CuMen:基于最大频繁序列模式的聚类算法及其在基因拼接中的应用*)

黄 东 唐 俊 汪 卫 施伯乐
(复旦大学计算机与信息技术系 上海 200433)

摘 要 基因组序列拼接的主流方法是将整条序列随机打断成小片段,然后根据片段间重叠关系连接成长序列。由于较多噪音存在,算法复杂度高,加之生物数据的海量增长,序列拼接处理导致巨大的时空开销而无法完成。本文提出一种基于最大频繁序列模式的聚类算法,将整个数据集分成若干个子集,分别高效地处理,实现了一个基因拼接网格系统、透明动态的资源管理,大大扩展了基因拼接计算能力。基于最大频繁序列模式聚类算法及挖掘算法,针对生物数据的特性做出了优化。

关键词 最大频繁序列模式,序列聚类,序列拼接,网格

CuMen: Clustering Sequences Based on Maximal Frequent Sequential Pattern and its Application in Genome Sequence Assembly

HUANG Dong TANG Jun WANG Wei SHI Bai-Le
(Department of Compater Science and Engineering, Fudan University, Shanghai 200433)

Abstract Sequencing genomes is a fundamental aspect of biological research. A variety of assembly programs have been previously proposed and implemented. Because of great computational complexity and increasingly large size, they incur great time and space overhead. In realistic applications, sequencing process might come to become unacceptably slow for insufficient memory even with a mainframe with huge RAM. This paper offeres a clustering algorithm based on maximal frequent sequential patterns, aiming at divide the whole dataset into several parts which can be processed independently and efficiently in limited memory. Some techniques are applied to optimize the mining and clustering procedure. This approach is introduced into grid environment, exploiting parallelism and distribution for improving scalability further.

Keywords Maximal frequent sequential pattern, Sequence clustering, Sequence assembly, Grid

1 引言

在生物信息学领域,大规模测序是基因组研究的最基本任务。当前最流行的大规模测序方法是鸟枪法测序,这种策略首先将一条完整的目标序列随机打断成小片段,分别测序,然后根据片断间区域的重叠关系,将它们拼接成一条一致的长序列。

实际的生物序列通常包含相当量的噪声,测序错误、由于突变产生的替换和缺失情况随处可见。基因组中还有一定量的重复序列,这些序列分散在各处多次出现,或完全一致,或者具有极高的相似度,使拼接处理过程复杂度以指数级增长,甚至会导致一些拼接错误。迄今为止,已经有多种拼接算法被提出并实现,例如 PHRAP[1]。这些方法的计算复杂性较高,导致巨大的时空开销,如 PHRAP 运行时的内存需求通常是原始数据大小的十几倍到几十倍,硬件上的升级已经无法跟上基因组数据的海量增长。

聚类分析在数据挖掘领域被广泛研究,它将对象的集合分组成由类似对象构成的多个类。通过对序列进行聚类,我们可以将基因序列数据集分成若干个分组,每个分组由能够互相拼接到一起的序列组成,这些分组可以单独地得到处理。这种方式有效地降低了数据规模及复杂性,并且提供了并行计算的可能。

RePS[3]系统使用 BLAST[2]同源性搜索工具,进行序列的两两比对,得到序列间的相似度,进行聚类,每个聚类中的序列较有可能互相拼接在一起,代表某一相对独立的基因组序列区域,而与其他聚类中的序列相关性较小。这些分组可以独立地进行拼接处理,在多处理器环境下得到执行。针对重复序列的情况,RePS 扫描一遍数据库,找出所有重复出现、长度为 20 的子序列,并将出现区域遮盖起来(例如标记为拼接软件忽略的字符 N),以降低重复序列对拼接处理的影响。

Phusion[5]根据序列间共同出现的长度为 k 的字符串(模式)进行聚类,使用数学定义的重复(MDR)去除重复序列的干扰。它的做法通过扫描数据库,统计所有长度为 k 的字串的出现频率,出现大于重复度阈值 D 的字符串被认为来源于重复序列并在聚类阶段将其忽略。再次扫描数据库,记录所有满足重复度阈值 D 的模式出现的序列,生成模式-序列列表的倒排结构。之后,根据此结构计算序列相关性矩阵,将拥有超过 M 个共有模式的两个序列归入一个聚类,并递归地计算,直到所有的序列都被归入某个聚类。

Phusion 系统需要保存所有长度为 k 字符串的出现频率,而模式的长度 k 需随着数聚集的增大而增大,在数据量大的情况下,将导致巨大的内存开销。产生的模式-序列列表的倒

*)本课题得到教委高校网格项目 200309 和上海科委重大项目 03dz15027 资助。黄 东 硕士研究生,主要研究方向为数据挖掘、生物信息学;唐 俊 硕士研究生,主要研究方向为数据挖掘、网络、数据库与知识库;汪 卫 教授,主要研究方向为数据挖掘、安全数据库、XML;施伯乐 首席教授,博导,主要研究方向为数据库与知识库。

排结构通常非常庞大，后面的聚类过程，面临更大的资源需求挑战。同时，在某种生物基因的重复序列的种类、数量、分布特性不甚明确之前，重复度阈值 D 的设定合理与否将对聚类的效果产生较大的影响。

GiSA[4]系统采用了和 RePS 类似的聚类策略，将生物基因拼接引入网格框架，使分布的计算资源动态透明地引入，大大地提高了生物基因拼接处理能力。由于 BLAST 工具旨在搜索生物意义上同源的相关序列，每次得到的聚类规模较小，聚类之间存在一定量的重叠，在分组数据集的时候需要很多次的迭代。

针对以上情况，我们提出了一种基于最大频繁序列模式的聚类算法：CuMen，它大大减少了序列模式的数量，拥有更高的准确性。挖掘和聚类过程中所用的参数易于被用户理解，并且形成的聚类对参数不是很敏感。

本文的贡献：①提出了一种基于最大频繁序列模式的聚类算法，并且针对模式数量庞大、内存开销大的情况作了一些实现上的改进。②一种最大频繁序列模式的挖掘算法，并针对生物数据的特性给出了一些优化方法。③实现了一个基因序列拼接网格系统。④最大频繁序列挖掘的结果，可以作为生物基因的重复序列的核心部分，为进一步挖掘模糊的重复序列提供了条件。

文章的结构是：第 2 节介绍预备知识；第 3 节介绍最大频繁序列模式挖掘算法；第 4 节介绍基于最大模式的聚类算法；第 5 节给出基因网格拼接系统的整体框架；第 6 节给出实验评估；最后进行总结。

2 预备知识和问题描述

定义 1 字符集 F

序列中所有可能出现的符号集合，例如生物 DNA 序列的字符集就是 $\{A,C,T,G\}$，蛋白质序列的字符集则包含 20 种氨基酸的符号。

定义 2 序列和序列数据库

$s=a_1a_2\cdots a_{n-1}a_n$ 表示一个序列，其中 $a_i\in F(1\leqslant i\leqslant n)$。

$D=\{s|s=a_1a_2\cdots a_{n-1}a_n,\ a_i\in F(1\leqslant i\leqslant n)\}$ 表示序列数据库。

定义 3 支持度 supp(P)和频繁序列模式

$a_{k1}a_{k2}\cdots a_{km}$ 表示一个模式序列 P，其中 $a_{ki}\in F(1\leqslant i\leqslant m)$。如果序列 $S=a_1a_2\cdots a_{n-1}a_n$，存在 t，使得 $a_{t+1}=ak1$，$a_{t+2}=a_{k2}$，…，$a_{t+m}=a_{km}$，则称序列 S 满足序列模式 P。supp(P)表示模式 P 在数据库中的出现次数。

如果 supp(P)＞minSup，minSup 是用户指定的阈值，则称 P 是频繁的。

定义 4 闭合频繁序列模式和最大频繁序列模式

如果模式 P 是频繁的，而且不存在 P 的某一超模式 P'，P 和 P' 拥有相同的支持度，则称 P 是闭合频繁序列模式。

如果模式 P 是频繁的，而且不存在 P 的某一超模式 P'，P' 也是频繁的，则称 P 是最大频繁序列模式。

举个例子来说明上面的定义。表 1 所示的是包含 4 条序列的数据库，假定最小支持度 minSup 为 2。考察以下 3 个序列模式：$p1=ct$，$p2=act$，$p3=aact$，supp($p1$)＝3，因此是频繁模式，但不是闭合频繁模式，因为存在 supp($p2$)＝3。$p2$ 是 $p1$ 的超模式，$P2$ 是闭合模式，但不是最大频繁模式，因为存在 supp($p3$)＝2，是频繁的，并且 $p3$ 是 $p2$ 的超模式。$P3$ 是最大频繁序列模式。

表 1 示例数据库

编号	序列
1	aactagaa
2	aactattc
3	tcactaac
4	ccgtaacg

3 最大频繁模式挖掘算法 MaxMiner

3.1 最大频繁序列模式

如果某一频繁序列模式 p，无法前向或后向做长度为 1 的模式扩展，则该序列模式是最大频繁序列模式。也就是说，对于模式 $p=a_{k1}a_{k2}\cdots a_{km}$，不存在一个模式 $p'=b\,a_{k1}a_{k2}\cdots a_{km}$，或者 $p'=a_{k1}a_{k2}\cdots a_{km},b,b,a_{ki}\in A(1\leqslant i\leqslant m)$，$p'$ 也是频繁模式。

3.2 为什么要挖掘最大频繁序列模式

至今为止，已提出多个频繁序列模式的挖掘算法，而最大频繁序列模式的挖掘仍是一个有相当挑战性的问题。之所以挖掘最大频繁序列模式，原因在于最大频繁序列模式也是频繁模式，是频繁模式集的一个边界，模式数量大大减少的同时又不丢失太多的信息。由于生物基因数据量非常大，而且字符集较小，长度较短的模式几乎毫无例外是频繁。这样的频繁模式对描述序列特征、聚类的贡献很小，而最大频繁模式则能最大限度地捕捉相关序列共同的数据特性。

3.3 算法基础与思想

我们借鉴了 PrefixSpan[5]的整体框架，它是 Jian Pei 等提出的一种基于后缀投影数据库的频繁序列挖掘方法，采用深度优先搜索策略，不需要产生候选模式集。首先，扫描序列数据库，生成所有长度为 1 的序列模式，创建相应的投影数据库。投影数据库可用伪投影的方式实现，以(id,offset)二元组的列表表示，id 代表序列编号，offset 代表模式在序列中出现的第一个位置。在相应的投影数据库上重复上述步骤扩展模式，直到在相应的投影数据库上不能产生长度为 1 的序列模式为止。

例如，对于表 1 中的数据库，模式 $p=ac$ 的投影数据库为 $PDB_p=\{(1,2),(2,2),(3,3),(3,7),(4,6)\}$。模式增长的时候，在该投影数据库中扩展长度 1。例如模式 $p'=act$，则可以得到 $PDB_{p'}=\{(1,2),(2,2),(3,3)\}$。

3.4 最大频繁序列模式挖掘算法 maxMiner

如图 1 所示的伪代码，maxMiner 算法首先扫描数据库，生成长度为 1 的频繁模式及其投影数据库(行 1～4)，然后递归调用。首先做前向模式增长(行 8～12)，如果无法向前扩展，则尝试向后扩展(行 13～16)。如果前后都无法扩展，则找到一个最大模式，输出该模式及其出现序列列表(行 17～19)。

3.5 基于闭合频繁序列模式的剪枝策略

如图 2 算法所示，如果某一个模式 p 的前向或者后向扩展的符号都相同，则 p 不是闭合频繁序列模式，可以放弃该分枝的搜索。因为必然存在某个 p'，是 p 的超模式，并且由 p' 出发，可以找到所有分枝 p 以下的所有频繁模式。

例如，如果模式 $p=at$ 出现的每个地方，前一个字符都是 c，则我们必然可以找到模式 $p'=cat$，p 的支持度等于 p' 的支持度。

3.6 Init-I 优化方法

通常较短模式的投影数据库都非常巨大，遍历需要较大的时间开销。而在数据量较大、字符集较小、支持度较低的情况下，短模式几乎无一例外频繁。在保证$|F|I<<|D|$时，可以直接构建长度为 I 的模式的投影库。例如，大小为 1M 的 DNA 序列，最小支持度为 5，基本上所有长度 3 以内的模式都频繁。因此，在初始化的时候，直接构建长度为 3 的模式的投影库，就是个很合理的想法。

Algorithm 1 maxMiner(D,F,minSup,pdb,p)

Input: D: the sequence database
F: set of symbols in sequences
minSup: minimal support of pattern
pdb: projected database to be searched
p: current sequential pattern

Output: maximal sequential patterns with their occurrences list

```
1. if pdb is null
2.     for each symbol a in F
3.         initial p with a
4.         generate len-1 projected database pdb
5.         maxMiner(D,F,inSup,pdb,p)
6. else
7.     if(closePrunce(D,F,minSup,pdb,p))return;
8.     for each symbol a in F //extend forward
9.         extend to p' with a,p'=p+a
10.        generate projected database pdb' for p'
11.        if supp(p')is not less than minSup
12.            maxMiner(D,F,minSup,pdb',p')
13. if supp(p')is less than minSup for all symbol a
14.     //can't extend forward
15.     extend p to p'' with a,p''=a+p,similar
16.     process as above//extend backward
17. if p can't extend forward nor backward
18.         //find a maximal frequent sequential pattern
19.         output this pattern with its occurrences list
```

图 1 最大频繁序列模式挖掘主算法

Algorithm 2 closePrunce(D,F,minSup,pdb,p)

Input: same as maxMiner()

Out: true if can be pruned, false otherwise

```
1. if all forward extended symbols are the same
2.         return true;
3. if all backward extended symbls are the same
4.         return true;
5. return false;
```

图 2 基于闭合频繁序列模式的剪枝算法

3.7 最小支持度参数的选择

选择基因数据的覆盖度一般就可以达到较好的效果，覆盖度相当于对序列测序的重复次数，这是用户十分容易理解和把握的。

4 基于最大频繁序列模式的聚类

4.1 算法思想

我们以两条序列之间共有的最大频繁序列模式的长度之和计算序列间的相似度。

如图 3 所示，该算法首先依次读入频繁模式及其出现列表，建立序列相似度矩阵。然后从一条序列出发，递归地找到相互间相似度大于最小阈值 minAsso 的一组序列作为一个聚类。

举个简单的例子，最大频繁模式挖掘结果如表 2 所示，则可建立起表 3 序列相似度矩阵。矩阵中的每一行代表一个序列，后面的二元组串(id,asso)表示与其存在共有模式的序列 id 及相似度 asso。假设 minAsso 为 15，则从序列 1 出发，可以递归地找到 2,4,7,6，形成一个聚类。

表 2 模式及其出现序列列表

最大频繁模式	出现序列列表
acctgggcat	1,2,6
atccgtaac	1,2,4,7
ctaagcctaggg	4,5,7
tttgacctaccg	2,4,6

表 3 序列相关度矩阵

序列	相关序列及支持度
1	(2,19),(4,9),(6,10),(7,9)
2	(4,21),(6,22),(7,9)
4	(5,13),(6,12),(7,22)
6	(1,10),(2,22),(4,12)
7	(1,9),(2,9),(4,21),(5,12)

4.2 减小重复序列的影响

较长的重复序列将对拼接效率和正确性产生影响。在做聚类的过程中，可以简单地忽略非常长的模式，因为这些模式往往代表着重复序列中的核心部分。这种简单自然的方式，将大大减少由重复序列引起的聚类错误合并。

4.3 控制相关度矩阵的大小

最大频繁序列模式集合可能仍然规模很大，而且由于偶然因素，某个模式可能在两个完全不相关的序列中都存在。因此如果在聚类的过程，如果保存所有相关序列的信息，将导致内存无谓的巨大开销。我们设置参数 K 调控相似度矩阵的规模，对于每条序列我们只保存与之相关度最大的 K^* minSup 条序列信息。实验表明，在 K 不是很小的情况下，这种方法能有效地降低内存需求，且对聚类规模和数量的影响可以忽略不计。

Algorithm 3 cluster(patternFile,clusterSet,minAsso)

Input: patterns with their occurrences list
minAsso: minimal similarity between seqs

Out: clusterSet: each containing related seqs to be joined

```
1. read each pattern and its occurrences list
2.     update association matrix accordingly
3. for each un-clustered sequence
4.     find related sequences recursively, which have association large
than minAsso
```

图 3 基于最大频繁序列模式的聚类算法

5 基因拼接网格系统

通过以上基于最大频繁序列模式的聚类，我们把原始的基因数据按类划分为许多组。由于每组内的基因数据具有相当高的相似度，所以可以对于这些类分别进行拼接，可以使拼接得到很好的收敛，然后再把各自拼接的结果汇总即可。

对于原始数据的拼接任务来说，在每个类的基因数据上的拼接就是拼接子任务。较大的原始数据量会导致产生的聚类数目可能非常多。这样，在单台计算机上进行大量的子任务拼接会很耗时。由于各个拼接子任务之间的数据相对独立，因此我们希望能够通过拼接子任务的并发执行来提高整体的拼接速度。

近年来，网格[6]的出现给我们提供了很好的平台和环境。网格把网络资源整合成一台巨大的虚拟超级计算机，来实现计算资源、存储资源、信息资源、数据资源和知识资源等的全

面共享和协同。网格对于资源的管理是动态和透明的。这样，我们就可以吸收更多的硬件资源，来分担整个拼接子任务的计算负担，协调和组织让更多的计算机来协同地完成拼接任务。而且，对于这些资源的管理是动态、灵活的，系统的扩展性得到了保证，资源的利用率也得到了提高。这就是我们实现基因拼接网格系统的出发点。

我们采用基于OGSA[7]体系结构的Globus Toolkit[8,9]来实施我们的系统。根据前面对于拼接过程的描述和分析，我们设计两种类型的结点。以下分别介绍这两种结点以及总体的系统体系结构。

5.1 主控聚类结点

主控聚类结点负责对原始基因数据进行基于最大频繁序列模式的聚类，以及在聚类后以中央控制的方式协调和组织辅助拼接结点来进行子任务的拼接。在主控聚类结点上，我们部署了以下组件：

- 基于最大频繁序列模式的聚类算法。通过这个组件，系统进行原始基因数据的聚类划分。
- 拼接子任务发送和结果回收。它负责子任务相关数据的发送以及拼接结果相关数据的回收。
- 远程拼接调用。它通过远程调用来命令相应的辅助拼接结点进行子任务的拼接。

主控聚类结点的体系结构图可以参见图4。

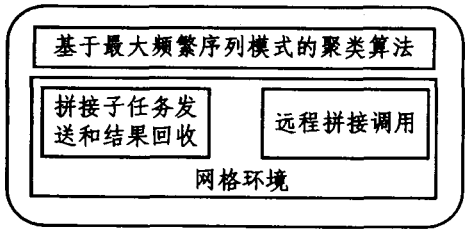

图4 主控聚类结点的体系结构图

5.2 辅助拼接结点

辅助拼接结点相对比较简单，它为系统提供拼接需要的计算能力。在辅助拼接结点上我们部署的组件如下：

- 拼接子任务接受和结果返回。它负责接受主控聚类结点分发来的子任务相关数据，在拼接完成后返回子任务的拼接结果。
- 拼接服务程序。它包装了调用PHRAP来进行基因数据拼接的过程。

辅助拼接结点的体系结构图可以参见图5。

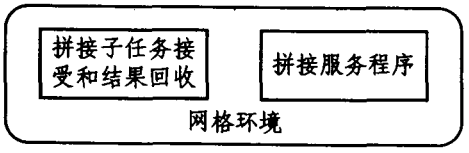

图5 辅助拼接结点的体系结构图

5.3 系统体系结构

根据上面对于拼接过程的分析可以看出，在整个系统运行中，主控聚类结点和辅助拼接结点之间保持着密切的通信，包括文件传输和远程调用。主控聚类结点处于中央的控制协调地位，其他的辅助拼接结点作为计算资源被使用。图6说明了这样的体系结构。

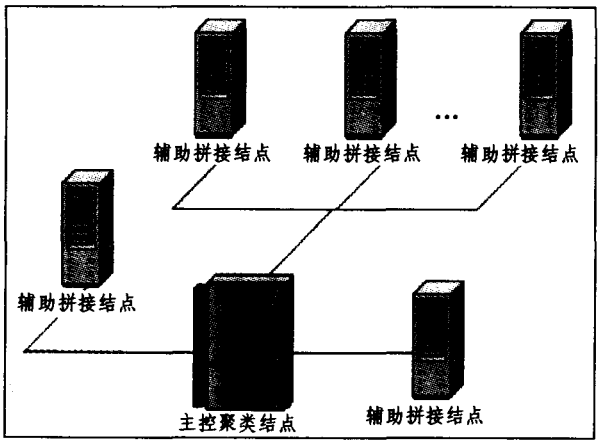

图6 系统体系结构

6 实验评估

我们使用番茄溃疡病菌(Clavibacter michiganensis)真实数据集测试我们的聚类算法。这个数据集约为32.5M，包含61042条序列，数据覆盖度约10X，可以从ftp://ftp.sanger-ac.uk/pub/pathogens/cm/下载该数据集。

实验环境是奔四2.8C处理器，512M内存，Windows 2000 Server系统。算法使用C++编写。

表4给出了部分重要参数说明。

表4 重要参数及说明

参数	说明
minSup	最小支持度(可设为基因序列的覆盖度)
minAsso	序列间最小相似度
K	聚类过程中，只维护相关度最高的K*minSupp条相关序列

CM数据集实验结果如表5所示。

表5 CM数据集运行结果(K=10，minSup=10)

阶段	执行时间	内存峰值
最大频繁模式挖掘(带Close + Init-I优化)	118s	174.5M
最大频繁模式挖掘(不带优化)	758s	58.8M
基于最大模式的聚类	31s	139.3M

由表5可以看出，基于闭合模式的优化策略大大加快了挖掘的进程。更多的实验数据表明，执行耗用的内存与数据集大小成正比，约2～3倍。

为了考察参数K对聚类结果的影响，我们又做了表6所示的实验，其中聚类数量中不包含单条序列的聚类，即孤立的序列，N50是指至少包含50%序列的聚类集中最小的聚类规模。

表6 参数K对聚类结果的影响(minAsso = 350)

K值	聚类数量(个)	最大聚类(条)	N50聚类大小(条)	内存消耗
10	1684	12992	51	139.3M
12	1672	13203	51	154.8M
14	1665	13441	52	168.4M
16	1663	13441	52	180.4M
18	1662	13464	52	197.5M
无限制	1662	13464	52	492.2M

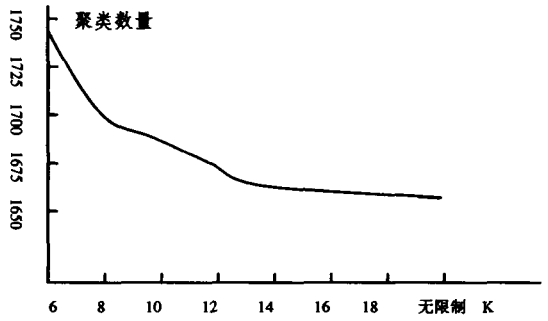

图 7 参数 K 对聚类数量的影响

由表 6 及图 7 可以看出，当 K 不太小时，使用该策略基本不影响聚类的效果。

将序列聚类之后，成功地在单机和网格系统中完成拼接，使用两个计算节点耗时 28 分钟拼接完成。同样的数据集，GiSA 系统执行时间超过 3 小时。而未经分组的原始数据在拼接过程中，由于内存不足而出错退出。

小结及未来工作 本文提出了一种基于最大频繁序列模式的聚类算法，能较准确地分组序列，同时给出了挖掘最大频繁序列模式的高效算法。在此基础上，我们实现了一个基因拼接网格系统，扩展计算能力，使序列聚类能够独立并行地得到处理。

未来的工作包括：基于硬盘的挖掘和聚类算法、挖掘模糊的最大频繁序列模式的算法以及将现有成果应用到日本血吸虫的基因组拼接。

参 考 文 献

1 http://www.phrap.org
2 http://www.ncbi.nlm.nih.gov/blast/
3 Wang J, Wang J, Yang HM, et al. RePS A: Sequence Assembler That Masks Exact Repeats Identified from the shotgun Data. Genome Research, 2002, 12: 824～831
4 Tang J, Huang D, Wang C, et al. GiSA: A Grid System for Genome Sequences Assembly(Industrial Full Paper). In: Proc. of 23th Intl. Conf. on Conceptual Modeling(ER'04), 2004
5 Jian P, Han JW, Morta-zavi-Asl B, et al. Mining Sequential Patterns by Prefix-Projected Growth. ICDE, 2001. 215～224
6 Foster I, Kesselman C. The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1998
7 OGSA(Open Grid Services Architecture) Documents. http://www.globus.org/ogsa
8 Globus: Research in Resource Management. http:// www.globus.org/research/
9 Foster I, Kesselman C. The globus project: A status report. In: Proc. The Heterogeneous Computing Workshop, 1998. 4～18
10 Mullikin J C, Ning Z. The Phusion Assembler. Genome Research, 2003, 13(1): 81～90
11 Wang JY, Han JW. BIDE: Efficient Mining of Frequent Closed Sequences. In: 20 Intl. Conf. on Date Engineering

(上接第 134 页)

虫(0,1,2;0.6)，蛔虫(0,1,2;0.6)，布氏嗜典阿米巴(0,1;0.6)，微小内蜒阿米(0,1;0.6)，巴人酵母菌(0,1;0.6)。属性后括号内的格式为(属性;重要度)，因而属性 1～12 的属性值个数分别为：2,4,7,4,3,4,6,3,3,2,2,2。

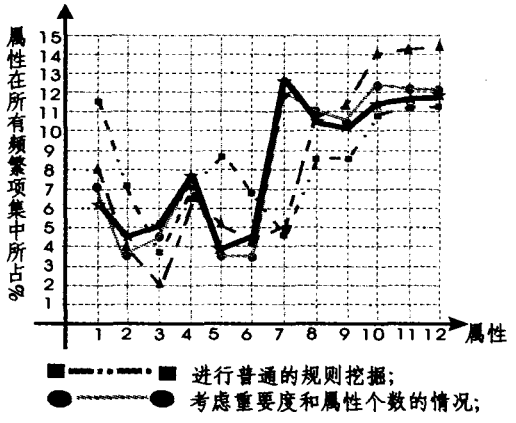

图 1 (supp(2%))

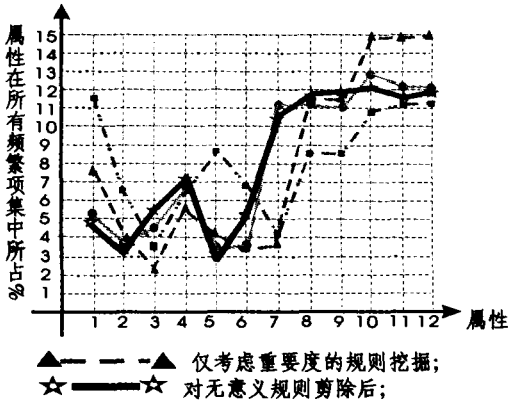

图 2 (supp(3%))

从上面两个图表可以看出，对于不同的支持度有类似的结果：如果不考虑属性重要度，则在挖出的频繁项集中，含属性值较多的属性很少，因而也就很难得到对应属性的规则；如果仅考虑属性重要度，可以看出，对于属性值较少重要度较高的属性可以得到很高的出现率，如属性 10,11,12，但对于属性值较多的属性尽管其重要度高也没有多大作用，如属性 7。因此仅对重要度的设置根本达不到预期的目的；当考虑属性的重要度兼属性个数时，从总体上来说，属性重要度大的属性其在频繁项集中出现的频度也较高。但我们看不出剪除无意义频繁项集的效果，因从百分比上看剪枝前后没有明显的差别。而从我们对 1000 条记录进行处理来看，支持度为 2%和 3%时，频繁项集中项的累计出现次数分别从 2702 下降至 1495 和从 1055 下降至 573。可见在频繁项集中无意义的项集占去了很大一部分，剪除无意义的频繁项集是必要的。

小结 本文开发了信息系统中根据用户的主观重视程度设计的规则挖掘方法，并归结了信息系统中规则发现的特点，提出综合属性值的个数和用户的主观关注程度的模式综合重要度，进而调整不同模式的最小支持度，从而得到合理的频繁项集。并通过实验证明该方法是可行并有效的。但本文只涉及到离散型的属性值，而在现实中很多数据是连续的，这是下一步要进行的工作。

参 考 文 献

1 程继华，郭建生，施鹏飞. 挖掘所关注规则的多策略方法研究. 计算机学报，2000(1)
2 陆建江. 加权关联规则挖掘算法的研究. 计算机研究与发展，2002(10)
3 武鹏程，袁兆山. 混合关联规则及其挖掘算法. 小型微型计算机系统，2003(5)
4 Duntsh I, Gediga G. Simple data filtering in rough set systems. International Journal of Approximate Reasoning, 1998, 18: 93～106