基于集合运算的关联规则采掘算法

A Set Operation Based Algorithm for Association Rules Mining

铁治欣[1] 陈 奇[2] 俞瑞钊[2]

(许继日立有限公司 河南许昌461000)[1] (浙江大学人工智能研究所 杭州310027)[2]

Abstract Mining association rules are an important data mining problem. In this paper, an association rules mining algorithm, ARDBSO, which is based on set operation, is given. It can find all large itemsets in the database while only scan the database once. So, the time for I/O is reduced enormously and the efficiency of ARDBSO is improved. The experiments show that the efficiency of ARDBSO is 80～150 times of Apriori's.

Keywords Data mining, Knowledge discovery, Association rules, Large items

1 引言

近年来,随着商业、政府和科学数据库的急剧增长和存贮设备的不断升级,给我们带来了大量的数据。面对这"堆积如山"的数据,用传统的数据分析手段无法理解并有效地利用它们。人们希望找到能够自动地、智能地理解分析它们的方法,于是就出现了数据采掘技术。它的提出为我们有效识别出存在于这些数据中的有效的、新颖的、具有潜在效用的乃至最终可理解的模式提供了可靠的科学方法,并成为当今国际人工智能和数据库等领域新兴且热门的研究课题。[1,2]

在事务数据库中发现关联规则是由 R·Agrawal 等人首先提出的[3]。目前已成为数据采掘领域中的一个非常重要的研究课题[1,3~12]。有一个关联规则的例子就是"面包⇒牛奶(20%,80%)",其直观意义为在所有的购买交易中,同时购卖面包与牛奶的概率为20%,在所有购卖面包的交易中有80%交易同时也购买牛奶。关联规则的应用主要包括顾客购物分析、目录设计、商品广告邮寄分析、追加销售、仓储规划、网络故障分析等。

本文给出一个基于集合运算的关联规则发现算法 ARDBSO (Association Rules Discovey Based on Set Operation),它只需对数据库进行一次扫描即可得到所有存在于数据库中的大项集。算法 ARDBSO 在扫描数据库时,得到所有的大1项集,并记录下支持大1项集中每一元素的记录的 TID 号,形成大项集的支持 TID 集,然后利用集合的"并"与"交"运算直接求出大 $k(k\geq 2)$ 项集及其支持 TID 集和支持数。

2 问题描述

2.1 关联规则的采掘

设 $I=\{i_1,i_2,\cdots,i_m\}$ 是由 m 个不同的项目组成的集合,给定一个事务数据库 D,其中的每一个事务 T 是 I 中一组项目的集合,即 $T\subseteq I$,T 有一个唯一的标识符 TID。

定义1 若项集 $X\subseteq I$ 且 $X\subseteq T$,则称 T 包含 X 或 T 支持 X。事务数据库 D 中支持项集 X 的总条数,称为 X 的支持数,用 $X.\sup$ 表示。

定义2 一条关联规则就是形如 $X\Rightarrow Y$ 的蕴涵式,其中 $X\subseteq I$,$Y\subseteq I$,$X\cap Y=\phi$。关联规则 $X\Rightarrow Y$ 成立的条件是:①它具有支持度 s。即事务数据库 D 中至少有 $s\%$ 的事务包含 $X\cup Y$。②它具有置信度 c。即在事务数据库 D 中包含 X 的事务至少有 $c\%$ 同时也包含 Y。

关联规则的采掘问题就是在事务数据库 D 中找出具有用户给定的最小支持度 $S_{\min}$ 和最小置信度 $C_{\min}$ 的关联规则。

定义3 设 $|D|$ 为事务数据库 D 中的事务总条数,对于具有 k 个元素的项集 X,若 $X.\sup\geq\min\sup\times|D|$,则称 X 是大 k 项集,否则称 X 是弱 k 项集。所有大 k 项集的集合称为大项集,所有弱 k 项集的集合称为弱项集。

采掘关联规则问题可以分解为以下两个子问

铁治欣 博士,主要研究方向为人工智能、数据采掘、电力系统自动化等。陈 奇 副教授,主要研究方向为人工智能、智能软件、ERP 等。俞瑞钊 教授,博士生导师,主要研究方向为人工智能、智能软件、决策支持系统等。

题[3,4]：①找出存在于事务数据库中的所有大项集；②利用大项集生成关联规则。对于每个大项集 A，若 $B \subset A$，$B \neq \phi$，且 support(A)/support(B)≥minconf，则有关联规则 B⇒(A−B)。Agrawal 等人已经给出了比较好的解决第二个子问题的办法[4]，目前大多数的研究工作主要集中在第一个子问题上，本文也只讨论第一个子问题。

2.2 相关工作

目前已有不少学者提出了许多种采掘关联规则的算法，如 Agrawal 等人提出的 AIS[3]、Apriori 和 AprioriTid[4]，Cumulate 和 Stratify[5]，Houtsma 等人提出的 SETM[6]，Park 等人提出的 DHP[7]，Savasere 等人的 PARTITION[8] 以及 Toivonen 提出的抽样算法 Sampling[9]以及 Han 等人提出的采掘多层关联规则的 ML_T2L[10]，Agrawal 等人提出的并行采掘算法 CD[11]，Cheung 等人提出的分布式采掘算法 DMA[12] 等等。

我们知道，要想提高关联规则采掘算法的效率，就必须从以下两个问题入手：①减少对数据库的扫描次数。因为关联规则采掘的数据对象一般是海量的，若能有效地减少对数据库的扫描次数，就会大大减少 I/O 负载，从而提高算法的效率；②生成较小的候选大项集。因为候选大项集越小需要与数据库中的每一记录相匹配的模式就越小，从而也能达到提高算法效率的目的。

目前已有的关联规则采掘算法中，大多数需要对数据库作多次扫描，从而导致较大的 I/O 负载。算法 PAPTITON[3]巧妙地将要发现关联规则的数据库 D 分为 n 个互不相交的且每一个的大小能够容纳在内存之中的分数据库 $D^1, D^2, \cdots, D^n$，从而使算法仅对整个数据库进行两次扫描就可以得到全部的大项集；算法 Sampling[9]利用对数据库进行随机抽样的方法，使算法能够在最多对数据库进行两次扫描的情况下得到存在于数据库中全部的大项集；文[4,7]中提到的"扫描缩减"(Scan reduction)技术也能有效地减少对数据库的扫描次数。

上述几种算法或方法虽然能够有效地减少对数据库的扫描次数，但减少的次数毕竟有限。

3 采掘关联规则高效算法 ARDBSO

为描述方便，我们在算法 *ARDBSO* 引入如下的符号。

$(TID, \langle I_1, I_2, \cdots, I_m \rangle)$：数据库中的某一记录，$TID$ 为该记录的标识号，在整个数据库中唯一，$I_1, I_2, \cdots, I_m$ 表示该记录中所含有的 m 项目

L_k：所有大 k 项集组成的集合，习惯上称其为大 k 项集

$L_{k,m}$：所有大 k 项集组成的集合中的第 m 个元素

$L_{k,m}[i]$：大 k 项集的第 m 个元素的第 i 个项目

$X.Set$：组成项集 X 的所有项目的集合，即项集 X 的项集

$X.Set[i]$：项集 X 所包含的第的 i 个项目

$X.SupSet$：项集 X 的支持 TID 集

$X.SupSet[i]$：项集 X 的支持 TID 集中的第 i 个 TID 号

$X.TID$：数据库中记录 X 的 TID 号

$|X|$：对象 X 的大小，若 X 是项集时，表示项集中的元素数；若 X 是数据库时，则表示数据库中的记录的条数

设数据库记录的 TID 号在整个数据库中是唯一的，在准备采掘数据时，我们将数据库记录按其 TID 号的大小进行排序。

定义4 对于任意一个项集 X，由数据库 D 中包含项集 X 的记录的 TID 号所组成的集合，我们称之为项集 X 的支持 TID 集。

在算法 ARDBSO 中，每一个项集都由两个域组成。域 Set 为组成该项集的所有项目的集合；域 $SupSet$ 为该项集的支持 TID 集。

由于数据库记录的 TID 号在整个数据库中是唯一的，由定义1和定义4我们可以知道，项集 X 的支持数等于其支持 TID 集中的元素数目，即：$X.Sup = |X.SupSet|$。

定理1 任意一个 k 项集 X，设 Y, Z 是两个不同的项集，且 $Y, Z \subset X$，$Y \neq \phi$，$Z \neq \phi$，$Y \cup Z = X$，则 $X.SupSet = Y.SupSet \cap Z.SupSet$。

证明：由于 $Y, Z \subset X$，根据定义1，则有 $X.SupSet \subseteq Y.SupSet$，$X.SupSet \subseteq Z.SupSet$，所以对于任意 $tid \in X.SupSet$，必有 $tid \in Y.SupSet$，$tid \in Z.SupSet$，即有 $tid \in Y.SupSet \cap Z.SupSet$。另一方面，对于任意 $tid \in Y.SupSet \cap Z.SupSet$，即 TID 号为 tid 的数据库记录既支持项集 Y 又支持项集 Z，由于 $X = Y \cup Z$，所以 TID 号为 tid 的数据库记录必支持项集 X，即 $tid \in X.SupSet$。于是我们可得结论 $X.SupSet = Y.SupSet \cap Z.SupSet$。

定义5 设有项集 X, Y，$X.Set$ 与 $Y.Set$ 所包含项目的个数分别为 m, n，将 $X.Set$ 中的 m 个项目按其在 $X.Set$ 中的顺序排成一排，形成一个字符串，记为 SX，同样将 $Y.Set$ 生成的字符串记为 SY，若按字典顺序有：①$SX < SY$，则称 X 小于 Y，记为 $X < Y$；②$SX > SY$，则称 X 大于 Y，记为 $X > Y$；③$SX = SY$，则称 X 等于 Y，记为 $X = Y$。

算法 ARDBSO 的具体实现见算法1。算法的主要思想为：首先对数据库进行一次扫描，得到大1项集 L_1，并对 L_1 中的元素按定义5所定义的大小关系进行升序排列；然后以 L_1 为基础进行多次循环，生成大 $k(k\geq 2)$ 项集 L_k。在第 $k(k\geq 2)$ 次循环中，对于任意两个大 $k-1$ 项集 $L_{k-1,m}, L_{k-1,n}\in L_{k-1}$，若条件

$$L_{k-1,m}[1]=L_{k-1,n}[1], L_{k-1,m}[2]=L_{k-1,n}[2], \cdots, L_{k-1,m}[k-2]=L_{k-1,n}[k-2], L_{k-1,m}[k-1]<L_{k-1,n}[k-1] \quad (1)$$

成立，则我们可以根据定理1生成一个 k 项集 $newl$，使得 $newl.Set=L_{k-1,m}.Set\cup L_{k-1,n}.Set=\{L_{k-1,m}[1], L_{k-1,m}[2], \cdots, L_{k-1,m}[k-2], L_{k-1,m}[k-1], L_{k-1,n}[k-1]\}$，$newl.SupSet=L_{k-1,m}.SupSet\cap L_{k-1,n}.SupSet$，若 k 项集 $newl$ 的支持数大于或等于最小支持度 S_{min} 与数据库中记录总条数的乘积，则 k 项集 $newl$ 为大 k 项集，否则为弱项集。根据以上大项集的生成过程，我们可以知道：对于大 k 项集 L_k 中的任意元素 X，必有 $X.Set[1]<X.Set[2]<\cdots<X.Set[k]$、$X.SupSet[1]<X.SupSet[2]<\cdots X.SupSet[k]$；对于大 k 项集 L_k，必有 $L_{k,1}.Set<L_{k,2}.Set<\cdots<L_{k,k}.Set$。所以若条件(1)不成立，则我们可以断定 $L_{k-1,m}$ 与 $L_{k-1,n+i}(n+i<k)$ 也不会使条件(1)成立，即用 $L_{k-1,m}$ 与 $L_{k-1,n+i}(n+i<k)$ 肯定不会生成大 k 项集，没有必要对它们再进行运算，我们称此剪枝技术为排序剪枝技术。

算法 ARDBSO 可以分为三部分。第一部分主要完成对数据库的扫描，生成所有大1项集的集合 L_1，并对 L_1 中元素按其域 Set 中元素的大小进行升序排列；第二部分以 L_1 为基础进行多次循环，生成大 $k(k\geq 2)$ 项集 L_k；第三部分即函数 $DiffLast$，其作用为判断用两个大 $k-1$ 项集能否生成一个大 $k(k\geq 2)$ 项集，若能则生成该大 k 项集。

算法用一个链表来存贮 $L_k(k=1,2,\cdots)$，每一个节点上存贮一个大 k 项集 $X\in L_k$，$X.Set$ 与 $X.SupSet$ 也分别用链表来存贮。在第 k 循环，若 L_{k-1} 较大，不能完全读入内存，则对 L_{k-1} 进行分组，每一组中的元素要求组成它们的前 $k-2$ 个项目是相同的；然后根据内存的多少尽量多地读入 L_{k-1} 分组，由它们生成大 k 项集，然后再读入剩下的 L_{k-1} 分组，再由它们生成大 k 项集，直到将 L_{k-1} 分组读完为止。

算法1　算法 ARDBSO

```
输入：数据库 D，最小支持度 Smin。
输出：大项集 L=∪Lk
方法：
  //第一部分，扫描数据库，生成大1项集。
L1=φ;
for all transactions t∈D{
  for all items c∈t{
    for all items l∈L1{
      if c∈l.Set then
        l.SupSet=l.SupSet∪{t.TID};
      else
        newl.Set={c}; newl.SupSet={t.TID}; L1=L1∪{newl};
    }
  }
}
L1={l|l∈L1, |l.SupSet|≥Smin*|D|};
Sort(L1); //用定义5对 L1进行排序
//第二部分，生成大 k(k>1)项集。
for(k=2; Lk-1≠∅; k++){
  Lk=φ;
  LABEL1:
  for(m=1; m<|Lk-1|; m++){
    for(n=m+1; n≤|Lk-1|; n++){
      ret=DiffLast(Lk-1,m, Lk-1,n, newl);
      if ret=0 then Lk=Lk∪{newl};
      if ret=2 then goto LABEL1;
    }
  }
}
L=∪kLk;
//第三部分，用两个大 k-1项集生成大 k 项集。
Function DiffLast(Lk-1,m, Lk-1,n, newl)
If Lk-1,m[1]=Lk-1,n[1], Lk-1,m[2]=Lk-1,n[2], ...,
  Lk-1,m[k-2]=Lk-1,n[k-2], Lk-1,m[k-1]<Lk-1,n[k-1]then{
  newl.SupSet=Lk-1,m.SupSet∩Lk-1,n.SupSet;
  if |newl.SupSet|≥Smin*|D|then{
    newl.Set=Lk-1,m.Set∪Lk-1,n.Set
    ={Lk-1,m.Set[1], Lk-1,m.Set[2], ..., Lk-1,m[k-1],
      Lk-1,n.Set[k-1];
    return 0;
  }
  else
    return 1;
}
else
    return 2;
```

算法 ARDBSO 主要有以下特点：

①只需对数据库扫描一次，大大减少了 I/O 时间和候选大项集与数据库记录进行匹配的时间；

②不需生成候选大项集，直接用两个大 $(k-1)$ 项集的项集通过集合"并"运算得到大 k 项集的项集；用两个大 $(k-1)$ 项集的支持 TID 集通过集合"交"运算得到大 k 项集的支持 TID 集，节省了存贮空间；

③采用了排序剪枝技术，节省了大项集之间的匹配时间；

④项集的支持 TID 集是有序的，从而使支持 TID 集的"交"运算能很快地执行。

4　测试结果与分析

为了验证算法 ARDBSO 的性能，我们在一台主频为266MHZ、内存64M、内装 Windows95操作系统的奔腾计算机上对其进行了测试。

测试数据库的有关参数意义和生成算法与文[4]相同。生成测试数据库的有关参数取值为：$N=1000$，$|L|=2000$，生成了十二个测试数据库，分别为 T20.I6.D10k、T10.I4.D1000k 和 T10.I4.D10～100k（从10k 到100k 每增加10k 为一个测试数据库）。算法 ARDBSO 与算法 Apriori 执行结果的比较如图1～5。

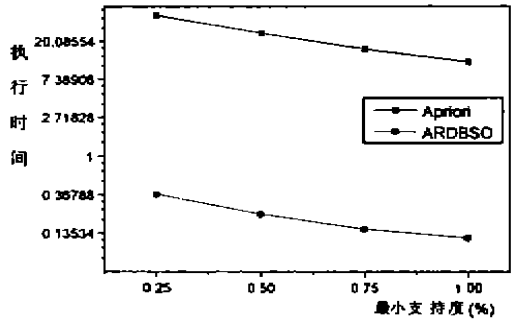

图1 算法Apriori与 算法ARDBSO的比较(T10I4D10K)

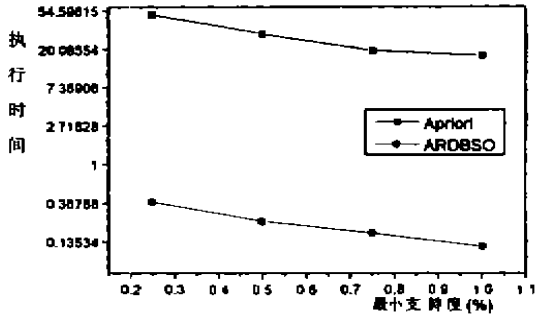

图2 算法Aprion与 算法ARDBSO的比较(T20I6D10K)

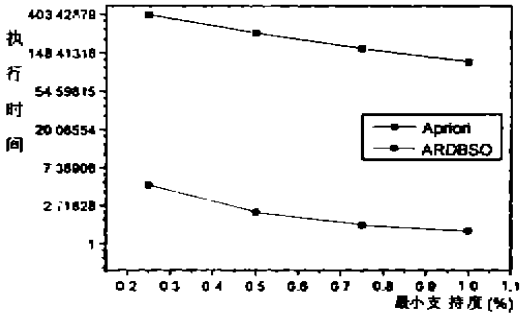

图3 算法Aprion与 算法ARDBSO的比较(T10I4D100K)

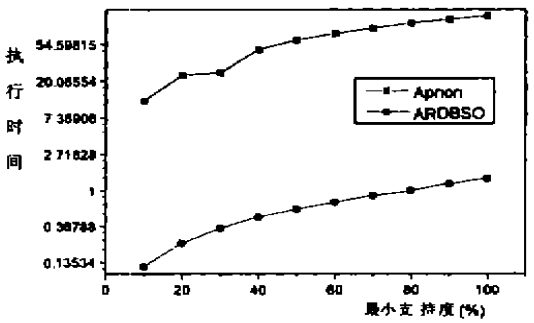

图4 算法Apriori与 算法ARDBSO的比较(S_{min}=1%)

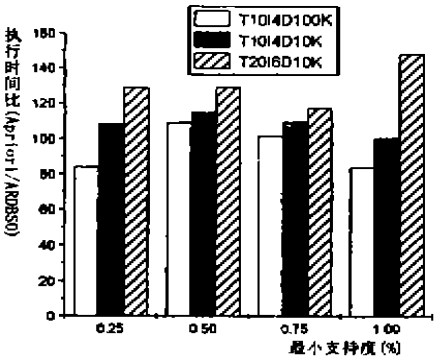

图5 算法 Apriori 与算法 ARDBSO 的效率比较

图1～3反映了在不同数据库下，随着最小支持度的改变算法 ARDBSO 与算法 Apriori 的执行时间情况。从图1～3中我们可以知道：在同样的数据库下，随着最小支持度的减小，算法 ARDBSO 与算法 Apriori 的执行时间都有增长的趋势，但算法 ARDBSO 的执行时间的增长速度要比算法 Apriori 慢得多；算法 ARDBSO 所用的执行时间总比算法 Apriori 要少。

图4反映了在相同最小支持度下，当数据库尺寸增加时，两个算法的执行时间比较情况。从图4我们可以知道：这两个算法都有很好的扩放性，但随着数据库尺寸的增大，算法 Apriori 所需用的执行时间增加幅度较大，而算法 ARDBSO 所需用的执行时间增加幅度较小。

图5反映了在不同数据库下算法 ARDBSO 与算法 Apriori 的效率的比较情况。从图中我们可以看出算法 ARDBSO 的效率是算法 Apriori 的80～150倍。

从以上的测试结果可知，算法 ARDBSO 是有效的，其执行效率比算法 Apriori 高得多。

结束语 本文对关联规则采掘问题进行了深入的研究，提出了一个基于集合运算的关联规则发现算法 ARDBSO。它只需对数据库进行一次扫描即可得到所有存在于数据库中的大项集，大大减少了 I/O 时间；不需生成候选大项集，直接用集合"并"、"交"运算生成大项集及其支持 TID 集，节省了存贮空间；采用了排序剪枝技术，节省了大项集之间的匹配时间；算法中项集的支持 TID 集是有序的，从而使支持 TID 集的"交"运算能很快地执行。从算法的测试结果可知算法是可行且有效的。

参 考 文 献

1 Chen M S, et al. Data mining: an overview form database perspective. IEEE Transactions on knowledge and data engineering, 1996.8(6): 866～883

2 Agrawal R, et al. Database Mining: A Performance Perapective. IEEE Transactions on knowledge and data engineering, 1993.5(6): 914～925

3 Agrawal R, et al. Mining aaaociation rules between sets of items in large databases. In: Proc. ACM SIGMOD int'l conf. management of data. Washington, DC, May 1993. 207～216

（下转第57页）

29到30)中的每对节点都要考查其间的所有路由。在延迟约束条件 D 下,本文描述的算法才执行和允许确定其路径。对每一对源和终节点对(s,e),要考查所有路径的延迟,找到最大延迟(D_{MAX})和最小延迟(D_{MIN})后才能确定 D。D 等于($D_{MIN}+t(D_{MAX}-D_{MIN})$)的整数部分,其中 $t\in[0,1]$。这样可确保满足约束条件的路由存在,并用 t 接近0表示约束条件是非常严格的。显然,如果 t=1说明所有路径都满足约束条件。

在不同的约束条件下,寻找几乎最佳路径的算法的性能如表1所示。表中使用的两种质量的度量是本算法找到最佳开销路由占总路径的百分比和最佳路径的开销与本算法找到的路径开销之比。从表1可看出该算法运行非常成功。我们多次用随机产生更强的连通性的方法改变网络的拓扑结构,该算法都可以找到最佳路径。

表1 算法的模拟结果

t	满足约束条件的平均路径数	满足约束条件的最大路径数	找到最佳路径所占的百分比	最佳开销与实际开销的比值
0.01	3.4	17	100	1
0.02	6.8	35	100	1
0.03	8.2	40	100	1
0.04	9.7	56	100	1
0.05	11.2	64	100	1
0.06	13.5	87	99.6	0.976
0.07	15.3	90	100	1
0.08	18.9	121	100	1
0.09	24.5	172	99.6	0.976
0.10	28.8	181	99.6	0.976
0.12	42.3	238	100	1
0.14	59.7	310	100	1
0.16	79.6	412	100	1
0.18	106.3	531	100	1
0.20	135.4	735	100	1
0.30	385.8	2224	100	1
0.99	2878.5	5472	100	1

结论 本方法是设计来用于有服务质量要求的交换机和路由器,例如 ATM 交换机与 IP 交换机和路由器。使用本算法的数据通信网中的交换机和路由器,通过减少网络通信不必要的网络通信开销(overhead)能几乎最优化地使用网络资源。因此,使用本算法最终会降低数据通信网的运营成本。我们也将这种算法扩充到去寻找一条具有几乎最佳开销的路径,且满足多个附加的 QoS 约束条件。这个研究成果将在另一篇论文中发表。

参考文献

1 ATM Forum,Private Network-Network Interface Specification Version 1.0(PNNI 1.0),1996

2 Crawley E,et al. A Framework for QoS-based Routing in the Internet,RFC 2386,August 1998

3 Data Connection,DC-PNNI specification,1997

4 Guerin R,Kamat S,Orda A,Przygienda T. QoS routing mechanism and OSPF extensions,Jan. 1998(IETF Internet Draft)

5 Iwata A,et al. PNNI Routing Algorithms for Multimedia ATM Internet. NEC Res. & develop. ,1997,38(1)

6 Jaffe J M. Algorithms for finding Paths with Multiple Constraints Networks,Networks and ISDN,1984,14:95～116

7 Lee W C,et al. Rule-based Call-by-Call Source Routing for Integrated Communication Networks,Infocom 3,1993,987～993

8 Lee W C,et al Multi-Criteria Routing Subject to Resource and Performance Constraints. ATM Forum,Mar. 1994 94～0280

9 Wang Zheng,Crowcroft J. Quality of Srvice Routing for Supporting Multimedia Applications. IEEE Journal on selected areas of communications,1996,14(17):1228～1234

10 Zhang Zhaohui,et al. Quality of Service Extensions to OSPF or Quality of Service Path First Routing QOSPF. IETF Internet Draft,Sep. 1998

11 Lin Chuang,et al. Differentiated Services in the Internet A Survey. Chinese Journal of Computers,2000,23(4):419～432

(上接第86页)

4 Agrawal,R. Srikant R. Fast algorithms for mining association rules. In:Proc. 20th int'l conf. very large databases Santiago,Chile,Sept. 1994. 478～499

5 Srikant R,Agrawal R. Mining generalized association rules. In:Dayal U,Gray PMD,Nishio S,eds. Proc. of the int conf. on Very Large Databases. San Francisco,CA:Morgan Kanfmann Press,1995. 406～419

6 Houtsma M,Swami A. Set-oriented mining for association rules in relational databases. In:YU P,Chen A,eds. Proc. of the Int Conf. on Data Engineering. Los Alamitos,CA:IEEE Computer Society Press,1995. 25～33

7 Park J S,et al. Using a hash-based method with transaction trimming for mining association rules. IEEE Transactions on knowledge and data engineering,1997,9(5):813～825

8 Savasere A,Omiecinski E,Navathe S. An efficient algorithm for mining association rules. In:Proc. of the 21st int conf. on very large databases,Zurich,Switzerland,Sept. 1995 432～444

9 Toivonen H. Sampling large databases for association rules. In,Proc. of the 22nd int conf. on very large databases,Bombay,India,1996. 134～145

10 Han Jia-wei,Fu Yong-jian. Discovery of multiple-level association rules from large databases. Same to[5],420～431

11 Agrawal R,et al. Parallel mining of association rules. IEEE Transactions on knowledge and data engineering,1996,8(6):962～969

12 Cheung D W,et al. Efficient mining of association rules in distributes databases. IEEE Transactions on knowledge and data engineering,1996,8(6):910～921