基于LSH的中文文本快速检索

蔡 衡[1] 李舟军[1] 孙 健[2] 李 洋[2]
(北京航空航天大学计算机学院 北京 100083)[1]
(新浪网技术(中国)有限公司研发中心-搜索-新技术部 北京 100191)[2]

摘 要 目前,高维数据的快速检索问题已经受到越来越多的关注。当向量空间的维度高于10时,R-tree,Kd-tree,SR-tree的检索效率反而不如线性检索,而位置敏感的哈希(Locality Sensitive Hashing,缩写为LSH)算法成功地解决了高维近邻数据的快速检索问题,因而受到国内外学术界的高度关注。首先介绍了LSH算法的基本原理和方法,然后使用多重探测的方法对二进制向量的LSH算法做了进一步改进。最后实现了这两种LSH算法,并通过详细的实验验证表明:在改进后的算法中,通过增加偏移量可以提高检索的召回率,而在不提高时间复杂度的情况下则可降低空间复杂度。

关键词 高维数据,相似性检索,位置敏感的哈希,近邻,多重探测

Fast Chinese Text Search Based on LSH

CAI Heng[1] LI Zhou-Jun[1] SUN Jian[2] LI Yang[2]
(School of Computer, Beihang University, Beijing 100083, China)[1]
(Sina Corporation R&D Center-Product R&D-Search Tech., Beijing 100191, China)[2]

Abstract The query of High dimension data attracts more and more attention. When dimension of a space vector is higher than 10, R-tree, Kd-tree, SR-tree and Quadtrees perform worse than linear query. However, Locality Sensitive hashing (LSH) algorithm successfully deals with this problem. Nowadays LSH is playing a more and more important role in high dimension query. In the paper, the basic algorithm and principle of LSH were introduced firstly, then binary vector LSH Search Algorithm was improved by means of the multi-probe. Finally, we implemented the two kinds of LSH algorithms. The experience we have designed verified that the revised algorithm has better performance than the original one in two aspects. On the one hand, as the increment of setover, the proportion of retrial recall enlarges. On the other hand, the complexity of space decreases without the change of time complexity.

Keywords High dimension data, Similarity search, Locality sensitive hashing, Near neighbor, Multi-probe

1 引言

相似性快速检索在各种领域特别是在视频、音频、图像、文本等含丰富特征信息的快速检索领域中(例如数据库、数据挖掘和搜索引擎、自然语言处理)的应用已经变得越来越重要。丰富的特征信息一般都代表了高维向量,所以相似性检索是通过高维数据的K近邻或者近似近邻来实现[6]。

一个理想的相似性检索模式一般需要满足以下4条性质:

- 准确性:检索操作返回的结果和线性查找的结果接近。
- 空间效率:检索只需占用少量的内存空间。理想状态下,空间复杂度随数据集呈线性增长,但不会远大于数据集的大小。
- 时间效率:检索的时间复杂度最好是O(1)或者O(log$N$)的时间,其中N是被检索数据的条目数。
- 高维度:检索模式应该能很好地处理高维数据。

此外,检索模式应能快速地构造索引数据结构,并且可以完成插入、删除等操作。

现有的很多检索算法并不能同时满足以上的所有性质。以前主要采用基于空间划分的算法——tree算法,例如:R-tree[12]、Kd-tree[13]、SR-tree。这些算法返回的结果都是精确的,然而它们在高维数据集上时间效率并不高。文献[11]的试验指出在维度高于10之后,基于空间划分的算法的时间复杂度反而不如线性查找。

1998年,MIT的P. Indy和R. Motwani提出了LSH算法的理论基础[3]。1999年P. Indy和R. Motwani使用哈希的办法解决高维数据的快速检索问题[14],这也是Basic LSH算法的雏形。2004年,P. Indy提出了LSH算法在欧几里德空间(2范数)下的具体解决办法[1]。同年,在自然语言处理领域中,Deepak Ravichandran使用二进制向量和快速检索算法改进了Basic LSH算法[5],并将其应用到大规模的名词聚类中,但改进后的算法时间效率并不理想。2005年,Mayank

到稿日期:2008-09-18 返修日期:2009-04-22 本文受国家自然科学基金项目(60573057,90718017)资助。
蔡 衡(1982—),男,硕士研究生,研究方向为文本挖掘,E-mail:ch198299@163.com;李舟军(1963—),男,教授,博士生导师,研究方向为高可信软件技术、安全协议的形式化验证、数据挖掘与文本挖掘。

Bawa, Tyson Condie 和 Prasanna Ganesan 提出了 LSH Forest 算法[15]，该算法使用树形结构代替哈希表，具有自我校正参数的能力。2006 年，R. Panigrahy[4] 用产生近邻查询点的方法提高 LSH 空间效率，但却降低了算法的空间效率。2007 年，William Josephson 和 Zhe Wang[9] 使用多重探测的方法改进了欧几里德空间(2 范数)下的 LSH 算法，同时提高了算法的时间效率和空间效率。

理论研究[9]表明，未被检索到的近邻点一般分布在待检索点哈希值的附近，传统的单次检索 LSH 算法并不能召回哈希表中所有的近邻点。为了使 LSH 算法更好地解决文本挖掘中文本近邻检索问题，本文将多重探测的原理应用于二进制向量的 LSH 算法[5]中。改进后的算法在不提高时间复杂度的情况下降低了空间复杂度。

2 问题定义

使用 l_p^d 代表空间 R^d 上的 l_P 范数。对于点 $v\in R^d$，$|v|$ 定义为向量 v 的长度。$M=(X,d)$ 为任意一个向量空间，且 $v\in X$，半径为 r 且圆心为 v 的球有如下定义：

$B(v,r)=\{q\in X|d(v,q)\leqslant r\}$。

给定一个包括 n 个向量的数据集合 Z，一个松弛变量 $\varepsilon>0$，一个检索点 q，则检索点 q 的 $(1+\varepsilon)$-近似近邻 $a\in Z$ 满足：对于任意 $b\in Z$，$|a-q|\leqslant(1+\varepsilon)|b-q|$。

目标是建立一个数据结构 S，据此可得到检索点 q 的所有 $(1+\varepsilon)$ 近似近邻点。

3 LSH 算法原理及其改进

一种解决 $(1+\varepsilon)$ 近似近邻问题的重要方法就是 LSH 算法。为此，先给出如下定义。

定义 1 设距离 $r_2>r_1>0$，p_1，p_2 为 LSH 算法返回概率，且 $1>p_1>p_2>0$，点 $q,p\in R^d$，$H=\{h|h:R^d\to R\}$ 称为 (r_1,r_2,p_1,p_2) 敏感的：

- 如果点 $p\in B(q,r_1)$，则 $P[h(q)=h(p)]\geqslant p_1$。
- 如果点 $p\notin B(q,r_2)$，则 $P[h(q)=h(p)]\leqslant p_2$。

下面将从 LSH 算法的基础——P 稳定分布开始介绍，接着将介绍 LSH 算法及其改进。

3.1 P 稳定分布

一个分布 D 被叫做 P ($P=1$ 或 $P=2$) 稳定的，如果 x，$x_1,x_2,\cdots,x_n$ 是 $n+1$ 个独立同分布且服从分布 D 的随机变量，且对于任意实数 $r_1,r_2,\cdots,r_n$ 均满足：$x(\sum|r_i|^p)^{\frac{1}{p}}$ 和 $\sum r_ix_i$ 具有相同的分布[1]。

P 稳定分布保证了距离较近的点有更高的可能性可以和哈希在一起。设 q 和 p(分别用向量 v_1 和 v_2 表示)为 R^d 空间上两个点，X 为 d 个服从 P 稳定分布 D 的随机变量组成的向量，x 为一个服从 P 稳定分布 D 的随机变量，一个松弛变量 $\varepsilon>0$，则根据 P 稳定分布的定义，有：

$$P[-\varepsilon<v_1X-v_2X<\varepsilon]=P[-\varepsilon<(v_1-v_2)X<\varepsilon]=$$

$$P[-\varepsilon<|v_1-v_2|x<\varepsilon]=P[-\frac{\varepsilon}{|v_1-v_2|}<x<\frac{\varepsilon}{|v_1-v_2|}]$$

从上面的推导结果可以看出，两个点之间的距离越接近，v_1X 和 v_2X 的值接近的概率越大；两个点之间的距离越远，v_1X 和 v_2X 的值接近的概率越小。

对于 l_P 范数而言，如果 $P=1$，那么 D 就是柯西分布；如果 $P=2$，那么 D 就是标准正态分布。

文献[7]指出，如果 $|v|$ 是向量空间下的余弦距离，则 D 就是球对称随机分布。

3.2 方程 $h(v)$

方程 $h(v)$ 利用了 P 稳定分布位置敏感的特性，把近邻的点 v(向量)影射到集合 S 内的同一个元素上。Basic LSH 是将点 v 影射到整数集合 Z 上，而二进制向量的 LSH 算法则将点 v 影射到集合 $\{0,1\}$ 上。若两个点之间的距离越接近，则 $h(v)$ 相等的概率越大。

对二进制向量的 LSH 算法，方程 $h(v)$ 定义如下：

$$h(v)=\begin{cases}1 & av>0\\ 0 & av\leqslant 0\end{cases}$$

其中 a 是 d 个服从 P 稳定分布的抽样组成的向量，$v\in R^d$。

假设 c 是两点(用 v_1 和 v_2 表示)之间的距离，r_0 为向量 v_1 和 v_2 归一化后的长度，方程 $f(x)$ 是 P 稳定分布的密度函数，那么 v_1 和 v_2 在方程 $h(v)$ 下，哈希值相等的概率为：

$$p(c)=P[h(v_1)=h(v_2)]=\int_0^{+\infty}\frac{1}{c}f(\frac{t}{c})(1-\frac{1}{r_0}\int_{-\infty}^{t}f(x)\mathrm{d}x)\mathrm{d}t$$

3.3 方程 $g(v)$

方程 $g(v)$ 的形式如下：

$g(v)=(h_1(v),h_2(v),\cdots,h_k(v))$

方程 $g(v)$ 将 k 个方程 $h_t(v)(t=1,\cdots,k)$ 组成一个长度为 k 的向量，并将所有哈希值 $g(v)$ 与检索点 q 的哈希值 $g(q)$ 相等的点 v 作为返回点。为了保证距离接近的点返回的概率增大，同时距离较远的点返回的概率减小，进一步引入 l 个方程 $g_i(v)(i=1,\cdots,l)$，并将 l 个方程 $g_i(v)$ 的返回点集合的并集作为 LSH 算法的返回结果。

LSH 算法把离检索点距离为 c 的点返回的概率 $LSH(c)$ 为：

$LSH(c)=l-(1-p(c)^k)^l$

当 $p=2$ 时，使用 Matlab 绘图，如图 1 所示。

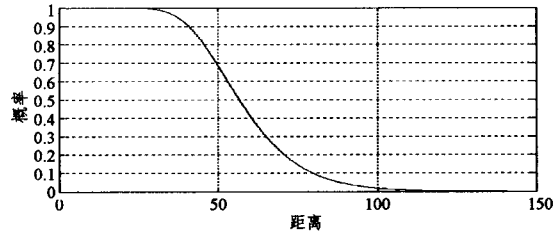

图 1 近邻点返回概率图像

从图 1 可以看出，距待检索点越近，返回概率越大；距待检索点越远，返回概率越小。该特性可保证 LSH 算法能返回较好的结果。

3.4 LSH 算法

根据 LSH 算法的基本原理，可以知道 LSH 算法可分为 2 部分：LSH 模型建立和 LSH 结果检索。LSH 模型建立是生成 l 个表(即 l 个方程 $g_i(v)$)的过程，每个表存放了哈希值和对应的数据信息。LSH 结果检索是在模型中检索近邻数据点的过程。

下面给出 LSH 模型建立和 LSH 结果检索的算法描述，分别见图 2 和图 3。

Input a set of points P, l(number of hash tables),

Output Hash tables T_i, i = 1……l

Foreach i = 1……l

Initialize LSH Algorithm T_i by generating a random hash function $g_i(.)$

Foreach i = 1……l

Foreach j = 1……n

Store point p_j on bucket $g_i(p_j)$ of hash table T_i

图 2 建立 LSH 模型

在本文中，将建立后的模型存为一个外存文件，在启动检索程序的时候，只需载入模型文件即可。这将 LSH 模型建立和 LSH 结果检索分割成两个独立的过程，并减少了重复计算。

Input a query point q,

m (number of approximate. nearest neighbors)

Access To hash tables T_i, i=1,…,l generated by the preprocessing algorithm

Output m (or less) approximate nearest neighbors

S←∅

Foreach i=1,…,l

S→S ∪{points found in $g_i(q)$ bucket of table T_i}. Return the m nearest neighbors of q found in set S

图 3 LSH 结果检索

3.5 二进制向量 LSH 检索算法改进

William Josephson 和 Zhe Wang 在文献[9]中指出，LSH 模型中，近邻点在每个表中的哈希值和检索点的哈希值要么相等，要么很接近(距离为 1 和 2)。其关系如图 4 所示[9]。

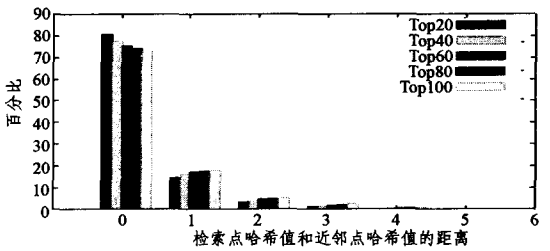

图 4 近邻点在检索点附近的分布

根据这一现象，多重探测的实质是巧妙地使用多个探测向量，在每个表中最近邻点出现概率高的地方进行若干次检索。多重探测在每个表中不仅检索与待检索点哈希值相等的近邻点，而且也会检索与其距离为 1 和 2 的近邻点。为了找到这些出现概率较大的近邻点，William Josephson 和 Zhe Wang[9]针对文献[1]中的算法提出了 Step-Wise Probing 和 Query-Directed Probing 两种检索方法。但这些检索算法不适用于二进制向量，本文根据多重检测的思想，对二进制向量的 LSH 检索算法进行改进。

若两个点哈希值(二进制向量)的海明距离越小，则这两个点距离接近的概率越大。图 5 为检索点 q 的最近邻点概率分布。由图 5 可知，对于检索点 q，aq 的绝对值越大，则近邻点和检索点哈希值相等的概率越大；如果 aq 的绝对值越小，则近邻点和检索点哈希值相等的概率值就越小。

这里，由于哈希值是一个长度为 k 的二进制向量，则距离为 1 的向量有 k 个，距离为 2 的向量有 $k*(k-1)/2$ 个。为了减小无用的检索，须在原点两侧设置长度为 *Setover* 的偏移长度。若 av 的绝对值大于或等于 *Setover*，就认为近邻点和检索点的哈希值一定相等；若 av 的绝对值小于 *Setover*，就认为近邻点和检索点的哈希值可能不等。显然，*Setover* 值为 0 时，为二进制向量的 LSH 算法。

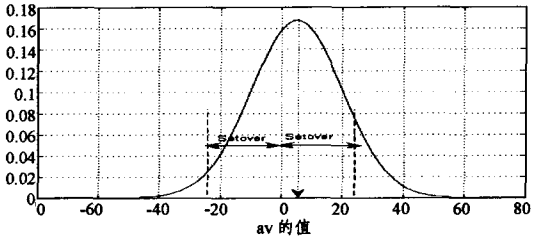

图 5 检索点最近邻点的 av 值分布

若长度为 k 的二进制向量中有 $r(r<k)$ 个位置 av 的绝对值小于 *Setover*，则距离为 1 的待检索向量有 r 个，距离为 2 的待检索向量有 $r*(r-1)/2$ 个，待检索向量集合 Q 共有 $1+r+r*(r-1)/2$ 个元素。对每个方程 $g_i(v)$，将所有哈希值 $g_i(v)$ 与 Q 中所有的待检索向量相等的点 v 作为返回点。最后，将 l 个方程 $g_i(v)$ 返回点集合的并集作为 LSH-pro 的结果。图 6 描述了 LSH 改进算法 LSH-pro 的流程。

Input a query point q,

m (number of approximate. nearest neighbors)

Access To hash tables T_i, i=1,…,l generated by the preprocessing algorithm

Output m (or less) approximate nearest neighbors

S←∅

Foreach i=1,…,l

Q←{$g_i(q)$}∪{vectors which are near $g_i(q)$,according to their |av| and Setover}

Foreach v in Q

S→S ∪{points found in v bucket of table T_i}. Return the m nearest neighbors of q found in set S

图 6 LSH 改进算法 LSH-pro

4 实验

4.1 评价标准

所有相似性检索系统都可从以下 3 个方面进行度量：检索质量、检索的时间复杂度和检索的空间复杂度。在理想状态下，一个检索系统应该在最短的时间用最少的空间做出高质量的检索。

检索质量可以通过召回率和错误率来描述。给定一个检索点 q，令 $I(q)$ 是理想结果(例如：距离 q 最近的 K 个点)，$A(q)$ 是 LSH 算法返回的结果，那么

$$召回率=\frac{|A(q)\cap I(q)|}{|I(q)|}$$

在理想状态下，召回率是 1.0，这意味着最近的 k 个点全部都返回了。

为比较算法的优劣，可用错误率来描述近似近邻的检索效率。定义如下：

$$错误率=\frac{1}{|Q|K}\sum_{q\in Q}\sum_{k=1}^{K}\frac{d_{LSH_k}}{d_k^*}$$

其中，Q 是实验中检索点的集合，K 为错误率和召回率的衡量标准(例如：K 为 30 表示选取最邻近的 30 个点作为评价集合)，d_{LSH_k} 是通过 LSH 算法找到的第 $k(0<k\leqslant K)$ 个近邻点到查询点的距离，且 d_k^* 是真实的第 k 个近邻到查询点的距

离。换句话说，这个公式代表了 LSH 算法找到的 k 近邻的距离和真实 k 近邻距离对比的结果。在理想状态下，错误率为 1.0。若返回的结果越差，错误率就越大。

检索时间复杂度可以用检索时间来衡量。空间复杂度可以用所有哈希表的个数来表示，或者用内存总的使用情况来衡量。

4.2 实验细节

本文采用了真实的中文文本作为试验数据，选择了 2000 个特征，并将 1900 篇文本投影到 2000 维的高维空间中（每个向量都进行了归一化处理），使用 2 范数作为距离度量方式，高斯分布作为随机分布。

本文已经实现了两种 LSH 算法：二进制向量 LSH 算法和 LSH 改进算法。第一种算法和文献[5]中的算法一致，使用了海明距离；第二种算法在坐标原点附近增加了 *Setover* 作为阈值。通过改变 *Setover* 的大小，可以控制待检索向量的数量，并根据待检索向量得到最后的返回结果。

为了说明增加偏移量可以提高检索的召回率（如表 1 所列），这里选取 l 为 25，k 为 15。

实验运行的机器 CPU 是 dual-processor Intel(R) Pentium(R) D CPU 2.80GH，内存大小为 1GB，硬盘是 SATA 7200 RPM，操作系统是 SUSE 10.1。

4.3 实验结果

为了说明 *Setover* 对检索结果的影响，在 l 和 k 数值不变的情况下，改变 *Setover* 的值，观察算法召回率的变化（l 和 k 不变，算法的空间复杂度不会发生变化）。从表 1 可以看出，随着 *Setover* 变大，召回率随之上升。从理论上讲，由于 *Setover* 变大，增加了待检索向量的个数，检索程序可以从近邻点出现概率较大的多个地方进行检索，因此结果的召回率得到了提升。但是，随着 *Setover* 的增大，检索时间也会增加。一方面，检索次数的增加会提高算法的时间复杂度；另外一方面，召回点个数的增加也会影响算法的时间复杂度。

表 1 试验 1（控制偏移量对召回率的影响）

Setover	T5	T10	T20	T30	T40
0	0.7044	0.569	0.4362	0.3969	0.3220
10	0.8044	0.6712	0.5382	0.4686	0.4201
20	0.878	0.7542	0.6266	0.5593	0.5114
30	0.9216	0.8196	0.7045	0.6392	0.5946
40	0.9484	0.8598	0.7587	0.7012	0.6592
50	0.9736	0.8934	0.8053	0.7532	0.7183
60	0.9872	0.9162	0.8401	0.7935	0.7586
70	0.9992	0.9344	0.8649	0.8237	0.7940
80	1.000	0.948	0.8865	0.8517	0.8245
90	1.000	0.958	0.9035	0.8731	0.8477
100	1.000	0.966	0.9157	0.8894	0.8657

为了比较二进制向量 LSH 算法和 LSH 改进算法的时间复杂度和空间复杂度，这里特别选择了两组不同的输入参数，以保证这两种算法召回率基本相同。在第二个实验当中（如表 2 所列），选取 l 为 75，k 为 9 作为二进制向量 LSH 算法的输入参数；选取 l 为 25，k 为 14，*Setover* 为 180 作为 LSH 改进算法的输入。从表 2 可以看出，在召回率相同的情况下，LSH 改进算法的空间复杂度有所下降。由于改进算法可以在一张表中多个不同的位置检索近邻点，用更少的表就可以得到相同的结果，因此改进算法的空间复杂度有所下降。

表 2 试验 2 结果

	LSH	LSH-pro
时间	0.10	0.10
内存	1.2	0.8
Top20 召回率	0.81151	0.82459
Top20 错误率	1.01160	1.01074
Top30 召回率	0.75613	0.77121
Top30 错误率	1.01597	1.01454
Top40 召回率	0.71400	0.72777
Top40 错误率	1.0202	1.0179

结束语 在本文中，使用多重探测的方法改进了二进制向量的 LSH[5] 算法，该方法降低了算法的空间复杂度。不仅如此，同文献[5]相比，快速检索算法实际上是用时间换检索质量，而多重探测的方法在保证检索精度的同时并没有降低算法的时间效率。

另外，本文中的改进算法可以使用 LSH Forest[15] 来代替表结构，该结构具有自我校正参数的功能。

目前，LSH 算法的理论正日臻完善，并逐步应用于各种领域中。我们认为 LSH 算法的理论研究和应用研究潜力巨大，将为解决各种应用领域的相似性快速检索问题提供一个高效而恰当的方法和手段。

参 考 文 献

[1] IndykP, DatarM, ImmorlicaN. Locality-SensitiveHashingScheme Based on p-Stable[C]// Annual Symposium on Computational Geometry. 2004

[2] Arya S, Mount D. Ann: Library for approximate nearest neighbor search[OL]. http://www.cs.umd.edu/~mount/ANN/

[3] Indyk P, Motwani R. Approximate nearest neighbors: Towards removing the curse of dimensionality[C]// Jeffrey V, ed. Proc. of the 30th Annual ACM Symp. on Theory of Computing. New York: ACM Press, 1998: 604-613

[4] Panigrahy R. Entropy based nearest neighbor searchin high dimensions[C]// Proc. of ACM-SIAMSymposium on Discrete Algorithms(SODA). 2006

[5] Ravichandran D, Pantel P, Hovy E. Randomized Algorithms and NLP: Using Locality Sensitive Hash Function for High Speed Noun Clustering[M]. Information Sciences Institute University of Southern California, 2004

[6] Cai Rui, Zhang Chao, Zhang Lei, et al. Scalable Music Recommendation by Search [C] // International Multimedia Conference. 2007

[7] Charikar M S. Similarity Estimation Techniques from Rounding Algorithms[C]// Annual ACM Symposium on Theory of Computing. 2002

[8] Qin L, Josephson W, Wang Zhe, et al. A TimeSpace Efficient Locality Sensitive Hashing Method for Similarity Search in High Dimensions

[9] Qin L, Josephson W, Wang Zhe, et al. MultiProbe LSH: Efficient Indexing for High Dimensional Similarity Search [C] // Very large data bases archive proceedings of the 33rd international conference on very large data bases. 2007

[10] Motwani R, Naor A, Panigrahy R. Lower bounds on locality sensitive hashing[C]// Proc. of the 22nd ACM Symposium on Computational Geometry (SCG). 2006: 154-157

（下转第 230 页）

由算法 2 的第 1)步得到，计算 $|U/(C\cup D)|$ 的 SQL 操作语言如下：

SELECT DISTINCT COUNT(*) FROM DECISION;

其中 DECISION 就表示决策表 1，它的条件属性和决策属性分别为：$\{a,b,c,d,e\}$ 和 $\{D\}$。

故 $|U/(C\cup D)|=8$。同理可得：$|U/(C-\{a\}\cup D)|=8$；$|U/(C-\{b\}\cup D)|=8$；$|U/(C-\{c\}\cup D)|=8$；$|U/(C-\{d\}\cup D)|=8$；$|U/(C-\{e\}\cup D)|=8$。故 $Core(C)=\emptyset$；$reduc(C)=\emptyset$。

由算法的第 2)步得：$B=\{a,b,c,d,e\}$。

由算法 2 的第 3)步得：

$$merit(a)=\left\{1-\frac{|U/((reduc(C))\cup D)|}{|U/((reduc(C)\cup\{a\})\cup D)|}\right\}$$

$$=1-\frac{|U/D|}{|U/(\{a\}\cup D)|}=1-\frac{2}{5}=\frac{3}{5}$$

同理可得：$merit(b)=3/5$；$merit(c)=2/4$；$merit(d)=4/6$；$merit(e)=4/6$。故值最大的有两个，即 $\{d,e\}$。选择 $\{d\}$。

由算法 2 的第 4)步得：

$reduc(C)=\{d\}$，算法转第 2)步，得到 $B=\{a,b,c,e\}$。

由算法 2 的第 3)步得：

$$merit(a)=\left\{1-\frac{|U/((reduc(C))\cup D)|}{|U/((reduc(C)\cup\{a\})\cup D)|}\right\}$$

$$=1-\frac{|U/(\{d\}\cup D)|}{|U/(\{d,a\}\cup D)|}=1-\frac{6}{8}=\frac{2}{8}$$

同理可得：$merit(b)=2/8$；$merit(c)=2/8$；$merit(e)=2/8$。故值最大的有 4 个，即 $\{a,b,c,e\}$。选择 $\{a\}$。

由算法 2 的第 4)步得：

$reduc(C)=\{d,a\}$，算法转第 2)步，得到 $B=\{b,c,e\}$。

由算法 2 的第 3)步得：

$$merit(a)=\left\{1-\frac{|U/((reduc(C))\cup D)|}{|U/((reduc(C)\cup\{a\})\cup D)|}\right\}$$

$$=1-\frac{|U/(\{d,a\}\cup D)|}{|U/(\{d,a,b\}\cup D)|}=1-\frac{8}{8}=1$$

同理可得：$merit(c)=1$；$merit(e)=1$。故值最大的有 3 个，即 $\{b,c,e\}$。但它们的值均为 1，故算法终止，输出属性约简 $reduc(C)=\{d,a\}$。

结束语 由于由数据库模型的属性约简定义而设计的属性约简算法结合了数据库技术，故该算法的效率较好，其中经常使用的数据库运算有：投影、选择和计数。使用数据库中相关的运算是一种较好的方法。基于系统熵的属性约简算法的时间复杂度为 $O(|C|^3|U|\log|U|)$。这一时间复杂度对处理大数据集时是不理想的。由于基于系统熵的属性约简与基于数据库模型的属性约简是相互等价的，故可以用设计基于数据库模型的属性约简算法的方法去设计基于系统熵的属性约简算法，即可以用数据库的相关操作设计基于系统熵的属性约简算法，其时间复杂度为 $O(|C|^2|U|)$。这样就可以将高效数据库的相关运算应用到属性约简算法中，从而提高算法的运行效率，使其能够处理大数据集。故这样的算法具有很好的可扩展性。

参 考 文 献

[1] Pawlak Z, Grzymala-Busse J, Slowinski R. Rough sets [J]. Communications of the ACM, 1995, 8(1): 89-95

[2] Pawlak Z. Rough set theory and its application to data analysis [J]. Cybernetics and Systems, 1998, 9(5): 661-668

[3] Xu Z Y, Yang B R, Song W. Comparative study of different attribute reduction based on decision table[J]. Chinese Journal of Electronics, 2006, 15(4A): 953-956

[4] 邓大勇，黄厚宽，李向军. 不一致决策系统中属性约简的比较[J]. 电子学报，2007，35(2)：252-255

[5] 徐章艳，杨炳儒，宋威，等. 几种不同属性约简的比较[J]. 小型微型计算机系统，2008，29(5)：848-853

[6] Hu X H, Lin T Y, Han J C. A new rough sets model based on database systems [J]. Fundamenta informaticae, 2004, 59(1): 135-152

[7] Skowron R, Rauszer C. The discernibilinity matrices and functions in information systems[M]. R. Slowincki, ed. Intelligent decision support-handbook of applications and advances of the rough sets theory. Dordrecht: Kluwer, 1992: 316-362

[8] Hu X H, Cercone N. Learning in relational databases: a rough set approach [J]. International Journal of computational intelligence, 1995, 11(3): 323-338

[9] Wang G Y. Rough reduction in algebra view and information view [J]. International Journal of Intelligent System, 2003, 18(5): 679-688

[10] Zhao J, Wu Z F, Li H. System entropy and its application in feature selection [J]. The Journal of China Universities of Posts and Telecommunications, 2004, 11(1): 100-105

(上接第 204 页)

[11] Weber R, Schek H, Blott S. A quantitative analysis and performance study for similarity search methods in high dimensional spaces[C]// Proc. of the 24th Intl. Conf. on Very Large Data Bases (VLDB). 1998: 194-205

[12] Guttman A. R-trees: A dynamic index structure for spatial searching[C]// Proc. of ACM Conf. on Management of Data (SIGMOD). 1984: 47-57

[13] Bentley J L. K-D trees for semi-dynamic point sets[C]// Proc. of the 6th ACM Symposium on Computational Geometry (SCG). 1990: 187-197

[14] Gionis A, Indyk P, Motwani R. Similarity search in high dimensions via hashing[C]// Proceedings of the 25th International Conference on Very Large Data Bases (VLDB). 1999

[15] Bawa M, Condie T, Ganesan P. LSH Forest: SelfTuning[C]// Indexes for Similarity Search International World Wide Web Conference Proceedings of the 14th International Conference on World Wide Web. 2005

[16] Stein B. Principles of hash-based text retrieval [C] // Annual ACM Conference on Research and Development in Information Retrieval Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. 2007

[17] Athitsos V, Potamias M, Papapetrou P, et al. Nearest Neighbor Retrieval Using Distance-Based Hashing[C] // Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on. 2008