挖掘最大频繁项集的遗传蚁群优化算法

黄红星，黄习培，王秀丽
（福建农林大学 计算机与信息学院，福州 350002）

摘　要：为了提高挖掘的效率和精度，采用代数定义最大频繁项集并建立其数学模型，通过二进制编码将支持度的计算、蚁群算法和遗传算法求解有机地融合，从而提出一种求解该数学模型的遗传蚁群算法。实验表明，该算法挖掘最大频繁项集是有效的，具有良好的伸缩性。

关键词：关联规则；最大频繁项集；遗传算法；蚁群算法
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2010)07-2505-04
doi:10.3969/j.issn.1001-3695.2010.07.029

Genetic ant colony optimization for mining maximal frequent itemsets

HUANG Hong-xing, HUANG Xi-pei, WANG Xiu-li
(*College of Computer & Information, Fujian Agriculture & Forestry University, Fuzhou* 350002, *China*)

Abstract: In order to improve the efficiency and accuracy of mining, adopted the algebraic definition for maximal frequent itemsets and established the mathematical model for it. The computing of support, ant colony algorithm and genetic algorithm were merged organically by the binary code. Thus, this paper proposed a genetic ant colony algorithm to solve this mathematical model. Experimental results show that the proposed algorithm for mining maximal frequent itemsets is effective and scalable.
Key words: association rules; maximal frequent itemsets; genetic algorithm; ant colony algorithm

0　引言

挖掘频繁项集是关联规则和序列模式等数据挖掘应用中的关键技术和步骤。目前，对于在支持度较高情况下的短的稀疏数据集，已经得到较好的解决，主要是 Apriori 算法[1]和 FP-Growth 算法[2]，其他算法大多数是它们的改进或者衍生。Apriori 算法是一种生成—剪枝的算法，基本思想是对数据库进行排序和分层，采用自底向顶和(或)自顶向底搜索的策略，通过连接生成候选项集，然后利用先验知识对候选频繁集进行剪枝。FP-Growth 算法是一种模式增长算法，基本思想是将数据库投影到一棵频繁模式树 FP-Tree 上，然后通过寻找 FP-Tree 中节点路径来穷尽挖掘频繁项集。该算法无须生成候选项集，只需两次扫描数据库，因此挖掘效率明显提高。然而，对于支持度较低情况下的长的稠密数据集，这些算法产生的频繁项集数以万计，大多数是冗余无意义的。一种替代的方法是挖掘最大频繁项集。基于 Apriori 算法的主要有 Max-Miner[3]和 Pincer-Search[4]，基于 FP-Growth 算法的主要有 FPMax[5]和 FPMax*[6]。但是这些算法都是基于 Apriori 算法或 FP-Growth 算法，Apriori 算法需要重复地扫描数据库，产生大量的候选集，通过模式匹配检查一个很大的候选集合。FP-Growth 算法虽然扫描数据库的次数少，但要递归地创建大量的条件 FP-Tree，在由节点前缀路径形成条件 FP-Tree 来导出频繁项集时，同样要分析大量的候选频繁项集。总之，这些精确算法虽然能找到数据库里面的所有(最大)频繁项集，但是需要消耗大量时间和空间，并且最终导致很多无意义的规则。

近些年来，人们从自然界得到启发，开发了以遗传算法[7]和蚁群算法[8]为代表的生物启发式近似算法，它们在解决复杂组合优化问题上已经得到了广泛的认可。虽然它们不能像精确算法那样理想地得到问题的全局最优解，但一般能得到问题的近似解，是在时间和空间有限的条件下求得的满意解，这更具有现实意义。已经有一些关于遗传算法和蚁群算法分别直接挖掘关联规则的研究。文献[9,10]首先给出遗传算法挖掘关联规则，取得了较好的效果，其他大多数相关文献都是此基础上的改进或者衍生。文献[11]首先提出蚁群算法挖掘关联规则的思想，但是该算法本质上是发现数据库中项的相关程度，而不是关联规则。文献[12]利用蚂蚁系统在由数据集中属性和属性值对应超顶点和子顶点而构成的无向图中挖拙关联规则，该算法可以较快的挖掘出关联规则，具有较好的规则质量。文献[13,14]利用蚁群算法挖掘多维约束的关联规则，能够快速得到准确而精简的结果。文献[15]给出一种基于遗传算法和蚂蚁算法相结合的多维关联规则挖掘算法，但只是针对稀疏特性的多维关联规则的挖掘。文献[16]在文献[12,15]的基础上提出两种遗传算法与蚁群算法相混合求解关联规则的算法，其本质上只是两种算法的简单混合，没有将蚁群算法和遗传算法有机融合。文献[17]首先利用频繁项集构造一个完全图，然后利用蚁群算法在该图上挖掘规则，但是对包含许多项的数据库构造完全图本身就是一个挑战。实际上，这些挖掘关联规则的算法都是顺序覆盖算法，借鉴了解决分类规则问题的思想，搜索一个规则，移去它覆盖的样例，从而得到共同覆盖样例的一组规则。但是，关联规则与分类规则有着本质

收稿日期：2009-12-23；**修回日期**：2010-01-28
作者简介：黄红星(1979-)，男，湖北红安人，讲师，硕士，主要研究方向为智能计算方法及其应用等(hhx825@126.com)；黄习培(1977-)，男，讲师，硕士，主要研究方向为数值计算与图像处理等；王秀丽(1963-)，女，教授，主要研究方向为代数及其应用等.

的不同[18],关联规则挖掘中属性之间是对称的,而分类规则中的属性是不对称的,所以不能简单地将挖掘分类规则算法移植。并且,对于一个给定的数据库和最小支持度,挖掘关联规则是确定性的任务,而分类规则是不确定的,因此对于关联规则通常使用精确型算法挖掘而不是近似算法,这也是上述各种算法关键性能瓶颈之所在。另外,这些算法也只是直接针对关联规则的挖掘,而不是应用更广泛的频繁项集。

频繁项集是满足约束条件的数据库中项的组合,最大频繁项集是频繁项集的约简表示,挖掘数据库中的最大频繁项集是一个 NP-难问题[19],而近似算法在求解 NP-难问题中体现了良好性能。文献[20]将最大频繁项集归结为一个非线性规划问题,提出利用遗传算法进行求解的 GAM 算法,但该算法处理长的稠密数据集时性能显著下降。文献[21]将最大频繁项集抽象为带约束条件的子集问题,提出利用蚁群系统进行求解的 ACSM 算法,该算法运行时间长,项集提取率低。这些算法虽然提高了最大频繁挖掘的性能,但是由于遗传算法和蚁群算法内在的缺点,对于支持度较低情况下的长的稠密数据集,仍存在性能瓶颈。遗传算法有比较强的全局搜索能力,特别是当交叉概率比较大时,能产生大量的新个体,提高了全局搜索范围,但是对于系统中的反馈信息利用不够,当求解到一定范围时往往作大量无为的冗余迭代,求精确解效率低;蚁群算法采用信息正反馈原理并融入启发式搜索思想,具有局部搜索能力强,但容易陷入局部最优的问题,导致收敛速度慢、参数敏感。为了扬长避短,文献[22,23]首先提出将遗传算法和蚁群算法相结合的思想,随后针对不同问题,出现了大量混合算法,均取得了良好的效果。一般来说,这些混合算法并不具有通用性,不同问题的结合策略往往不同。

本文通过频繁项集的代数定义和二进制编码将支持度的计算、问题的解(最大频繁项集)、蚁群算法和遗传算法四者有机地融合,给出挖掘最大频繁项集的遗传蚁群算法。算法利用了遗传算法和蚂蚁算法共有的良好全局搜索能力,并克服了遗传算法局部搜索能力弱和蚂蚁算法搜索速度慢的缺陷。该算法只需要扫描一次数据库,然后利用蚂蚁的正反馈原理和遗传算子发现最大频繁项集。最后在数据集上的测试表明,该算法在时间效率上优于蚁群算法,在求精解效率上优于遗传算法,是时间效率和求解效率都比较好的挖掘最大频繁项集的算法。

1 频繁项集的代数定义

为了建立最大频繁项集挖掘的数学模型,首先采用向量和矩阵定义相关概念[20]。设 $I=\{i_1,i_2,\cdots,i_j,\cdots,i_n\}$ 是数据集中所有项(item)的集合,I 中的任何非空子集称为项集(itemsets)。数据库 D 是事务(transaction)的集合,$D=\{t_1,t_2,\cdots,t_i,\cdots,t_m\}$,其中每个事务是项的集合,即 $t_i\subseteq I,1\leqslant i\leqslant m$。

定义 1 项 i_j 的位向量 $\boldsymbol{x}=\{x_1,x_2,\cdots,x_i,\cdots,x_n\}$,其中 $x_i=\begin{cases}1 & \text{若 } i=j\\ 0 & \text{否则}\end{cases}$,即向量的第 j 位为 1,其余为 0。项集位向量的长度等于项集数 n。

定义 2 事物位向量矩阵 $\boldsymbol{V}=(v_{ij})_{m\times n}$,其中 $v_{ij}=\begin{cases}1 & \text{若 } i_j\in t_i\\ 0 & \text{否则}\end{cases}$,即若项 i_j 在 t_i 事务中出现,则该位向量的第 i 位置 1,否则置 0。事物位向量的长度等于事务数 m。

为了减少将来的遍历次数,缩小搜索空间,对上述矩阵增加一行和一列,分别用来存储某项在数据库出现的次数和某事物包含项的个数,即

$$v_{(m+1)j}=\sum_{i=1}^{m}v_{ij}(j=1,2,\cdots,n),v_{i(n+1)}=\sum_{j=1}^{n}v_{ij}(i=1,2,\cdots,m)$$

进一步,利用内积对定义 2 中的矩阵定义如下运算:

定义 3 向量矩阵 $\boldsymbol{V}$ 满足下列运算:

加法:$v_i\oplus v_j=[v_{1i}\wedge v_{1j},v_{2i}\wedge v_{2j},\cdots,v_{mi}\wedge v_{mj}]$,其中 $\wedge$ 表示逻辑“与”运算;

数乘:$x_j\otimes v_j=[x_j\vee v_{1j},x_j\vee v_{2j},\cdots,x_j\vee v_{mj}]'$,其中 $\vee$ 表示逻辑“或”运算;

范数:$\|v_j\|=\sum_{i=1}^{m}v_{ij}$。

定义 4 项 i_j 支持度计数为 $\text{Support}\{i_j\}=\|v_j\|$,即 $\text{Support}\{i_j\}=\|v_j\|=\sum_{i=1}^{m}v_{ij}=v_{(m+1)j}$。

定义 5 k 项集 $\{i_1,i_2,\cdots,i_k\}$ 的支持度计数 $\text{Support}\{i_1,i_2,\cdots,i_k\}=\|\bigoplus_{j=1}^{n}(x_j\otimes v_j)\|$。

定义 6 项集 X 的支持度是数据库 D 中包含 X 的事务数的百分比,记做 $\text{Support}(X)/|D|$。如果 X 的支持度不小于用户指定的最小支持数阈值 $\min\_s$,即 $\text{Support}(X)/|D|\geqslant\min\_s$,则称 X 为频繁项集;否则称 X 为非频繁项集。

定义 7 项集 X 的所含项的个数 k 称为 X 的模,记做 $k=|X|$,并称 X 为 k-项集。进一步,如果项集 X 是频繁的,则称 X 为频繁 k-项集。所有频繁 k-项集组成的集合记做 L_k。

2 最大频繁项集的数学模型

定义 8 对于给定的数据库 D 和最小支持数阈值 $\min\_s$,对于 $X\subseteq I$,若 $\text{Support}(X)/|D|\geqslant\min\_s$,但对于任意的 $Y\subseteq I$,$X\subset Y$,有 $\text{Support}(X)/|D|<\min\_s$,则称 X 为最大频繁项集。

根据定义 1,$\boldsymbol{x}=\{x_1,x_2,\cdots,x_j,\cdots,x_n\}$ 是一组 n 个离散变量的集合,$D_j=\{0,1\}$ 是变量 x_j 的值域。若 $x_j=1$,表示项 i_j 在当前解中;若 $x_j=0$,表示项 i_j 不在当前解中。一个可行解是满足最小支持度的 $\boldsymbol{x}$。根据以上定义,最大频繁项集问题的数学模型为

$$\max f(x)=\sum_{j}^{n}x_j$$

$$\text{s.t.}\begin{cases}\left\|\bigoplus_{j=1}^{n}(x_j\otimes v_j)\right\|\geqslant m\times\min\_s\\ x_j\in\{0,1\},j=1,2,\cdots,n\end{cases}$$

这样,实际上将最大频繁项集问题转换为 0-1 整数规划问题。该大规模的 0-1 整数规划问题是一个 NP-难问题,目前没有多项式时间算法,可以应用近似算法求解。另外,对于约束优化问题,可以通过罚函数将其转换为无约束优化问题求解,但是如何设置罚因子是该问题的瓶颈。因此,本文应用遗传蚁群算法求解该问题,算法不使用罚函数,允许不可行解的存在,最后通过修正算子将不可行解修改为可行解。

3 挖掘频繁项集的遗传蚁群优化算法

对于一个数据库,满足最小支持度的最大频繁项集不是惟一的,这是与其他最优化问题的显著区别,也是该问题的难点。本文不是只求包含项最多的最大频繁项集,而是尽可能求出全部最大频繁项集。遗传算法具有全局搜索能力强的特点,如果

采用遗传算法产生的初始解作为蚁群算法的初始种群,那么算法只是求解最长的最大频繁项集而不是尽可能多的最大频繁项集。由于蚁群算法局部搜索能力强,为了尽可能多地求出最大频繁项集,本文采取蚁群算法产生初始种群,然后进行遗传算子操作,发现最大频繁项集。

3.1　初始解的构建

设 N_{ant} 表示蚁群中蚂蚁的总数,$\boldsymbol{x}^a=(x_1^a,x_2^a,\cdots,x_j^a,\cdots x_n^a)$ 表示蚂蚁 a 构造的解。在算法的初始时刻,$x_j^a=0(j=1,2,\cdots,n)$,项上的信息素相等,令 $\tau_j(0)=\tau_0(\tau_0$ 为常数)。蚂蚁 a 从 I 中随机选择一项出发,设在 t 时刻构造的部分子集为 $L_k^a(t)$,当 $i_j\in I/L_k^a(t)$ 加入后的子集是频繁的,则令 $x_j^a=1$,否则放弃选择 i_j。这样通过增加一个来自于集合 $I/L_k^a(t)$ 中的可行项,不断地扩展直至成为一个最大频繁项集。每只蚂蚁根据项上残留的信息素和启发式信息独立地选择下一项。$p_j^a(t)$ 表示 t 时刻蚂蚁 a 选择 i_j 的概率:

$$p_j^a(t)=\begin{cases}\dfrac{[\tau_j(t)]^{\alpha}[\eta_j(t)]^{\beta}}{\sum\limits_{l\in \text{allowed}_a}[\tau_l(t)]^{\alpha}[\eta_l(t)]^{\beta}} & \text{如果 } j\in \text{allowed}_a\\ 0 & \text{否则}\end{cases}\tag{1}$$

最大频繁项集挖掘是无序组合优化问题,构造解的过程中添加到频繁项集中的项并不需要考虑最近刚刚加入的项,所以信息素应该与点相关而不是边。设 $\tau_j(t)$ 表示 t 时刻项 i_j 上的信息素量,该项越大表示 i_j 在候选解中被选择的期望程度越高;$\eta_j(t)$ 表示 t 时刻 i_j 的启发式信息,令 $\eta_j(t)=\text{Support}\{i_j\}$,该项越大表示 i_j 加入部分解后对支持度的贡献度越高;tabu_a $(a=1,2,\cdots,m)$ 记录蚂蚁 a 所走过的节点,当所有 n 个节点都加入到 tabu_a 中时,蚂蚁 a 便完成了一次循环,$\text{allowed}_a=I-\text{tabu}_a$ 表示蚂蚁 a 下一步允许选择的所有节点。

3.2　信息素更新

1)局部信息更新　当一个蚂蚁 a 经历时间 Δt 得到一个局部最大频繁项集 $\text{MFI}_{ka}^a(x^a)$ 后,算法进行信息素局部更新。更新公式为

$$\tau_j(t+\Delta t)=\begin{cases}(1-\lambda)\tau_j(t)+\lambda\tau_0 & \text{若 } i_j\in \text{MFI}_{ka}^a(x^a),\lambda\in(0,1)\\ \tau_j(t) & \text{否则}\end{cases}\tag{2}$$

2)全局信息素更新　当某次迭代所有的蚂蚁 a 经历时间 ΔT 都分别构造完一个最大频繁项集 $\text{MFI}_{ka}^a(x^a)$ 后,算法进行全局信息素更新。设本次迭代最大频繁项集为 $\text{MFI}_{ib}(x^{ib})$,迄今全局最大频繁项集为 $\text{MFI}_{gb}(x^{gb})$。更新公式为

$$\tau_j(t+\Delta t)=(1-\rho)\tau_j(t)+\rho\tau_j(t,t+\Delta T),\ \rho\in(0,1)\tag{3}$$

$$\Delta\tau_j(t,t+\Delta T)=\begin{cases}\dfrac{1}{1+|\text{MFI}_{gb}(x^{gb})|-|\text{MFI}_{ib}x^{ib}|} & \text{若 } i_j\in \text{MFI}_{ib}(x^{ib})\\ 0 & \text{否则}\end{cases}\tag{4}$$

3.3　遗传算法求解

与问题的解相对应,这里采用二进制编码,问题的一个解 $\boldsymbol{x}=\{x_1,x_2,\cdots,x_j,\cdots,x_n\}$ 即为种群的一个个体。种群数量与蚂蚁数相等,即每个蚂蚁产生一个个体。设 $P(g)$ 为第 g 代种群,则初始种群 $P(0)=\{x^1,x^2,\cdots,x^a,\cdots,x^{N_{ant}}\}$。其中:$\boldsymbol{x}^a=\{x_1^a,x_2^a,\cdots,x_j^a,\cdots,x_n^a\}$,$a=1,2,\cdots,N_{ant}$。适应度函数取 $f(x)=\sum\limits_j^n x_j$。遗传算子如下:

a)选择。对于适应度最高的个体,直接复制到下一代;其余个体按照轮盘赌方式进行选择。复制概率 $p_s=0.1\sim0.2$。

b)交叉。采用单点交叉,即在个体编码串中随机地设置一个交叉点,然后在该点相互交换配对个体的部分基因。交叉概率 $p_s=0.5\sim0.8$。

c)变异。采用二元变异,即编码位取反,"1"变成"0","0"变成"1"。变异概率 $p_m=0.001\sim0.1$。

d)修正。由于交叉或者变异后的个体可能是非法的,随机修改某位"1"为"0",直到满足最小支持度。

基于以上分析,挖掘最大频繁项集的遗传蚁群算法 GACM 流程图如图 1 所示。

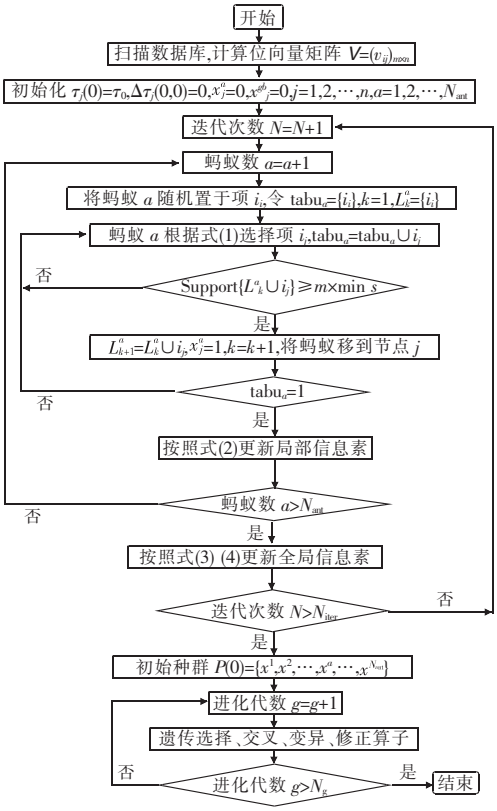

图 1　挖掘最大频繁项集的遗传蚁群算法 CACM 流程图

4　实验结果与分析

根据 2003 年频繁项集挖掘实现会议(FIMI03),当时挖掘最大频繁项集所有算法中,能在大多数数据集上获得最好性能的是 FPMax*[6] 算法。为了验证算法的性能,将本文提出的 GACM 算法分别与 FPMax* 算法、遗传算法挖掘算法 GAM[20] 以及蚁群系统挖掘算法 ACSM[21] 进行比较。实验环境为:Windows XP SP3 操作系统,Pentium4 3.2 GHz CPU,1 GB DDR 内存,SQL Server 2005 数据库,VC++.NET 2005 编译器。实验数据集为 Mushroom[24],该数据集包含 8 124 个事物,23 个属性,119 个项,转换为布尔编码长度为 58。实验参数组合为:$\alpha=1,\beta=2,\lambda=0.1,\rho=0.2,\tau_0=1/|L_1|,N_{ant}=30,N_{iter}=50,N_g=50$。

下面从挖掘最大频繁项集提取率、平均运行时间、平均迭代次数以及算法伸缩性四个方面进行比较分析。平均项集数 $N=\sum\limits_{i=1}^{N}N_i/N$,其中 N 表示实验所做的次数,N_i 表示第 i 次提取的项集数;项集提取率为 N_{app}/N_{pre},其中 N_{pre} 表示精确算法提取的平均项集数,N_{app} 表示近似算法的提取的平均项集数。

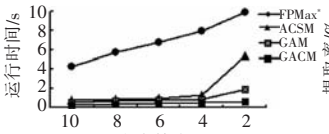

图 2　运行时间比较

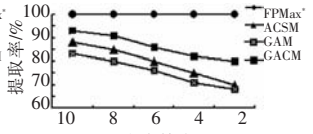

图 3　提取率比较

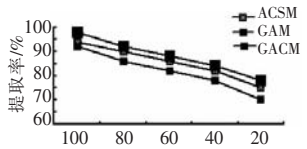

图 4　迭代代数与提取率比较

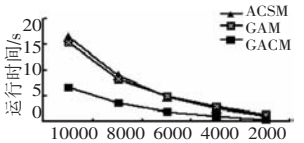

图 5　算法伸缩性比较

在数据集中随机抽取 1 000 条事物对上述四个算法进行验证,各近似算法迭代相同代数,得到不同算法在不同支持度阈值下的运行时间(图 2)和提取率(图 3)、固定支持度为 5% 时迭代代数与提取率(图 4)以及伸缩性(图 5)的比较。图 2 表明,GACM 算法在执行速度方面不但比精确算法 FPMax* 平均提高了 90%,而且比 ACSM 算法平均提高了 70%,比 GAM 算法平均提高了 50%。另外基于遗传机制的 GAM 算法执行速度要快于基于信息素机制 ACSM 算法。值得注意的是,GACM 算法所需的时间只与潜在的最大频繁项集的个数和迭代代数有关,与大多数精确算法随着支持度越小运行时间呈指数变化不同,它对支持度不敏感,时间随支持度呈线性变化。图 3 表明,精确算法 FPMax* 能够发现所有的最大频繁项集,它的提取率是 100%,而有限迭代的近似算法都无法找到所有的最大频繁项集。但在相同支持度下,GACM 比 ACSM 算法提取率平均提高了 9%,比 GAM 算法提取率平均提高了 13%,平均提取率在 85% 以上。同时,由于 ACSM 算法采取了正反馈机制,它比随机搜索的 GAM 算法提取率高,尽管要耗费更多时间。图 4 表明,各近似算法随着迭代代数和提取率的增加,在相同迭代代数下,GACM 比 ACSM 算法提取率平均提高了 3%,比 GAM 算法提取率平均提高了 8%,当迭代代数足够大时基本上可以发现所有的最大频繁项集。图 5 表明,GACM 算法随事物数执行时间呈线性增长,并且比另外两种近似算法性能高。因此,本文提出的 GACM 算法执行速度快,提取率高,具有可伸缩性。

5　结束语

本文提出的通过二进制编码将支持度的计算、蚁群算法和遗传算法有机融合的模型和算法,为最大频繁项集的挖掘开辟了新的方法,为蚁群算法和遗传算法的融合提供了新的思路。与大多数精确算法随着支持度越小运行时间呈指数增长不同,该算法所需的时间只与潜在的最大频繁项集的个数和迭代代数有关,它对支持度不敏感,时间随支持度呈线性变化。在数据集上的测试表明,其在时间效率上优于蚁群算法,在求精解效率上优于遗传算法,并且伸缩性强。但是由于遗传算法和蚁群算法涉及到较多参数,如何结合最大频繁项集的挖掘合理设置参数以进一步提高挖掘效率,将是下一步研究的方向。

参考文献:

[1] AGRAWAL R, SRIKANT R. Fast algorithms for mining association rules[C]//Proc of the 20th International Conference on Very Large Database. San Francisco, CA: Morgan Kaufmann, 1994: 487-499.

[2] HAN Jia-wei, PEI Jian, YIN Yi-wen. Mining frequent patterns without candidate generation[C]//Proc of 2000 ACM-SIGMOD International Conference on Management of Data. New York: ACM Press, 2000: 1-12.

[3] BAYARDO R J. Efficiently mining long patterns from databases[C]//Proc of the ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 1998: 85-93.

[4] LIN D I, KEDEM Z M. Pincer-search: a new algorithm for discovering the maximum frequent set[C]//Proc of the 6th European Conference on Extending Database Technology. Heidelberg: Springer-Verlag, 1998: 105-119.

[5] GRAHNE G, ZHU J F. High performance mining of maximal frequent itemsets[C]//Proc of the 6th SIAM Int'l Workshop on High Performance Data Mining. 2003: 135-143.

[6] GRAHNE G, ZHU J F. Effciently using prefix-trees in mining frequent itemsets[C]//Proc of IEEE ICDM Workshop on Frequent Itemset Mining Implementations. 2004.

[7] HOLLAND J H. Adaptation in natural and artificial system[M]. Michigan: University of Michigan Press, 1975.

[8] DORIGO M, MANIEZZO V, COLORNI A. The ant system: optimization by a colony of cooperating agents[J]. IEEE Trans on System, Man, and Cybernetics-Part B, 1996, 26(1): 29-41.

[9] FLOCKHART I W, RADCLIFFE N J. A genetic algorithm-based approach to data mining[C]//Proc of International Conference on KDD. 1996: 299-302.

[10] CHOENN R. On the suitability of genetic-based algorithms for data mining[C]//Proc of ER Workshops AIDT. London, UK: Springer-Verlag, 1998: 55-67.

[11] SU B D. Discovering association rules through ant systems[D]. Taiwan: National Chin-Hwa University, 2002.

[12] 屠莉, 陈　. 挖掘关联规则的蚁群算法[J]. 南京邮电大学学报: 自然科学版, 2006, 26(5): 36-39.

[13] ZHU Xing-liang, LI Jian-zhang. An ant colony system-based optimization scheme of data mining[C]//Proc of the 6th IEEE International Conference on Intelligent Systems Design and Applications. 2006: 400-403.

[14] KUO R J, SHIH C W. Association rule mining through the ant colony system for national health insurance research database in Taiwan[J]. Computers and Mathematics with Applications, 2007, 54: 1303-1318.

[15] 沈国强, 覃征. 一种新的多维关联规则挖掘算法[J]. 小型微型计算机系统, 2006, 27(2): 291-294.

[16] 许珂. 基于群体智能的关联规则挖掘方法及应用[D]. 济南: 山东师范大学, 2008.

[17] 唐文志. 蚁群算法在关联规则学习中的研究与应用[D]. 北京: 北京工业大学, 2009.

[18] FREITAS A A. Understanding the crucial differences between classification and discovery of association rules[C]// Proc of ACM SIGKDD Explorations. New York: ACM Press, 2000: 65-69.

[19] YANG Gui-zhen. The complexity of mining maximal frequent itemsets and maximal frequent patterns[C]//Proc of ACM SIGKDD International Conference on Knowledge Discovery in Databases. New York: ACM Press, 2004: 344-353.

[20] 张军. 基于遗传算法的频繁项挖掘算法[J]. 计算机工程与应用, 2008, 44(12): 161-165.

[21] 宋洁, 刘华, 谭庆, 等. 蚁群算法在最大频繁项集挖掘问题中的应用[J]. 计算机工程与设计, 2008, 29(20): 5290-5292.

[22] ABBATTISTA F, ABBATTISTA N, CAPONETTI L. An evolutionary and cooperative agents model for optimization[C]//Proc of the IEEE International Conference on Evolutionary Computation. Washington DC: IEEE, 1995: 668-671.

[23] WHITE T, PAGUREK B, OPPACHER F. ASGA: improving the ant system by integration with genetic algorithms[C]//Proc of the 3rd Annual Genetic Programming Conference. 1998: 610-617.

[24] GOETHALS B. Frequent itemset mining implementations repository [EB/OL]. (2004). http://fimi.cs.helsinki.fi/.