Methods for Calculating Autocorrelation Coefficients

J. Rice & J. C. Muzio

Dept. of Computer Science
University of Victoria
Victoria, BC

Canada

Autocorrelation Coefficients

- measure of similarity of the function to itself shifted by a given amount
- uses in optimization and synthesis
- limited usefulness due to complexity of calculation

Notes on Intro slides:

- blah

Calculating the Autocorrelation Coefficients

$$B(u) = \sum_{v=0}^{2^n-1} f(v)f(v \oplus u)$$

$$\mathbf{B} = \frac{1}{2^n}\mathbf{T}^n \cdot \mathbf{R}^2$$

Notes on Autocorrelation Slides

- these can also be done using the $+1, -1$ encoding of the output vector; we call the resulting coefficients $C(u)$
- $\mathbf{R}^2$ is the spectral vector $\mathbf{R}$ with each element squared
- for multiple output functions sum individual coeffs for each fn
- using the $f(v) \cdot f(v \oplus u)$ equation requires 2^{2n} operations
- using the T^n equation requires 2^{3n+2} operations

Example - Part 1

$$f(x) = \overline{x}_1 + x_2 x_3$$

$x_3 x_2 x_1$	$f(x)$
000	1
001	0
010	1
011	0
100	1
101	0
110	1
111	1

$$\begin{aligned}
\mathbf{B} &= \tfrac{1}{2^n} \cdot \mathbf{T}^n \cdot \mathbf{R}^2 \\
&= \tfrac{1}{2^3} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} 25 \\ 9 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} 5 \\ 2 \\ 4 \\ 2 \\ 4 \\ 2 \\ 4 \\ 2 \end{bmatrix}
\end{aligned}$$

Example - Part 2

$$f(x) = \overline{x}_1 + x_2 x_3$$

$x_3x_2x_1$	$f(x)$
000	1
001	0
010	1
011	0
100	1
101	0
110	1
111	1

$$B(u) = \sum_{v=0}^{2^n-1} f(v)f(v \oplus u)$$

$$\begin{aligned}
B(000) &= f(000) \cdot f(000 \oplus 000) \;+ \\
&\quad\; f(001) \cdot f(001 \oplus 000) \;+\; ... \;+ \\
&\quad\; f(111) \cdot f(111 \oplus 000) \\
&= 1+0+1+0+1+0+1+1 \\
&= 5
\end{aligned}$$

$$\begin{aligned}
B(001) &= f(000) \cdot f(000 \oplus 001) \;+ \\
&\quad\; f(001) \cdot f(001 \oplus 001) \;+\; ... \;+ \\
&\quad\; f(111) \cdot f(111 \oplus 001) \\
&= 0+0+0+0+0+0+1+1 \\
&= 2
\end{aligned}$$

$$B(010) = 4$$

$$B(011) = 2$$

$$B(100) = 4$$

$$B(101) = 2$$

$$B(110) = 4$$

$$B(111) = 2$$

Technique 1 - Disjoint Cubes

Algorithm:

- ensure that the input is a list of disjoint cubes
- sum the contribution of each cube for each coefficient

- the contribution is based on the number of don't cares in each cube
- the method takes advantage of two things to increase speed:
  1. only the on-set of the function(s) are given, and
  2. don't cares are used to reduce this list even further

Notes on Disjoint Cubes Technique

- disjoint cube list is generated using technique from Falkowski *et. al.* in Midwest Symposium on Circuits and Systems 1990 ("A Fast Computer Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions")

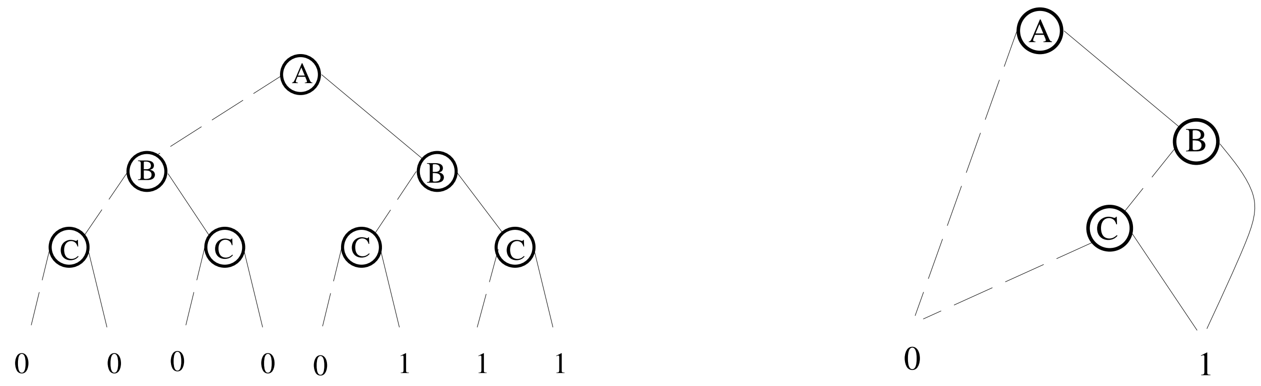
- technique based on previous technique for calculation of Rademacher-Walsh coefficients, introduced by Falkowski *et. al.* in EDAC 1992

- is very similar to method described by Varma and Trachtenburg in *Efficient Spectral Techniques for Logic Synthesis*, ed. Sasao, 1993 and in Trans. on CAD 1989

- algorithm: for each coefficient e.g. 0001, for each cube:
  - if - is in position i then contribution is 2^d where d is the number of don't cares (-) in the cube
  - otherwise compare the cube with each other cube; for each cube where the orig. cube is the "same" as the comparison cube with only the variable at position i changed, then the contribution is is $2 * 2^{d-same}$ where $d - same$ is the number of don't cares in the same positions in both cubes

- requires $m * (n * \frac{p*(p+1)}{2})$ operations, where n is the num of variables, m is the num of outputs, and p is the number of disjoint cubes (initial overhead is incurred in building the disjoint cube list)

Technique 2 - simple ROBDD-based method



The BDD and ROBDD for $f = AB + AC$

- takes advantage of the fact that a ROBDD need not store all 2^n truth table values

Notes on Technique 2

- very simple method: create the ROBDD for the function
- create the ROBDD for the function shifted by the appropriate amount
- compare the outputs
- running time?

Technique 3 - recursive ROBDD method

- recursively compare parts of the ROBDD to itself to determine each coefficient

- current implementation requires reordering of the variables for each coefficient calculated; this could be avoided

Notes on Technique 3

- running time?
- example?

Experimental Results

method	**number of benchmarks that failed**	**percentage within 5 sec of lowest time**	**avg time over all benchmarks**
disjoint cubes	4	87 %	43.33 sec
BDD-simple	3	100 %	0.72 sec
BDD-recursive	10	68 %	24.52 sec
brute force	64	60 %	98.90 sec

Notes on Experimental Results

- 130 benchmarks were tested
- reasons for failure included running out of memory, taking too long (*e.g.* over 10 hours of CPU time)
- internal failure
- brute force is an implementation of the simple definition of the autocorrelation equation

Conclusions & Future Work

- blah

Notes on Conclusions & Future Work

- blah