文章编号: 1001-0920(2007)05-0520-05

用有序 FP-tree 挖掘最大频繁项集

于 红[1,2], 王秀坤[1], 孟 军[1]
(1. 大连理工大学 电信工程学院, 辽宁 大连 116023; 2. 大连水产学院 信息工程学院, 辽宁 大连 116023)

摘 要: 提出了完全前缀路径和有序 FP-tree 的概念,给出根据数据项所在的层建立有序 FP-tree 的方法,利用有序 FP-tree 表示数据. 提出用有序 FP-tree 中的完全前缀路径进行最大频繁项集挖掘的算法——MFIM 算法,该算法利用有序 FP-tree 中的完全前缀路径对挖掘算法进行优化. 实验结果表明,该算法对于浓密数据集中挖掘长模式具有较好的性能.

关键词: 最大频繁项集; 有序 FP-tree; 数据挖掘; MFIM 算法
中图分类号: TP311.133 **文献标识码**: A

Mining maximal frequent itemsets using the ordered FP-tree

YU Hong[1,2], *WANG Xiu-kun*[1], *MENG Jun*[1]
(1. School of Electronics and Information Engineering, Dalian University of Technology, Dalian 116023, China; 2. School of Information Engineering, Dalian Fisheries University, Dalian 116023, China. Correspondent: YU Hong, E-mail: myulee@sina.com)

Abstract: The ordered FP-tree and complete prefix path are proposed. A method of constructing the ordered FP-tree is presented according to the level of nodes. The ordered FP-tree is used to compress the dataset. The maximal frequent itemsets mining algorithm is presented. The complete prefix path is used to optimize the maximal frequent itemsets mining. The experiment result shows that the algorithm has better performance for the long pattern mining in density datasets.

Key words: Maximal frequent itemsets; Ordered FP-tree; Data mining; MFIM algorithm

1 引 言

根据挖掘结果的不同,频繁项集挖掘可分为 4 类:1) 完全频繁项集挖掘[1-6]. 这类挖掘输出所有满足最小支持数阈值的频繁项集,但对于维数较高的频繁项集挖掘,结果集太大,挖掘结果的价值受到影响. 2) 频繁闭项集挖掘[7,8]. 这类挖掘过于强调支持数,因此结果集仍然很大. 3) 频繁表示项集挖掘[9]. 这类挖掘对挖掘的频繁项集进行二次挖掘,即对频繁项集进行聚类,用表示项集来表示每类频繁项集. 这样可减小结果集的规模,但表示项集的支持数阈值小于频繁项集的支持数阈值,挖掘结果的可信度受到影响. 4) 最大频繁项集挖掘[3,10]. 这类挖掘只输出满足最小支持数阈值的最大频繁项集,最大频繁项集的子集均为频繁项集,因此这类挖掘具有更强的实际意义和应用价值. 本文研究最大频繁项集的挖掘算法.

频繁项集挖掘算法主要有两大类:1) Apriori 类算法[1,2]. 这类算法利用频繁 k-1-项集生成频繁 k-项集,需扫描数据库 k 次,由于扫描的次数较多,算法的代价较大. 2) 基于 FP-tree 的算法[3-5,10]. 这类算法利用压缩 FP-tree 存储结构,只需扫描数据库两次,具有较高的效率.

Grahne 对 FP-tree 结构进行改进,提出了基于数组的 FP-tree,并在此基础上提出最大频繁项集挖掘算法——FPMAX* 算法[3]. 该算法在对数据库进行扫描并建立 FP-tree 的过程中,用数组记录频繁项对出现的次数,不必统计子 FP-tree 中频繁项出现的次数,从而只需扫描每个 FP-tree 一次,大大减少了系统开销. 该方法对于稀疏数据集的效果显著,对于浓密数据集建立数组的开销较大. Han 等针对

收稿日期: 2006-02-04; **修回日期**: 2006-04-25.
基金项目: 国家 973 预研项目(2001CCA00700); 辽宁省教育厅攻关项目(05L090); 大连市青年基金项目(2005J22JH038).
作者简介: 于红(1968—),女,辽宁瓦房店人,教授,博士生,从事数据库技术、决策支持系统等研究; 王秀坤(1945—),女,辽宁辽阳人,教授,博士生导师,从事数据库技术、决策支持系统等研究.

含单前缀路径的 FP-tree,提出了 PFG 挖掘算法[5]. 该算法可在一定程度上降低系统开销.

通过对数据集的分析和实验发现,含单前缀路径的 FP-tree 很少,多数 FP-tree 含多条路径. 有些路径的项集之间存在包含关系,若频度高的项集包含频度低的项集,就不必处理频度低的路径. 因此本文提出完全前缀路径和有序 FP-tree 的概念,给出了建立有序 FP-tree 的算法,并提出了基于有序 FP-tree 的最大频繁项集挖掘算法 ——MFIM 算法. 对算法的性能进行实验研究,实验结果表明,MFIM 算法对于浓密数据集中挖掘长模式效果较好. 该算法具有广阔的应用前景,如汉语认知脑成像数据集是有长模式的浓密数据集,MFIM 算法对这类数据挖掘具有较大的应用价值.

2　问题定义

设 $I = \{i_1, i_2, \cdots, i_m\}$ 是 m 个不同项的集合,事务数据库 $DB = \{T_1, T_2, \cdots, T_n\}(T_i \subseteq I)$,项集 $X \subseteq I$. X 在 DB 中的支持数是指 DB 中满足 $X \subseteq T_i$ 的事务数,记为 $X.\text{support}$; X 在 DB 中的支持度 $X.\text{sup}$ 是指 DB 中满足 $X \subseteq T_i$ 的事务占 DB 中事务总数的百分比,即

$$X.\text{sup} = \frac{X.\text{support}}{n} \times 100\%. \qquad (1)$$

假设给定的事务数据库 DB 和频繁项集的最小支持数阈值为 θ,对于任意给定的项集 X,如果 $X.\text{support} \geq \theta$,则称 X 为 DB 中的频繁项集[4,5]. 对于频繁项集 X,若任意项集 $Y \subseteq X$,均有 $Y.\text{support} < \theta$,则称 X 为 DB 中的最大频繁项集[4,5]. 给定事务数据库 DB 和最小支持数阈值 θ,找到 DB 中所有最大频繁项集称为最大频繁项集挖掘.

为减少对事务数据库的扫描次数,降低挖掘开销,Han 等提出了压缩 FP-tree 这一紧凑的数据结构,可对事务数据库的数据进行压缩[4].

FP-tree 是如下定义的树结构[4,5]:

1) 由一个标号为 NULL 的根、一个项前缀子树集作为根的子女、一个频繁项头表构成.

2) 项前缀子树中的每个节点由 4 个域组成: item-name, count, node-link, node-parent. 其中: item-name 是该节点表示的项,count 是到达该节点的路径中的事务数,node-link 指向 FP-tree 中同名项的下一节点, node-parent 链接到父节点.

3) 频繁项头表中的每个表项由两个域组成: item-name 和 node-link 头. node-link 头指向 FP-tree 中该 item-name 的第 1 个节点.

基于以上定义,FP-tree 按以下步骤创建:

1) 扫描 DB 一次,产生频繁项集 F 及其支持数. 按支持数降序排列 F,生成频繁项列表 FLIST.

2) 创建 FP-tree 的根节点 T,标号为 NULL. 对于 DB 中的每个事务 T_i,作如下处理:

①选择 T_i 中的频繁项并将它们按 FLIST 中的次序排列,设排序后的频繁项列表为 $[p \mid P]$,其中 p 是第 1 项, P 是剩余项列表;

②调用 insert_tree$([p \mid P], T)$;

③若 P 非空,则递归调用 insert_tree$([p \mid P], N)$.

函数 insert_tree$([p \mid P], T)$ 执行如下:如果 T 有子女 N,使得 $N.\text{item-name} = p.\text{item-name}$,则 $N.\text{count}$ 加 1;否则创建一个新节点 N,将其 item-name 初始化成 $p.\text{item-name}$,count 设置为 1,node-parent 链接到 T,node-link 链接到具有相同 item-name 的节点.

3　有序 FP-tree

定义 1　$I = \{i_1, i_2, \cdots, i_m\}$ 是事务数据库 DB 中的 m 个不同项的集合,事务数据库 DB 的最小支持数阈值为 θ,DB 中共有 s 个频繁项,排序后的频繁项列表为 $FLIST = \{a_1, a_2, \cdots, a_s\}$ $(a_i \in I, i = 1, 2, \cdots, s)$. 如果在 FP-tree 中存在长度为 k 的前缀路径 $P = \{a_1, a_2, \cdots, a_k\}(k \leq s)$,包含了 FLIST 中的前 k 项,在路径 P 中 $a_k.\text{count} \geq \theta$,且 a_k 有子女 a_{k+1},则在路径 $P' = \{a_1, a_2, \cdots, a_k, a_{k+1}\}(k+1 \leq s)$ 中 $a_{k+1}.\text{count} < \theta$,称 P 为完全前缀路径.

为了尽早找到完全前缀路径,并用它进行最大频繁项集挖掘算法的优化,应将完全前缀路径中的项放在靠近 FP-tree 头表的位置. 作者对 FP-tree 进行改进,提出了有序 FP-tree,并给出了构建有序 FP-tree 的算法.

定义 2　有序 FP-tree 是以下定义的树结构:

1) 由一个标号为 NULL 的根、一个项前缀子树集合、一个频繁项头表构成.

2) 项前缀子树中的每个节点由 5 个域组成: item-name, count, level, node-link, node-parent. 其中: item-name 表示该节点所表示的项,count 表示包含从根到达该节点的路径中项集的事务数,level 表示该节点是根的子孙的代数,node-link 指向 FP-tree 中同名项的下一节点,FP-tree 中的节点按 level 由高到低的顺序链接起来,node-parent 链接到父节点.

3) 频繁项头表中的每个表项由 3 个域组成: item-name, index 和 node-link 头. node-link 头指向 FP-tree 中该 item-name 的第 1 个节点, index 表示该项在 FLIST 中的位置.

基于以上定义,有序 FP-tree 按如下算法创建:

OFPTreeC 算法(有序 FP-tree 创建)

输入: 事务数据库 DB, 最小支持数阈值 θ;

输出: 有序 FP-tree.

1) $I \leftarrow \phi$, $F \leftarrow \phi$ HEADTABLE $\leftarrow \phi$ // 项集初始化

2) for each T_i in DB // 扫描数据库

3) for each item in T_i

4) if item $\in I$ then item.count ++

else $I = I \cup \{$item$\}$, item.count = 1

5) for each item in I // 选择满足大于 θ 的项

6) if item.count > = θ then

7) $F = F \cup \{$item$\}$

8) FLIST $\leftarrow$ sort on the item.count of F

9) For $item_i$ in FLIST

10) $item_i$.node-link = NULL, $item_i$.index = i,

HEADTABLE = HEADTABLE $\cup$ $\{item_i\}$

11) $T \leftarrow$ NULL // 创建有序 FP-tree 的根节点

12) for each T_i in DB

13) temp $\leftarrow \phi$;

14) for each item in T_i // 选出 T_i 中的频繁项

15) if item in FLIST then

16) temp = temp $\cup \{$item$\}$

17) $[p \mid P] \leftarrow$ sort on temp according to FLIST

18) lv = 1, insert_tree($[p \mid P]$, T, lv);

19) if P is not null then

20) lv = lv + 1, insert_tree($[p \mid P]$, N, lv)

21) Return

算法中的多数行已在算法描述中作了注释, 部分行因为空间太小没有注释, 说明如下: 第 8 行对 F 按支持数降序排列生成有序频繁项列表; 第 9 行根据 FLIST 建立有序 FP-tree 头表函数; 第 17 行中 p 是第 1 项, P 是剩余项列表; 第 20 行 insert_tree($[p \mid P]$, T, lv) 执行如下: 如果 T 有子女 N, 使得 N.item-name = p.item-name, 则 N.count + 1; 否则创建一个新节点 N, 将其 item-name 初始化成 p.item-name, count 设为 1, level = lv, node-parent 链接到 T, 根据 level 调整 node-link 指针, 将该节点插入到具有相同 item-name 的节点链的合适位置.

算法分析如下: 比较 FP-tree 和有序 FP-tree 的创建可以看出, 有序 FP-tree 的空间开销稍大于 FP-tree, 这是由于需要一定的空间存放头表中的 index 和 FP-tree 节点中的 level. 通过对实验数据集的研究发现, 对于具有 n 个事务的数据库, 平均每个事务的频繁项数为 m, 则增加的空间开销为 $O(m \times n)$. 如果创建 FP-tree 时, 将每个新建节点插入到同名节点链的链头, 则创建有序 FP-tree 的时间开销要稍大于创建 FP-tree 的时间开销; 如果创建 FP-tree 时, 将每个新建节点插入到同名节点链的链尾, 则创建有序 FP-tree 的最大时间开销等于创建 FP-tree 的时间开销.

4 MFIM 算法

首先证明有序 FP-tree 的性质和相关定理, 然后给出用有序 FP-tree 挖掘最大频繁项集的算法.

性质 1 在有序 FP-tree 中, 若含长度为 k 的完全前缀路径 $P = \{a_1, a_2, \cdots, a_k\}$, 则该路径一定是以有序 FP-tree 头表中 a_k 的 node-link 为终点, 且节点的 level = k 的路径. 其他任何以 a_k 为终点的路径均为 P 的子路径.

证明 根据定义 1, 长度为 k 的完全前缀路径是所有以 a_k 为终点的路径中最长的路径. 根据 OFPTreeC 算法, 所有以 a_k 为终点的路径可通过头表中 a_k 的 node-link 获得, 其中最长的路径是以头表中 a_k 的 node-link 指向的节点为终点的路径. 若该路径是完全前缀路径, 则该路径一定包含 k 项, 且在 a_k 之前有 $k-1$ 项, 因此 a_k 的 level = k. 其他以 a_k 为终点的路径长度一定小于 k, 且这些路径一定是 P 的子路径. □

定理 1 在有序 FP-tree 中, 若含长度为 k 的完全前缀路径 $P = \{a_1, a_2, \cdots, a_k\}$, 则项集 $\{a_1, a_2, \cdots, a_k\}$ 的任何子集均不能成为最大频繁项集.

证明 根据定义 1, 长度为 k 的完全前缀路径 $P = \{a_1, a_2, \cdots, a_k\}$, a_k.count $\geq \theta$. 根据 FP-tree 的构建过程, 可知 P 中的任何 a_i.count $\geq a_k$.count $(i = 1, 2, \cdots, k-1)$, 因此项集 $\{a_1, a_2, \cdots, a_k\}$ 的支持度 $\geq a_k$.count $\geq \theta$. 根据频繁项集的定义, $\{a_1, a_2, \cdots, a_k\}$ 是频繁项集; 根据最大频繁项集的定义, $\{a_1, a_2, \cdots, a_k\}$ 任何子集一定不是最大频繁项集. □

基于以上定理, 这里给出用有序 FP-tree 挖掘最大频繁项集的算法:

MFIM 算法

输入: 事务数据库 DB, 最小支持数阈值 θ;

输出: 最大频繁项集集合 MF.

1) OFP = OFPTreeC (DB, θ)

2) MF = { }, α = NULL // 数据结构初始化

3) CALL MFP-growth(OFP, α)

其中: 第 1 行调用 OFPTreeC 构建有序 FP-tree; 第 3 行调用 MFP-growth 挖掘最大频繁项集.

Procedure MFP-growth (tree, α)

1) if tree 只含一个单一路径 P then

2) MF = MF $\cup\{P \cup \alpha\}$

3) $\{P \cup \alpha\}$.support = P 中节点的最小支持数

4) else

5) for each a_i in HEADTABLE of tree

6) if a_i.index = a_i.node-link.level and a_i.node-link.count > θ then

7) if NOT ContainSubSet(MF, $\alpha \cup \{a_1, a_2, \cdots, a_i\}$) then

8) MF = MF $\cup\{\{a_1, a_2, \cdots, a_i\} \cup \alpha\}$

9) support = a_i.node-link.count

10) goto (14)

11) else

12) $\alpha = \alpha \cup \{a_i\}$, α.support = a_i.support

13) 构建 α 的频繁模式库 DB_α,令 FI_α = DB_α 中的频繁项集

14) if NOT ContainSubSet (MF, $\alpha \cup FI_\alpha$) then

15) $OFP\text{-}tree_\alpha$ = OFPTreeC (DB_α, θ)

16) if $OFP\text{-}tree_\alpha$ < > NULL then

17) MFP-growth ($OFP\text{-}tree_\alpha$, α)

18) else

19) if NOT ContainSubSet (MF, α) then

20) MF = MF $\cup\{\alpha\}$

21) Return

算法说明如下:在 MFP-growth 中,1) ～ 3) 处理 FP-tree 为单一路径的情况,此时该路径 P 与模式 α 的并集为最大频繁项集,其支持度为路径 P 中节点的最小支持度;4) ～ 20) 处理 FP-tree 有分支的情况,此时对头表中的项按支持度由低到高的顺序,挖掘包含该项的最大频繁项集;5) 按头表中的项从后向前的顺序进行;6) 根据性质 1 和定理 1 判断路径是否为完全前缀路径,如果是,则无需建立子 FP-tree,也不必处理其他路径,否则要建立每项对应的子 FP-tree,并继续处理其他项;7) ～ 10) 判断当前项集是否为某个最大频繁项集的子集,如果不是,则加入最大频繁项集的集合;11) ～ 20) 通过建立子 FP-tree 处理非完全前缀路径的情况;13) 检查 $\alpha \cup FI_\alpha$ 是否为 MF 中最大频繁项集的子集,如果 $\alpha \cup FI_\alpha$ 是 MF 中某个项集的子集,则 $\alpha \cup FI_\alpha$ 及其所有子集均不能成为最大频繁项集,不需要进一步处理,否则要构建 α 的有序频繁模式树,并对该有序频繁模式树进一步挖掘;15) 构建 α 的有序频繁模式树.

5　实验研究和性能分析

为研究算法性能,作者用 C + + 实现了 MFIM 算法和 FPMAX* 算法[3].实验环境为 800 MHz 的 PⅢCPU,256 MB 内存和 40 GB 硬盘,Microsoft Windows 2000 Server 操作系统.为综合评价算法的性能,实验数据集选用具有不同浓密性和模式长度的数据集 Pumsb(2 113 项,49 046 个事务,平均每个事务 74 项,文件大小 16 299 KB),Connect-4(129 项,67 557 个事务,平均每个事务 43 项,文件大小 9 039 KB),Mushroom(119 项,8 124 个事务,

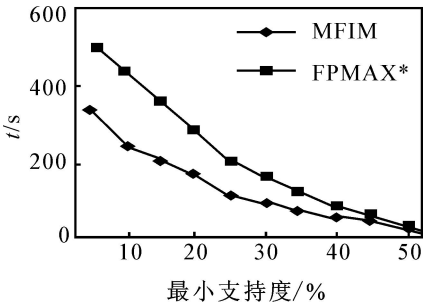

图 1　数据集 Pumsb 的运行结果

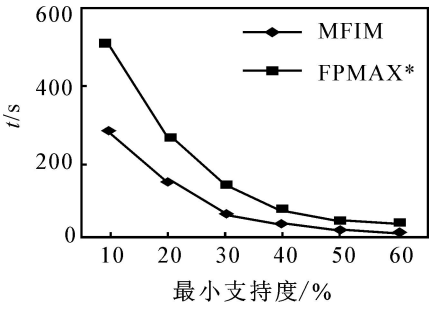

图 2　数据集 Connect-4 的运行结果

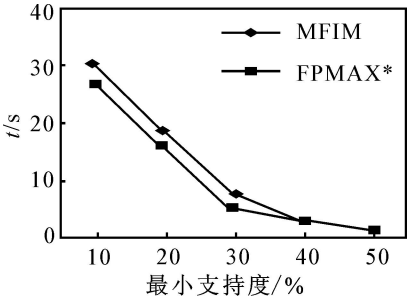

图 3　数据集 Mushroom 的运行结果

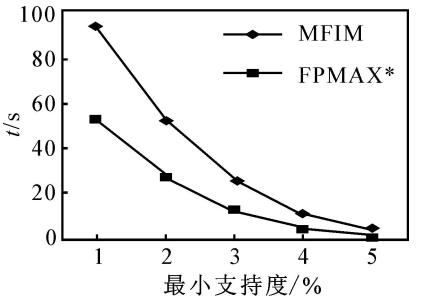

图 4　数据集 T100120D 的运行结果

平均每个事务 23 项,文件大小 558 KB),T100120D (870 项,100 000 个事务,平均每个事务 10 项,文件大小 998 KB).实验结果如图 1～图 4 所示.

从实验结果可以发现,对于数据集 Pumsb 和 Connect-4,MFIM 算法的性能好于 FPMAX*;对于数据集 Mushroom 和 T100120D,MFIM 算法的性能不如 FPMAX*.通过对数据集的特征研究发现,前两个数据集属于浓密数据集,后两个数据集属于稀疏数据集.浓密数据集的 FP-tree 中包含较多的完全前缀路径,本文算法无需对完全前缀路径逐项建立子 FP-tree,减少了冗余操作,因此算法性能有了较大的提高.本文算法对于 Pumsb 数据集的效果要好于对于 Connect-4 数据集的效果. Pumsb 中的模式属于长模式,其中的完全前缀路径的长度较长,算法减少的冗余操作较多,因此性能提高幅度较大.对于稀疏数据集,其中没有或少有完全前缀路径,因此算法优化的幅度不大.因为 FPMAX*利用数组降低了建立 FP-tree 的代价,所以 FPMAX*的性能要好于 MFIM 的性能.通过以上分析可知,本文算法对于浓密数据集的长模式挖掘效果较好.

6 结 语

通过对基于 FP-tree 的最大频繁项集挖掘算法的研究发现,该类算法中有一些不必要的冗余操作.为进一步对该类算法进行优化,本文提出了完全前缀路径和有序 FP-tree 的概念,给出了建立有序 FP-tree 的算法.在此基础上,提出了用有序 FP-tree 进行最大频繁项集挖掘的算法 ——MFIM 算法.该算法利用有序 FP-tree 中的完全前缀路径包含右边子路径这一特点,对挖掘算法进行优化.实验结果表明,该算法对于浓密数据集中长模式的挖掘具有较大的应用价值.下一步工作将结合实际应用,对算法进一步改进和完善.

参考文献(References)

[1] Agrawal R, Imielinski T, Swami R. Mining association rules between sets of items in large databases [C]. SIGMOD '93. Washington, 1993: 207-216.

[2] Agrawal R, Srikant R. Fast algorithms for mining association rules[C]. VLDB '94. Santiago, 1994: 487-499.

[3] Grahne G, Zhu J F. Efficiently using prefix-trees in mining frequent itemsets [C]. FIMI '03. Melbourne, 2003.

[4] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[C]. Proc of ACM SIGMOD '00. Dallas, 2000: 1-12.

[5] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation: A frequent-pattern tree approach [J]. Data Mining and Knowledge Discovery, 2004, 8 (1): 53-87.

[6] He Z, Deng S, Xu X. A FP-tree based approach for mining all strongly correlated pairs[C]. Proc of Conf on Computational Intelligence and Security. Xi'an, 2005: 735-740.

[7] Han J, Wang J, Lu Y, et al. Mining top-*K* frequent closed patterns without minimum support [C]. ICDM '02. Washington, 2003: 211-218.

[8] Zaki M J, Hsiao C. CHARM: An efficient algorithm for closed itemset mining [C]. SDM '02. Arlington, 2002: 457-473.

[9] Xin D, Han J W, Yan X F, et al. Mining compressed frequent-pattern sets[C]. Proc of the 31st VLDB Conf. Trondheim, 2005: 709-720.

[10] Grahne G, Zhu J. High performance mining of maximal frequent itemsets[C]. Proc of the 6th SIAM Int'l Workshop on High Performance Data Mining. San Francisco, 2003: 135-143.

(上接第 519 页)

[6] Xu Z S, Da Q L. An overview of operators for aggregating information [J]. Int J of Intelligent Systems, 2003, 18 (9): 953-969.

[7] Yager R R. Families of OWA operators[J]. Fuzzy Sets and Systems, 1993, 59(1): 125-148.

[8] Herrera F, Herrera Viedma E, Verdegay J L. Direct approach processes in group decision making using linguistic OWA operators[J]. Fuzzy Sets and Systems, 1996, 79(2): 175-190.

[9] Yager R R. Induced ordered weighted averaging operators [J]. IEEE Trans on Systems, Man and Cybernetics: Part B, 1999, 29(2): 141-150.

[10] Xu Z S, Da Q L. The uncertain OWA operator[J]. Int J of Intelligent Systems, 2002, 17(6): 569-575.

[11] Keeney R L, Raiffa H. Decisions with multiple objectives: Preferences and value trade-offs[M]. New York: Wiley, 1976.

[12] Dubois D, Prade H. A review of fuzzy sets aggregation connectives[J]. Information Science, 1985, 36(1): 85-121.