二元判决图变量排序新方法*

李翔宇** 陈光禩

（电子科技大学 CAT 室 成都 610054）

【摘要】 从遗传算法入手，利用已有的几种启发式变量排序方法的结果作为最初的染色体群体，并适当加入随机变量顺序，然后按照遗传算法的方法进行运算，经过多代循环找到近似最优解。实验结果表明，该方法比已有的启发式方法更有效，能够在全局范围内搜索最优解，对 BDD 的变量编序和遗传算法的运用具有参考价值。

关 键 词 二元判决图； 变量排序； 遗传算法； 适应性； 变异； 交叉

中图分类号 TP31

布尔函数是计算机科学和数字系统设计的基石，如何高效地表示布尔函数以及实现快速有效的运算，对它的运用具有至关重要的影响。二元判决图(BDD)是一种表示布尔函数的有效方法[1,2]。自从 Bryant 对 BDD 方法进行完善后，BDD 得到了广泛地应用。但是，构造一个函数的 BDD 时，如何选取它的变量顺序，一直是 BDD 应用过程中的障碍。对于一个布尔函数，在有些变量顺序下，构造出的 BDD 可能很小，但在另外一些变量顺序下，它的 BDD 却很大，有时甚至根本无法产生。例如，对于函数 $x_1x_2+x_3x_4+x_5x_6$，当变量顺序为 1,2,3,4,5,6 时，它的 BDD 如图 1a 所示，它有 8 个节点。当变量顺序为 1,3,5,2,4,6 时，其 BDD 如图 1b 所示，有 16 个节点。可见，如何选取函数的变量顺序对构造它的 BDD 影响巨大。

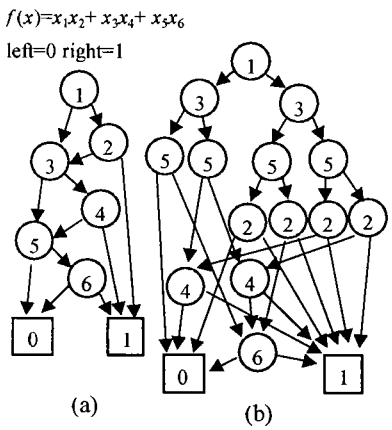

图 1 变量顺序对 OBDD 的影响

1 BDD 变量排序方法

自 BDD 提出以来，有关专家不断地对 BDD 变量顺序作深入地研究。尽管找到一个理想的变量顺序可使 BDD 的体积很小，但变量排序问题本身却等同于 NPC 问题[2]。S.J.Friedman 等人提出了一种寻找 BDD 最佳变量顺序的方法[3]，它的时间复杂度为 $O(n^3 3^n)$，比原始的寻找最佳变量顺序方法的时间复杂度 $O(n!2^n)$有所提高[3]，但仍然不能摆脱指数时间复杂度。目前，BDD 变量排序算法大致分为以下两种。

1.1 动态变量排序方法[4,5]

这种方法是以一个原始的变量顺序为基础，在构造 BDD 的过程中，通过动态地调整变量的顺序，并观察 BDD 体积的大小，从而找到一个较好的变量顺序。动态变量排序法对初始变量顺序的依赖性较强，如果初始变量顺序不好，就无法产生出 BDD。

1.2 启发式变量排序方法[6~8]

这种方法主要是根据电路本身所提供的各种启发式信息来指导变量排序。如果所采用的启发信息对某个电路较合适，可能产生一个满意的变量顺序。同时，也能为动态变量排序提供一个初始的变量顺序。但是，对于各种各样的组合电路，要想找到一个合适的启发信息是十分困难的。目前，启发式变量排序方法大致可分为：

1) 直接计算原始输入端对输出端的影响，根据其影响的大小排定变量顺序。文献[6]采用的方法是，首先对输出端赋一个值，然后按照一定的方法，计算出每一个输入变量的值，把赋值最高的输入变量排在变量顺序的前面，并从电路中删除。然后再对电路中余下的变量循环进行同样的处理，直到确定所有

1998 年 9 月 15 日收稿，1998 年 10 月 27 日修改定稿

* 国家科委大规模集成电路测试技术研究重点科研项目

** 男，28 岁，博士生

变量的顺序。对于多个输出的电路，从深度最大的输出端开始进行处理。

2) 采用深度优先的方法对电路进行遍历，按照被访问的先后次序排定变量顺序。当对多个输出的输入顺序进行合并时，文献[7]采用了附加法，文献[8]采用插入法。

上述对变量进行排序的方法各有利弊，各个适应的电路范围不同，没有一个统一的解决办法。本文把遗传算法引入变量排序中，利用它在全局范围内寻求最优解的性能，并把前面所提到的方法考虑在内，能得到比较满意的结果。

2　基于遗传算法的 BDD 变量排序方法

2.1　遗传算法

遗传算法是模仿自然界的生物进化而产生的一种寻求最优解的方法[9]。它是把问题的解表示成一个染色体，然后随机选取一群染色体，组成最初的繁殖群体。对繁殖群体中的每一个染色体，按照一定的方法计算其适应性，根据适应性的大小选取参加遗传的父代。对父代中的每个个体进行变异，然后两两配对，由一定的法则交叉而产生子代，再对子代中的个体进行同样的处理，循环结束的条件是达到某个预定的目标或者预定循环多少代。

2.2　运用于 BDD 变量排序的遗传算法操作

设函数 f 有 n 个输入变量，分别为 x_{k1}，x_{k2}，…，x_{kn}，在 BDD 中的顺序表示为 k_1，k_2，…，k_n（从 BDD 的根节点到叶节点），我们把变量顺序 k_1，k_2，…，k_n 作为遗传算法中的一个染色体。例如，一个 6 变量的函数 $f(x_1, x_2, x_3, x_4, x_5, x_6)$构造其 BDD 时，采用的变量顺序为 $x_4x_3x_5x_2x_1x_6$，则数字串 4,3,5,2,1,6 代表对应的变量顺序，4,3,5,2,1,6 就是一个染色体，其中每个数字是它的一个基因。

2.2.1　染色体群体初始化

一般的遗传算法，染色体群体的初始化是随机选取的。但对于 BDD 的变量顺序来说，随机选取的变量顺序往往不能在合理的时间内构造出对应的 BDD，即使能构造，BDD 的体积也相当大，在循环运算过程中收敛的速度很慢。因此，采用把已知的变量编序结果和随机选取的变量顺序相结合的办法，组成最初的染色体群体。设选取的染色体个数为 C，存入到染色体类的对象数组 P 中。

2.2.2　计算适应性

适应性是衡量一个染色体好坏的标准。要衡量一个染色体(变量顺序)的好坏，最直接的方法就是构造出该变量顺序所对应的 BDD，然后计算出 BDD 的节点数目。数目越小，则该染色体的适应性越高。但是，构造一个变量顺序的 BDD 很费时。因此必须采用一种快速衡量染色体好坏的方法，缩短整个算法的运行时间。文献[10]提出了一种快速衡量变量顺序的方法，它是根据在一段确定的时间内，部分构造某个变量顺序的 BDD，然后计算电路中已构造出 BDD 的基本多输入门的个数，由个数的多少衡量变量顺序的好坏。通过改进，把这种方法应用于计算染色体的适应性上。设在一个时间段 T 内，对每一个染色体构造相应的 BDD，如果完成 BDD 的构造，则计算出 BDD 的节点个数 B 和被构造出 BDD 的基本多输入门的个数 E，如果没有完成 BDD 的构造，则只计算其已被构造出 BDD 的基本多输入门个数，其 BDD 节点个数 B 为 0。设某个染色体 i 所对应的这两个变量分别为 B_i 和 E_i，则该染色体的适应性 F_i 为

$$F_i=S(\max B-B_i+E_i) \qquad B_i\neq 0$$

$$F_i=SE_i, \quad B_i=0$$

其中　$\max B=\max\{B_1, B_2, \ldots, B_{10}\}$，$S$ 为放大倍数。

2.2.3　选择父代个体

对于每一代群体，使用轮盘算法选择参加变异和交叉的父代个体。为简单起见，设 10 个染色体的适应性分别为 4，8，24，32，4，12，56，64，8，16，父代个体的选择如表 1 所示。

表 1 轮盘算法选择父代个体

染色体	1	2	3	4	5	6	7	8	9	10
适应性	4	8	24	32	4	12	56	64	8	16
累加和	4	12	36	68	72	84	140	204	212	228
随机数	10	94	196	180	85	219	198	12	216	82
父代个体选择	2	7	8	8	7	10	8	2	10	6

表 1 中第一、二行分别对应 10 个染色体的编号和适应性。进行父代选择时，首先按照各个染色体的排列顺序，计算出累加和。每一个染色体的累加和都是它前面所有染色体(包括自身)的适应性总和，如表 1 中第三行所示。然后，对每个染色体产生一个介于 1 到最大累加和之间的随机数(这里为 1~288 之间)，如表中第四行所示。该随机数确定选择哪个父代个体，其选择的原则是，把染色体的随机数与累加和从小到大进行比较，如果某个染色体的累加和是第一个大于等于该随机数的染色体，则它被选中。各个被选中的染色体的编号如表中第五行所示。可以看出，适应性越大的父代个体被选中的机会越大。

```
mutation(h){
    for(k=0; k<n; k++){
        R=randomnumber;              //between 0 and 100
        if(R<M){
            ep=randomnumber;         //between 0 and n-1
            if(ep!=k){
                tmp=h[k];
                h[k]=h[ep];
                h[ep]=tmp;
            }
        }
    }
    return (h);
}
```

图 2 变异流程图

2.2.4 变异

被选中的父代个体要进行变异的处理，这是为了避免算法在寻找最优解的过程中陷入局部最优而采取的一种措施。这里，变异的方法是让染色体中的某一个基因随机地与另一个基因交换位置。变异的流程图如图 2 所示。

图 2 中，h 是指向某个染色体的指针，它的每个基因表示为 $h[i]$，其中：$0<=i<n$，n 为基因个数。变异时，对染色体中的某个基因 $h[k]$产生一个随机数 R，如果该随机数小于所设置的变异率 M，则对该基因进行变异。产生一个随机的变异位置 ep，当 k 不等于 ep 时，让基因 $h[k]$和 $h[ep]$交换位置。

例如，对一个 8 变量的函数，它的一个染色体（变量顺序）是 3,1,5,7,6,4,8,2。用 mutation 程序进行处理时，首先处理第一个基因 x_3，如果 $R<M$，那么产生一个随机数 ep，假设 $ep=4$，那么变异后的染色体就是 6,1,5,7,3,4,8,2。然后处理第二个变量。如果 R 不满足小于 M 的条件，则不进行变异处理。如此循环，直到最后一个基因。

2.2.5 交叉

交叉是由两个父代个体通过一定的规则产生两个新的子代个体的过程。设两个父代个体分别为 $h1$ 和 $h2$，它们的基因分别为 $h1[i]$和 $h2[i]$，其中：$0<=i<n$，n 为基因个数。进行交叉时，产生一个随机的切入点 pos($0<=pos<n-1$)，把两个父代个体在基因 $h1$[pos]和 $h2$[pos]后切断，重新组成两个新的个体 $n1$ 和 $n2$。例如，设两个五变量的染色体 $h1$ 和 $h2$ 分别为 5,1,4,3,2 和 1,5,4,2,3，随机产生的切入点 pos 为 2，则交叉后产生的两个新个体为 5,1,4,2,3 和 1,5,4,3,2，如图 3 所示。

从图 3 还可以看出，进行交换的两部分基因的集合必须相同(如 2 和 3)，否则，交叉之后得到的子代个体会丧失某些变量，而某些变量又会重复。例如，图 4 中，进行交换的变量集合不一样，交换之后，得到的两个子代个体中，$n1$ 丢失了变量 2，变量 1 却有重复，$n2$ 丢失了变量 1，变量 2

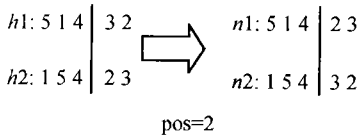

图 3　交叉示意图(交换变量相同)

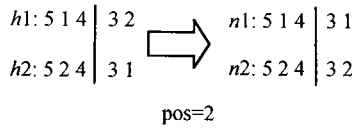

图 4　交叉示意图(交换变量不相同)

却有重复。为了使每次交叉始终都能顺利进行，采取如下的交叉方法：

设两个父代个体为 $h1$ 和 $h2$，基因个数为 n，随机产生的交叉点位置为 pos，产生的两个子代个体为 $n1$ 和 $n2$，则

$n1[0] = h1[0]$，…，$n1[pos-1]=h1[pos-1]$

$n2[0] = h2[0]$，…，$n2[pos-1]=h2[pos-1]$

$n1[pos]$到 $n1[n-1]$分别取自 $h2$ 中的基因顺序，$n2[pos]$到 $n2[n-1]$分别取自 $h1$ 中的基因顺序。如图 5 所示。

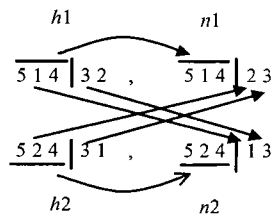

图 5　改进后的交叉

图 5 中，pos=2，$n1$ 的前一部分基因从 $h1$ 中获得，各基因位置不变，后一部分从 $h2$ 中获得，保持各基因在 $h2$ 中的顺序不变。$n2$ 也同样产生。交叉操作的流程图如图 6 所示。

2.3　遗传算法的流程

1) 读取电路文件;

2) 产生 C 个初始染色体 P;

3) 计算每个染色体的适应性 F;

4) 完整构造适应性最大的染色体的 BDD，并保留在 O 中;

5) 设循环执行次数 g=1;

6) 利用轮盘算法选择父代个体;

7) 进行变异处理，变异率设置为 M;

8) 进行交叉处理，得到一群新的染色体 n;

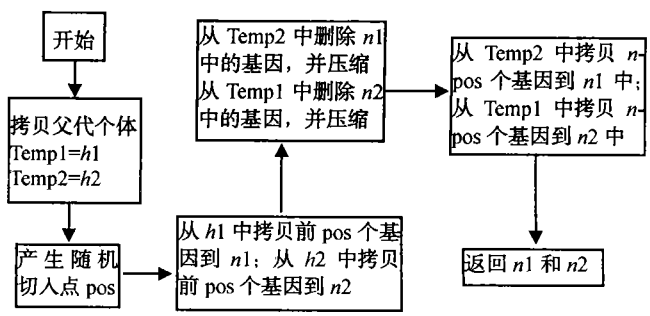

图 6　交叉操作流程图

9) 计算每个新染色体的适应性 F;

10) 找出适应性最大的染色体，完整构造其 BDD，如结果优于 O，则保留新结果在 O 中:

11) $g=g+1$，若 $g<\max G$，则返回 6);

12) 输出最优结果 O;

3　结　论

在 PC 机(K6-200，64MRAM，Windows98)上，我们用 C++语言实现了上述算法，其中，C 取 10，M 取 6，max$G$ 取 10，计算结果如表 2 所示。

变量编序 1 是按照输入端作用，输出端的多少来进行排序，变量编序 2 是附加法的排序。从表 2 可看出，由遗传算法所构造的 BDD，每种电路都有一定程度的提高，但时间却稍长。从 BDD 的应用来看，节点越少，运算速度会越快，因此，这将缩短 BDD 以后的计算时间。

BDD 的变量编序是一个广泛研究的问题，利用遗传算法对排序进行研究，吸收了遗传算法的长处，使得能够在全局范围内搜索最优解，最大限度地避免了以前变量排序方法陷入局部最优解的可能性。当然，这种方法仍有其不足之处，需作继续研究。

表 2 遗传算法的结果比较

电路名	变量编序 1		变量编序 2		遗传算法编序	
	BDD 节点	CPU/s	BDD 节点	CPU/s	BDD 节点	CPU/s
c432	44 949	7.69	31 178	3.57	1 733	199.43
c499	46 170	4.89	55 866	133.1	40 658	789.56
c880	9 470	0.66	10 348	0.6	7 072	259.25
c1 908	23 652	2.86	17 493	1.98	10 590	303.57

参 考 文 献

1 Akers S B. Binary decision diagrams. IEEE Trans Computer, 1978, C-27: 509~516

2 Bryant R E. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Trans Computer, 1986, C-35: 677~691

3 Friedman S J, Supowit K. Finding optimal variable ordering for binary decision diagrams. IEEE Trans Computer, 1990, 39(5): 710~713

4 Rudell R. Dynamic variable ordering for ordered binary decision diagrams. Proc ICCAD'93, 1993, 42~47

5 Ishiura N, Sawada H, Yajima S. Minimization of Binary Decision Diagrams Based on Exchanges of Variables. IEEE ICCAD'91, 1991: 472~475

6 Minato S I, Ishiura N, Yajima S. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. Proc 27th Design Automation Conference, 1990: 52~57

7 Butler K M, Ross D E, Kapur R *et al.* Heuristic to compute variable ordering for efficient manipulation of ordered binary decision diagrams. Proc 28th Design Automation Conference, 1991: 417~420

8 Fujii H, Ootomo G, Hori C. Interleaving based variable ordering methods for ordered binary decision diagrams. Proc ICCAD'93, 1993:38~41

9 吴 斌，吴 坚，涂序彦.快速遗传算法研究.电子科技大学学报，1999, 28(1):49~53

10 Long Wangning, Min Yinhua, Yang Shiyuan. Short-time scaling of variable ordering of OBDDs. J of Comput Sci & Technol, 1997: 366~371

A New Method of Binary Decision Diagram Variable Ordering

Li Xiangyu Chen Guangju

(CAT Lab., UEST of China Chengdu 610054)

Abstract In this paper, how to find an optimal order using genetic algorithm is discussed. This paper also uses some variable orders found by heuristic method and random orders for population initialization, and utilizes a fast method to compute an order's fitness. The experiment results indicate that this method can find the optimal order in global scope. This method has some reference to the variable ordering of BDD and to the application of genetic algorithm.

Key words binary decision diagram; variable ordering; genetic algorithm; fitness; mutation crossover