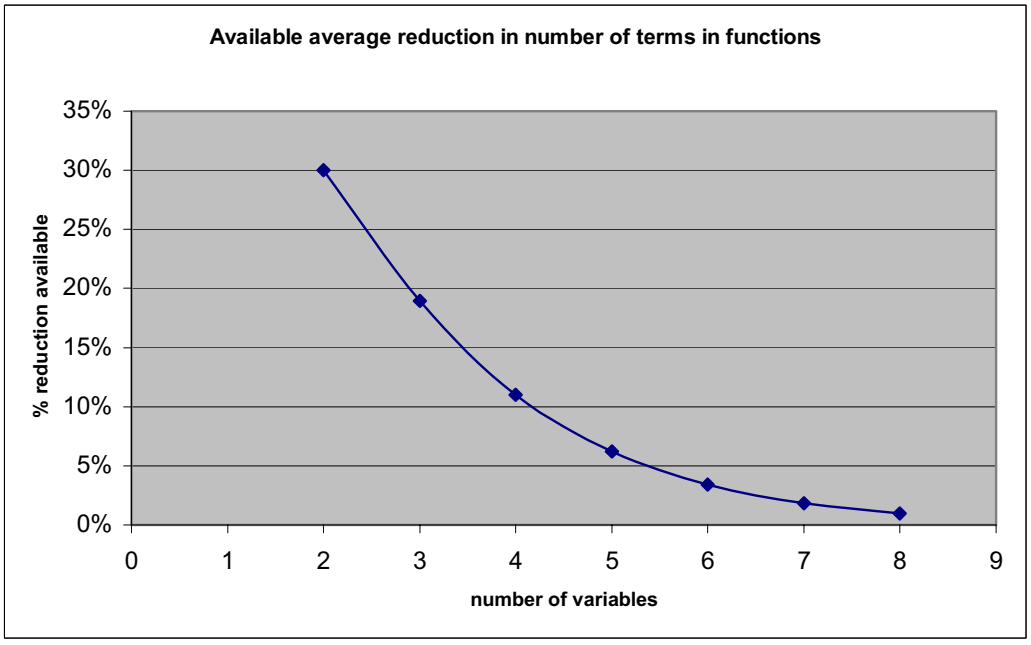

Figure 2 Expected number of terms compared with average minimum number of terms for *n*-variable functions (Table 3)

0195-623X/04 $20.00 © 2004 **IEEE**

IEEE COMPUTER SOCIETY

n	Expected number of terms E_n	Average possible minimum	Average as % of E_n	Average % reduction available in number of terms
2	12	8.4	70%	30%
3	48	39	81%	19%
4	192	171	89%	11%
5	768	720	94%	6%
6	3072	2967	97%	3%
7	12288	12060	98%	2%
8	49152	48664	99%	1%

Table 3 Expected number of terms compared with average minimum number of terms for *n*-variable functions (Figure 2)

n	Exact Optimisation (secs)	Optimisation using GA (secs)	Sample Size
5	0	0.25	20,000
6	3	2.2	2,000
7	43	14.2	2,000
8	1007	111	400
9	18070	721	10

Table 4 Time need for optimisation of n-variable functions over four polarities (*poolsize*=24, *nCrossover*=10, *nMutation*=10)

Number of generations	GT GF(4)	Cumulative % of functions optimised	LD MOD 4	Cumulative % of functions optimised
25	3006	60.1%	4022	80.4%
50	841	76.9%	519	90.8%
75	469	86.3%	201	94.8%
100	257	91.5%	113	97.1%
125	160	94.7%	55	98.2%
150	105	96.8%	35	98.9%
175	63	98.0%	22	99.3%
200	28	98.6%	11	99.6%
225	31	99.2%	5	99.7%
250	14	99.5%	7	99.8%
Not found in 250 generations	26	0.5%	10	0.2%
Average number of generations	33.7		18.9	

Table 5 Effectiveness of GA in optimising a random sample of 5,000 5-variable functions for Green/Taylor and Level Detector bases (*poolsize*=24, *nCrossover*=10, *nMutation*=10)

Number of Variables	Optimisation using GA (secs)	Sample Size
6	1.7	2,000
7	10.2	2,000
8	58	1,000
9	431	500

Table 6 Time needed to find numbers of terms in functions randomly constructed from fixed numbers of terms ranging through 100, 200, … 1,000 (*poolsize*=24, *nCrossover*=10, *nMutation*=10)

IEEE COMPUTER SOCIETY

	5 variables (2,000 functions)			
Basis	Average minimum found	Theoretical average minimum	Difference	%
GT	725	720	5	0.69%
LD	729	720	9	1.25%
CQ	726	720	6	0.83%
DM	725	720	5	0.69%
MA	725	720	5	0.69%
RM	725	720	5	0.69%
Computation time: 5.4 hours				

	6 variables (2,000 functions)			
Basis	Average minimum found	Theoretical average minimum	Difference	%
GT	2982	2967	15	0.51%
LD	2985	2967	18	0.61%
CQ	2981	2967	14	0.47%
DM	2981	2967	14	0.47%
MA	2984	2967	17	0.57%
RM	2981	2967	14	0.47%
Computation time: 27.7 hours				

	7 variables (500 functions)			
Basis	Average minimum found	Theoretical average minimum	Difference	%
GT	12105	12060	45	0.37%
LD	12097	12060	37	0.31%
CQ	12098	12060	38	0.32%
DM	12107	12060	47	0.39%
MA	12107	12060	47	0.39%
RM	12097	12060	37	0.31%
Computation time: 35.8 hours				

	8 variables (100 functions)			
Basis	Average minimum found	Theoretical average minimum	Difference	%
GT	48772	48664	108	0.22%
LD	48760	48664	96	0.20%
CQ	48769	48664	105	0.21%
DM	48767	48664	103	0.21%
MA	48779	48664	115	0.23%
RM	48765	48664	101	0.21%
Computation time: 47.7 hours				

Table 7 Average minimum numbers of coefficients achieved for random functions by the GA compared with theoretical (average) minimum possible (*poolsize*=24, *nCrossover*=10, *nMutation*=10, *nGeneration*=40)

IEEE COMPUTER SOCIETY