BDD 最小化問題之改良演算法

林順喜
國立台灣師大資訊工程研究所
台北市汀州路四段八十八號
linss@csie.ntnu.edu.tw

魏君任
國立台灣師大資訊教育研究所
台北市和平東路一段 162 號
cjwei@ice.ntnu.edu.tw

摘要

自二元決策圖（Binary Decision Diagram）[1]用來表示布林函數的概念和其相關運算的演算法被提出以來，就獲得許多研究者的採用。然而要找出表達函數的最佳變數順序，計算複雜度相當高[9]，於是在過去的十年間，BDD 最小化的問題受到許多人的討論和研究。雖然如此，數十到數百個輸入的電路至今仍無實用的演算法能夠求出更為精確的解。本論文針對這樣的問題，以亂數演算法，將過去只能求出近似解的篩選（sifting）演算法[3]的計算能力，提升到相當於求出精確最佳解演算法一樣，效能上也有極佳的表現；benchmark LGSynth91 中輸入數小於 500 以下的電路，除了 C6288.blif 以外，都已經找到了很好的解。而我們提出的新方法更容易平行化，以符合更大型電路求解的需求。

關鍵詞：Minimization, Binary Decision Diagram, Randomized Algorithm

一、BDD 介紹與問題定義

設一布林函數 f（或邏輯電路）的 n 個變數，以 $x_1, x_2, \cdots, x_n$ 表示。當其中某一變數 x_i 以常數 b，b ∈ {0, 1}，取代（稱為 restriction of f，或 cofactor）[14]，則表示為 $f|_{x_i=b}$，也就是：

$$f|_{x_i = b}(x_1, x_2, \cdots, x_n) = f(x_1, x_2, \cdots, x_{i-1}, b, x_{i+1}, \cdots, x_n)$$

依此方法，原函數式 f 可表示為 Shannon 分解（decomposition）[10]：

$$f = x_i \cdot f|_{x_i = 1} + \overline{x_i} \cdot f|_{x_i = 0}$$

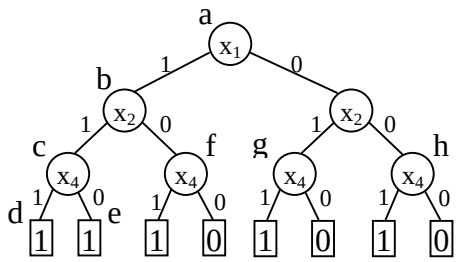

圖 1-1、全部展開（未化簡）的 BDD

若將一個布林函數的 n 個變數依 Shannon 分解展開，轉換成 BDD，那麼會得到一個完整二元樹，如圖 1-1，其中方形的端點稱為終端點，端點 a 到端點 d 代表 $x_1 \cdot x_2 \cdot x_4 = 1$，端點 a 到端點 e 則代表 $x_1 \cdot x_2 \cdot \overline{x_4} = 1$，各個最小項對應的輸出值依其欲表示的函數而定，此處以函數 $x_1 \cdot x_2 + x_4$ 為例指定各終端點之值。

左右子樹相同的端點和同構的子圖都可被省略，稱為化簡（reduced）。例如圖 1-1 中端點 c 可省略，由端點 b 的左子樹直接連接到值為 1 的終端點，而端點 f、g 和 h 為同構，故只需一個。同樣的，代表 0 和 1 的終端點也只各需一個即可。於是經過化簡後，只需要 5 個端點即可表示原函數（如圖 1-2）。此 BDD 的大小（size）就是 5。

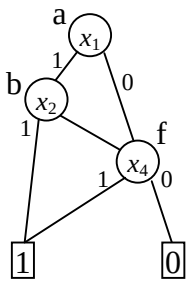

圖 1-2、化簡（reduced）的 BDD

任何一個布林函數，無論其函數式是否已化簡、或以何種方式呈現，只要是以同一變數順序，轉換成 BDD，化簡後都會是標準的形式（canonical form）。BDD 能夠以標準的形式表示，而且能有效率的運算的優點，讓它被廣泛的運用在邏輯設計驗證（logic design verification）、測試產生（test generation）、錯誤模擬（fault simulation），和邏輯合成（logic synthesis）等方面[11]。

但是 BDD 表示法有一個嚴重的問題[7]：舉例而言，下面由左至右二個圖分別是以< $x_1, x_2, x_3, x_4, x_5, x_6$ >和< $x_1, x_3, x_5, x_2, x_4, x_6$ >的變數順序表示函數 $x_1x_2 + x_3x_4 + x_5x_6$ 的 BDD：

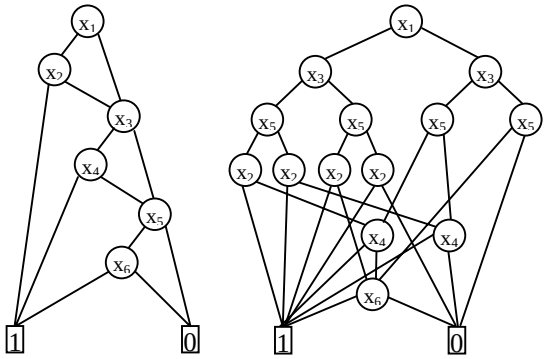

圖 1-3、變數順序對 BDD 大小的影響

這個 n = 6 個變數的布林函數以< x_1, x_2, x_3, x_4, x_5, x_6 >的變數順序做轉換時，得到的 BDD 大小為 8 = n + 2（個端點），相當於 O(n)的數量，但是若以< x_1, x_3, x_5, x_2, x_4, x_6 >的變數順序表示原函數時，將會得到含有 $16 = 2^{\lceil n-\log n \rceil}$ 個端點的 BDD，其大小（端點數量）變成指數大小。在處理上就極耗時間及空間。

二、我們所提出的亂數演算法

在本章中，我們將提出一個全新的概念，用不同於以往的方式求解二元決策圖（BDD）最小化（minimization）的問題，實驗的數據將顯示這個新方法會有極佳的效能。

整個過程由數個實驗組成。全部實驗分成三個部分：首先，第一個實驗要試試看新的方法是否有效；如果新方法的確在效能上有所增進的話，則進行第二個實驗，將所有已知最佳解的電路，都以新方法實際求解，推算出求取最佳解所需的計算成本；第三個實驗則是以第二個實驗所得的結果，嘗試將 benchmark 中尚未有最佳解的電路，一一求解。

實驗一：

我們以 benchmark LGSynth 91 中的電路 cordic.blif（benchmark 中，已知最佳解的一個電路）為例子，說明求解的過程（.blif 格式的檔案是 benchmark LGSynth91 中的一種電路描述格式，用來描述邏輯電路。LGSynth91 則可由 International Logic Synthesis Workshop[15] 取得）：

1. 程式讀入電路中各輸入的代號，cordic.blif 這個電路有 23 個輸入，各以 a6, a4, a3, a2, a5, v, x0, x1, x2, x3, y0, y1, y2, y3, z0, z1, z2, ex0, ex1, ex2, ey0, ey1, ey2 為代表的符號，而這個代號就是這個電路所代表之布林函數的變數名稱，其順序就用來作為產生 BDD 時的初始變數順序。
2. 得到初始的變數順序之後，程式就以此順序，以篩選演算法[8]，計算所能得到最小化的 BDD。
3. 記錄這個新 BDD 的大小（端點數量）、耗費的計算時間…等。
4. 比較新 BDD 的大小，是否為最佳解。若已得到最佳解的話，就記錄下最佳解的變數順序。若不是最佳解的話，就隨機的（以亂數產生器）產生一個新的變數順序，重複步驟 2 ~ 4。
5. 對於每一個電路，重複的隨機產生變數順序，一直到篩選演算法得到最佳解為止，稱為一次的實驗。每個電路都重複的做 100 次實驗，記錄每一次的實驗中，到底產生了多少個變數順序（作為化簡演算法的初始變數順序）才得到最佳解、和全部的計算花費了多少的時間。

在（步驟 5 提到的）100 次實驗中，每一次實驗我們都用一個不同的亂數種子（seed），第一次實驗的 seed 是就 1，第二次實驗的 seed 是 2，第三次實驗的 seed 是 3，…，第 k 次實驗的 seed 是就 k；藉由不同的 seed 值的亂數改變變數順序，也方便後續研究時能夠重覆實驗做確認。

以下是我們實驗過的一個電路，cordic.blif，的計算過程：

程式先以電路的變數順序作為初始的變數順序：

初始變數順序：

a6 a4 a3 a2 a5 v x0 x1 x2 x3 y0 y1 y2 y3 z0 z1 z2 ex0 ex1 ex2 ey0 ey1 ey2

經過 0.32 秒的計算，以此順序所能計算得到的最小化 BDD 的大小為 43（個端點），大於已知的 42（個端點）。於是隨機的重排變數順序：

第二個初始變數順序為：

a3 x2 z2 y2 a4 z1 y0 y3 ey0 ex0 z0 a2 ey2 x3 a5 ex1 ex2 a6 ey1 v y1 x0 x1

以此新變數順序作為初始變數順序，計算 0.31 秒後，得到最小化的 BDD 是以 a2 a4 a3 a6 a5 z2 z1 z0 y2 y0 y3 y1 x2 x3 x0 x1 ey0 ey2 ey1 v ex0 ex1 ex2 為變數順序，大小為 42，已經是最小值了。也就是說，在這一次的實驗中，程

式產生了二個變數順序（作為初始變數順序），才使得篩選演算法得到最佳解。全部的計算時間是 0.63 秒。

我們接著改變亂數產生器的種子（seed），重做一次實驗，看看結果如何：
改變 seed 值之後，產生出的變數順序為：

x1 v z2 y1 y2 y0 ey2 a2 x2 y3 a6 z0 a4 z1 ex0 x0 a3 ex1 x3 a5 ex2 ey1 ey0

以此新變數順序作為初始變數順序，在 0.31 秒的計算後，得到最小化的 BDD 大小為 42(以 a2 a3 a4 a6 a5 y1 y2 y0 y3 x1 x2 x0 x3 z2 z0 z1 ex0 ex1 ex2 v ey2 ey1 ey0 為變數順序)，是已知的最小值。所以在這一次的實驗中，篩選演算法依照亂數產生器產生的初始變數順序在第一次就得到最佳解了。在這裡，我們注意到 BDD 的最佳變數順序可能不只一種！

再以不同的 seed，進行第三次實驗：
這一次的實驗中，第一次的變數順序是：

y1 a4 a5 y2 x2 a3 x1 y3 z2 x3 z1 ex0 ex2 z0 ex1 ey0 y0 ey1 ey2 a6 a2 v x0

而化簡的計算花了 0.28 秒，只能得到最小化的 BDD 大小為 55。
第二次的變數順序是：

z0 a6 z1 x3 z2 ex1 y0 ex2 y2 y1 a3 ex0 ey0 y3 ey1 x0 ey2 a4 x1 x2 a2 a5 v

花費了 0.29 秒的計算得到最小化的 BDD 大小為 55。
第三個初始變數順序為：

z2 x2 x1 a5 ex0 a3 a2 x0 y2 ex1 z0 z1 y1 ex2 y3 y0 a4 ey0 v ey1 ey2 a6 x3

在 0.25 秒的計算後，所能得到最小化 BDD （以 a2 a4 a3 a6 a5 y2 y1 y3 y0 x2 x1 x0 x3 z2 z0 z1 ex0 ex1 ex2 v ey0 ey1 ey2 為變數順序）的大小為 42。

所以這一次的實驗需產生 3 個變數順序，共花了 0.82 秒得到最佳解。

因為是以亂數的方式隨機的產生變數順序，所以每次可能需要產生不同數量的變數順序（作為初始變數順序）才能得到最佳解。於是，以不同的亂數種子（seed），重複進行實驗，看變數順序數量的需求是否有一可預測的趨勢。

100 次實驗中，在第 50 次、第 91 次和第 94 次的實驗，需要超過 10 個（分別是 11 個、17 個和 13 個）初始變數順序才能得到最佳解。

限於篇幅，以下只繪出第 91 次的實驗中各變數順序所花費的計算時間(如圖 2-1)：

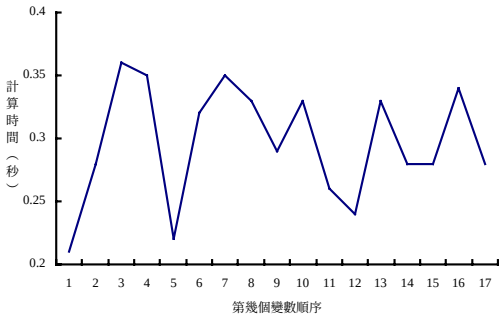

圖 2-1、第 91 次的實驗各變數順序的計算時間

cordic.blif 這個電路的各次實驗的計算時間如圖 2-2 所示：

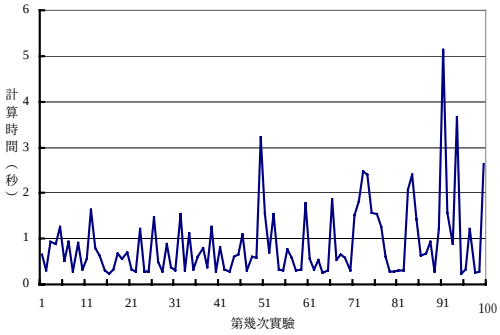

圖 2-2、cordic.blif 電路的計算時間

cordic.blif 這個電路在各次實驗中，變數順序的需要量如圖 2-3：

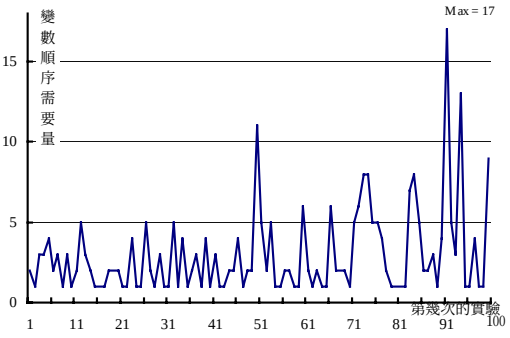

圖 2-3、cordic.blif 電路的變數順序需要量

變數順序需要量的統計如下頁之圖 2-4 所示：

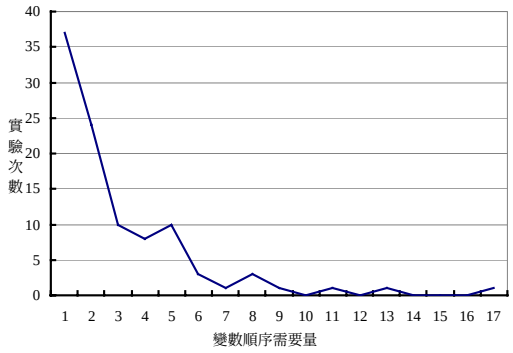

圖 2-4、cordic.blif 電路的變數順序需要量

在 100 次的實驗中，只有三次需要產生 10 個以上的變數順序，有高達 89 次的實驗，都可以在 5 個（$\lceil log_2 n \rceil = 5$）變數順序內得到最佳解。

對於 cordic.blif 這個電路而言，平均只要產生 2 個變數順序（平均計算時間 0.95 秒），就可以得到最佳解。計算時間的統計如圖 2-5 所示。

Drechsler 等人在 2000 年發表的精確最小化（exact minimization）演算法[9]是以前得到結果最佳的演算法，但是此法需耗用大量的計算時間。在該論文中有 31 個 benchmark 電路的最小化 BDD 被求出；下面的列表 2-1 是以前所知最佳的求解方法之計算時間，和我們提出的新方法所用的計算時間的比較。

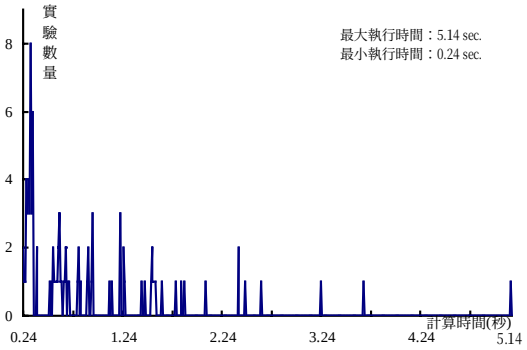

圖 2-5、cordic.blif 電路計算時間的統計

本實驗中，我們提出的新方法是在 Sun Ultra 1 上執行，和 Drechsler 之論文中所用的平台是一樣的，作業系統是 UNIX System V Release 4.0，編寫程式所用的語言都是 C++，前述步驟 2 中的最小化計算程式也是在相同的平台下，使用 Fabio Somenzi 教授所編寫的 CUDD 2.3.1[12]之收斂式篩選化簡演算法。由於篩選演算法可能仍有改進的空間，為了避免模糊本論文中亂數演算法在 BDD 最小化問題上效能表現的焦點，所有實驗皆直接採用[8]中的篩選演算法，程式可於網站[16]下載。

表 2-1、在 Ultra 1 – 140 (UNIX System V Release 4.0)上面執行 BDD 最小化演算法的結果

電路名稱	輸入數	最小 size	Drechsler 之演算法[9]的計算時間(秒數)	我們的新方法計算時間(秒數)		變數順序之數量	
				最大值	平均值	最大值	平均值
cc	21	46	753.3	**2.92**	**0.7**	**12**	**2**
cm150a	21	33	3858.8	**0.39**	**0.31**	**1**	**1**
cm162a	14	30	4.2	**0.8**	**0.43**	**3**	**1**
cm163a	16	26	7.7	**0.77**	**0.43**	**3**	**1**
cmb	16	28	0.1	0.28	0.25	1	1
comp	32	95	47605.4	**0.97**	**0.72**	**2**	**1**
cordic	23	42	23.8	**5.14**	**0.95**	**17**	**2**
cu	14	32	6.5	**0.77**	**0.46**	**3**	**1**
i1	25	36	198.7	**1.03**	**0.29**	**4**	**1**
lal	26	67	4595.4	**0.51**	**0.3**	**2**	**1**
mux	21	33	3872.7	**0.37**	**0.29**	**1**	**1**
parity	16	17	0.1	0.28	0.23	1	1
pcle	19	42	60.1	**0.3**	**0.26**	**1**	**1**
pm1	16	40	3.8	**0.32**	**0.26**	**1**	**1**
s1488	14	369	90.0	**16.51**	**3.82**	**42**	**9**
s208.1	18	41	57.1	**1.94**	**0.44**	**7**	**1**
s298	17	74	93.4	**1.19**	**0.35**	**5**	**1**
s344	24	104	11519.2	**7.79**	**1.61**	**28**	**5**
s349	24	104	11546.3	**8**	**1.67**	**28**	**5**
s382	24	119	6637.6	**10.25**	**1.9**	**35**	**6**
s386	13	109	14.1	**11.17**	**2.27**	**41**	**8**
s400	24	119	6846.8	**10.27**	**1.91**	**35**	**6**
s444	24	119	6934.0	**12.18**	**2.73**	**43**	**9**

s526	24	113	8809.1	**7.41**	**0.89**	**24**	**2**
s820	23	220	17145.3	**7.5**	**1.98**	**19**	**5**
s832	23	220	17953.8	**11.1**	**1.78**	**28**	**4**
sct	19	48	60.6	**1.21**	**0.36**	**5**	**1**
t481	16	21	1.2	3.11	**0.96**	4	**1**
tcon	17	25	3.7	**0.31**	**0.21**	**1**	**1**
ttt2	24	107	7087.2	**3.35**	**0.85**	**11**	**2**
vda	17	478	641.6	**11.65**	**1.73**	**26**	**3**

我們的程式以作業系統提供的 g++編譯程式（compiler）編譯後執行。表 2-1 中所有電路都是在同一台機器上執行的結果；雖然使用相同的機器，但我們的機器上只有 64Mb 的記憶體（比起 Drechsler 所用的 300Mb 要來得少）。不能確定是否是記憶容量的關係，在我們的機器上並無法達成[9]中所敘述的效能，所以，我們並未重複 Drechsler 的實驗，表中「Drechsler 之演算法的計算時間」是由 Drechsler 的論文中直接抄錄而來。

表 2-1 中的粗體字表示我們所提出的新方法可於較短的時間內，找出最佳解，達成和精確解演算法一樣的效果。在表中可以發現，Drechsler 的精確解演算法無法在很短的時間算出的電路，我們所提出的新方法卻有很好的表現：即使是最耗時的電路 comp，新方法也可以在 1 秒內得解。

我們的新方法雖然是用隨機的方式產生變數順序，但是看來有相當穩定的表現（見下圖），對新方法而言，最耗時的電路（s1488）也只用了不到 20 秒的時間就得到最佳解了。

本文中新方法的計算時間如圖 2-6 及 2-7 所示：

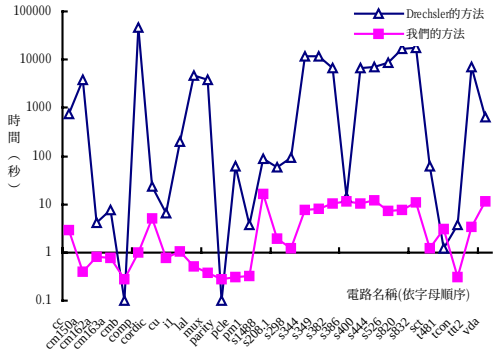

圖 2-6、本文中新方法計算時間的最大值和以前最好的演算法之比較

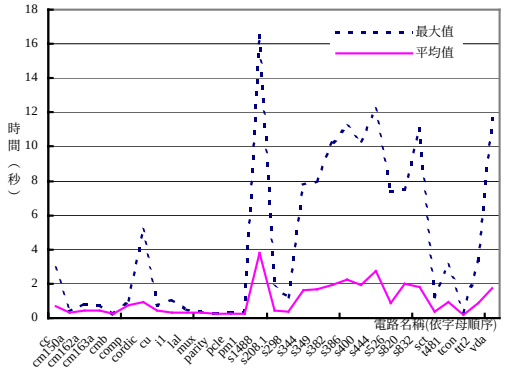

圖 2-7、新方法的計算時間

實驗二：

本實驗針對 benchmark (LGSynth 91) 中，已知最佳解的 48 個電路，嘗試以收斂式對稱篩選演算法[12]執行 BDD 最小化運算，每次皆先以隨機的方式產生初始變數順序求解，接著重複產生變數順序，直到求得最佳解為止，看看需要產生多少個變數順序，才能得到最佳解；並記錄其計算時間，觀察整個結果，看是否能得到一些資訊，來估計出到底要多少個初始變數順序，才能夠得到最佳解。

在這個部份的實驗中，我們不再用 Ultra 1 – 140 的機器，而改用 Pentium II 450。Ultra 1 – 140 的主時脈是 143MHz，計算的速度比起現在普遍的 Pentium II 慢。而這個部份的電路並不是很複雜，我們不用 Pentium IV，因為其計算速度相當快，將會使得數據都很小（當然，在 Pentium IV 越來越普及後，實際的運用上改以更快的機器亦無不可）。電路全部都是在同一台機器（Pentium II 450）上執行的結果。因為硬體的改變，作業系統也改用最普及的 Microsoft Windows 2000 作業系統；不過執行收斂式篩選演算法的程式（CUDD 2.3.1），需要 UNIX 的系統呼叫，所以我們依照其說明安裝了軟體 Cygwin 程式（可於網站 http://sourceware.redhat.com/cygwin取得），然後以它附帶的編譯器 g++ 重新編譯所有的程式來執行。

下頁表 2-2 列出了執行的結果（限於篇幅只列出了 20 個電路）：在此 48 個電路中有 17 個電路，不受到初始變數順序的影響，可直接以收斂式的對稱篩選演算法[9]求得最佳解。其他約 64.6%的電路就沒那麼幸運了，選用了不好的變數順序時，就無法直接得到最佳解（必須藉助我們的新方法才能得到最佳解）。

同樣的，有 17 個電路（只是數量上的巧合），即使改變初始變數順序，收斂式的對稱篩選演算法也無法在第一次求出最佳解，對於這一類的情形，若不用耗時的精確解演算法或用本文中的新方法，就無法得到最佳解了。

表 2-2、在 PII 450 (Windows 2000) 上的實驗結果（共計 48 個電路，只列出了 20 個電路）

電路名稱	輸入數	最小 size	計算時間(秒數)		變數順序之數量	
			最大值	平均值	最大值	期望值
9symml	9	25	0.04	0.03	1	1
alu4	14	350	13.73	2.87	123	26
C17	5	7	0.07	0.02	3	1
cm162a	14	30	0.17	0.03	5	1
cm85a	11	28	0.06	0.04	2	1
comp	32	95	0.23	0.15	1	1
cordic	23	42	0.79	0.33	12	3
decod	5	32	0.04	0.02	1	1
i1	25	36	0.11	0.05	3	1
majority	5	8	0.04	0.01	1	1
mux	21	33	0.1	0.09	1	1
parity	16	17	0.04	0.02	1	1
s1488	14	369	4.86	1.31	51	13
s208.1	18	41	0.26	0.08	4	1
s349	24	104	1.96	0.3	34	5
s386	13	109	2.89	0.42	59	9
s400	24	119	2.34	0.39	39	6
s526	24	113	0.82	0.26	11	3
s832	23	220	2.73	0.55	25	5
t481	16	21	0.72	0.26	3	1

在表 2-2 數據中可得知，欲求得最佳解需要產生之變數順序數量的平均值如圖 2-8：

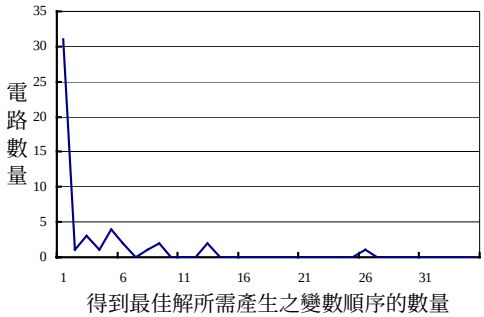

圖 2-8、欲求得最佳解所需產生之變數順序數量的平均值

此外，97.9% 的電路平均只需（以亂數隨機）產生 13 個不同的變數順序做為初始順序（initial ordering），就能求得最佳解。欲求得最佳解需要產生之變數順序數量的最大值則如下圖所示：

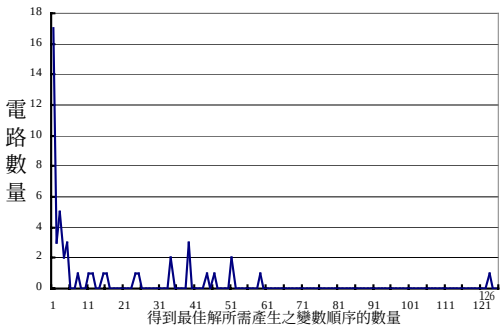

圖 2-9、欲求得最佳解所需產生之變數順序數量的最大值

由此可知，最多需要（隨機）產生 123 個不同的變數順序，以得到最佳解。同樣的，只需產生到 59 個不同的變數順序，就可求解 97.9% 的電路。計算時間之平均值則如下頁圖 2-10 所示：

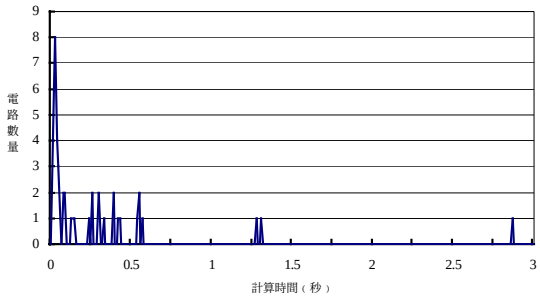

圖 2-10、欲求得最佳解所需計算時間的平均值

每個電路重複做 100 次實驗，其平均計算時間如上圖。幾乎所有的電路都可在 1.31 秒內得到最佳解。計算時間之最大值如下圖：

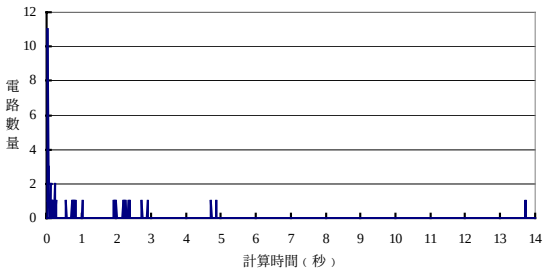

圖 2-11、欲求得最佳解所需之計算時間的最大值

在最壞的狀況下，需要計算 13.73 秒以求得最佳解。同樣的，在最壞的狀況下，97.9% 的電路仍可於 4.86 秒內得到最佳解。

實驗三：

由於 BDD 最小化計算的複雜度相當高，所以許多 benchmark 中的電路都還沒有最佳解。在接下來的實驗中，我們要試看看新的方法是否能讓 BDD 最小化的能力更向前推進一步。我們先整理一下實驗二所得的數據（如下圖 2-12 所示），橫（x）軸代表的是電路的輸入數，縱（y）軸所代表的是求出真正最佳解需要（隨機）產生多少個初始變數順序。由圖 2-12 可以看出，輸入數少於 5 個的電路，收斂式篩選演算法幾乎不受初始變數順序的影響，可以直接求出最佳解。當輸入數為 5 時，就會受到初始變數順序的影響了，最壞的狀況下，要（隨機）產生 5 個初始變數順序，收斂式篩選演算法才能求得最佳解！

就這些已經知道真正最佳解的電路而言，需要最多初始變數順序的是一個 14 個輸入的電路（alu4.blif）；在我們的實驗中，這個電路需要超過 123 個初始變數順序才能求得最佳解。

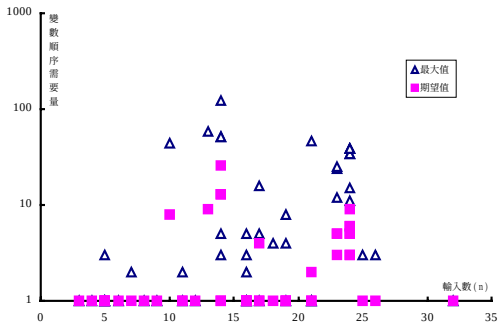

圖 2-12、欲求得最佳解所需計算時間的最大值

根據實驗二所得的數據（如圖 2-12 所示），求出最佳解所需的變數順序數量都未曾超出 $n^{1.8}$ 或是 $n(\log n)^2$；也就是說：在已知的數據中顯示，若以隨機產生的 n^2 個變數順序做為初始變數順序，計算所得到最好的答案，幾乎就是最佳解了。

所以，對於每一個尚未有最佳解的電路，我們將隨機產生 n^2 個變數順序，經由對稱篩選方式計算 BDD 最小化，計算至收斂演算法，求取所能得到的最佳解。

由於尚未有最佳解的電路不少（超過五十個），而且有許多電路的輸入數都超過 100，要產生 n^2 個變數順序來求最佳解必須花費大量的時間；所以我們以 11 台 Pentium IV 1.4GHz 的電腦（配備記憶體 128Mb）分別運算，每台電腦上都是安裝 Winodws 2000 professional edition 的作業系統，再加上 Cygwin 這個軟體以提供 CUDD 執行的需要。這 11 台電腦各別的執行數個電路的最小化計算。幾個輸入數大於 400 的電路各由一台電腦負責最小化計算（因為其 n^2 很大，要計算較長的時間），而其他的電腦則負責數個 n 值較小的電路。

表 2-3 中的每一個電路，我們都隨機產生 n^2 個變數順序來求取最佳解，並記錄下程式發現的最佳解是在第幾個變數順序時找到的，以供後續的研究和確認。

我們以 mm30a.blif 這個電路來說明表中的數據：這個電路有 123 個輸入，當程式（隨機的）產生了 15129（$123^2 = 15129$）個變數順序作為收斂式對稱篩選演算法的初始變數順序來做最小化運算，其中有 3149 個初始變數順序導致程式運算的時間超過計算時間的上限（300 秒）。在以第 1454 個變數順序做化簡時，得到了最佳變數順序，而依此變數順序建立的 BDD 大小為 11504（個端點）。

表 2-3、在 Pentium IV - 1.4GHz 128Mb, Windows 2000 + Cygwin 上最小化各個電路的結果

電　路　名　稱	輸入數 n	全部產生了多少個變數順序(n^2)	最小 BDD 的 size	其中失敗(逾時 300 秒)了多少次	最小的 BDD 是在第幾次發現的
apex6.blif	135	18225	490	0	604
apex7.blif	49	2401	214	0	4
b9.blif	41	1681	97	0	10
bigkey.blif	486	236196	1565	0	64
C1355.blif	41	1681	25866	0	17
C1908.blif	33	1089	5705	0	34
C2670.blif	233	54289	1784	0	13574
C3540.blif	50	2500	23828	13	1
C432.blif	36	1296	1132	0	908
C499.blif	41	1681	25866	0	20
C5315.blif	178	31684	1744	0	605
C7552.blif	207	42849	5249	0	19098
c8.blif	28	784	80	0	1
c880.blif	60	3600	4053	0	60
cht.blif	47	2209	90	0	1
clma.blif	415	172225	725	0	441
clmb.blif	415	172225	561	0	3004
count.blif	35	1225	81	0	1
dalu.blif	75	5625	689	0	3
des.blif	256	65536	2928	0	43716
dsip.blif	452	204304	2456	10	1
example2.blif	85	7225	265	0	2
frg1.blif	28	784	72	0	66
frg2.blif	143	20449	849	0	6291
i8.blif	133	17689	1276	0	5
i9.blif	88	7744	908	0	7
k2.blif	45	2025	1243	0	37
mm30a.blif	123	15129	11504	3149	1454
mm9a.blif	39	1521	1111	0	24
mm9b.blif	38	1444	1527	0	9
mult16a	33	1089	116	0	907
mult16b	47	2209	93	0	1
mult32a	65	4225	235	0	1840
my_adder.blif	33	1089	82	0	1
pair.blif	173	29929	2370	0	8451
pcler8.blif	27	729	86	0	1
rot.blif	135	18225	2837	0	18186
s1196.blif	32	1024	598	0	1
s1423.blif	91	8281	1677	0	625
s420.1.blif	34	1156	81	0	1
s510.blif	25	625	146	0	12
s5378.blif	199	39601	1873	0	21035
s641.blif	54	2916	381	0	7
s713.blif	54	2916	381	0	7
s838.1.blif	66	4356	161	0	1
s9234.1.blif	247	61009	2560	0	26252
sbc.blif	68	4624	904	0	1523
term1.blif	34	1156	75	0	3
too_large.blif	38	1444	302	0	102
unreg.blif	36	1296	82	0	1
x1.blif	51	2601	399	0	1789
x2.blif	10	100	31	0	1
x3.blif	135	18225	490	0	15
x4.blif	94	8836	320	0	25
z4ml.blif	7	49	17	0	1

當我們在考慮計算時間上限的設定時，我們發現有圖 2-13 所顯示的情形：

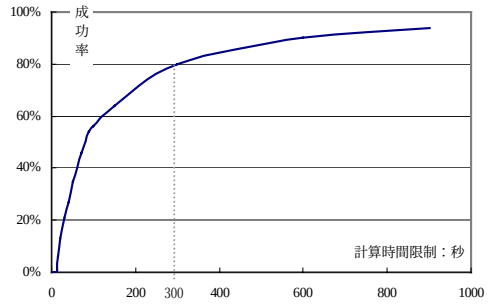

圖 2-13(a)、電路 mm30a.blif 的計算時間限制和失敗率的關係

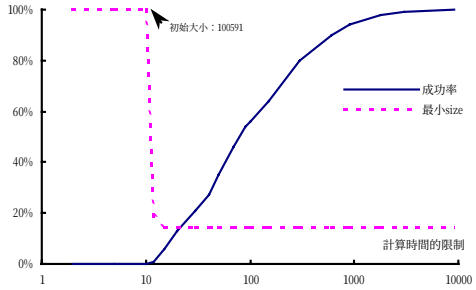

圖 2-13(b)、電路 mm30a.blif 的計算時間限制和失敗率的關係

圖 2-13 是我們以不同的計算時間上限值，來限制 mm30a.blif 這個電路求解的計算：當我們限制求解最小化的演算法只能計算 14 秒（或更短的時間）時，有 95%以上的初始變數順序都未能在時限內得解，在 100 個（隨機的）初始變數中能夠找到的最小 BDD 大小是 18801，而當時限增加到 15 秒之後，見圖 2-13(b)，無法得解的情形就大為減少了！同樣是 100 個（隨機的）初始變數，找到的最小 BDD 大小是 14506；可見當時限設得太小時，會錯過得解的機會。

反之，時限設得長，可以避免上述問題；但是上圖中可以看出，當時限超過 300 秒之後，不到 20%的初始變數順序無法在時限內求解，在 100 個（隨機的）初始變數中能夠找到的最小 BDD 相同於 15 秒時限所能找到的。即使再增加到 600 秒（或更長），無法求解的變數順序量也不會明顯的減少。當時限設得長的時候，雖然有更多的初始變數順序都可以求解，但是卻不會找到更小的 BDD，似乎浪費了許多時間在不必要的計算上。

對這個電路而言，15 秒就像一個臨界值一般，若想讓程式能夠順利的運作，時限便不得低於這個臨界值；在我們的觀察中顯示出，許多電路都有相同的狀況。在實際的應用上，倒不必去求出臨界值，計算時限大約設定為第一個變數順序計算時間的 3 ~ 5 倍即可；因為若設定得太剛好的話，也是有可能會錯過最佳解，所以設定計算時限的時候，設定的值應該比臨界值高。

在表 2-3 中，有 14 個電路在第 1 個變數順序就找到一個很好的變數順序（使得 BDD size 很小），之後（隨機）產生的 n^2-1 個變數順序都無法使 BDD 更小。

綜合我們所有的數據，有大約 30.1%左右的電路，可直接以收斂式的對稱篩選演算法求得最佳解的，也就是說，單純的用收斂式的對稱篩選演算法，雖然速度很快，但是有接近 70%的電路並不是最佳解！而採用新方法的話，幾乎都可得到最佳解，而運算時間和硬體配備都小於精確解演算法的需求。

對於過去尚未有最佳解的電路，我們收集到了 23 個電路的解，如表 3-1 的第四欄所示：電路 apex6.blif 在過去所能找到的最小 BDD 有 622 個端點，而採用新方法來求最佳解之後，找到的最小 BDD 只需要 490 個端點就可表示同一電路，有 21.2%的改進。

三、效能分析

雖然亂數演算法大大的增進了求取近似解演算法的能力，使得它有了相當於求取精確解演算法的能力，但是，近似解演算法的選用也相當重要。如下圖所示，我們亦嘗試過以模擬鍛鍊（simulated annealing）演算法[6]來求解，改以模擬鍛鍊演算法雖可於較少的變數順序內得到最佳解，但其計算時間就顯著的落後於收斂式對稱篩選演算法了（這似乎也說明了為何模擬鍛鍊演算法雖能找到較佳的解答，但篩選演算法卻受到較多重視和討論的原因）。

以各個不同方法化簡時，所需的變數順序數量（最大值）如下圖所示：

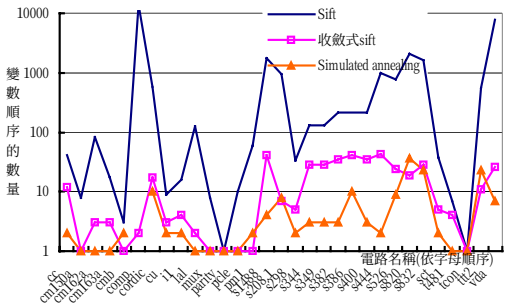

圖 3-1、各種不同方法化簡時，所需的變數順序數量（最大值）

表 3-1、新舊方法所得到的 BDD size 之比較（”-”表示未知最小 size）

電 路 名 稱	輸入數 n	新方法找到的最小 BDD 之 size	以前文獻所知的最小 size[8]	改進比率	Fabio 網站[13]上的最小 size	改進[13]的比率
apex6.blif	135	490	622	21.2%	498	1.6%
apex7.blif	49	214			-	
b9.blif	41	97			-	
bigkey.blif	486	1565			1595	1.9%
C1355.blif	41	25866	26756	3.3%	25866	0.0%
C1908.blif	33	5705	6380	10.6%	5526	**-3.2%**
C2670.blif	233	1784	2057	13.3%	1774	**-0.6%**
C3540.blif	50	23828	23850	0.1%	23828	0.0%
C432.blif	36	1132	1221	7.3%	1064	**-6.4%**
C499.blif	41	25866	26909	3.9%	25866	0.0%
C5315.blif	178	1744	2262	22.9%	1719	**-1.5%**
C7552.blif	207	5249	10072	47.9%	2212	**-137.3%**
c8.blif	28	80			-	
c880.blif	60	4053	4384	7.6%	4053	0.0%
cht.blif	47	90			-	
clma.blif	415	725			738	1.8%
clmb.blif	415	561			690	18.7%
count.blif	35	81			-	
dalu.blif	75	689	860	19.9%	689	0.0%
des.blif	256	2928	2948	0.7%	2945	0.6%
dsip.blif	452	2456			3033	19.0%
example2.blif	85	265			-	
frg1.blif	28	72			-	
frg2.blif	143	849	1434	40.8%	963	11.8%
i8.blif	133	1276	2181	41.5%	1276	0.0%
i9.blif	88	908			-	
k2.blif	45	1243	1319	5.8%	1246	0.2%
mm30a.blif	123	11504	17626	34.7%	11065	**-4.0%**
mm9a.blif	39	1111	2023	45.1%	1111	0.0%
mm9b.blif	38	1527	1683	9.3%	1527	0.0%
mult16a	33	116			-	
mult16b	47	93			-	
mult32a	65	235			-	
my_adder.blif	33	82			-	
pair.blif	173	2370			-	
pcler8.blif	27	86			-	
rot.blif	135	2837			-	
s1196.blif	32	598			598	0.0%
s1423.blif	91	1677	3277	48.8%	1796	6.6%
s420.1.blif	34	81			-	
s510.blif	25	146			-	
s5378.blif	199	1873	2010	6.8%	1932	3.1%
s641.blif	54	381			-	
s713.blif	54	381			-	
s838.1.blif	66	161			-	
s9234.1.blif	247	2560	3490	26.6%	3045	15.9%
sbc.blif	68	904	1043	13.3%	917	1.4%
term1.blif	34	75			-	
too_large.blif	38	302	336	10.1%	319	5.3%
unreg.blif	36	82			-	
x1.blif	51	399			-	
x2.blif	10	31			-	
x3.blif	135	490			-	
x4.blif	94	320			-	
z4ml.blif	7	17			-	

以各個不同方法化簡時，所需的計算時間（最大值）如下圖所示：

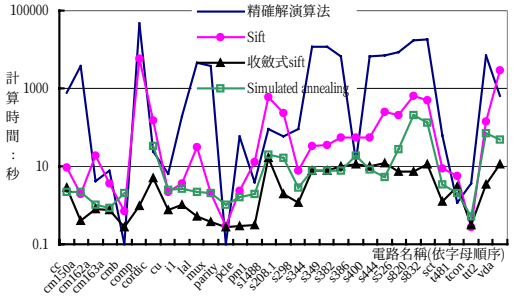

圖 3-2、各種不同方法化簡時，所需的計算時間（最大值）

截至目前，所有求解 BDD 最小化問題的演算法都會受到初始變數順序的影響。以某一種變數順序為初始變數順序時，使用 A 演算法所得的結果可能比使用 B 演算法的結果差，但是使用另一種初始變數時，使用 A 演算法的結果可能就比較好了，於是，很難斷定哪一個演算法比較好。當使用亂數產生器來產生多個變數順序，做為 BDD 最小化演算法的初始變數順序時，初始變數順序的影響就此被排除，在計算複雜度難以估算的困境下，仍可以圖 3-1 和圖 3-2 的方式比較出各個演算法的優劣。這也不失為亂數演算法的一個貢獻。

接下來的例子說明這個情形：在我們評估的幾種 BDD 最小化演算法中，收斂式的篩選演算法有最佳的整體表現，大部份的電路都可於最短的時間內求得最佳的變數順序，但是，在表 3-1 中可以發現 C7552.blif 這個電路以收斂式篩選演算法計算 n^2 個初始變數順序之後所能找到的最小 BDD 仍需要 5249 個端點（比起已知的最小 2212 個端點還大很多），當我們改用模擬鍛鍊演算法來求解時，在第 266 個初始變數的時候，就找到能夠以 2161 個端點表示 C7552 這個電路的變數順序了。模擬鍛鍊演算法計算 266 個初始變數所花費的時間（874,894 秒）近乎是收斂式篩選演算法（470,161 秒）的 2 倍。截至目前的資料顯示，收斂式篩選演算法無法在 n^2 個初始變數順序內找到（端點數誤差在 10%以內的）最佳變數順序的電路約有 1%左右。

上述例子顯示出收斂式篩選演算法有最佳的時間效能（但有少部份的電路無法得到最佳解），模擬鍛鍊演算法能找到較佳的變數順序（但是計算時間可能非常的長），各有其優劣；在尚未有更好整體表現的演算法出現之前，在實用上應混合兩個演算法，當找出一個較佳的解之後，就以此一新的 BDD 大小為上限，計算過程中，導致 BDD 大小超過此一上限的變數順序都無須再計算了。

至於計算的複雜度就如同之前篩選演算法和模擬鍛鍊演算法一樣，因為尚未有足夠的理論可以推算，所以無法確認計算的複雜度。只能說大約是 $O(n^c)$，c 是一個常數；不過可以知道的是，就亂數產生 n^2 個不同的初始變順順序而言，將增加 $O(n^2)$的複雜度。這相對於以往最佳演算法的複雜度 $O(3^n)$仍然是很小。

四、結論

在本論文中，我們提出了一個不同於以往的新概念來求解 BDD 最小化問題，實驗的數據顯示這個新方法有極佳效能；這樣一個新方法之所以能奏效，揭露了 BDD 最小化問題在求解時可利用的特性：

1. 相對於求精確解的演算法，求取近似解的演算法相當的快（計算複雜度低很多）：求取一次精確解所耗費的時間，可允許重複求取多個近似解；在數據中顯示，稍微複雜一點的電路，求取精確解的計算時間，約為求取近似解計算時間的數千倍到數萬倍不等，也就是說，可以讓求取近似解的演算法"算錯"數千到數萬次。當然，求取近似解的演算法並不是每次都找不到最佳解的。
2. BDD 化簡的最佳解，答案並不是唯一的，幾乎都有一個以上的解答（可使得 BDD 最小）。也就是說，"猜"中的機率不低。相反的，在求取精確解的過程中，不能排除的搜尋空間很大，以致於求精確解的計算時間居高不下。於是在兩者的比較下，亂數演算法顯然表現優異。
3. 我們的方法仍有賴於求取近似解的演算法：對於輸入數 n 小於 500 以下的電路，我們都已將最佳解列於第二章的列表中；而 C6288.blif 這個電路並未能完成計算，因為以最新版篩選演算法（cudd 2.3.1）來執行時，程式無法順利完成計算。

本文所介紹的新方法，不同於以往的概念，在計算的效能上亦有大幅的改進，深具實用的價值。除此，在任兩次的求解計算上，並沒有資料的相依性，這樣的演算法比以往的方

法更具實用性，可以非常容易的改成平行演算法，幾乎不需做任何的更動，當然更有助於大型電路的求解了。

四、參考文獻

1. R. E. Bryant, “Graph-based algorithms for Boolean function manipulation,” IEEE Trans. Comput., vol. 35, pp. 677-691, Aug. 1986.
2. S. J. Friedman and K. J. Supowit, “Finding the optimal variable ordering for binary decision diagrams,” IEEE Trans. Comput., pp. 710-713, May 1990.
3. R. Rudell, “Dynamic variable ordering for ordered binary decision diagrams,” Int. Conf. Computer-Aided Design, pp. 42-47, 1993.
4. S.-W. Jeong, T.-S. Kim, and F. Somenzi, “An efficient method for optimal BDD ordering computation,” in Proc. Int. Conf. VLSI and CAD, 1993.
5. S. Panda and F. Somenzi, “Who are the variables in your neighborhood,” in Int. Conf. Computer-Aided Design, pp. 74-77, 1995.
6. B. Bolling, M. Löbbing, and I. Wegener, “On the effect of local changes in the variable ordering of ordered decision diagrams,” Inform. Processing Lett., vol. 59, pp. 233-239, Oct. 1996.
7. B. Bolling, I. Wegener, “Improving the variable ordering of OBDDs is NP-complete,” IEEE Trans. Computer., vol. 45, pp 993-1002, Sep. 1996.
8. R. Drechsler, W. Günther, and F. Somenzi, “Using lower bounds during dynamic BDD minimization,” IEEE Trans. Computer-Aided Designs, vol. 20, pp. 51-57, Jan. 2001.
9. R. Drechsler, N. Drechsler, and W. Günther, “Fast exact minimization of BDDs,” IEEE Trans. Computer-Aided Designs, vol. 19, pp. 384-389, Mar. 2000.
10. J. C. Muzio, T. C. Wesselkamper, “Multiple-Valued Switching Theory,” Adam Hilger Ltd Bristol and Boston, 1986.
11. R. E. Brant, “Symbolic Boolean Manipulation with Ordered Binary-Decision Diagrams,” ACM, Comp. Surveys, vol. 24, 293-318, 1992.
12. F. Somenzi, CUDD: CU Decision Diagram Package – Release 2.3.1 Technical report, Dept. of Electrical and Computer Engineering, University of Colorado, Boulder, Colorado, Feb. 2001.
13. 網站http://vlsi.colorado.edu/~fabio
14. R. K. Brayton, G. D. Hachtel, C. T. McMullen and A. L. Sangiovanni-Vincentelli, "Logic Minimization Algorithms for VLSI Synthesis," Kluwer Academic Publishers, Boston. 1984.
15. 網站ftp://mcnc.mcnc.org
16. 網站http://vlsi.colorado.edu/~fabio