

# A\* 算法在 BDD 变量最优排序方法中的应用

胡东华, 张 旭

(郑州轻工业学院 计算机与通信工程学院, 河南 郑州 450002)

**摘 要:**介绍了二叉判定图(BDD)的相关知识及在构造 BDD 过程中变量顺序对其结点数的影响,在 Friedman 等提出的一种寻找最优变量排序算法的基础上,将广泛应用于人工智能的 A\* 搜索算法引入到最优变量排序方法中,提出了一种寻找变量最优排序的新方法。该方法在寻求 BDD 最优变量排序的过程中,使处理器的处理时间和存储器的空间需求上都有很大的改善。

**关键词:**BDD;最优变量排序;A\* 搜索算法;状态空间;估价函数

**中图分类号:**TP301.6;O223

**文献标识码:**A

**文章编号:**1673-629X(2007)07-0070-03

## Application of A\* Algorithm in Optimal Variables Ordering Method of BDD

HU Dong-hua, ZHANG Xu

(Sch. of Computer and Communication Eng., Zhengzhou Univ. of Light Industry, Zhengzhou 450002, China)

**Abstract:** Gave a brief introduction of BDD and the order of the variables reflects the size of a BDD related to a given Boolean function. In paper, bring the widely used searching algorithm in AI—A\* algorithm into the optimal variable ordering. Compared with the algorithm of others, in finding the BDD optimal variable ordering, this algorithm has more improvement in the management time of CPU and the need of storage.

**Key words:** BDD; optimal variable ordering; A\* searching algorithm; state spaces; cost function

### 0 引言

二叉判定图(BDD)<sup>[1]</sup>是表示布尔函数的有效工具,被广泛地应用到逻辑综合、布尔电路的模拟和测试等领域。一个有效和简化的 BDD 将极大地提高模型检验所能验证的系统规模以及验证和测试的生成效率。但在现在常用的方法中,变量顺序的选择是影响布尔函数的 BDD 的结构和大小的一个非常重要的因素。目前 BDD 的变量排序的基本方法大致可以分为三种:启发式变量排序、动态变量排序、最优变量排序。其中以 Friedman 等人提出最优变量排序法<sup>[2]</sup>的结果最好,时间复杂度为  $O(n^2 3^n)$ 。

笔者在 Friedman 等人提出的一种寻找最优变量排序的算法的基础上将广泛应用于人工智能的 A\* 搜索算法<sup>[3]</sup>引入到最优变量排序方法中,提出了一种寻找

变量最优排序的新方法。该算法利用人工智能中的 A\* 算法,把寻求变量的最优排序问题转变成了在状态空间中寻找最优路径的问题,因此这种办法比起 Friedman 的最优变量排序算法在处理器的处理时间上和存储器的空间需求上都有很大的改善。

### 1 BDD 简介

**定义 1** (二叉判定图 BDD): BDD 是一个有限个结点的有向无环图  $G = \langle V, E \rangle$ , 其中  $V$  是  $G$  中所有结点的集合,  $E$  是  $G$  中所有边的集合。 $V$  中的结点分为端结点(用方框表示)和非端结点(用圆框表示)。每个端结点  $v_i$  用 0 或 1 标识,无射出边;每个非端结点  $v_n$  用变量  $\text{var}(v_n)$  标识,并且有两条射出边:一条指向右子结点  $\text{hi}(v_n)$  的 then-边(用实线表示,表示  $\text{var}(v_n)$  被赋值 1),另一条指向左子结点  $\text{lo}(v_n)$  的 else-边(用虚线表示,表示  $\text{var}(v_n)$  被赋值 0)。

图 1 是布尔函数  $f = x_1 \cdot x_2 + x_3 \cdot x_4$  的 BDD 表示,只用了 8 个结点。从图中可以看出,从 BDD 根结点到一个个终结点的一条迹就对应着变量的一组赋值,由该分支的端结点所标识的值就是变量在这组赋值下所

收稿日期:2006-10-08

基金项目:河南自然科学基金(0411010500);郑州轻工业学院校内科研基金

作者简介:胡东华(1976-),男,河北唐山人,硕士,助教,研究方向为硬件模型检测。

对应的函数值。

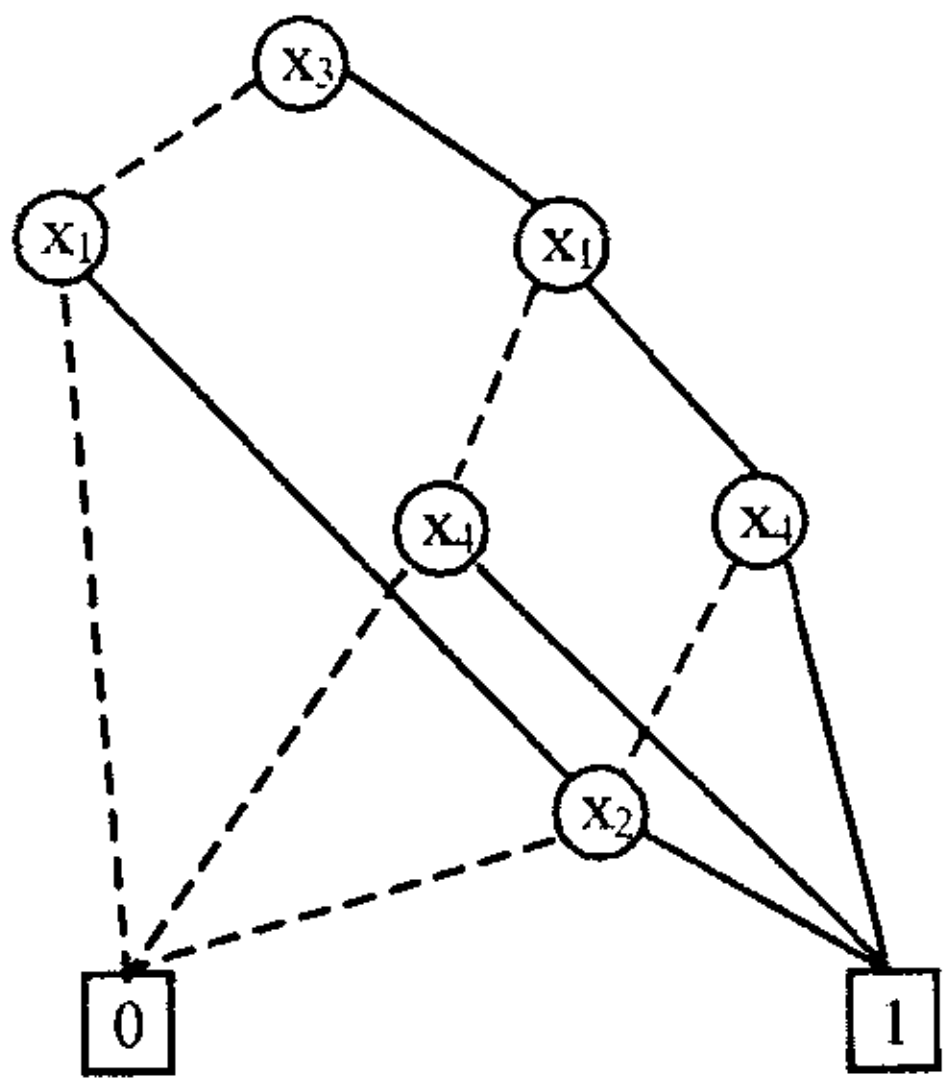


图 1  $f = x_1 \cdot x_2 + x_3 \cdot x_4$  的 BDD 表示

注意图 1 的二叉判定图,处于同一层所有的结点都是相同的变量,如果所有从根结点到终端结点的路径上各变量按同一顺序出现,那么这样的二叉判定图称为有序的二叉判定图<sup>[4]</sup>(OBDD)。图 1 就是布尔函数  $f = x_1 \cdot x_2 + x_3 \cdot x_4$  在变量顺序为  $x_3 < x_1 < x_2 < x_4$  下的 OBDD 表示。对于一个布尔函数  $f$ ,在给定一个变量排序  $\pi$  下的 BDD 表示记作  $BDD(f, \pi)$ 。图 1 所示函数  $f$  的 BDD 记作  $BDD(x_1 \cdot x_2 + x_3 \cdot x_4, (3, 1, 4, 2))$ 。尽管 BDD 是表示布尔函数的非常有效的方法,能够在很大程度上减少存储的结点数,但是一个布尔函数的 BDD 的结构和大小却依赖于变量的顺序,变量顺序不同结点数有很大差别。 $BDD(x_1 \cdot x_2 + x_3 \cdot x_4, < 3, 4, 1, 2 >)$  如图 2 所示,其结点数仅为 6 个。对于较大的  $n$  值,不同变量顺序对计算时间和存储空间的要求有很大的差别,这样就会对算法的执行效率有很大的影响,而确定 BDD 变量最优排序的问题被证明了是一个 NP 完全问题<sup>[5]</sup>,实际执行时间复杂度是  $O(n!2^n)$ 。目前 S. J. Friedman 提出最优变量排序方法时间复杂度也为  $O(n^2 3^n)$ ,文中将在下面重点介绍在前人研究的基础之上提出了一种改进的最优变量排序方法。

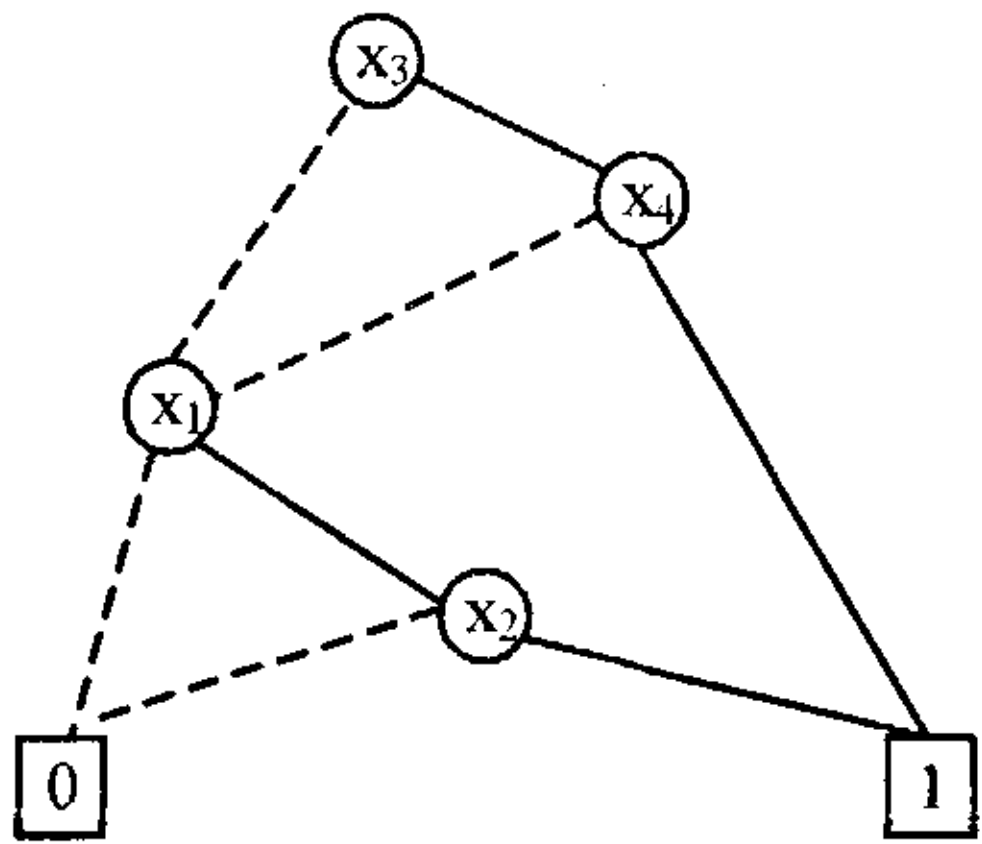


图 2  $BDD(x_1 \cdot x_2 + x_3 \cdot x_4, < 3, 4, 1, 2 >)$

## 2 A\* 算法在最优变量排序方法的应用

状态空间的求解问题就是确定一个操作算子的序列的过程,使其从初始状态  $S_0$  转化为目标状态  $G$ :  $S_0 \xrightarrow{O_1} S_1 \xrightarrow{O_2} S_2 \xrightarrow{O_3} \dots \xrightarrow{O_k} G$ 。在具体的求解过程中,一

个重要的办法就是利用与该问题有关的信息去简化搜索过程,称此类信息为启发信息,这种利用启发信息的搜索过程称为启发式搜索。启发信息往往被反映在估价函数上,估价函数的作用就是估计待搜索结点的“有希望”程度,并以此给所有结点排序。记估价函数为  $\varphi(q)$ ,其一般形式是:  $\varphi(q) = g(q) + h(q)$ 。其中  $g(q)$  是从初始结点到当前结点  $q$  的最小代价值(也记作  $g^*(q)$ ),而  $h(q)$  是从当前结点  $q$  到目标结点的最优路径的估计代价(也记作  $h^*(q)$ ),这种启发式搜索算法称为 A\* 搜索算法。

在此方法中,通过把 A\* 算法应用于寻找一个 BDD 的最优变量排序来实现对 Friedman 的算法的改进。对于含有  $n$  个变量的布尔函数,它的所有变量的排序空间总数是  $n!$  个,而 Friedman 的最优变量排序算法操作在  $2^n$  个状态空间上,所谓一个状态空间就是一个变量集合  $q \subseteq X_n$  (其中  $X_n = \{x_1, x_2, \dots, x_n\}$ ),利用 A\* 算法,虽然状态空间的个数仍为  $2^n$  个,但是大部分状态空间在搜索的过程中被删除了,所以该方法的执行效率有了很大改善。

首先来解释为什么寻求变量的最优排序问题可以转化为在状态空间中寻求从初始状态到目标状态的最小代价路径。

定义 2: 一个变量的集合  $q$ , 对于任意一个变量  $x_i \in X_n \setminus q$  加入到集合  $q$  中,记作  $q \xrightarrow{x_i} q \cup \{x_i\}$ 。

在本算法中,集合  $q$  的初始状态为  $\emptyset$ ,到达目标状态后为  $X_n$ 。通过上一节的论述,知道 A\* 算法能够找到一条从  $\emptyset$  到  $X_n$  的路径  $p(t)$  并且具有最小代价,这个最小代价就是  $\emptyset \xrightarrow{x_{i_1}} \{x_{i_1}\} \xrightarrow{x_{i_2}} \dots \xrightarrow{x_{i_n}} X_n$  的累加转移代价,而路径  $p(t)$  上的变量的顺序就是变量的最优排序。

$g(q)$  和  $h(q)$  定义如下:  $g(q)$ : 由公式  $g(q') = g(q) + \text{cost}(q, q')$  可知,  $g(q)$  是一个转移代价的累加值,转移函数  $\text{cost}(q, q')$  定义为变量  $x_i$  在经过  $q \xrightarrow{x_i} q \cup \{x_i\}$  后得到的 BDD 中对应变量的结点的个数。 $h(q)$ : 定义为变量  $x_i$  在 BDD 中对应的指向非终端结点的结点的出度之和。对  $g(q)$  和  $h(q)$  的定义依赖于下述定理:

定理 1<sup>[2]</sup>: 如果  $I \subseteq \{1, 2, \dots, n\}$ , 且  $k = |I|$ ,  $v \in I$ , 那么存在一个常量  $c$ , 使得对于任意的  $\pi \in \Pi(I)$ , 都有: 如果  $\pi(k) = v$  则  $\text{cost}_v(f, \pi) = c$ 。

定理 2<sup>[2]</sup>: 位于前  $k$  层变量所对应的结点数仅依赖于这  $k$  个变量的顺序,而与另外  $n - k$  个变量无关。

本算法中  $h(q)$  是变量  $x_i$  在 BDD 中对应的结点的

指向非终端结点的出度之和,  $h(q)$  越小, 下层非终端结点个数越少, 定义的  $h(q)$  可以表示从当前结点  $q$  到目标结点的最优路径的估计代价。对于  $g(q)$ , 由于转移代价  $\text{cost}(q, q')$  是变量  $x_i$  在经过  $q \xrightarrow{x_i} q \cup \{x_i\}$  后得到的 BDD 中对应变量  $x_i$  的结点的个数, 所以累加转移代价  $\emptyset \xrightarrow{x_{i_1}} \{x_{i_1}\} \xrightarrow{x_{i_2}} \dots \xrightarrow{x_{i_n}} X_n$  的就是路径  $p(t)$  上与变量顺序  $x_{i_1} < x_{i_2} < \dots < x_{i_n}$  一致的各种排序对应的 BDD 中结点数最少的 BDD 的结点数目, 将结点数最少的 BDD 对应的变量排序记作  $l$ , 由定理 2 可知, 对于剩余的  $n - k$  个变量的状态扩展不会改变该 BDD 的结点数; 由定理 1 可知, 由排序  $l$  经过剩余的  $n - k$  个变量的状态扩展得到的  $n$  个变量对应的 BDD 仍然是  $x_1, x_2, \dots, x_k$  各种排序经过剩余的  $n - k$  个变量的状态扩展得到的  $n$  个变量对应的 BDD 中结点数最少的 BDD。所以本算法中定义的  $g(q)$  可以表示从初始状态到当前状态  $q$  的最优路径。更进一步, 当  $k = n$  时即到达了目标状态  $X_n, g(X_n) = \varphi(X_n)$  表示  $n$  个变量各种排序对应的 BDD 中结点数最少的 BDD 的结点数目, 所以最优路径  $p(t)$  上的变量顺序就是一个 BDD 的最优变量排序。

### 3 算法及算法分析

下面给出 A\* 搜索算法寻求最优变量排序的算法描述, 在该算法中设一个变量 `upper_bound`, 其初始值是未调整的 BDD 的结点, 但是该变量在算法运行的过程中不断通过 `upper_bound := update_upper_bound()` 进行更新, 使其总是等于当前已知的 BDD 的最小结点数。由于  $\varphi, g, h$  仅仅计算非终端结点的个数, 因此用  $C^* + 2$  表示最优 BDD 结点的个数 (即加入两个终端结点 0 和 1)。算法如下:

```
compute_optimal_ordering(BDD F, int n) { /* F represents f */
    upper_bound := update_upper_bound(); /* first upper_bound:
size of initial BDD */
    g[hash(Φ)]:=0; h[hash(Φ)]:=1;
    states[hash(Φ)]:=Φ; insert Φ into OPEN;
    while 1 do /* until */
        determine q ∈ OPEN with minimal g[hash(q)] + h[hash(q)];
        if g[hash(q)] + h[hash(q)] + 1 = upper_bound then
            Reconstruct ordering that yielded upper_bound; doomsday:
            = 1;
        end-if
        if q = X_n then
            reconstruct ordering as the sequence of the path to q; dooms-
            day:=1;
        end-if;
```

```
    if doomsday then exit while loop; end-if
    reconstruct an appropriate ordering π with π(q) = q with path re-
    construction and establish π on F;
    upper_bound := update_upper_bound();
    for each x_i ∈ X_n \ q do
        q' := q ∪ {x_i};
        if q' ∈ states then g[hash(q')]:=∞; end-if
        shift x_i to level |q| + 1;
        new cost = label(F, |q'|) + g[hash(q)];
        if q' ∈ states or new cost < g[hash(q')] then
            g[hash(q')]:=new cost;
            p(q') = x_i / * needed for the path reconstruction */
            if q' ∈ states then states[hash(q')]:=q'; end-if
            if h[hash(q')] not yet computed and g[hash(q')] + n - |q'|
            + 1 ≤ upper_bound then
                upper_bound := update_upper_bound();
                h[hash(q')]:=compute_lower_bound();
            end-if
            if h[hash(q')] computed and g[hash(q')] + h[hash(q')] + 1
            ≤ upper_bound then
                if OPEN too crowded, resize it and garbage-collect out-
                dated entries;
                insert q' into OPEN;
            end-if
        end-if
    end-for
end-while
}
```

首先计算本算法中的状态空间的个数。从所给的状态空间的定义中可以计算出该算法的状态空间的总数为:  $C_n^0 + C_n^1 + \dots + C_n^n = (1+1)^n = 2^n$  (其中  $C_n^0$  是  $\emptyset$ ), 等于 Friedman 的最优变量排序算法的状态空间的个数, 但是由于: a. Friedman 的最优变量排序算法需要遍历整个状态空间 (即  $2^n$  状态空间), 但是文中的算法使用了带有启发函数的 A\* 搜索算法, 每次仅对 open 表中具有最小估价函数值的状态进行扩展, 使得大部分状态空间在搜索的过程中被删除了。算法运行的时间和空间效率高于 Friedman 的最优变量排序算法, 该算法在最坏的情况下才退化为 Friedman 的最优变量排序算法。b. 算法 27 行中, 对于一个状态  $q$  如果有  $\varphi(q) + 2 > \text{upper\_bound} \geq C^* + 2$ , 不把  $q$  插入 open 表; 算法 7 行中, 如果经过有计算  $\varphi(q) + 2 = \text{upper\_bound}$ , 就有  $\text{upper\_bound} = C^* + 2$ , BDD 结点的个数已经到了最小值, 就不再对状态空间中还未扩展的状态进一步扩展, 可以结束算法了, 使该算法的状态空间得到了更进一步的缩减。c. 该算法利用 BDD 的形式来存储布

(下转第 76 页)

用一个实数来描述,DFDT 算法的条件属性使用一个动态模糊数进行描述。实验如下:

实验数据从数据库中任意抽取一组训练样例,样例数目分别为 30、100、200 组。建立树后分别用训练样例、数据库其他数据进行验证。

(1)使用训练样例验证:

DFDT: 86.21% (30 组) 91.32% (100 组)  
92.67% (200 组)

C4.5: 82.10% (30 组) 87.67% (100 组)  
88.10% (200 组)

(2)随机从数据库抽取验证样例(修剪之前):

DFDT: 80.14% (30 组) 86.12% (100 组)  
87.67% (200 组)

C4.5: 78.30% (30 组) 82.37% (100 组)  
83.40% (200 组)

(3)随机从数据库抽取验证样例(修剪之后):

DFDT: 85.07% (30 组) 89.92% (100 组)  
91.07% (200 组)

C4.5: 77.10% (30 组) 81.07% (100 组)  
81.60% (200 组)

从实验数据中可知,在处理动态模糊性问题时 DFDT 比 C4.5 等算法更加有效。对已建立的 DFDT 修剪后,它的预测准确性比未修剪前更高。

## 5 结 论

DFS 理论在描述动态模糊信息时,能够有效地表

示事物真实情况。而使用静态精确数据描述动态模糊信息时会造成信息缺失。DFDT 高效、准确地处理 DFS 理论表示的训练样例,能够有效地处理动态模糊问题。和模糊决策树不同,DFDT 对任何输入只有唯一的判断结果,匹配算法的时间复杂度低于模糊决策树。

### 参考文献:

- [1] Mitchell T M. 机器学习[M]. 曾华军, 张银奎译. 北京: 机械工业出版社, 2003.
- [2] 杨宏伟, 赵明华, 孙 娟, 等. 基于层次分解的决策树[J]. 计算机工程与应用, 2003 (23): 108 - 110.
- [3] 李凡长, 刘贵全, 余玉梅. 动态模糊逻辑引论[M]. 昆明: 云南科技出版社, 2005.
- [4] Olarn C, Wehenkel L. A Complete Fuzzy Decision Tree Technique[J]. Fuzzy Sets and Systems, 2003, 138 (2): 221 - 254.
- [5] Myles A J, Brown S D. Induction of Decision Trees using Fuzzy Partitions[J]. Journal of Chemometrics, 2003 (17): 531 - 536.
- [6] Aoki K, Watanabe T, Kudo M. Design of Decision Tree Using Class - Dependent Features Subsets[J]. Systems and Computers in Japan, 2005, 36(4): 37 - 47.
- [7] 王熙照, 孙 娟, 杨宏伟, 等. 模糊决策树算法与清晰决策树算法的比较研究[J]. 计算机工程与应用, 2003 (21): 72 - 75.
- [8] 李道国, 苗夺谦, 俞 冰. 决策树剪枝算法的研究与改进[J]. 计算机工程, 2005, 31 (8): 19 - 21.

(上接第 72 页)

尔函数,不仅可以利用中动态变量交换技术方便地实现变量  $x_i$  和  $x_{i+1}$  的交换,而且在得到最优变量序的同时,也构建出了与之对应的 BDD,此外对存储空间的需求也减少了。

从上面的分析中可以看到,把 A\* 算法引入到基于 Friedman 的最优变量的排序问题中,在处理器的处理时间上和存储器的空间需求上都有很大的改善,提高了算法的执行效率、验证和测试的生成效率。

## 4 小 结

将 A\* 搜索算法引入到求解最优变量排序的问题中,由于 A\* 搜索算法仅对 open 表中具有最小估价函数值的状态进行扩展,使得  $2^n$  个状态空间的大部分状态空间在搜索的过程中被删除了,同时在该算法引入了暂缓插入的条件和提前结束的条件,使该算法的状态空间得到了更进一步的缩减。因此该算法在搜寻变量最优序的过程中,有更好的执行效率,为最优排序算

法向实际的工业应用提供了一个新思路和新方法。

### 参考文献:

- [1] Bryant R E. Symbolic manipulation of Boolean functions using a graphical representation [C] // Proceedings of the 22nd ACM/IEEE conference on Design Automation. [s. l.]: [s. n.], 1985: 688 - 694.
- [2] Friedman S J, Supowit K J. Finding the Optimal Variable Ordering for Binary Decision Diagrams[J]. IEEE Trans Computers, 1990, 39(5): 710 - 713.
- [3] Hart P, Nilsson N, Raphael B. A formal basis for the heuristic determination of minimum cost paths[J]. IEEE Trans Syst Sci Cybern, 1968, 2: 100 - 107.
- [4] Bryant R E. Graph - based algorithms for Boolean function manipulation[J]. IEEE Transactions on Computers, 1986, 35 (8): 677 - 689.
- [5] Bollig B, Wegener I. Improving the variable ordering of OBDDs is NP - complete[J]. IEEE Trans on Computers, 1996, 45: 993 - 1002.