面向凝聚式层次聚类算法实现的矩阵存储数据结构研究*)

张振亚[1,2] 程红梅[3] 王 进[2] 王煦法[2]

(中国科学技术大学电子工程与信息科学系 合肥 230027)[1] (中国科学技术大学计算机系 合肥 230027)[2]

(安徽师范大学数学系 芜湖 241000)[3]

摘 要 快速查找、扩张、收缩是凝聚式层次聚类算法快速实现对相似度/距离矩阵存储的基本要求。本文提出了基于十字链表和平衡二叉树的复合数据结构CrossAVL用于矩阵的存储,给出了查找、扩张,收缩操作的实现并对其时间复杂度进行了分析。实验结果表明,CrossAVL对快速要求能够较好地满足。

关键词 凝聚式层次聚类,矩阵,十字链表,平衡二叉树

An Approach on the Data Structure for the Matrix Storing Based on the Implementation of Agglomerative Hierarchical Clustering Algorithm

ZHANG Zhen-Ya[1,2] CHENG Hong-Mei[3] WANG Jin[2] WANG Xu-Fa[2]

(EEIS Department of University of Science and Technology of China, Hefei 230027)[1]

(CS Department of University of Science and Technology of China, Hefei 230027)[2]

(Mathematics Department of Anhui Normal University, Wuhu 241000)[3]

Abstract Searching, expanding, shrinking instantly is the precondition of the similarity/distance matrix storing for the implementation of agglomerative hierarchical clustering algorithm. This paper presents a new compound data structure named as CrossAVL based on cross list and AVL tree for the matrix storing. The time complexity for the implementation of searching, expanding and shrinking method based on CrossAVL are given. Experimental results show that all those method can be running instantly.

Keywords Agglomerative hierarchical clustering, Matrix, Cross list, AVL tree

1 简介

当前,互连网络已经成为信息发布的重要载体,如何从互连网上海量的信息中有效地获取信息也为研究者所关注[1,2]。对互连网上海量信息的处理,数据挖掘技术成为强有力的工具。利用数据挖掘技术,可以有效地对海量信息进行分类、聚类、关联分析等处理[1,3~5]。

凝聚式层次聚类算法[5]是对数据进行聚类分析经常使用的技术。当前类别的相似度/距离矩阵是凝聚式层次聚类算法的一个重要数据。相似度矩阵的有效组织,是凝聚式层次聚类算法高效实现的关键[6]。为使凝聚式层次聚类算法高效实现,相似度矩阵的组织必须满足:

a) 查找:快速查找矩阵中最大/最小分量;

b) 扩张:快速使矩阵增加一行、一列;

c) 收缩:快速删除矩阵的一行、一列;

所谓快速,是指:对$n\times n$阶矩阵,操作的时间复杂度小于$O(n^2)$。

2维数组、行(列)链表、十字链表是三种常用的矩阵存储数据结构[7]。对一个$n\times n$阶矩阵:若矩阵作为一个2维数组存储,则:a)查找矩阵中最大/最小分量的时间复杂度为$O(n^2)$;b)收缩或扩张操作需要重新构造、填充存储矩阵的2维数组,时间复杂度为$O(n^2)$。

若矩阵作为行链表存储,则:a)查找矩阵中最大/最小分量的时间复杂度为$O(n^2)$;b)扩张时,增加一行的时间复杂度为$O(n)$,增加一列的时间复杂度为$O(n^2)$,因此,扩张操作的时间复杂度为$O(n^2)$。c)收缩时,删除指定一行的时间复杂度为$O(n)$,删除指定一列的时间复杂度为$O(n^2)$,故收缩操作的时间复杂度为$O(n^2)$。

若矩阵作为十字链表存储,则:a)查找矩阵中最大/最小分量的时间复杂度为$O(n^2)$;b)扩张时,增加一行的时间复杂度为$O(n)$,增加一列的时间复杂度为$O(n)$,因此,扩张操作的时间复杂度为$O(n)$;c)收缩时,删除指定一行、列的时间复杂度均为$O(n)$,故收缩操作的时间复杂度为$O(n)$。

显然,在使用以上三种常见的数据结构存储矩阵时,三种操作的快速要求无法同时满足。本文提出了以十字链表和平衡二叉树为基础的复合数据结构CrossAVL,满足了三种操作的快速要求。第2节给出了CrossAVL的定义以及三种操作的实现与时间复杂度分析。实验结果在第3节给出,最后进行了总结。

2 CrossAVL及其性能分析

2.1 CrossAVL的定义

对$n\times n$阶矩阵,若以十字链表存储,矩阵的收缩、扩张操作的时间复杂度均为$O(n)$。若在以十字链表存储矩阵的同

*)基金项目:中国博士后基金资助金(2004036463)。张振亚 博士后,主要研究领域为信息检索、数据挖掘,机会/征兆发现;程红梅 讲师,主要研究领域为计算经济学;王 进 博士生,主要研究领域为信息检索、数据挖掘与半结构化数据;王煦法 教授,博导,主要研究领域为人工智能、进化计算。

时，链表中的各接点以某种合适的树结构组织，则查找最大分量操作的时间复杂度可降为 O(log(n))。平衡二叉树是一种较好的树结构[7]：1)由于树深度的复杂度为 O(log(n))，故查找最大分量操作的时间复杂度为 O(log(n))；2)插入一个新节点的时间复杂度为 O(log(n))；3)删除一个节点的时间复杂度为 O(log(n))。表 1 给出了使用十字链表和平衡二叉树存储矩阵时所定义的数据结构。称这种复合的数据结构为十字平衡二叉树 (CrossAVL，CrossAVL Tree)。

表 1 中，结构 MatrixElement 用来存储矩阵的一个分量以及相关的树、链表维护所必需的指针；结构 RowColTag 用以存储行、列链表的头指针；CAVLMatrix 对基于 MatrixElement 和 RowColTag 的矩阵的存储以及查找、删除、增加操作进行了定义。其中，按照“子树根节点表示的元素比其左子树上任何节点表示的元素都大；子树根节点表示的元素比其右子树上任何节点表示的元素都小”的原则使用 AVL 树组织矩阵的元素。算法 1 给出了将矩阵的一个分量插入 CrossAVL 操作的形式描述。本文中，矩阵每个分量是一个双精度数。

表 1 CAVL 的定义

算法 1 AddNewElement//将矩阵的一个分量插入 CrossAVL
输入：分量所在行 rowCode、所在列 colCode 以及分量的取值 value
输出：插入的 AVL 树节点的指针

```
(1) node = NewTreeNode(rowCode, colCode, value);//构造新的
    CrossAVL 节点
(2) if(node! =NULL){
(3)             AddNewElementToCrossList(node); //新节点插入
                  Cross List
(4)             AddNewElementToAVLTree(node); //新节点插入
                  AVL 树中
    }
(5)return node;
struct MatrixElement
{ double value;//距离矩阵的一个元素的值
   MatrixElement * LChild;
   MatrixElement * RChild;
   //AVL 树中该节点的左、右孩子节点
   MatrixElement * Parent;
   //AVL 树中该节点的父节点
   int balanceFactor;//平衡因子
   struct RowColTag * rowPointer;
   struct RowColTag * colPointer;
   //指示所在行、列的标记的指针
     MatrixElement * RowBefore;
   /*在行链表中，该元素直接前驱的元素，第一个元素的取值为
     NULL*/
   MatrixElement * RowNext;
   /*在行链表中，该元素直接后继的元素，最后一个元素的取值为
     NULL*/
   MatrixElement * ColBefore;
   /*在列链表中，该元素直接前驱的元素，第一个元素的取值为
     NULL*/
   MatrixElement * ColNext;
   /*在列链表中，该元素直接后继的元素，最后一个元素的取值为
     NULL*/
   };
   //矩阵行列信息存储结构的定义
   struct RowColTag
   { int id;
     MatrixElement * FirstElement;
     RowColTag * next;
   };
   //十字平衡二叉树
   class CAVLMatrix
   { int RowSize; //矩阵的行数
     int ColSize; //矩阵的列数
     RowColTag *Row;//行信息存储
     RowColTag *Col;//列信息存储
     RowColTag * *RowBuffer;
     //按行号顺序存储的行信息指针
     RowColTag * *ColBuffer;
     //按列号顺序存储的列信息指针
     MatrixElement * root;//存储矩阵的每个分量
   public:
     MatrixElement* FindMax(…);
     MatrixElement* AddNewElement(…);
     void DelRowCol(…);
     void AddNewCol(…);
     ……
};
```

命题 1 若 CrossAVL 已经有 n 个节点，使用算法 1 出入一个新的矩阵分量的时间复杂度为 O(log(n))。

证明：使用算法 1 插入新的矩阵分量时：

∵(2)、(5)步骤所使用的时间是常数

∴(2)、(5)步骤的时间复杂度为 O(1)

∵(1)步骤仅仅是 MatrixElement 结构的构造，结构的每个分量需要且仅需要构造一次

∴(1)步骤的时间复杂度为 O(1)

∵(3)步骤是将新节点作为第一个节点插入十字链表，只需要指针的调整

∴(3)步骤的时间复杂度为 O(1)

∵(4)步骤是将新节点插入 AVL 树

∴n 节点 AVL 树的高度是 O(log(n))

∵4)步骤的时间复杂度是 O(log(n))

∴用算法 1 出入一个新的矩阵分量的时间复杂度为 O(1)+O(1)+O(1)+ O(log(n))+O(1)

∴用算法 1 出入一个新的矩阵分量的时间复杂度为 O(log(n))

推论 1 若 CrossAVL 已经有 n^2 个节点，使用算法 1 出入一个新的矩阵分量的时间复杂度为 O(log(n))。

证明：由命题 1 知此时用算法 1 出入一个新的矩阵分量的时间复杂度为 $O(\log(n^2))$。而 $O(\log(n^2))= O(\log(n))$，故推论显然成立。

2.2 最大元素查找操作的实现及其性能分析

在 CrossAVL 中，由于矩阵各分量组成的 AVL 树中，“子树根节点表示的元素比其左子树上任何节点表示的元素都大；子树根节点表示的元素比其右子树上任何节点表示的元素都小”，故矩阵的最大分量只能出现在当前 AVL 树的右子树或根节点中(此时，根节点的右子树为空)。具体的查找过程由算法 2 给出。

算法 2 FindMaxValue //矩阵最大元素查找

输入：空
输出：最大元素所在 AVL 树节点的指针

```
(1)p=root;
(2)if(p! =NULL){
(3)             while(p->RChild! =NULL)
(4)               p=p->RChild;
                }
(5)return p;
```

命题 2 若 $n\times n$ 阶矩阵使用 CrossAVL 存储，则使用算法 2 查找矩阵最大分量的时间复杂度为 O(log(n))。

证明：∵$n\times n$ 阶矩阵有 n^2 个分量

∴CrossAVL 中，以全部矩阵分量构成的 AVL 树的高度为 $O(\log(n2)=O(\log(n))$

∵算法 2 的(3)、(4)步骤至多执行 O(log(n))次

∵算法 2 的其它步骤的执行仅需要常数的时间

∴使用算法 2 查找矩阵最大分量的时间复杂度为 O(log(n))。

2.3 收缩操作的实现及其性能分析

在删除以 CrossAVL 形式存储的矩阵的一行、一列时，需要进行两方面的工作：一方面，由于被删除的矩阵的分量的有效存储中含有十字链表的信息，因此需要维护十字链表的完整性；另一方面，由于被删除的矩阵的分量的有效存储中含有 AVL 树节点的信息，因此需要维护 AVL 树的完整性。具体的在 CrossAVL 上删除指定行列操作的形式描述如算法 3。

算法 3 DelRowCol//删除指定的矩阵的某行列

```
输入:被删除的行的行号 rowCode,被删除的列的列号 colCode
输出:删除指定的矩阵的某行列后的矩阵
(1)p=RowBuffer[rowCode-1];u=p->FirstElement; //获取被删除行的第一个被删除分量的指针
(2)while(u! =NULL)
(3){DelNode(u); //删除 AVL 树的一个节点
(4)   u=p->FirstElement; //下一个被删除节点
   }
(5)p=ColBuffer[colCode-1];u=p->FirstElement; //获取被删除列的第一个被删除分量的指针
(6)while(u! =NULL)
(7){ DelNode(u); //删除 AVL 树的一个节点
(8)   u=p->FirstElement; //下一个被删除节点
   }
(9)RowSize--; ColSize--; //矩阵的一行、一列被删除后,矩阵的行、列数都减 1
(10)DelSequenceRowColInformation();//销毁十字链表的顺序存储缓冲
(11)GenSequenceRowColInformation();//重新构造十字链表的顺序存储缓冲
```

算法 3 需要特别说明的是步骤(1)、(5)、(10)和(11)。为了使步骤 1、5 能够根据被删除行、列的行、列号快速定位十字链表中相应行、列的头指针,CrossAVL 定义了数组 RowBuffer 和 ColBuffer。RowBuffer[i]取值为第 i 行行链表头指针,ColBuffer[j]取值为第 j 列列链表头指针。步骤 10 用以在指定的行列删除后,RowBuffer、ColBuffer 的销毁,步骤 11 用以新的 RowBuffer、ColBuffer 的构造。对于 $n\times m$ 阶矩阵,由于 RowBuffer 是长度为 n 的数组,ColBuffer 是长度为 m 的数组,组各分量的取值只需要分别顺序扫描 CrossAVL 中十字链表的行链表和列链表一次即可获得。由于行链表中有 n 个节点,列链表中有 m 个节点,故步骤 10、11 的时间复杂度为 $O(m+n)$。

命题 3 若 $n\times n$ 阶矩阵使用 CrossAVL 存储,则使用算法 3 删除矩阵一行、一列操作的时间复杂度为 $O(n\times \log(n))$。

证明:∵$n\times n$ 阶矩阵使用 CrossAVL 存储

∴ 算法 3 中除步骤 1～4、6～8、10、11 以外的各步骤的时间复杂度均为 O(1)

∴ 算法 3 中除步骤 1～4、6～8、10、11 以外的各步骤的时间复杂度之和为 O(1)

∵ $n\times n$ 阶矩阵使用 CrossAVL 存储

∴ 步骤 3 为从 AVL 树中的删除一个节点,n^2 个节点的 AVL 数的高度为 $O(\log(n))$

∴步骤 3 的时间复杂度为 $O(\log(n))$

∵ 步骤 2、步骤 4 的时间复杂度为 O(1)

∵ 步骤 2～4 重复执行的次数为行链表中有效节点的数 n

∴ 步骤 2～4 的时间复杂度为 $O(n\times(O(\log(n)+O(1)+O(1)))=O(n\times\log(n))$

∴ 同理,步骤 6～8 的时间复杂度为 $O(n\times\log(n))$

∵步骤 10、11 的时间复杂度均为 $O(n)$

∴ 算法 3 的时间复杂度为 $O(1)+O(n\times\log(n))+O(n\times\log(n))+O(n)+O(n)$

∴算法 3 的时间复杂度为 $O(n\times\log(n))$

2.4 扩张操作的实现及其性能分析

在 CrossAVL 中矩阵的扩张操作可以快速实现:十字链表中增加一个节点可以在常数时间内完成,而在一个高度为 $O(\log(n))$的 AVL 树中插入一个节点的时间复杂度为 $O(\log(n))$。而 $m\times n$ 阶矩阵的扩张,只需要插入 $n+m+1$ 个节点而已。具体的描述如算法 4。

算法 4 AddNewRowCol//矩阵的扩张操作

```
/*设当前矩阵的阶 n×m,本算法将其扩张为(n+1)×(m+1)阶矩阵:首先增加行号为 n+1、列号为 m+1 的空行、空列,然后把相应的分量加入以 CrossAVL 形式的存储的矩阵中*/
输入:newRowBuffer :有 m+1 个分量的数组,与扩张后的矩阵的新行对应
     newColBuffer : 有 n+1 个分量的数组,与扩张后的矩阵的新列对应
输出:扩张完成后的矩阵
(1) n=RowSize;m=ColSize; //当前矩阵的行列数
(2) AddNewBlankRowCol();//当前矩阵的行列数都增 1,空行、列被作为 n+1 行、m+1 列加入矩阵
(3) DelSequenceRowColInformation();//销毁十字链表的顺序存储缓冲
(4) GenSequenceRowColInformation();//重新构造十字链表的顺序存储缓冲
(5) For i=1 to m
(6)     AddNewElement(n+1,I,newRowBuffer[i]);//插入新的分量
(7) For i=1 to n
(8)     AddNewElement(i,m+1,newColBuffer[i]); //插入新的分量
(9) AddNewElement(n+1,m+1,newRowBuffer[m+1]) //插入新的分量
```

命题 4 若 $n\times n$ 阶矩阵使用 CrossAVL 存储,则使用算法 4 插入矩阵一行、一列操作的时间复杂度为 $O(n\times\log(n))$。

证明:∵$n\times n$ 阶矩阵使用 CrossAVL 存储

又∵步骤(2)使矩阵的行列数都增 1,十字链表添加第 $n+1$ 行、列的头指针且其指示的链表为空

∴步骤 2 的时间复杂度为 $O(n)$

∵步骤 3 为销毁长度为 n 的一维数组

∴步骤 3 的时间复杂度为 $O(n)$

∵步骤 4 是扫描十字链表的头指针建立长度为 $n+1$ 的一维数组

∴步骤 4 的时间复杂度为 $O(n)$

∵由命题 1 推论 1 知步骤 6、8 的时间复杂度均为 $O(\log(n))$

∴步骤 5、6 以及步骤 7、8 的时间复杂度都是 $O(n\times\log(n))$

∵步骤 9 的时间复杂度为 $O(\log(n))$

∵步骤 1 的时间复杂度为 O(1)

∴算法 4 的时间复杂度为 $O(1)+O(n)+O(n)+O(n)+O(n\times\log(n))+O(n\times\log(n))+O(\log(n))$

∴算法 4 的时间复杂度为 $O(n\times\log(n))$

3 实验结果

本研究对以 CrossAVL 形式存储的矩阵的查找最大分量、收缩、扩张操作的性能进行了测试。实验使用文本挖掘的凝聚式层次聚类分析所使用的相似度矩阵。这种矩阵的特点为:(1)方阵:矩阵的行、列数相同;(2)正对角线分量取值均为 1:正对角线分量的取值为各个文本对象自身的相似度。正对角线分量不参加聚类分析,可以不存储;(3)实对称阵。对称性的存在使得可以只存储矩阵的上三角或下三角部分。

实验中,随机生成矩阵的上三角部分,每个分量的取值为 0～1 间的实数。矩阵的最大行列数为 4000。矩阵的行列数从 1 到最大行列数变化。为取得三种操作所使用的时间,每个实验重复 1000 次。图 1、2、3 给出了三种操作使用时间的图示。图 1、2、3 中,各坐标图的横坐标表示矩阵的行数,纵坐标表示相应操作使用的时间(单位:秒)。显然,这三种操作,在基于 CrossAVL 表示的矩阵中都可以快速完成。

实验中使用机器的主要硬件配置为 Intel P4 1.7G CPU 以及 DDR 266 内存 1G。

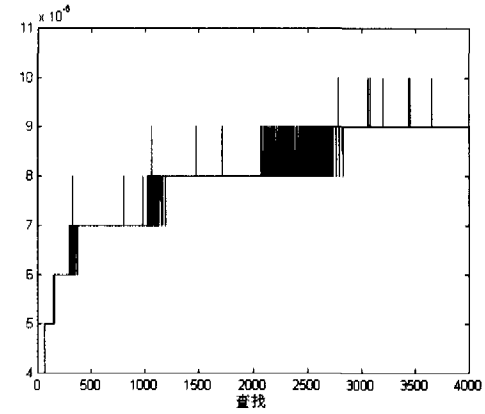

图 1　最大分量查找操作耗时

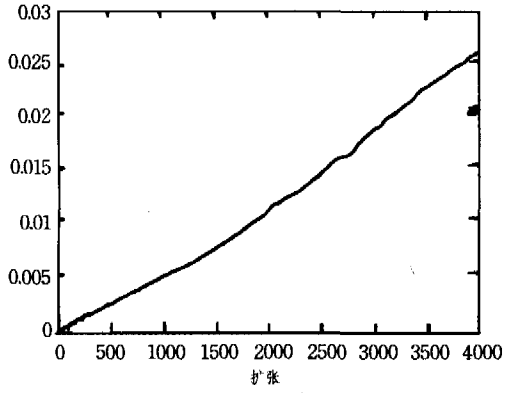

图 2　扩张操作耗时

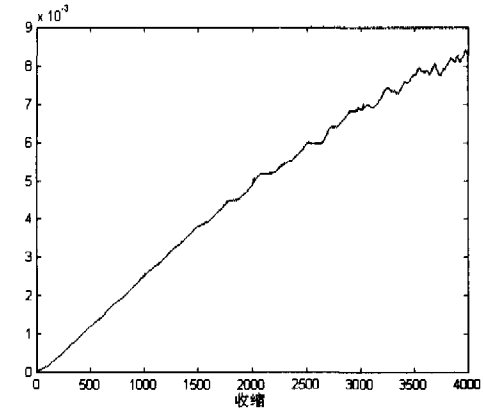

图 3　收缩操作耗时

结论与研究展望　凝聚式聚类是数据挖掘与模式识别中一类典型的聚类算法。数据对象间的距离/相似度矩阵的有效组织是算法性能提升的关键。以 CrossAVL 形式存储的矩阵为基础，可以快速完成矩阵收缩、扩张以及最大元素的查找。对规模为 $n\times n$ 的矩阵，这三种操作的时间复杂度都是 $O(n\times\log(n))$。三种操作的快速实现是凝聚式聚类算法快速实现的基础。

若矩阵采用 CrossAVL 形式组织，矩阵最小元素的查找与最大元素的查找类似，时间复杂度一致。

矩阵的有效存储是正在进行的机会/征兆发现研究中相关算法快速实现的基础。CrossAVL 可以作为基本的数据结构之一。事实上，在数据挖掘技术研究中，如层次式聚类分析技术，CrossAVL 可以用来作为矩阵的有效存储而广泛应用。

致谢　本文受"面向 21 世纪教育振兴行动计划"部分资助。

参 考 文 献

1　Fayyad U, Uthurusamy U. Data mining and knowledge discovery in databases [J]. Commun. ACM, 1996,39:24～27

2　Mitra S, Pal S K, Mitra P. Data Mining in Soft Computing Framework: A Survey [J]. IEEE Trnas. on Neural Networks, 13(1):3～14

3　Ohsawa Y, Nara Y. Decision process modeling across internet and real world by double helical model for chance discovery [J]. New Generation Computing, 2003, 21(2):109～121

4　Shu B, Kak S. A neural network-based intelligent metasearch engine [J]. Information Sciences, 1999,120(1):1～11

5　ChiaHui, Chang. Enabling Concept-Based Relevance Feedback for Information Retrieval on the WWW [J]. IEEE Trans. on Knowledge and Data Engineering, 1999,11(4):595～609

6　Arwar. Item-Based collaborative filtering recommendation algorithms [A]. In: Proc. of the 10th Intl. World Wide Web Conf. (WWW10) [C], 2001. 285～295

7　Jain A, Dubes R C. Algorithms for Clustering Data [M]. Prentice Hall, 1988

8　Karypis G, Han E, Kumar V. Chameleon: A hierarchical clustering algorithm using dynamic modeling [J]. IEEE Computer, 1999, 32(8):68～75

9　Larsen K S. AVL trees with relaxed balance [J]. Journal of Computer and System Sciences, 2000,61(3):508～522

10　Mehlhorn K, Tsakalidis A. An amortized analysis of insertions into AVL trees[J]. SIAM Journal on Computing, 1986,15(1): 22～33

(上接第 13 页)

3　http://www.napster.com

4　Stoica I, Morris R, Karger D, et al. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In: Proc. of ACM SIGCOMM 2001

5　Stoica I, Morris R, David K D, et al. Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications. To Appear in IEEE/ACM Transactions on Networking. http://www.pdos.lcs.mit.edu/chord/papers/paper-ton.pdf

6　Ratnasamy S, Francis P, Handley M. A Scalable Content-Addressable Network, In: Proc. of ACM SIGCOMM, 2001

7　Rowstron A, Druschel P. Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-Peer Systems. In: IFIP/ACM Intl. Conf. on Distributed Systems Platforms (Middleware), Heidelberg, Germany, 2001. 329～350

8　Zhao B Y, Kubiatowicz J D, Joseph A D. Tapestry: An infrastructure for fault-tolerant wide-area location and routing:[Tech. Rep]. CSD-01-1141, Berkeley, 2001

9　Gruber T R. A translation approach to portable ontologies. Knowledge Acquisition, 1993 , 5(2):199～220

10　Wache H, Vogele T, Visser U. Ontology-Based Integration of Information -A Survey of Existing Approaches. In: Proc. of the IJCAI-01 Workshop on Ontologies and Information Sharing, Seattle, USA, Aug. 2001. 108～118

11　Sperber D, Wilson D. Relevance: Communication and Cognition. Oxford: Basil Blackwell, 1986

12　Xu L, Embley D W. Combining the Best of Global-as-View and Local-as-View for Data Integration. Department of Computer Science, Brigham Young University, Provo, Utah 84602, U. S. A

13　Iamnitchi A, Ripeanu M, Foster I. Small-World File-Sharing Communities. IEEE INFOCOM, 2004