第 35 卷 第 7 期 计 算 机 工 程 2009 年 4 月
Vol.35 No.7 Computer Engineering April 2009
·软件技术与数据库· 文章编号：1000—3428(2009)07—0064—04 文献标识码：A 中图分类号：TP393

基于余弦向量法的 Web 数据并行抓掘系统

徐文杰，陈庆奎
(上海理工大学计算机与电气工程学院，上海 200093)

摘 要：为了提高 Web 海量数据的抓掘效率，引入并行机群抓掘机制。为使机群中每个计算节点的能力得到充分发挥，应用向量度量技术解决抓取任务和计算节点能力匹配的问题。对抓取任务向量、计算节点向量进行定义，提出余弦向量匹配算法，描述相关并行算法。理论分析和实验表明，基于余弦向量匹配算法的挖掘任务分配模型具有良好的分配适应性和负载平衡性。

关键词：并行抓取；余弦向量法；计算机机群

Parallel Crawling System for Web Data Based on Cosine Vector

XU Wen-jie, CHEN Qing-kui
(School of Computer and Electrical Engineering, University of Shanghai for Science and Technology, Shanghai 200093)

【Abstract】This paper proposes a parallel cluster crawling model to improve the mining efficiency of massive data on Web. For fully using of the ability of parallel nodes in computer cluster, a vector measurement technology is introduced to solve the matching problem between crawling task and computer node. After giving the definitions of crawling task vector and computer node vector, cosine vector similarity formula is described, and the parallel crawling algorithms is designed. Experimental results show that the system is effective in distribution adaptability and load balance.

【Key words】parallel crawling; cosine vector; computer cluster

1 概述

面对互联网上资源呈指数级增长并且不断更新[1]，搜索引擎[2]已成为用户经常使用的网络功能。为了保证搜索引擎及时、准确地更新信息资料，要求抓掘(Crawler)系统在尽可能短的时间内对已有的网络信息资源进行抓取。

采用并行抓掘系统[3-4]可以提高工作效率，且具有良好的扩展性。多个抓掘器并行搜集 Web 信息，各抓掘器之间通过分配策略相互协调，完成信息搜集任务。文献[3]提出了一种并行 Crawler 算法，其研究主要集中于在多个 Crawler 协作下载时如何降低重复下载率、如何保持下载质量、如何高效地运用通信带宽等，文中提出的 Web 划分策略及划分的依据对提高抓掘效率很有意义。文献[4]中的实验通过每天对 270 个网站持续 4 个多月的跟踪，得出了 Web 随时间变化的情况，并根据这些结论对增量搜索引擎进行改进。然而，现有的文献对于并行抓掘系统任务分配策略缺少论述。本文利用并行计算模型计算节点并行工作以提高处理能力的特点，通过余弦向量法[5]将抓取任务分配到合适的计算节点上运行，提升抓掘系统的运行效率。不同的抓取任务对于带宽、处理器能力等要求是不一样的。因此，将同一任务分配到不同的抓掘器上运行可获得不同的运行效率，而这种运行效率的差异是由抓掘器的特性决定的。余弦向量法通过构建向量空间，将抓取任务和抓掘器向量化，通过比较抓取任务与不同抓掘器向量夹角的差异，找到最适合运行该任务的抓掘器。同时，对抓掘器进行实时管理可以实现动态调整抓取任务的分配，也有助于提升系统的负载平衡，提升总体抓掘效果。

2 抓取任务及计算节点能力的量化定义

一个网络信息资源抓取任务在某个计算节点上运行的效率涉及到很多因素，它们在一定的约束条件下相互影响。不同的抓取任务对计算节点的要求不同，其中包括计算节点的能力(ability)、可信任度(reliability)、网络带宽(bandwidth)等。抓取任务之间也会存在较大差异，如信息资源大小、更新频度[6-7]。

定义 1 抓取任务向量及计算节点向量

为了用一个向量表示一个抓取任务(assignment)，对其做如下定义：抓取任务是一个 n 维向量 $\boldsymbol{A}=\{a_1,a_2,\cdots,a_n\}$，$n$ 为自然数，n 维向量的每一个元素用来描述该任务的一项基本特征，a_i 是一个实数，可以是抓取任务的文件大小、计算量以及访问信息资源所需要的时间，例如，抓取上海理工大学首页的任务可以描述为：$\boldsymbol{A}$={172,79,36}，其中，172 表示对页面做信息摘要需要 172 ms；79 表示文件大小约为 79 KB；36 表示访问页面的网络延时为 36 ms。

同时，对应该抓掘任务向量应该有相应的计算资源支持，如 CPU 和网络带宽。因此，对应于某个计算节点(Crawler)按照相应的特征也可以构造 n 维向量，得到计算节点向量。将其定义为：$\boldsymbol{C}=\{c_1,c_2,\cdots,c_n\}$，$n$ 为自然数。例如，一个 CPU 3.0 GHz、硬盘 60 GB、带宽 10 MB 的计算节点可以描述为：$\boldsymbol{C}$={3,60,10}。

2 个向量对应元素之间有对应关系 $c_i \leftrightarrow a_i(i=1,2,\cdots,n)$，$c_i$ 是用于支持 a_i 因素的资源描述情况。这样就可根据向量的特征选取特定的机器来支持特定的任务。如果抓取任务某一个分量为 0，表示该分量对该抓取任务没有意义。同样，如果计算节点某一分量为 0，说明该计算节点不支持该类资源。

基金项目：国家自然科学基金资助项目(60573108)；上海教委发展基金资助项目(06QZ002, 07ZZ92)；上海教委科研创新基金资助重点项目(08ZZ76)；上海市重点学科建设基金资助项目(S30501)

作者简介：徐文杰(1983－)，男，硕士研究生，主研方向：计算机机群，并行抓掘；陈庆奎，教授、博士生导师

收稿日期：2008-09-10 **E-mail：**chenqingkui@tom.com

定义 2 任务集

任务集(***R***)表示整个需要抓取任务的集合，其可以用向量集合表示 $\boldsymbol{R}=\{\boldsymbol{A}_1, \boldsymbol{A}_2, \cdots, \boldsymbol{A}_p\}$，其中，$p$ 为抓取任务的数目。

定义 3 计算节点集

计算节点集(***G***)表示参与计算机群的集合，其可以用向量集合表示 $\boldsymbol{G}=\{\boldsymbol{C}_1, \boldsymbol{C}_2, \cdots, \boldsymbol{C}_l\}$，其中，$l$ 为参与抓取的计算节点的数目。

3 抓取任务和计算节点的能力匹配

不同的抓取任务对于计算节点的性能要求也不同。提高抓取任务和计算节点间的匹配程度可以提升抓掘系统的性能。由于机群中各个计算节点的配置不一，因此所处的网络环境等也存在差别。针对特定的抓取任务，所选择的计算节点是否合适将会对执行效率产生直接影响。下面给出一些基本定义。

定义 4 夹角余弦值

余弦向量法通过计算向量间的夹角余弦值来判断向量间的匹配程度，余弦值的取值范围为(0, 1)，余弦值越接近 1，表示向量间匹配程度越高，在本文即表现为计算任务和计算节点的匹配程度高。

为了求解给定的任意一个抓取任务与计算节点之间的匹配程度，可以将抓取任务表示为 $\boldsymbol{A}_i=\{a_{i1}, a_{i2}, \cdots, a_{in}\}$，计算节点表示为 $\boldsymbol{C}_j=\{c_{j1}, c_{j2}, \cdots, c_{jn}\}$，它们所对应的匹配程度则可以表示为两者向量的夹角余弦值：

$$\cos(\boldsymbol{A}_i,\boldsymbol{C}_j)=\frac{\boldsymbol{A}_i\cdot\boldsymbol{C}_j}{\|\boldsymbol{A}_i\|\cdot\|\boldsymbol{C}_j\|}=\frac{\sum_{k=1}^{n}a_{ik}\square c_{jk}}{\sqrt{\sum_{k=1}^{n}a_{ik}{}^2\cdot\sum_{k=1}^{n}c_{jk}{}^2}}\cos(\boldsymbol{A}_i,\boldsymbol{C}_j)\in(0,1) \quad (1)$$

定义 5 加权夹角余弦值

由于向量中不同分量对抓取任务效率的影响不同，因此有必要引入相应的权系数 $\{\lambda_1,\lambda_2,\cdots,\lambda_n\}$，其中，$0\leqslant\lambda_i\leqslant1$，且 $\sum_{i=1}^{n}\lambda_i=1$，此时式(1)可改写为

$$\cos(\boldsymbol{A}_i,\boldsymbol{C}_j)=\frac{\boldsymbol{A}_i\cdot\boldsymbol{C}_j}{\|\boldsymbol{A}_i\|\cdot\|\boldsymbol{C}_j\|}=\frac{\sum_{k=1}^{n}\lambda_k\left(a_{ik}\square c_{jk}\right)}{\sqrt{\sum_{k=1}^{n}\lambda_k\left(a_{ik}{}^2\right)\cdot\sum_{k=1}^{n}\lambda_k\left(c_{jk}{}^2\right)}}\cos(\boldsymbol{A}_i,\boldsymbol{C}_j)\in(0,1) \quad (2)$$

定义 6 最佳匹配

通过将一个抓取任务与各个计算节点进行带权值的夹角余弦值计算，就可以找到一个最适合运行此抓取任务的计算节点。针对抓取任务 $\boldsymbol{A}_i$ 的最佳匹配计算节点可以描述为

$$C_{Result}=\max\left\{\cos ine(\boldsymbol{A}_i,\boldsymbol{C}_1),\cos(\boldsymbol{A}_i,\boldsymbol{C}_2),\cdots,\cos(\boldsymbol{A}_i,\boldsymbol{C}_l)\right\}$$

4 并行算法构建

实验中机群的构建采用了集中式模型，即由一台主机(master)负责向机群中的其他计算节点分派任务。Master 掌握并维护机群所有计算节点的信息。

4.1 并行算法相关定义

定义 7 计算节点的任务就绪队列

由于分配抓取任务的速度和计算节点实际运行任务的速度存在差异，因此在每个计算节点上都存在一个任务就绪队列 Q_i，用于存放该计算节点就绪抓取任务列表。为了保持任务分配的平衡，任务就绪队列的长度也作为该计算节点状态的分量，作为计算节点向量 ***C*** 的第 n+1 个分量，计算节点向量的长度扩展为 n+1。该值在运行过程中动态发生变化，其长度大小表示该计算节点当前就绪任务队列的长度。

同理，为了使余弦向量法能够正确计算，也对抓取任务向量 ***A*** 给出第 n+1 个分量，其取值为 0。

定义 8 任务就绪队列矩阵

任务就绪队列矩阵用于 Master 存放整个抓掘系统抓取任务分配状况的信息。设某一时刻系统有 l 个计算节点，它们的任务就绪队列可以表示为 $Q_1, Q_2, \cdots, Q_l$，队列的长度可分别用 $q_1, q_2, \cdots, q_l$ 表示。取 $t=\max\{q_1, q_2, \cdots, q_l\}$，则任务就绪队列矩阵为一个 $l\times t$ 矩阵：$\boldsymbol{M}_r=(a_{i,j})_{l\times t}$。其中，$a_{i,j}$ 表示第 i 个计算节点第 j 个就绪抓取任务，当 $a_{i,j}$ 为空时表示是一个空任务。

定义 9 计算节点向量矩阵

计算节点向量矩阵用于 Master 记录参与运算的计算节点的实时状态。描述计算节点实时状态的消息随着心跳信号传入 Master。计算节点的状态信息可以描述为：$M_s=(c_{i,j})_{l\times(n+1)}$。其中，$c_{i,j}$ 表示第 i 个计算节点第 j 个分量所对应的值。

4.2 计算节点间的负载平衡

从定义 8 可以看出，为了保持抓取任务分配的平衡性，需要保持各计算节点就绪队列的长度尽可能一致。而余弦向量法可以根据就绪队列的长度变化实现任务分配的调整，以达到计算节点间负载平衡的目的。

在表示计算节点的向量 $\boldsymbol{C}_j$ 中，就绪队列的长度对应于第 n+1 个分量的值，即 $c_{j,(n+1)}$。可知，对于计算节点 j，就绪队列越长，$c_{j,(n+1)}$的值越大。在抓取任务向量 $\boldsymbol{A}_i$ 中，其第 n+1 个分量的值始终为 0，即 $a_{i,(n+1)}=0$。

根据加权夹角余弦公式可知：就绪队列变长对公式的分子部分没有影响。由于 $a_{i,(n+1)}=0$，因此无论 $c_{j,(n+1)}$如何改变都不影响公式的分子部分 $\boldsymbol{A}_i\cdot\boldsymbol{C}_j$ 的值。然而此时观察公式的分母部分可知，由于 $c_{j,(n+1)}$增大，$\|C_j\|$也相应增大，而 $\|\boldsymbol{A}_i\|$ 保持不变。因此，公式的分母部分 $\|\boldsymbol{A}_i\|\cdot\|\boldsymbol{C}_j\|$ 增大。

综上所述，就绪队列越长，对应的余弦向量值 $\cos(\boldsymbol{A}_i,\boldsymbol{C}_j)$ 越小，从而减小了计算节点 j 对应于抓取任务 i 的匹配程度。

结合定义 6 可知，某一计算节点的就绪队列越长，其被分配到抓取任务的可能性就越小，从而实现了计算节点间的负载平衡。

4.3 相关算法

算法 1 任务分配算法

设 *Assignments* 是一批需要实现的抓取任务，*Batch* 是这批抓取任务的一个子集。则有：

(1)Master 读入一批抓取任务 $Batch=\{\boldsymbol{A}_i\mid\boldsymbol{A}_i\in Assignment\}$，$Batch\subseteq Assignments$，$Assignments=Assignments-Batch$；

(2)选取一个抓取任务 $\boldsymbol{A}_i$；

(3)计算 $\boldsymbol{A}_i$ 与各个计算节点的余弦值，获得 $\boldsymbol{C}_{\text{Result}}$；

(4)向 $\boldsymbol{C}_{\text{Result}}$ 对应的计算节点发送任务；

(5)将 $\boldsymbol{A}_i$ 添加到任务就绪队列矩阵 $\boldsymbol{M}_r$ 中相应节点的就绪队列；

(6)如果还有抓取任务则回到(2)。

算法 2 计算节点信息更新算法

Switch(Master 收到信号)

Case 收到计算节点 $\boldsymbol{C}_i$ 申请加入抓取任务的请求：

Master 在任务就绪队列矩阵 $\boldsymbol{M}_r$ 和计算节点状态矩阵 $\boldsymbol{M}_s$ 中创建相应的存储空间。

Break；

Case 收到计算节点 $\boldsymbol{C}_i$ 退出抓取任务的请求：

(1)Master 将任务就绪队列矩阵中分配到 $\boldsymbol{C}_i$ 的抓取任务

退回抓取任务队列 $Assignments=Assignments\cup Q_i$；

(2)将 Master 中任务就绪队列矩阵 $\boldsymbol{M}_r$ 对应于 $\boldsymbol{C}_i$ 部分的抓取任务队列删除；

(3)删除 Master 中存储于状态信息矩阵 $\boldsymbol{M}_s$ 中 $\boldsymbol{C}_i$ 部分的信息。

Break；

Case 收到计算节点 i 基于心跳机制发送的信息：

(1)修改 Master 中计算节点状态矩阵 $\boldsymbol{M}_s$ 中相应位置的节点信息；

(2)在任务就绪队列矩阵 $\boldsymbol{M}_r$ 相应位置处删除已经完成的抓取任务。

Break；

End Switch

算法 3 抓取任务执行算法

在抓掘系统中，所有计算节点都维护一张抓取任务资源链接表，抓取任务资源链接表采用二级存储结构：抓取任务对应的站点域名；抓取任务链接。如果系统存在新增的资源链接，则先找到抓取任务对应的站点域名，并在其下插入新增的资源链接。如果新增的资源链接对应的站点域名在系统中不存在，则先扩展站点域名列表，然后在其下插入新增的资源链接。抓取任务执行及资源链接表更新算法如下：

Switch(计算节点收到来自 Master 的信号)

Case 收到抓取任务：

(1)计算节点 $\boldsymbol{C}_k$ 获得由算法 1 分配到的抓取任务；

(2)查看计算节点是否存在运行的抓取任务，如果计算节点空闲就转到(4)，否则将任务放入就绪任务队列 Q_k；

(3)从就绪任务队列 Q_k 中取出一个抓取任务 $\boldsymbol{A}_i$；

(4)根据抓取任务访问相应的网络资源，实现抓取；

(5)对抓取资源提取网络链接，并与计算节点本机的链接库进行比对，保存新增的链接；

(6)对所抓得的资源进行数字摘要计算；

(7)从就绪任务队列 Q_k 中删除此抓取任务 $\boldsymbol{A}_i$；

(8)如果就绪任务队列非空则回到(1)；

Break；

Case 收到要求发送心跳信息的信号：

将计算节点 Q_k 新增的链接发送到 Master；

Break；

End Switch。

5 模拟试验结论及分析

实验构建了由 4 台配置不同的 PC 组成的机群网络，其中，一台作为 Master 负责任务的分配和结果的汇总；3 台用于运行抓取任务的计算节点环境如下：赛扬 2.1 GHz，512 MB 内存，60 GB 硬盘；奔腾 2.93 GHz，512 MB 内存，80 GB 硬盘；速龙 2.1 GHz，1 GB 内存，150 GB 硬盘。系统配置：操作系统：WindowsXP SP2，JVM：Sun J2SE 1.5 Win32 Edition。网络配置：校园网出口 10 Mb/s 带宽。

实验针对上海理工大学网站及沪江论坛进行了网页抓取，共抓取页面 1.3 万个。对所抓页面的相关分析也在抓取任务完成后直接进行处理，抓掘系统对页面的处理过程分为 3 个部分：页面存取，URL 提取，对页面做信息摘要。

实验中对于计算向量进行了筛选，选取了对抓取效率影响较大的分量。计算节点参考了 CPU 利用率、网络带宽、内存占有率这几个参数，并结合了计算节点就绪队列的长度构建向量空间。实验表明，在相应权值为 0.4, 0.2, 0.1, 0.3 时，实验效果较好。

抓取任务向量和计算节点向量的构建如下：

以抓取百度首页的任务为例，可以构建抓取任务向量：$\boldsymbol{A}$={5,4,26,0}，其中，172 表示对页面做信息摘要需要 5 ms；4 表示文件大小约为 4 KB；26 表示访问页面的网络延时为 26 ms。

以配置为赛扬 2.1 GHz、512 MB 内存、60 GB 硬盘、10 Mb/s 带宽的计算节点在 CPU 占有率 20%、内存占有率 40% 的情况为例，构建计算节点向量：$\boldsymbol{C}$={16.8,30.7,10,5}，其中，16.8 由 2.1×10×(1-CPU 占有率)计算得到；30.7 由 512/10×(1-内存占有率)计算得到；10 表示 10 Mb/s 带宽，5 表示该节点当前就绪任务队列长度为 5。

5.1 通过余弦向量法实现计算节点间负载平衡的验证

图 1 给出了抓取 $\boldsymbol{A}_{\text{百度首页}}$ 任务在计算节点不受其他因素干扰的情况下，就绪队列长度从 1 增加到 40 对余弦向量值计算结果的影响。

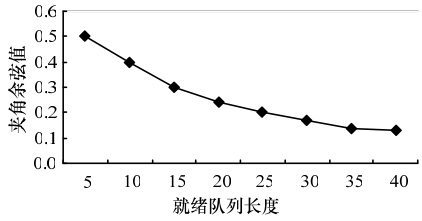

图 1　就绪队列长度对夹角余弦值的影响

从图中可以发现，计算节点在不受其他因素干扰的情况下，随着就绪队列变长，夹角余弦值会减小。根据定义 5 可知，夹角余弦值减小意味着此计算节点被分配到该抓取任务的机会将减少，系统实现了计算节点之间的负载平衡。

5.2 余弦向量法、轮循法和随机法的比较

轮循法和随机法都属于经典负载平衡算法，其使用的是静态任务分配策略，优点是实现方便且运行开销小，缺点是不能根据运行状况调整任务的分配。

实验中使用余弦向量法、轮循法、随机法对第 1 小时到第 6 小时的运行效率(图 2)和整个更新任务运行完成所花费的时间(图 3)两方面进行了比较。

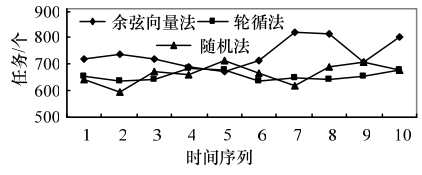

图 2　单位时间完成任务数对比

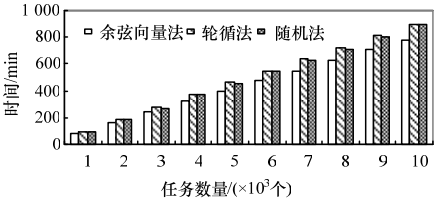

图 3　完成 1 万个抓掘任务所需时间的对比

图 2 反映了 3 种分配策略在单位时间内处理任务的数量的差异。可以看到，在采用余弦向量法分配任务时，系统平均抓掘性能比采用轮循法和随机法分配任务时的性能高 10%~15%。虽然有时可能发生余弦向量法的运行效率和轮循法、随机法运行效率相近的情况，这是因为实验中分配任务时只考虑了计算节点就绪队列的长度，没有考虑就绪队列中

每个抓取任务运行所需时间的差异。

图 3 反映了 3 种分配策略在完成全部任务所需的总时间上的差异。可以看到，随着抓掘任务数量的增加，系统使用余弦向量法分配抓取任务时的运行效率明显高于轮循法和随机法率。系统的运行效率除了受抓取任务分布在不同节点上所造成的抓掘效率差异的影响外，还受到计算节点负载差异的影响。由于向量法和轮循法不能动态调节各计算节点就绪队列任务负载的不平衡，因此影响了整体的运行效率。

5.3 单机抓取页面数和余弦向量法并行抓取页面数对比

表 1 示出了从第 1 小时到第 6 小时的单机抓取页面数和余弦向量法并行抓取页面数的对比。笔者分别使用 2 个抓取节点、3 个抓取节点、4 个抓取节点进行了实验。可以看到，采用余弦向量法的并行抓掘系统其单个节点平均抓取页面数与单机抓取页面数的比值分别为 0.707, 0.704, 0.702。

表 1 单机抓取和并行抓取的对比

时间序列	单机抓掘	并行抓掘		
		2 台抓掘节点	3 台抓掘节点	4 台抓掘节点
1	341	483	722	959
2	351	495	739	982
3	339	483	718	953
4	329	461	689	915
5	312	438	652	868
6	331	476	712	947

采用余弦向量法的并行抓掘系统与单机抓取页面效率存在差异的原因在于：系统对每个计算节点实时信息的检测和采集存在开销；此外，计算节点与 Master 间的通信也存在开销。同时可以看到，计算节点从 2 个增加到 4 个对并行系统单个计算节点的运行效率并未产生较大影响。这表明系统具有较好的扩展能力，可以进一步扩展计算节点而不会造成性能大幅度下降。

6 结束语

目前，针对搜索引擎的并行抓掘系统正越来越受到关注。并行抓掘系统的表现对搜索引擎也有很大的影响。本文所提出的并行抓掘任务分配策略针对并行抓掘的特点，较好地兼顾了分配适应性和负载平衡性。

参考文献

[1] Sung Jin Kim, Sang Ho Lee. An Empirical Study on the Change of Web Pages[C]//Proc. of Conf. on Web Technologies Research and Development. Heidelberg, Germany: Springer, 2005: 632-642.

[2] 王晓宇, 周傲英. 万维网的链接结构分析及其应用综述[J]. 软件学报, 2003, 14(10): 1768-1780.

[3] Cho J, Garcia-molina H. Parallel Crawlers[C]//Proceedings of the 11th International World Wide Web Conference. [S. l.]: IEEE Press, 2002.

[4] Cho Junghoo, Garcia-molina H. The Evolution of the Web and Implications for an Incremental Crawler[C]//Proceedings of VLDB'0. Seou, Korea: [s. n.], 2000.

[5] Salton G, Buckley C. Term-weighting Approaches in Automatic Retrieval[J]. Information Processing and Management, 1998, 24(5): 513-523.

[6] De Bra D, Post R D. Searching for Arbitrary Information in the WWW: The Fish Search for Mosaic[C]//Proceedings of the 2nd World Wide Web Conference. Chicaco, IL, USA: [s. n.], 1994.

[7] Fetterly D, Manasse M, Najork M, et al. A large-scale Study of the Evolution of Web Pages[C]//Proceedings of the 12th World Wide Web Conference. New York, NY, USA: ACM Press, 2003.

编辑 张正兴

（上接第 60 页）

m_{n+2}，…，m_{2n} 后得到 $L_{n\times 2n}(K)$，对应的概念比较次数为 $\int_0^n n(n+x)\delta \mathrm{d}x$；第 2 阶段为在 $L_{n\times 2n}(K)$ 插入对象 o_{n+1}，o_{n+2}，…，o_{2n} 后得到 $L_{2n\times 2n}(K)$，对应的概念比较次数为 $\int_0^n 2n(n+x)\delta \mathrm{d}x$；总的时间复杂度为 $O(9/2n^3\delta)$。

图 1(b)采用的基于对象和属性交叉渐进概念格生成算法包含 2 个部分。第 1 部分为在 $L_{n\times n}(K)$，$L_{(n+1)\times(n+1)}(K)$，…，$L_{(2n-1)\times(2n-1)}(K)$ 插入对象 o_{n+1}，o_{n+2}，…，o_{2n}，对应的概念比较次数为 $\int_0^n (n+x)(n+x)\delta \mathrm{d}x$。第 2 部分为在 $L_{(n+1)\times n}(K)$，$L_{(n+2)\times(n+1)}(K)$，…，$L_{2n\times(2n-1)}(K)$ 插入属性 m_{n+1}，m_{n+2}，…，m_{2n}，对应的概念比较次数为 $\int_0^n (n+x+1)(n+x)\delta \mathrm{d}x$；总的时间复杂度为 $O(14/3n^3\delta+3/2n^2\delta)$。

在交叉渐进式和非交叉渐进式概念格生成算法的时间复杂度比较中，忽略了低阶因素的影响，交叉渐进式生成算法在时间复杂度上降低了((14/3)/(9/2)-1)，即 3.7%。为此，可以结合基于索引树的概念格快速生成算法[6]，利用辅助索引树缩小新生格节点的父节点和子节点的搜索范围，确定已有的概念节点和新增对象或属性之间的关系，快速地调整概念之间的相互关系，提高调整操作元运算的效率来弥补上述不足。

5 结束语

本文提出了一种基于对象和属性交叉渐进式概念格生成算法。该算法模拟人类基于对象和属性交叉渐进式生成概念的过程，从空概念格开始，逐个将形式背景中的对象和属性交叉插入到概念格中，再调整概念之间的相互关系，实现对概念格的渐进式构造。基于对象和属性交叉渐进式概念格生成算法的重要贡献在于解决了以往非交叉渐进式生成算法对于属性和对象交叉渐增更新需要重新构造概念格的问题。该算法的时间复杂度为 $O(14/3n^3\delta)$，相比非交叉渐进式概念格生成算法的时间复杂度 $O(9/2n^3\delta)$，效率减低了 3.7%，为此本文借助辅助索引树方法提高调整操作元运算的效率来弥补。

参考文献

[1] Ganter B, Wille R. Formal Concept Analysis: Mathematical Foundations[M]. Berlin: Springer-Verlag, 1999.

[2] 王 虹, 张文修. 形式概念分析与粗糙集的比较研究[J]. 计算机工程, 2006, 32(8): 42-44.

[3] Xie Z, Liu Z. Concept Lattice and Association Rule Discovery[J]. Journal of Computer Research and Development, 2000, 37(12): 1415-1421.

[4] Nourine L, Raynaud O. A Fast Algorithm for Building Lattices[J]. Information Processing Letters, 1999, 71(5): 199-204.

[5] Godin R, Missaoui R, Alaoui H. Incremental Concept Formation Algorithms Based on Galois(Concept) Lattices[J]. Computational Intelligence, 1995, 11(2): 246-267.

[6] 谢志鹏, 刘宗田. 概念格的快速渐进式构造算法[J]. 计算机学报, 2005, 25(5): 490-496.

编辑 张正兴