无圈与或图搜索的符号 OBDD 算法研究

王雪松 赵岭忠 古天龙

(桂林电子科技大学计算机与控制学院 桂林 541004)

摘 要 与或图搜索是人工智能领域一项重要的问题求解技术。基于传统数据结构的与或图表示技术极大地限制了与或图搜索算法可求解问题的规模。在无圈与或图符号 OBDD 表示的基础上，给出了一种求解无圈与或图最小代价解图的符号搜索算法。实验结果表明，与 AO＊算法相比，该算法可处理问题的规模有较大的提高。

关键词 与或图，最小代价解图，OBDDs

OBDD Based Symbolic Algorithm for Searching Acyclic AND/OR Graphs

WANG Xue-song ZHAO Ling-zhong GU Tian-long

(School of Computer and Control, Guilin University of Electronic Technology, Guilin 541004, China)

Abstract Searching AND/OR graph is an important problem-solving technique in the area of artificial intelligence. The representation of AND/OR graph based on traditional data structures greatly limits the scale of the graph that could be handled with existing AND/OR graph searching algorithm. Based on the symbolic representation of acyclic AND/OR graph using OBDD(ordered binary decision diagram), we proposed a novel symbolic algorithm for searching minimal-cost solution graph of acyclic AND/OR graph. It is shown that the symbolic algorithm has lower space complexity and can be used to handle larger-scale acyclic AND/OR graphs.

Keywords AND/OR graph, Minimal-cost solution graph, OBDDs

1 引言

OBDD(有序二叉决策图)是一种对布尔函数进行表示和操作的图形数据结构，可用于离散对象的隐式(或符号)表示并具有较高的存储效率[1]。基于 OBDD 的符号技术可有效缓解实际应用中存在的状态空间爆炸问题，目前已被应用于包括大规模集成电路验证、大规模图论问题求解等在内的众多领域[2,3]。

与或图的搜索是人工智能领域的一个重要问题，诸如机器人任务规划[4]、装配/拆卸序列规划[5,6]等许多问题的求解均可转化为与或图上最小代价解图的搜索。早期的与或图搜索算法集中在无圈图的研究上，包括 Martelli 和 Montanari 提出的自底向上搜索算法和 HS 算法[7,8]。自底向上搜索算法的基本思想是从终结叶结点开始，自底向上标记可解结点，直至根结点。HS 算法是一个启发式搜索算法。搜索过程自顶向下迭代进行，通过建立显式图 G' 来实现。最初 G' 只含根结点 s。在每次循环中，选择 G' 的一个端结点 n 进行扩展，将 n 的子结点及相应的弧加到 G' 中。然后执行自底向上的代价更新过程：对每个或结点，选择代价最小的子结点并标记相应的弧；对每个与结点，选择所有的子结点并标记相应的弧。该过程对 G' 中的每个结点 n 都标记了一个潜在解图(potential solution graph)，被标记的潜在解图即是 n 的代价最小的潜在解图。算法以这种方式循环进行，直到根结点 s 的潜在解图是一个解图，算法成功并输出该解图；或者根结点 s 的潜在解图包含一个非终结叶结点，算法宣告失败。其后，Nilsson 通过引入 k 连接的概念对 HS 算法进行了改进，提出了 AO＊算法。此算法已成为求解无圈与或图最小代价解图的经典算法，其基本思想是自顶向下地建立待扩展的局部解图，每次扩展后计算根结点的各个局部解图的代价，从中选择代价最小的局部解图进入下一步的扩展。算法如此循环进行，直至找到一个解图或者所有解图均不可解。随后的改进包括 Mahanti 和 Bagchi 的 CF，CS 算法[10]等。CF 算法和 AO＊类似，如果启发函数满足可允许性条件，两者均可给出最小代价解图，且具有类似的时间和空间复杂度。如果启发函数不满足可允许性条件，两者均无法给出最小代价解图，所不同的是当 CF 的结果可作为另一个算法 CS 的输入且满足某特定条件时，CS 可以产生最小代价解图。在含圈与或图搜索方面，Chakrabarti 最早提出了 Iterative_revise 和 REV＊算法[11]，其改进包括 Jimenez 和 Torras 的 INT 算法和 CFCREV＊算法[12]、谢青松提出的贪心搜索算法[13]以及 Ambuj Mahanti 在新的与或图理论框架上提出的 S1，S2 算法[14]等。

以上提及的改进与或图搜索算法均致力于提高已有与或图搜索算法的执行效率和求解质量。这些算法均基于与或图的显式表示，即采用了传统的数据结构，如十字链表和邻接矩

到稿日期：2009-08-28 返修日期：2009-11-09 本文受国家自然科学基金(60803033，60663005)和广西青年科学基金(桂科青 0728093，桂科青 0542036)资助。

王雪松(1981－)，硕士，讲师，主要研究方向为符号计算及其应用，E-mail：songsong8131@126.com；赵岭忠(1977－)，博士，副教授，主要研究方向为符号计算、形式化技术等；古天龙(1964－)，教授，博士生导师，主要研究方向为实时混杂系统理论、形式化方法等。

阵，对与或图逐个进行表示和存储。与基于 OBDD 的隐式表示方式相比，显式表示的存储效率较低，从而制约了以上与或图搜索技术可求解问题的规模。为此，本文尝试把 OBDD 用于无圈与或图的隐式表示，并给出了一种基于该表示的无圈与或图符号搜索算法。实验结果表明，与经典的与或图搜索算法 AO* 相比，本文算法可处理问题的规模有较大的提高。

2 预备知识

定义 1(OBDD) 给定变量序 $x_1,\cdots,x_n$ 上的变量序 π，有序二叉决策图是表示布尔函数 $f(x_1,\cdots,x_n)$ 的有向无环图 $\langle V,E\rangle$，其中：

(1)V 是结点集，其中结点分为根结点、叶结点和内部结点 3 类。没有输入边的结点称为根结点；没有后继结点的结点称为叶结点；除根结点和叶结点之外的结点称为内部结点。叶结点仅有 2 个，分别标记为 0 和 1，并表示布尔常量 0 和 1；每个非叶结点(包括根节点和内部节点)有一个标记变量，节点 u 的标记变量记做 $u.var$。

(2)E 是边集，每个非终结点具有两个输出边，分别叫做 0 边和 1 边，并分别用虚线和实线表示。节点 u 的 0 边和 1 边对应的子结点分别记做 $u.low$ 和 $u.high$。

(3)OBDD 的任一有向路径上，标记变元 $x_1,\cdots,x_n$ 以变量序 π 所规定的次序依次出现，并且每个变元至多出现一次。

OBDD 中的非终结点 u 表示布尔函数 fu：

$fu=u.var\cdot fu.high\cdot\overline{u.val}\cdot fu.low=u.var\cdot fu.var=1\vee\overline{u.val}\cdot fu.var=0$，其中，$f_0=0$，$f_1=1$，$f_{u.var=1}$ 和 $f_{u.var=0}$ 分别表示布尔函数 f 对其中变元 x_i 取 1 和取 0 后所得到的布尔函数。

定义 2(与或图) 与或图 G 是一个有向图，定义为六元组 $G=(O,A,T,N,E,s)$，其中 O 是或结点集合；A 是与结点集合；$T\subseteq O$ 是终结叶结点集合且 $T\neq\varnothing$；$N\subseteq O$ 是非终结叶结点集合；$E\subseteq((O-T-N)\cup A)\times(O\cup A)$ 是有向弧集合；$s\in O$ 是 G 的外向根结点。T 和 N 中的结点又分别称作可解叶结点和不可解叶结点。

若 $N=\varnothing$，则称 G 是叶可解的；若 E 中的弧含有代价，即 $E\subseteq((O-T-N)\cup A)\times(O\cup A)\times Z^+$，则称 G 为代价与或图；若 G 中的有向弧连成圈，则称 G 为含圈与或图，否则称 G 为无圈与或图。若无特别说明，以下与或图均是指无圈代价与或图。

若与或图 $G=(O,A,T,N,E,s)$，则其中结点的深度由以下规则定义：

1)根结点 s 的深度为 0；

2)如果结点 *node* 的深度为 h，则其后继结点的深度为 $h+1$。

无圈与或图 G 的深度 $Dep(G)$ 定义为 G 中结点的最大深度；如果有向弧 e 的弧尾结点深度为 i，则称 e 所在的层次为 $i+1$。于是 G 中的有向弧共有 $Dep(G)$ 层，且第 $Dep(G)$ 层弧弧头指向的结点均为叶结点。

与或图 G 中的结点具有以下性质：

- G 中的任意一个结点要么是与结点要么是或结点。
- 与结点 n 可解当且仅当 n 的所有子结点可解。
- 或结点 n 可解当且仅当存在 n 的一个子结点可解。

定义 3(解图) 设 $G=(O,A,T,N,E,s)$ 为无圈与或图，m 是 $O\cup A$ 中的任一结点。根结点为 m 的解图 $D(m)$ 定义如下：

1)$m\in D(m)$；

2)若 $n\in O$ 且 $n\in D(m)$，则有且仅有一个 n 的后继结点属于 $D(m)$；

3)若 $n\in A$ 且 $n\in D(m)$，则 n 的所有后继结点均属于 $D(m)$；

4)$D(m)$ 中每个叶结点 *node* 均为 G 中的终结叶结点，即 $node\in T$。

下面把以 n 为根结点的解图简称 n 的解图。

定义 4(解图的代价) 解图 $D(m)$ 中任一结点 n 的代价 $h(n,D(m))$ 定义如下：

1)$h(n,D(m))=0$，若 n 是终结叶结点；

2)$h(n,D(m))=c(n,n')+h(n',D(m))$，若 n 是或结点，n' 是 n 在 $D(m)$ 中的直接后继。$c(n,n')$ 是弧尾为 n 弧头为 n' 的弧的代价。

3)$h(n,D(m))=\sum_{i=1}^{k}[c(n,n_i)+h(n_i,D(m))]$，若 n 是与结点，$n_1,\cdots,n_k$ 是 n 在 $D(m)$ 中的直接后继。

因此，$h(m,D(m))$ 表示 m 的解图 $D(m)$ 的代价。通常，与或图 G 的根结点存在多个解图，其中代价最小的解图称作 G 的最佳解图。与或图搜索的目的即寻找根结点的最佳解图。

定义 5(净化图) 设 G 是一个与或图，则 G 的最大叶可解子图被称作 G 的净化图，即如果 $G'=(O',A',T',\varnothing,E',s')$ 是 G 的一个叶可解子图，且 G 的其它叶可解子图均是 G' 的子图，则称 G' 为 G 的净化图。当 G 无这样的子图时，定义其净化图 $G'=\varnothing$。

由于与或图 G 的任一解图也是其净化图 G' 的一个解图，因而在求解最佳解图时直接搜索净化图具有更高的效率[13]。下面不做特别说明的与或图均是指净化图。

3 无圈与或图的 OBDD 表示

OBDD 可用于表示布尔函数之外的离散对象，其它离散对象(如与或图)的 OBDD 表示以集合的 OBDD 表示为基础。利用 OBDD 表示集合基于以下原理：对集合元素进行适当的编码之后，集合可以由其特征函数来表示。集合的特征函数是一个集合元素编码变量上的布尔函数，且函数值为真当且仅当变量赋值对应的元素属于该集合。于是一个集合可由其特征函数，进而由 OBDD 来表示。利用 OBDD 表示集合可实现不同集合元素在表示上的共享，减少信息冗余。

可见，集合元素 OBDD 表示的关键是对各元素进行编码并计算其特征函数。

例 1 给定集合 $S=\{a,b,c,d,e\}$，为了获得集合的特征函数，首先需要利用布尔变量对集合元素进行编码。集合中有 5 个元素，需要 $\lceil\log_2 5\rceil=3$ 个布尔变量 x_1,x_2 和 x_3。5 个元素可编码为 000 (a)，001 (b)，010 (c)，011 (d)，100 (e)。特征函数是变量 x_1,x_2 和 x_3 上的函数 $S(x_1,x_2,x_3)$，且 $S(x_1,x_2,x_3)$ 为真当且仅当以变量 x_1,x_2,x_3 的赋值为编码的元素属于集合 S。于是就得到集合 S 特征函数表达式为：

$$S(x_1,x_2,x_3)=\overline{x_1}\,\overline{x_2}\,\overline{x_3}\vee\overline{x_1}\,\overline{x_2}x_3\vee\overline{x_1}x_2\overline{x_3}\vee\overline{x_1}x_2x_3\vee x_1\overline{x_2}\,\overline{x_3}$$

如果规定变量序 $x_1<x_2<x_3$，以上表达式可以由图 1 所示的 OBDD 表示。

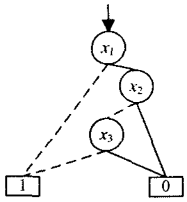

图 1　集合 $S=\{a,b,c,d,e\}$ 的 OBDD 表示

给定无圈的与或图 $G=(O,A,T,\emptyset,E,s)$，结点集 O, A, T 和 $\{s\}$ 的 OBDD 表示可直接采用上述方法实现。而有向弧集 E 的 OBDD 表示则需考虑任意边的起点、终点和代价的编码。为此，需要采用 3 组互不相交的布尔变量，其中用于起点和终点编码的布尔变量个数相同。设 G 中包含 v 个结点，则可用 $n(=\lceil \log_2 v\rceil)$ 个布尔变量，如 $(x_0,\cdots,x_{n-1})$ 对结点进行编码。对于代价的编码，我们规定：如果某有向弧的代价为 w，则该代价的编码为 w 的二进制表示。设 G 的最大解图的代价为 p，则可用 $m(=\lceil \log_2 p\rceil$ 个布尔变量，如 $(w_0,\cdots,w_{m-1})$ 对弧的代价进行编码。对于有向弧集 E，本文采用了分层表示和存储的策略，即把无圈与或图每一层的有向弧分别用一个 OBDD 表示。于是对任意弧 $(n,n',w)\in E$，如果分别用 $x'=(x_0'\cdots x_{n-1}')$ 表示弧的起点，$x=(x_0\cdots x_{n-1})$ 表示弧的终点，$w=(w_0\cdots w_{m-1})$ 表示弧的代价，则图 G 中的第 i 层弧的集合的特征函数可表示为定义在以上 3 组变量上的布尔函数 $InitialArc[i](x',x,w)$，满足 $InitialArc[i](x',x,w)$ 为真当且仅当 x' 和 x 所对应有向弧的层次为 i 且其代价为 w。于是代价与或图 G 中有向弧集 E 的特征函数为 $\bigvee_{1\leqslant i\leqslant Dep(G)} InitialArc[i](x',x,w)$。

若图 G 的结点集 O, A, T 和 $\{s\}$ 的特征函数分别表示为 $A(x)$, $O(x)$, $T(x)$ 和 $s(x)$，则无圈代价净化图 G 可表示为 $((O(x),A(x),T(x),0,\bigvee_{1\leqslant i\leqslant Dep(G)} InitialArc[i](x',x,w),s(x))$。把以上各个特征函数用 OBDD 表示，即可得到与或图 G 的符号 OBDD 表示。

例 2　图 2(a) 所示深度为 2 与或图中有向弧集合以及结点集 $O=\{0,2,3,4,5,6,7\}$，$A=\{1\}$ 和 $\{s\}=\{0\}$ 的特征函数表示如下。图中包含 8 个结点，需用 3 个布尔变量 (x_0,x_1,x_2) 进行编码。

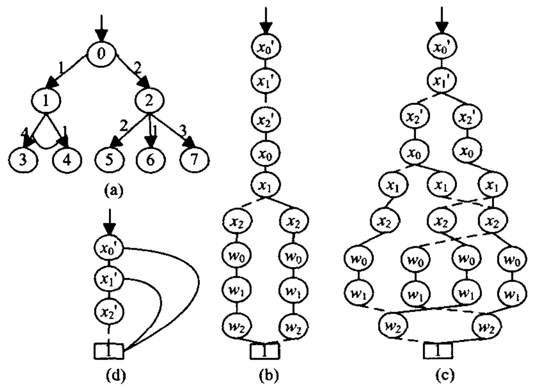

图 2　与或图及其 OBDD 表示

编码方案如下：

结点	0	1	2	3	4	5	6	7
编码	000	001	010	011	100	101	110	111

该与或图最大解图的代价不超过 6，可使用 $\lceil \log_2 6\rceil=3$ 个布尔变量 (w_0,w_1,w_2) 编码。

各层有向弧以及 A 和 O 的特征函数表达式如下：

$$InitialArc[1](x',x,w)=\overline{x_0'}\,\overline{x_1'}\,\overline{x_2'}\,\overline{x_0}\,\overline{x_1}x_2\overline{w_0}\,\overline{w_1}w_2\vee\overline{x_0'}\,\overline{x_1'}\,\overline{x_2'}\,\overline{x_0}x_1\overline{x_2}\,\overline{w_0}w_1\overline{w_2};$$

$$\begin{aligned}InitialArc[2](x',x,w)=&\overline{x_0'x_1'}x_2'\overline{x_0}x_1x_2w_0\overline{w_1}\,\overline{w_2}\vee\\&\overline{x_0'}\,\overline{x_1'}x_2'x_0\overline{x_1}\,\overline{x_2}\,\overline{w_0}\,\overline{w_1}w_2\vee\\&\overline{x_0'}x_1'\overline{x_2'}x_0\overline{x_1}x_2\overline{w_0}w_1\overline{w_2}\vee\\&\overline{x_0'}x_1'\overline{x_2'}x_0x_1\overline{x_2w_0w_1}w_2\vee\overline{x_0'}\\&x_1'\overline{x_2'}x_0x_1x_2\overline{w_0}w_1w_2;\end{aligned}$$

$$\begin{aligned}O(x')=&\overline{x_0'}\,\overline{x_1'}\,\overline{x_2'}\vee\overline{x_0'}x_1'\overline{x_2'}\vee\overline{x_0'}x_1'x_2'\vee x_0'\overline{x_1'}\,\overline{x_2'}\vee\\&x_0'\overline{x_1'}x_2'\vee x_0'x_1'\overline{x_2'}\vee x_0'x_1'x_2'=\overline{x_0'}\,\overline{x_1'}\,\overline{x_2'}\vee\\&x_0'\vee\overline{x_0'}x_1';\end{aligned}$$

$A(x')=\overline{x_0'}\,\overline{x_1'}x_2'$；$s(x')=\overline{x_0'}\,\overline{x_1'}\,\overline{x_2'}$。

图 2(b)，(c) 和 (d) 分别给出该与或图中有向弧集合 $InitialArc[1]$ 和 $InitialArc[2]$ 以及或结点集合 $O(x')$ 的 OBDD 表示。图中未画出的 1-边或 0-边均指向 0 结点。

4　无圈与或图搜索的符号 OBDD 算法

本节提出了一种无圈与或图搜索的符号 OBDD 算法。算法以自底向上的方式计算与或图 G 中各个结点最佳解图的代价。最后一层结点为终结叶结点，其最佳解图的代价均为 0。已知第 $i+1,\cdots,Dep(G)$ 层结点的最佳解图代价，第 i 层结点的最佳解图代价可按照定义计算。与传统搜索算法不同的是，以上计算过程均由基于 OBDD 的符号操作实现。

算法中需使用以下布尔表达式：

- $F(x,f)$ 为真当且仅当 x 结点的最佳解图代价为 f；
- $Sum(x',x,f)$ 为真当且仅当有向边 $\langle x',x\rangle$ 的代价与结点 x 的最佳解图代价之和为 f，并称 f 为结点 x' 的候选代价；
- $SumA(x',f)$ 为真当且仅当 x' 是与结点且 x' 的最佳解图代价为 f；
- $MinO(x',f)$ 为真当且仅当 x' 是或结点且 x' 的最佳解图代价为 f；
- $workedArc[i](x',x,w)$ 为真当且仅当结点 x' 经由第 i 层弧 (x',x,w) 才能获得最佳解图。
- $bsg[i](x',x,w)$ 为真当且仅当第 i 层有向弧 (x',x,w) 属于与或图的最佳解图。

符号算法的伪代码如图 3 所示。

输入：与或图 $((O(x'),A(x'),T(x'),0,\bigvee_{0<i<n+1} InitialArc[i](x',x,w),s(x'))$，其中 n 是与或图的深度；

输出：最佳解图 bsg 及其代价；

方法：

```
F(x,f)= T(x)∧(f=0);
i=n;
DO {
    Sum(x',x,f)=∃w∃f'(InitialArc[i](x',x,w)∧F(x,
    f')∧add(f',w,f);
    PreA(x',x,f)=Sum(x',x,f)∧A(x');
    PreO(x',x,f')=Sum(x',x,f)∧O(x');
    SumA(x',f)=0; MinO(x',f)=0;
    For each mini-term cube(x') consisting of variables in x' do
```

```
9      {
10         SumA(x',f)=SumA(x',f) ∨ (cube ∧ ∃x∃f'PreA(x',x,f')
                  ∧ f=Σ_{cube.PreA(x',x,f')} f');
11
12         MinO(x,f)=MinO(x',f) ∨ (cube ∧ ∃x∃f'PreO(x',x,f')
13                ∧ f=MIN_{cube.PreO(x',x,f')} f');
14     }
15     F(x,f')=SumA(x',f) ∨ MinO(x',f);
16       tmp1(x',x,w)=∃f(MinO(x,f) ∧ PreO(x',x,f)) ∧
                  InitialArc[i](x',x,w);
17     tmp2(x',x,w)=InitialArc[i](x',x,w) ∧ A(x');
18     workedArc[i](x',x,w)=tmp1(x',x,w) ∨ tmp2(x',
                  x,w);
19     i=i-1;
20     }
21 WHILE (i≥1)
22 l(x')=s(x');
23 For (i=1;i<n+1; i++)
24 {
25   bsg[i](x)=∃x'∃w(l(x') ∧ workedArc[i](x',x,w));
26   l(x')=bsg[i](x');
27 }
```

图 3 基于 OBDD 符号算法伪代码

该符号算法的执行可分为几个步骤。

初始化：第 1 行完成算法的初始化，把终结叶结点的最佳解图代价均赋值为 0。接下来，循环计算第 i 层弧弧尾结点的最佳解图代价。

步骤 1(第 4 行)：计算 $Sum(x',x,f)$，其中 x 是第 i 层弧弧头结点，$add(f',w,f)$ 为真当且仅当 f' 与 w 之和为 f。$add(a,b,c)$ 可根据下面的递归表达式实现：

$$add(a,b,c)=((b=0)\wedge(a=c))\vee\exists b',c'(inc(b,b')\wedge inc(c,c')\wedge add(a,b',c'))$$

式中，inc 表示所有形如 $(i,i+1)$ 的序偶集合的特征表达式。在变量 a,b,c 的取值为有限域的情况下，可通过不动点计算对递归函数 $add(a,b,c)$ 求值。

步骤 2(第 5－15 行)：计算第 i 层弧弧尾结点的最佳解图代价。由于与结点的最佳解图代价的计算方法和或结点不同，因此按照结点的种类将上一步得到的结果分为两部分 $PreA(x',x,f)$ 和 $PreO(x',x,f)$，然后分情况分别求取弧尾的最佳解图代价。第 10,11 行针对弧尾结点为与结点的情况：任意由 x' 中变量构成的小项均表示一个可能的弧尾结点，对 $cube(x')$ 的所有候选代价求和可得结点 $cube(x')$ 的最佳解图代价，从而得到 $SumA(x',f)$ 的特征表达式，其中条件 $\exists x\exists f'(PreA(x',x,f')$ 保证 $cube(x')$ 结点是第 i 层弧弧尾指向的与结点。类似地，第 12,13 行针对弧尾结点为或结点的情况：$cube(x')$ 所有候选最佳代价中的最小值为结点 $cube(x')$ 的最佳解图代价，从而得到 $MinO(x',f)$ 的特征表达式，条件 $\exists x\exists f'(PreO(x',x,f')$ 则保证 $cube(x')$ 对应的结点是第 i 层弧弧尾指向的或结点。

综合起来，第 i 层弧弧尾结点的最佳解图代价表达式 $F(x,f)$ 可由下面的公式计算：

$$F(x,f)=SumA(x',f)\vee MinO(x',f)$$

步骤 3(第 16－18 行)：在 $workedArc[i](x',x,f)$ 中保存第 i 层弧的弧尾结点 x' 为获得最佳解图在本层所选择的路径(弧)，如果 x' 为或结点，则只能选择一条弧；如果 x' 是与结点，则需选择所有的以该结点为弧尾的弧。第 16 行针对或结点的情况，第 17 行针对与结点的情况。保存 $workedArc[i](x',x,f)$ 的目的是为了在循环结束之后从与或图的根结点出发求解整个与或图的最佳解图。

循环执行以上步骤，自底向上求解每层弧的弧尾结点的最佳解图代价，直至求出根结点的最佳解图代价。

步骤 4(第 22－27 行)：从根结点出发，自顶向下从 $workedArc$ 中析取根结点的最佳解图。

下面，通过例子来说明符号 OBDD 算法的搜索过程。

以图 2 所示的与或图为例，按照上述算法步骤来求解无圈代价与或图的最佳解图。为描述方便，用 Set_f 表示布尔函数 f 所表示的离散对象的集合。

图 2 所示的与或图含有 2 层弧，$n=2$；

初始化　$Set_{F(x,f)}=\{(3,0),(4,0),(5,0),(6,0),(7,0)\}$；

第 1 次循环：

步骤 1　$Set_{Sum(x',x,f)}=\{(1,3,4),(1,4,1),(2,5,2),(2,6,1),(2,7,3)\}$；

步骤 2　$Set_{SumA(x',f)}=\{(1,5)\}$；$Set_{MinO(x',f)}=\{(2,1)\}$；$Set_{F(x,f)}=\{(1,5),(2,1)\}$；

步骤 3　$Set_{workedArc[2](x',x,f)}=\{(1,3,4),(1,4,1),(2,6,1)\}$；

第 2 次循环：

步骤 1　$Set_{Sum(x',x,f)}=\{(0,1,6),(0,2,3)\}$；

步骤 2　$Set_{SumA(x',f)}=\emptyset$；$Set_{MinO(x',f)}=\{(0,3)\}$；$Set_{F(x,f)}=\{(0,3)\}$；

步骤 3　$Set_{workedArc[1](x',x,f)}=\{(0,2,2)\}$；

由于第 1 层弧的弧尾结点即是与或图的根结点，因此由本次循环第 2 步可知根结点的最佳解图代价为 3。

步骤 4　$Set_{bsg[1](x',x,w)}=\{(0,2,2)\}$，$Set_{bsg[2](x',x,w)}=\{(2,6,1)\}$，因而根结点只有一个仅包含一条路径“0-2-6”的最佳解图。

至此，算法结束。

5　实验结果及分析

为了检验符号算法的性能，本文利用 Colorado 大学的 CUDD 软件包实现了该算法。以随机产生的无圈与或图为输入，将本文算法与传统算法进行实验对比。AO＊算法和 CF 算法在启发函数满足可允许性条件下，均可给出最小代价解图，且时间复杂度和空间复杂度均相当。这里仅以 AO＊算法为例进行比较。实验平台配置如下：操作系统 Windows XP，CPU P4 1.5GHz，内存 256M，缓存 220M。

表 1 给出了利用 AO＊算法和本文符号算法获得与或图最佳解图的时间。由表 1 可见，在求解较小规模的与或图时，AO＊算法的时间效果较好。但当与或图规模增大到一定程度(表中为 3×104 个结点)时，AO＊算法产生溢出出错，而本文的算法仍能在较短的时间内计算出最佳解图。在同样的运行平台上，本文算法求解问题的规模，以与或图结点数目计，可达 2×106，可见本算法所需的存储空间小于传统的 AO＊

算法。以上实验结果表明,基于 OBDD 的与或图符号搜索算法具有较低的空间复杂度,可处理问题的规模与使用显式存储技术的传统算法相比有较大的提高。

表 1 符号算法与 AO*算法执行结果比较

名称	结点数	与结点数	与结点百分比	符号算法运行时间(s)	AO*算法运行时间(s)
a1	101	13	13%	0.02	0.015
a2	1001	155	15%	0.20	0.046
a3	5001	801	16%	1.11	0.109
a4	10001	2039	20%	2.73	0.109
a5	15001	3006	20%	4.50	0.094
a6	20000	3246	16%	5.36	0.140
a7	23000	4626	20%	7.50	0.125
a8	30001	4873	16%	8.14	溢出
a9	50001	4998	10%	18.06	溢出
a10	60000	5367	9%	25.33	溢出
a11	80000	6481	8.1%	35.39	溢出
a12	100000	8068	8.1%	46.42	溢出
a13	200002	16271	8.1%	97.89	溢出
a14	300000	62256	20%	169.89	溢出
a15	500000	102434	20%	308.45	溢出
a16	1000000	202903	20%	733.00	溢出
a17	1500001	303279	20%	1223.03	溢出
a18	2000000	403752	20%	1740.52	溢出

结束语 与或图搜索是一项重要的人工智能问题求解技术,本文对基于 OBDD 的无圈与或图符号求解技术进行了探讨,主要贡献有:

1)与或图的符号表示技术,包括与结点、或结点、叶子结点和有向弧的 OBDD 表示,以及代价有向弧的分层表示和存储技术;

2)基于以上表示的与或图自底向上解图最小代价计算方法,以及自顶向下的解图析取技术。

以后将研究有圈与或图的符号搜索算法,以及这些算法在大型装配序列规划、机器人规划中的应用。

参考文献

[1] Bryant R E. Graph-based algorithms for Boolean function manipulation[J]. IEEE Trans. on Computers, 1986, 8: 677-691

[2] Drechsler R, Sieling D. Binary decision diagrams in theory and practice[J]. International Journal on Software Tools for Technology Transfer, 2001, 3(2): 112-136

[3] Bloem R, Gabow H N, Somenzi F. An algorithm for strongly connected component analysis in n log n symbolic steps[C]// Proceedings of International Conference on Formal Methods in Computer-Aided Design. 2000: 37-54

[4] Cao T, Sanderson A C. AND/OR net representation for robotic task sequence planning[J]. IEEE Trans. Systems Man Cybernet—Part C: Applications and Reviews, 1998, 28(2): 204-218

[5] DeMello L S H, Sanderson A C. A correct and complete algorithm for the generation of mechanical assembly sequences[J]. IEEE Trans. Robotics and Automation, 1991, 7(2): 228-240

[6] Homen de Mello L S. AND/OR graph representation of assembly plans[J]. IEEE Trans. Robotics and Automation, 1990, 6(2): 188-199

[7] Martelli A, Montanari U. Additive AND/OR Graphs[C]// Proceedings of the International Joint Conference on Artificial Intelligence. 1973: 1-11

[8] Martelli A, Montanari U. Optimizing Decision Trees Through Heuristically Guided Search[J]. Communications of the ACM, 1978, 21(12): 1025-1039

[9] Nilsson N J. Principles of Artificial Intelligence[M]. Palo Alto: Tioga Publishing Company, 1980

[10] Mahanti A, Bagchi A. AND/OR Graph Heuristic Search Methods[J]. Journal of the Association for Computing Machinery, 1985, 32(1): 28-51

[11] Chakrabarti P P. Algorithms for Searching Explicit AND/OR Graphs and Their Applications to Problem Reduction Search[J]. Artificial Intelligence, 1994, 65: 329-345

[12] Jiménez P, Torras C. An Efficient Algorithm for Searching Implicit AND/OR Graphs with Cycles[J]. Artificial Intelligence, 2000, 124: 1-30

[13] 谢青松,王岩冰,马绍汉. 显式与或图的一种新的贪心搜索算法[J]. 计算机研究与发展, 1997, 34(12): 887-892

[14] Mahanti A, Ghose S, Sadhukhan S K. A Framework for Searching AND/OR Graphs with Cycles[R]. The AI/0305001. Computing Research Repository, 2003

(上接第 143 页)

根据冲突所涉及的冲突因素,再分别实现消解。这一算法可以初步实现并行执行的 3 类冲突的自动消解。本方法建立在已知手工分类和消解的基础上,没有包含对可能产生的新的重构规则的自动识别及消解。

参考文献

[1] Corradini A, Montanari U, Rossi F, et al. Algebraic approaches to graph transformation I: Basic concepts and double pushout approach[C]//Rozenberg G. editor. Handbook of Graph Grammars and Computing by Graph transformation. Volume 1: Foundations. World Scientific, 1997

[2] Mens T. On the use of graph transformations for model refactoring[C]// Generative and Transformational Techniques in Software Engineering. LNCS 4143. Springer, 2006: 219-257

[3] Mens T, Van Eetvelde N, Demeyer S, et al. Formalizing refactorings with graph transformations[J]. Journal on Software Maintenance and Evolution, 2005, 17(4): 247-276

[4] Habel A, Hoffmann B. Parallel Independence in Hierarchical Graph Transformation[M]. ICGT, 2004: 178-193

[5] Taentzer TG, Runge O. Detecting structural refactoring conflicts using critical pair analysis[J]. Electronic Notes in Theoretical Computer Science, 2005, 127(3): 113-128

[6] Plump D. Critical pairs in term graph rewriting [C]// Proc. Mathematical. Foundations of Computer Science. volume 841 of Lecture Notes in Computer. London, UK: Springer-Verlag, 1994

[7] Mens T. A formal foundation for object-oriented software evolution[D]. Department of Computer Science, Vrije Universiteit Brussel, September 1999

[8] http://user.cs.tu-berlin.de/~gragra/agg/critical_pairs.html.

[9] 刘辉,麻志毅,邵维忠. 模型转换中的特性保持的描述与验证[J]. 软件学报, 2007, 18(10): 2369-2379

[10] Lambers L, Ehrig H, Orejas F. Efficient detection of conflicts in graph-based model transformation [C] // Proc. International Workshop on Graph and Model Transformation (GraMoT'05). Electronic Notes in Theoretical Computer Science, Tallinn, Estonia, Elsevier Science, 2005