Dempster证据合成法则的通用实现方法

傅仰耿[1] 巩晓婷[2] 张玺霖[1] 吴英杰[1]
(福州大学数学与计算机科学学院 福州 350108)[1] (福州大学公共管理学院 福州 350108)[2]

摘 要 证据推理是不确定性推理的一种重要方法,而Dempster合成法则是进行证据推理的核心。针对目前Dempster证据合成法则的算法实现不能处理合成结果的焦元为多个假设或命题的集合、近似算法的计算不够精确等问题,提出通过位向量表示识别框架子集,并利用线性表、平衡树等数据结构实现证据合成的3种精确、通用算法。理论分析与仿真实验表明,所实现的算法是有效的。

关键词 证据推理,不确定性,Dempster合成法则,算法实现
中图法分类号 TP181 **文献标识码** A

General Algorithms for Dempster's Combination Rule of Evidence

FU Yang-geng[1] GONG Xiao-ting[2] ZHANG Xi-lin[1] WU Ying-jie[1]
(College of Mathematics and Computer Science, Fuzhou University, Fuzhou 350108, China)[1]
(School of Public Administrator, Fuzhou University, Fuzhou 350108, China)[2]

Abstract Evidential reasoning is an important method of uncertainty reasoning, while Dempster's combination rule is the core of evidential reasoning. The focal element of synthesis results may be a collection of a number of assumptions or propositions, but the previous algorithms for Dempster's combination rule of evidence cannot handle this situation, and their approximation algorithms for the calculation are also not precise enough. Therefore, by implementing subsets of the frame of discernment with bit vectors, three precise general algorithms for Dempster's combination rule were proposed, which use linear list and balanced tree. Theoretical analysis and simulation results show that the algorithms are effective.

Keywords Evidential reasoning, Uncertainties, Dempster's combination rule, Algorithm implementatio

1 引言

证据理论(也称为D-S证据理论)[1,2]是由Dempster和Shafer提出来的一种用于不确定性推理的重要理论,目前已经广泛运用于信息融合、专家系统、模式识别、数据库与知识发现、不确定多准则决策等领域,并取得了良好的应用效果。其中最核心的就是用于合成两个和多个证据的Dempster合成法则,而该法则在计算上存在着潜在的组合爆炸问题。为了减少Dempster合成法则的计算量,人们提出很多实现方法,主要有3种途径[3,4]:

(1)针对特定证据组织结构而构造的快速算法。如,文献[5]针对支持或者否定识别框架的某个假设这样简单的证据结构,提出了一种快速实现的方法。

(2)通过减少焦元个数来达到简化计算的近似计算方法。如,文献[6]通过证明信度函数的贝叶斯近似的合成等于合成的贝叶斯近似提出了信度函数的贝叶斯近似计算方法;文献[7]通过处理嵌套焦元信度给出了一种一致近似的算法;文献[8]通过对信度函数值排序并设定阈值构造了(k,l,x)的近似计算方法,等等。

(3)针对基于规则的系统的应用困难而提出的修改方法。如,文献[9]将模糊集引入了证据理论。

然而,对于以上这3种途径,途径(1)只能面向特定结构的证据;途径(2)虽然可以大大提高证据合成的计算速度,但牺牲了计算的精确度,而且对于合成结果的焦元是多个假设或命题的情形不适用;途径(3)是通过引入其他理论对Dempster合成法则进行修改扩展,已经不是单纯的Dempster合成法则。由此可知,考虑到计算量的问题,目前Dempster合成法则的实现方法均未精确实现焦元可能是识别框架的任意子集的情形,而这种情形所实现的算法却可用于证据理论的所有应用领域。

例如,设$\theta=\{a,b,c\}$,$m_1(\{a,b\})=0.5$,$m_1(c)=0.3$,$m_1(\theta)=0.2$;$m_2(a,b)=0.3$,$m_2(c)=0.4$,$m_2(\theta)=0.3$,可得Dempster合成法则的合成结果为$m(a,b)=0.507042$,$m(c)=0.408451$,$m(\theta)=0.084507$。显然这种情形不适合使用途径

到稿日期:2012-03-19 返修日期:2012-05-07 本文受国家杰出青年科学基金(70925004),福建省教育厅科技项目(JA10035)和福州大学科技发展基金(2009-XY-17)资助。

傅仰耿(1981—),男,博士生,讲师,主要研究方向为不确定多准则决策、隐私保护,E-mail:fu@fzu.edu.cn;巩晓婷(1982—),女,硕士,讲师,主要研究方向为信息管理、信息隐藏;张玺霖(1988—),男,硕士生,主要研究方向为隐私保护;吴英杰(1979—),男,博士,讲师,主要研究方向为数据安全、数据挖掘和算法设计。

(1)和(2)解决,因为合成结果的焦元包含了多个假设或命题的情形($\{a,b\}$),所以研究这种通用情形下的 Dempster 合成法则的算法实现具有一定的现实意义。

本文简单介绍了证据理论的相关概念,主要阐述通过位向量表示识别框架子集,并利用线性表、平衡树等数据结构来实现 Dempster 合成法则的 3 种通用的精确计算方法,并对这 3 种实现方法进行了分析比较,最后给出了结论。

2 Dempster 证据合成法则的通用实现

2.1 相关概念

以下简要介绍证据理论的相关定义及性质[2]。

定义 1 假设 $H=\{H_1,H_2,\cdots,H_L\}$ 为某问题的各个互相独立、整体上完备的命题或者假设所构成的一个有限集合,则称 H 为识别框架。

定义 2 如果函数 $m:2^H\to[0,1]$ 满足 $m(\phi)=0$ 和 $\sum_{A\subseteq H}m(A)=1$,则称函数 m 为识别框架 H 上的基本概率分配(Basic Probability Assignment,BPA)。对于任意 $A\subseteq H$,$m(A)$ 也称为 mass 函数,用于表示证据支持命题或假设 A 发生的程度。这里 ϕ 表示空集,A 是 H 的子集,2^H 为 H 的幂集,包含所有 H 的子集,即 $2^H=\{\phi,\{H_1\},\{H_2\},\cdots,\{H_L\},\{H_1,H_2\},\cdots,\{H_1,H_L\},\cdots,H\}$。

定义 3 对于任意 $A\subseteq H$,如果 $m(A)>0$,则称 A 为证据的焦元(focal elements)。

定义 4 所有焦元的集合称为核。

定义 5 假设 m_1 和 m_2 是定义在 H 上的两个 mass 函数(即两个不同证据的 BPA),则用于合成这两个证据的 Dempster 合成法则可以表示为:

$$m(C)=[m_1\oplus m_2](C)=\begin{cases}0, & C=\phi\\ \dfrac{\sum_{A\cap B=C}m_1(A)m_2(B)}{1-\sum_{A\cap B=\phi}m_1(A)m_2(B)}, & C\neq\phi\end{cases}\tag{1}$$

这里,$\oplus$ 表示直和运算,很显然 A 和 B 是焦元,而 $[m_1\oplus m_2](C)$ 是由 m_1 和 m_2 合成的一个 BPA;分母 $1-\sum_{A\cap B=\phi}m_1(A)m_2(B)$ 被称为归一化因子,其中 $\sum_{A\cap B=\phi}m_1(A)m_2(B)$ 表示证据之间的冲突程度,易知:

$$1-\sum_{A\cap B=\phi}m_1(A)m_2(B)=\sum_{A\cap B\neq\phi}m_1(A)m_2(B)\tag{2}$$

Dempster 合成法则已经被证明满足交换律和结合律,即 $m_1\oplus m_2=m_2\oplus m_1$(交换律)且 $(m_1\oplus m_2)\oplus m_3=m_1\oplus(m_2\oplus m_3)$(结合律),这两个性质允许证据按任意顺序合成。因此,对于多个证据的合成,可以使用两两合成的方式。

2.2 通用实现

以下给出实现 Dempster 合成法则的算法框架,再分别介绍采用 3 种不同数据结构[10]的具体实现方法。

显然,对 Dempster 合成法则的计算机算法实现的主要任务是计算两个给定的证据的 BPA(即 mass 函数 m_1 和 m_2)的合成结果,即给出实现式(1)的计算方法。由式(1)可以看出,Dempster 合成的主要计算量在于需要不断地求两个核中焦元的交集。值得注意的是,每个核中焦元的规模最坏情况下可能达到 $2^{|H|}$($|H|$ 表示识别框架 H 中元素的个数),这也是为什么证据合成的计算量可能导致指数级爆炸的原因。以下是实现 Dempster 合成法则的算法框架:

```
void DempsterCombination(Set s1, Set s2){
  for each a in s1//枚举 s1 中所有的焦元
    for each b in s2//枚举 s2 中所有的焦元
      c=a∩b //求两个焦元的交集
        if (c≠ϕ) m(c)+=m1(a) * m2(b);
          //将交集不为空的 mass 函数值的乘积
          //累加到对应的焦元的 mass 函数
    end for
  end for
}
```

在最坏情况下,这个算法的两重循环的时间复杂度显然需要 $O(2^{|H|}*2^{|H|})$。,对于循环体内所要求的两个焦元的交集($c=a\cap b$),同时存储其 mass 函数值时还要定位其位置($m(c)$),不同的集合实现方式所需要的时间复杂度不一样。另外,上述算法框架及以下提出的 3 种实现方法均未包含归一化的步骤,这是因为由式(2)可知,归一化结果可以很容易由上述合成结果求得。

注意到焦元所包含的元素是识别框架 H 的子集,所有焦元的集合(核)最坏情况下是定义在集合 H 之上的幂集 2^H,因此对于任何一个集合 $A\subseteq H$,可以定义一个特征函数为:

$$\delta_A(H_i)=\begin{cases}1, & H_i\in A\\ 0, & H_i\notin A\end{cases}\tag{3}$$

即用一个 n 位的向量 v 来存储 A 的特征函数值,如 $n=8$ 时,若 $A=\{H_8,H_7,H_4,H_3,H_1\}$,则对应的位向量 v 为 11001101,这可以看成一个二进制数,若写成十进制的整型数,则为 205。由此,在 n 不太大(如,$n\leqslant 31$)的情况下,A 可以直接用一个整型数来表示(而对于 n 较大的情况,则可以使用无符号的整型数组来表示,具体参见文献[10]),这样可以很方便地利用按位与运算实现两个焦元的求交集。如,若 H 的子集 $B=\{H_7,H_5,H_4,H_3,H_2\}$(即对应的位向量为 01011110,整型数为 94),此时求 A 与 B 的交集,只需求它们对应整型数的按位与运算,即 205&94,可得到 00001100,即 $\{H_4,H_3\}$。

以下分别介绍 3 种运用不同数据结构来实现上述算法框架的方法,并分析其相应的时间复杂度。

算法 1 使用数组实现(dcArray)

在使用位向量表示识别框架子集的基础上,用数组来存放两个证据的 BPA,即可得到识别框架子集与其 mass 函数值存储位置的一一映射,实现随机存取合成结果的 BPA。

数据结构定义如下:

```
const int st=3; //识别框架 H 所包含元素的个数
const int N=1 << st;//H 的幂集所包含元素的个数
double m1[N],m2[N],m[N];//m1,m2 表示待合成的两个证据的
                        BPA,m 为合成结果的 BPA
```

算法描述如下:

```
void dcArray() {
  int i,j,k;
  initialBPA(m);//初始化合成结果的 BPA
  for (i=1;i < N;i++)
    for (j=1;j < N;j++)
      if (i&j) //通过按位与求两个焦元的交集
        m[i&j]+=m1[i] * m2[j];
```

```
        //将交集不为空的 mass 函数值乘积
    end for
      //累加到对应的焦元的 mass 函数
    end for
}
```

上述算法实现非常简洁，求交集仅需常数时间，最坏情况下总的时间复杂度仅为 $O(2^{|H|} * 2^{|H|})$。但是该算法不管实际焦元个数的多少，始终要开辟 3 个 $2^{|H|}$ 大小的数组，使得 $|H|$ 只能取一个很小的值，如小于等于 16(因为普通 PC 的内存和计算能力都很有限)。然而在现实生活中，很多应用的焦元个数却远远小于 $2^{|H|}$，在这种情况下仍然开辟指数级规模的数组不仅浪费空间，更是浪费时间。

算法 2 使用链表实现(dcLink)

为解决数组实现所存在的问题，考虑分别使用两个链表 a，b 来存放两个证据的焦元及对应的 mass 函数值，合成的结果也使用一个链表 c 存放。

数据结构定义如下：

```
typedef struct node * link;
typedef struct node { //链表节点
  bitset focalelem;//焦元
  double mass; //焦元对应的 mass 函数值
  link next;
} Node;
typedef struct llist * List;//链表定义
typedef struct llist{
    link first;
} llist;
```

算法描述如下：

```
void dcLink(List a,List b){
  枚举链表 a 和 b 中的所有节点对:p 和 q;
  bitset fe=q->focalelem & p->focalelem;//求两个焦元交集
  if (null! =fe) //如果 q->focalelem 与 p->focalelem 的交集不为空
    if (c)
      s=c->first;
      findpos(fe);//在链表 c 中查找焦元 fe 的位置
    else //否则在链表 c 的表首新建一个节点,将 fe 保存下来
        s->focalelem=fe;
    end if
  s->mass+=q->mass * p->mass;//累加上对应 mass 值
  end if
}
```

很显然上述算法枚举所有的交集 fe 需要的计算时间为 $O(2^{|H|} * 2^{|H|})$，求交集 fe 也仅需常数时间，但存储 fe 的 mass 函数值时，findpos 需要 $O(2^{|H|})$，所以最坏情况下总的计算时间复杂度为 $O(2^{|H|} * 2^{|H|} * 2^{|H|})$，这显然比数组实现的效率低很多。然而实际上，当待合成证据的核的规模 N、M 远远小于 $2^{|H|}$ 时，相应的复杂度也退化为 $O(N * M * N * M)$。这种情形下，使用链表实现的优势就可以充分发挥出来。

算法 3 使用向量与映射实现(dcVectorMap)

在上述使用链表实现的算法中，主要的瓶颈在于存储合成结果的焦元所对应的 mass 函数值时，需要耗费 $O(N * M)$ 来查找其在链表中对应的存储位置。若采用平衡树来存储合成结果，可以将上述算法的时间复杂度进一步降低到 $O(N * M * \log(N * M))$，然而平衡树和链表的实现细节较为繁琐。为了综合数组实现和链表实现的优点，可以直接使用 STL[11] 提供的向量和映射来实现 Dempster 合成法则。

数据结构定义如下：

```
struct point { //定义焦元与 mass 函数值的点对
  int v; //焦元
  double p;//焦元对应的 mass 函数值
};
vector<point> m1,m2;//m1,m2 表示待合成的两个证据的 BPA,存放在向量中
map<int,double> m;// m 为合成结果的 BPA,存放在映射中
```

具体算法描述如下：

```
void dcVectorMap() {
  m. clear(); //初始化合成结果的 BPA
  for (i=0;i < m1. size();i++)
  for (j=0;j < m2. size();j++)
  if (m1[i]. v&m2[j]. v)//求两个焦元的交集
  m[m1[i]. v&m2[j]. v]+=m1[i]. p * m2[j]. p;
  //将交集不为空的 mass 函数值乘积
  end for
    //累加到对应的焦元的 mass 函数
  end for
}
```

由此可见上述算法实现也很简洁，使用向量存储两个待合成证据的 BPA 可实现随机访问；使用映射存放合成结果的 BPA，因为映射的内部实现是平衡树，所以仅需对数级的查找定位时间。因此，该算法总的时间复杂度为 $O(N * M * \log(N * M))$，N、M 分别为待合成两个证据的核的规模，在最坏情况下，即当 $N=M=2^{|H|}$ 时，时间复杂度为 $O(|H| * 2^{|H|} * 2^{|H|})$。

以上介绍了 3 种实现 Dempster 合成法则的算法，其中 dcArray 算法由于开辟了 3 个 $2^{|H|}$ 规模的数组，不管待合成证据的核的规模如何，时间复杂度均为 $O(2^{|H|} * 2^{|H|})$；dcLink 算法则为 $O(2^{|H|} * 2^{|H|} * 2^{|H|})$，但在待合成的证据的核的规模 N、M 远远小于 $2^{|H|}$ 时，相应的复杂度也退化为 $O(N * M * N * M)$；而 dcVectorMap 算法则将这种情形下的时间复杂度进一步改进为 $O(N * M * \log(N * M))$，当 $N=M=2^{|H|}$ 时，时间复杂度为 $O(|H| * 2^{|H|} * 2^{|H|})$。因此，dcVectorMap 算法优于 dcLink 算法，而 dcArray 算法和 dcVectorMap 算法有不同的适用情形。在实际的应用中，可以根据待合成证据的核的规模大小来选择使用 dcArray 算法或者 dcVectorMap 算法。

3 实验比较与分析

3.1 实验结果

为了说明 dcArray 算法和 dcVectorMap 算法具有不同的适用情形，实现了上述两种算法，在保证不会出现证据完全冲突[12]的情形(即归一化时分母为零的情形)下，对核的规模为 $|H|$、$|H|^2$、$|H|^3$ 和 $2^{|H|}$ 的 4 种情形分别随机生成两组待合成

(下转第 219 页)

表明，这种方法能够有效去除因垄线不直存在的抖动问题，从而准确完成农机车载 GPS 的定位，具有较高的使用价值。

参 考 文 献

[1] Huang J-Y, Tsai C-H. Improve GPS positioning accuracy with context awareness [C]// IEEE International Conference on Ubi-Media Computing. 2008:94-99

[2] 陈娇，姜国权，杜尚丰，等. 基于垄线平行特征的视觉导航多垄线识别[J]. 农业工程学报，2009，25(12)：107-113

[3] 杨玲，李博峰，楼立志. 不同对流层模型对 GPS 定位结果的影响[J]. 测绘通报，2009(04)：9-11

[4] 朱志宇. 基于 ukf 的闪烁噪声机动目标跟踪[J]. 计算机仿真，2007，24(11)：120-123

[5] Abdel-Hafez M F. The Autocovariance least-squares technique for cps measurement nose estimation [J]. IEEE Transactions on Vehicular Technology, 2010, 59(2): 574-588

[6] 卢晓燕，卢京潮，周良荣，等. 野值存在下的 BP 网络自适应卡尔曼滤波[J]. 计算机仿真，2010，27(01)：158-161

[7] Hill J, Szew cyk R, Woo A, et al. System Architecture Directions for Networked Sensor[C]// Proceeding of International Conference on Architectural Support for Programming Languages and Operating Systems. 2009, 11: 93-104

[8] Mainwaring, Polastre J, Szewczyk R, et al. Wireless Sensor Networks for Habitat Monitoring[C]// Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications. 2010: 94-97

[9] 王金刚. GPS 数字中频信号仿真及捕获验证[J]. 重庆邮电大学学报：自然科学版，2010，22(2)：252-256

(上接第 183 页)

的 BPA 数据，并在一台硬件配置为 CPU2. 6G、内存2. 0G，装有 Windows 7 操作系统的 PC 上，选择 $|H|=5,9,13,15,16,21,31$，对两个算法的运行时间进行记录，得到表 1 所列的数据。

表 1 两种实现方法的计算时间比较

核的规模	$\|H\|$	$\|H\|^2$	$\|H\|^3$	$2^{\|H\|}$
焦元个数	5	25	125	32
dcArray	0.000	0.000	/	0.000
dcVectorMap	0.000	0.000	/	0.000
焦元个数	9	81	729	512
dcArray	0.002	0.000	/	0.000
dcVectorMap	0.000	0.000	/	0.169
焦元个数	13	169	2197	8192
dcArray	0.651	0.543	0.634	0.607
dcVectorMap	0.000	0.033	2.262	30.791
焦元个数	15	225	3375	32678
dcArray	6.507	6.550	6.648	6.873
dcVectorMap	0.000	0.066	6.183	>60.000
焦元个数	16	256	4096	65536
dcArray	25.558	25.600	25.757	26.193
dcVectorMap	0.001	0.085	9.473	>60.000
焦元个数	21	441	9261	924844032
dcArray	—	—	—	—
dcVectorMap	0.002	0.499	>60.000	—
焦元个数	31	961	29791	2147483648
dcArray	—	—	—	—
dcVectorMap	0.004	2.848	>60.000	—

3.2 结果分析

从表 1 的数据可以看出，在识别框架较小的情况下，如 $|H|=5,9$ 时，两种算法的效率相差并不大；随着识别框架中元素的不断增多，当 $|H|=13$ 时，dcAarry 算法依旧保持较好的性能，而 dcVectorMap 算法的性能则随焦元个数的增加而明显下降；当 $|H|=15,16$ 时，dcVectorMap 算法已经无法在 1 分钟之内解出规模为 $2^{|H|}$ 的合成结果，此时的 dcAarry 算法的计算时间也分别达到 6 秒和 25 秒；当 $|H|>16$ 时，其也不能在 1 分钟之内解出结果；相反，当 $32>|H|>16$ 时，dcVectorMap 算法还可以解出部分规模的结果。因此在实际应用中，当识别框架所包含的元素(命题或者假设)个数少于 16 时，可以直接使用 dcAarry 算法；反之，在焦元个数小于 8192 时，则可以采用 dcVectorMap 算法。

结束语 证据理论已经得到了广泛的应用，对于证据推理中的核心——Dempster 合成法则的算法实现，前人已经取得了很多有益的研究成果。针对目前 Dempster 合成法则的算法实现不够精确、不适用于合成结果的焦元包含了多个假设或命题的情形，提出 3 种精确实现 Dempster 合成法则的通用算法，即 dcArray、dcLink 和 dcVectorMap。通过比较分析与实验表明，在实际应用中，当识别框架不太大($|H|\leqslant 16$)时，一般可以选择 dcArray 算法，对 $|H|$ 较大而焦元个数较小的情形，则可以选择 dcVectorMap 算法。

参 考 文 献

[1] Dempster A P. A generalization of Bayesian inference(with discussion)[J]. Journal of the Royal Statistical Society Series B, 1968, 30(2): 205-247

[2] Shafer G. A Mathematical Theory of Evidence[M]. Princeton: Princeton University Press, 1976

[3] 徐从富，耿卫东，潘云鹤. Dempster-Shafer 证据推理方法理论与应用的综述[J]. 模式识别与人工智能，1999，12(4)：424-430

[4] 徐从富，耿卫东，潘云鹤. 面向数据融合的 DS 方法综述[J]. 电子学报，2001，29(3)：393-396

[5] Barnett J A. Computational methods for a mathematical theory of evidence[C]// Proceedings of 7th International Joint Conference on Artificial Intelligence, 1981. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981: 868-875

[6] Voobraak F. A computationally efficient approximation of Dempster-Shafer theory[J]. International Journal of Man-Machine Study, 1989, 30: 525-536

[7] Dubois D, Prade H. Consonant approximations of belief functions [J]. International Journal of Approximate Reasoning, 1990, 4: 279-283

[8] Tessem B. Approximations for efficient computation in the theory of evidence[J]. Artificial Intelligence, 1993, 61: 315-329

[9] Yen J. Generalizing the Dempster-Shafer theory to fuzzy sets [J]. IEEE Trans. on Systems, Man, and Cybernetics, 1990, 20 (3): 559-570

[10] 王晓东. 数据结构(C 语言描述)[M]. 北京：电子工业出版社，2007

[11] 侯捷. STL 源码剖析[M]. 武汉：华中科技大学出版社，2002

[12] Zadeh L A. Review of Shafer's a mathematical theory of evidence[J]. AI Magazine, 1984, 5: 81-83