Key-Tree:一种增强目录索引接口有限查询能力的方法*)

贺 凡 杨晓春 于 戈 李 琳 石 磊
(东北大学信息科学与工程学院 沈阳110004)

摘 要 限于目录索引接口的查询能力,为了优化采用这种接口的信息查询系统,讨论了通用的基于目录索引的信息查询系统,提出一种优化算法通过对查询条件和查询结果进行分析、提取,构造出相关文档的关键字树,并基于关键字树对查询进行重写,生成由关键字组成的新的查询序列,使用生成的关键字序列重新搜索文档,比较两次查询结果并对其进行优先级排序,输出优化后的查询结果。实验结果证明本文提出的查询优化方法能够获得具有更高查全率(recall)和查准率(precision)的查询结果。

关键词 查询重写,查询优化,目录索引,关键字查询

Key-Tree: An Approach for Enhancing Queriability of Form-Based Interface

HE Fan YANG Xiao-Chun YU Ge LI Lin SHI Lei
(School of Information Science & Engineering, Northeastern University, Shenyang 110004)

Abstract Web users often post queries through form-based interfaces on the Web to retrieve data from the Web; however, answers to these queries are mostly computed according to keywords entered into different fields special in a query interface, and their precision and recall could be low. An enhancement in answering this type of queries can be achieved by considering closely related, previous queries submitted through the same interface, along with their answers. In this paper, we present an approach for enhancing the retrieval of other relevant answers to a form-based Web query using previous, relevant queries and their answers. Experimental results show that our query-rewriting approach achieves higher average for precision and recall.

Keywords Query rewriting, Query optimization, Form-based query, Keyword query

1 引言

随着 Internet 的发展,网络上的信息变得复杂而又多样化、网站数量激增,搜索引擎对于网上冲浪的人来说变得更加重要起来。基于目录索引的查询系统是应用较为广泛的一种搜索引擎。然而,这种简单的基于目录索引的查询系统的能力有限。例如,在查询条件很模糊的情况下,一次对文档的标题进行查询的过程中,可能只会查询到与查询条件(如文档的标题)相关的占全部文档20%甚至更低的查询结果,查询的查准率和查全率往往会很低。一般说来,产生这种结果是由于系统以有限的文本属性(文档的标题)来对查询用户的请求进行匹配,在系统实现中,往往联系不到文本的实际内容,在极端的情况下还会产生文本内容与文本属性毫不相干的现象。为此,本文提出关键字树的概念,并基于此提高目录索引的查询能力。

在目录索引这种查询方式下,得到的查询结果之间是相互独立的,彼此之间没有任何联系,而且文档中的内容在查询过程中不被考虑,仅仅是对某些文档属性进行查询。例如:文档的名字、作者、出处等等。希望能够有一种查询机制来进一步加强文档之间的联系并且在系统查询的过程中能够更多地考虑文档的内容信息。为此,本文提出了一种通过改进目录索引的查询接口来优化查询系统的技术。通过对系统原始的查询结果进行分析,将原本相互独立的结果文档信息进行分析,计算文档之间的关联程度,从而对文档进行细致的分类,通过构造关键字树的模型来描述结果集中文档与文档之间的关系。利用构造的关键字树生成新的更能体现查询需求的查询条件,对系统进行二次查询。最后,综合两次的查询结果返回给用户。实验结果显示改进后的查询系统能够帮助系统获得更多的正确查询结果和更高的查全率。在性能方面,为实现查询优化所牺牲的系统性能则基本上是微乎其微的。

2 相关工作

信息查询领域,许多研究者都提出使用全文检索技术对数据进行分类,例如利用 SVM(支持向量机)[1]或决策树的技术。一些研究者也通过链接结构[2,3]和 XML[4~6]标签信息对文档信息进行分析、分类,通过运算构造关系模型,建立视图,在视图上进行查询。

应用视图[7~9]对数据进行查询是数据管理的一种设计方式,通过分析数据的关系、构造数据集、集成数据、设计查询规则来对数据查询进行优化[10]。一个查询视图可以看作是在一定范围内对于网络查询请求的一个结果集,用户通过查询接口在视图上提出查询请求,视图对该请求进行查询重写,通过访问数据源来得到实际的数据信息。

由于目录索引的查询方式从某种意义讲,它作为搜索引

*)本课题得到国家自然科学基金项目资助(60163051)。贺 凡 硕士研究生,主要研究领域为数据挖掘与聚类分析。杨晓春 博士,副教授,主要研究领域为信息集成与生物信息处理。于 戈 教授,博士生导师,主要研究领域为分布式数据库、Web 服务、数据流。李 琳 硕士研究生,主要研究领域为生物信息集成。石 磊 硕士研究生,主要研究领域为 Web 信息集成。

擎在检索能力上还不是很有效。使用全文检索、链接结构和XML的优化技术对于目录索引的查询系统并不适合，因此，引入了一种基于关键字树的技术来实现对目录索引系统的查询重写。它需要的仅是每一次的查询请求和查询结果。

3 问题描述

对于普通的目录索引系统，用户通过接口提交查询请求并获得查询结果。然而，当查询接口的能力有限时，用户可能无法获得希望得到的所有结果。下面对要解决的问题进行形式化描述。

定义1 给定一个数据源 D 及其接口定义 I。对应任意基于 I 的查询请求 q，q 的查询结果可表示为

$R(q)=\langle q(I), q(D)\rangle$，

其中 $q(I)$ 表示通过接口得到的查询结果集合，而 $q(D)$ 表示在数据源 D 中实际可以满足查询 q 的结果集合。显然，$q(I)\subseteq q(D)$。

注意，定义1给出任意一个查询系统的通用定义。当查询接口 I 可以针对数据源 D 的所有信息进行查询时，$q(I)=q(D)$。大部分提供接口的查询系统都属于这种情况。但对于如1节中提到的有限接口能力 I 的目录索引系统，$q(I)\neq q(D)$。本文重点解决 $q(I)\neq q(D)$ 情况下的查询处理问题。

问题1 给定有限接口能力 I 的目录索引系统和查询 q，如何获得一个重构的查询 q'，使 $q(D)$ 与 $q'(I)$ 相似，即可以近似地用 $q'(I)$ 回答查询 q。

理想情况下，希望找到一个 q'，使基于 I 的结果 $q'(I)$ 最接近于 $q(D)$，且 $q'(I)\subseteq q(D)$。此时，通过 q' 得到的结果 $q'(I)$ 很少是不正确的，即保证查准率接近100%。

Internet 上的目录索引系统为网络上的文档提供一种搜索界面，基于网络上的文档检索通常是不精确查询，因此，理想的假设条件 $q'(I)\subseteq q(D)$ 很难被满足。此时希望找到一个 q'，使其基于 I 的结果 $q'(I)$ 与 $q(D)$ 最相似。这种情况下，通过 q' 得到的结果 $q'(I)$ 中可能存在不正确的结果，即查准率可能小于100%，希望将不正确的结果缩小到最小。

由于查询接口 I 的查询能力有限，给定任意两个查询 q_1 和 q_2，$q_1(I)\subseteq q_2(I)$ 并不能说明 $q_1(D)\subseteq q_2(D)$，反之亦然。因此，不能简单地依赖 I 与 D 的关系来重构某一查询。

4 构造关键字树 Key-Tree

为了找到与 $q(D)$ 最相似的 $q'(I)$，必须要在查询 q' 与目标结果 $q(D)$ 间构造一些关联规则，以便查询 q 可以被一个能获得更接近目标结果的 q' 所替代。

为此，本文首先分析任意查询 q 的结果 $q(D)$，抽取出能代表结果集 $q(D)$ 的一组关键字；并将这些关键字组成一个树型结构 Key-Tree，称之为关键字树。然后，选择其中的部分关键字构成 q'。

4.1 关键字树

定义2 给定一个基于目录索引的查询 q 及其结果 $q(I)$，q 的关键字树被记为 $T_q=\langle N_k, N_l\rangle$，其中 N_l 为叶子结点集合，记录 $q(I)$ 中的文档标识，N_k 为内部结点集合，记录其所覆盖的叶子结点的文档的公共关键字。

一个关键字树由树根出发，按照文档相互之间的关联程度以及与该生成树的核心关键字的关联程度由高到底分布，树的根结点用查询 q 来表示。E 是边的集合，每一个边连接这两个结点，是有向的，指出方比被指出方的权值要高。

首先对于一个查询请求 q，应用原有的目录索引系统对 q 进行第一次查询，对返回的结果进行全文扫描，对文档的相关关键字进行词频统计，并依照词频参数构造出相应的关键字树，并依据关键字树对查询 q 进行查询重写。以下重点阐述了构造关键字树的全过程。

4.2 构造关键字树

构造关键字树的过程中引入了相似矩阵和图的概念，通过计算关键字在整体的查询结果文档中的权值来计算相似矩阵，并将构造的相似矩阵转化成一个非连通图，由非连通图内部划分出子连通图，再由连通图生成最大生成树，最后将每一棵子最大生成树合并，构成关键字树，再由关键字树来推导生成重写的关键字查询。整个过程是一个由文档到矩阵，再由矩阵到图，转而由图生成树的过程。是一种通过第一次查询结果来重新生成新的查询条件，并综合前后查询结果来优化查询的算法。

4.2.1 计算文档关键字权值

对一次基于目录索引的查询 q 的查询结果文档进行全文检索，过滤掉非关键字（副词、代词、常用动词、连词、形容词等），统计文档中的关键字及其出现的次数，并用散列表记录下来，通过下面的公式计算出关键字权值：

$$w_{k,i}=tf_{k,i}\times idf_k=\frac{f_{k,i}}{\max_l(f_{l,i})}\times\log\frac{N}{n_k} \tag{1}$$

其中 $tf_{k,i}$ 为第 k 个关键字 k_k 在第 i 个文档 d_i 中的频率比率，$f_{k,i}$ 是 k_k 在 d_i 中的出现次数，$\max_l(f_l,i)$ 是 k_l 在 d_i 中的最大出现次数。idf_k 为 k_k 在文档中的出现频率（定义成 $\log\frac{N}{n_k}$），N 是 q 的结果数，n_k 是 k_n 所出现在 q 中的结果的数量。

4.2.2 构造文档的相似矩阵

对于查询结果集 $q(I)=\{a_1,a_2,\cdots,a_n\}$，在文档与文档之间利用文档中包含的关键字建立起一个 $n\times n$ 的相似矩阵，矩阵中的每个元素 $S_{i,j}$ 代表文档 d_i 与文档 d_j 的相似度，计算公式如下：

$$S_{i,j}=\frac{\sum_{k=1}^{m}w_{k,i}\times w_{k,j}}{\sum_{k=1}^{m}w_{k,i}^2\times\sum_{k=1}^{m}w_{k,j}^2} \tag{2}$$

其中文档中关键字的权值 $w_{i,j}$ 可以通过式(1)得到。

将相似矩阵 $S_{i,j}$ 转换成一个全连通图 G，图中的点表示文档，边表示文档的相似度，边上的权值用 $S_{i,j}$ 表示。设置一个阈值 λ，删掉 G 中权值小于 λ 的边，构造出一个非连通图 G'。

4.2.3 生成关键字树

定义3（最大生成树） 假设一个相似矩阵 $Sim_{n\times n}$，由矩阵 $Sim_{n\times n}$ 可得到相应的一个非连通图 G'，则可以将 G' 分割成多个连通图，每一个子连通图对应一棵最大生成树，其中在最大生成树中约定：(1)在树中每一对结点之间有且仅有一条路径让他们彼此相连，(2)最大生成树的相似度之和在同一连通图中的所有生成树中最大。

将非连通图 G' 划分成多个非连通子图 $SG_1,\cdots,SG_n$。对于每个 SG_i，均对应生成一个最大生成树（即关键字树子树），每一个最大生成树对应一个由构成该最大生成树的相关文档中的关键字组成的关键字集 C_i。在每个 C_i 中，计算关键字 k_i 在关键字集 C_i 中的权值 $W_{i,c}$，如式(3)所示。

式(3)中 $P(A\wedge B)=P(A)\times P(B)$ 中 $P(A)$ 代表 k_i 出现在 C 中，$P(B)$ 表示 k_i 不出现在 C 中。计算过程中用到了之前的文档中关键字的权值。在所有结果中，选出最大权值所对应的

k_i 作为 C_i 的关键字来代表 C_i。C_i 对应的最大生成树则以关键字 k_i 作为根结点。

$$w'_{i,C}=P(A\wedge B)=P(A)\times p(B)$$
$$=\frac{1}{|C|}\sum_{a_j\in C}w_{i,j}\times\prod_{C'\neq C}(1-\frac{1}{|C'|}\sum_{a_j\in C'}w_{i,j}) \quad (3)$$

将所有的最大生成树 T_i 的根结点作为关键字树根结点的下级结点组合在一起，生成关键字树。

图1是一个关键字树的实例。其中查询的主题是 Operation System，通过对首次查询结果中文档内容的分析，生成了三棵子树分别为 linux、windows、unix。其中包含有关键字 linux 的子树由 kernel、segment、process 三个子树构成，kernel 指向的两个叶结点 doc1、doc2分别是两个查询结果 doc1、doc2的文档链接。

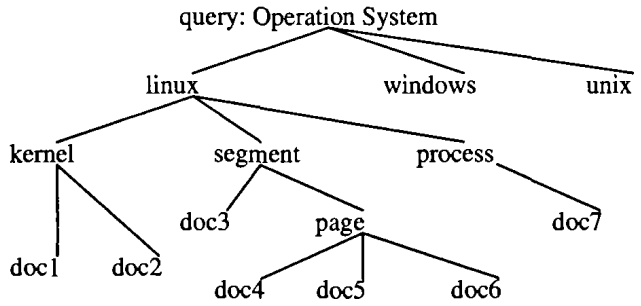

图1 由查询 query 生成的关键字树

5 基于 Key-Tree 的查询重构

利用分析文档得到的关键字树重新生成新的查询，对系统进行二次查询，其中关键字树的利用有两种查询重构的方式，一种是基于单一关键字树的查询重构，另外一种则是综合考虑每一次查询结果所生成的关键字树，即基于多个关键字树的查询重构。

5.1 基于单一关键字树的查询重构

系统通过分析当前的查询结果中文档的内容信息，构造出与之相关的关键字树。在新生成的关键字树中，首先提取根结点以及根结点下一级结点中的关键字，对于二级结点以及二级以下的结点通过设定阈值以过滤的形式对关键字进行提取，组成新的关键字序列，利用新生成的关键字序列，对原系统进行二次查询，得到第二次的查询结果综合分析两次查询结果，过滤掉重复的查询结果，并进行优先级排序，返回查询结果。查询重写规则如下：

设关于查询条件 q 的关键字树为 $T_q(N,D,K)$，其中 N 代表关键字树的所有结点集合，D 代表由 q 通过原查询系统所得到的文档集，K 代表所有文档中的关键字集合。对于每个结点 $n(k_n,l,d_n)\in N$，k_n 表示代表这一结点的关键字，l 代表该结点在关键字树 T_q 中的级别，d_n 代表在该结点下所包含的文档数量。则新的查询序列 q' 为：

$q'_1:-\bigcup_n k_n(k_n\in k_c)$，其中当结点 $n(k_n,l,d_n)$ 中 $l>l_{min}$ 或 $d<\lambda$ 时 $k_n\in K_c$（l_{min} 表示限定结点所在层次的最小值，λ 表示以当前结点为根结点的子树中所包含文档数量的最大值，即阈值）。

5.2 基于多个关键字树的查询重构

相对于单一关键字树来说，基于多个关键字树的查询重构更加复杂。

在某一段时间内，用户连续地对一定范围领域内提出查询请求时，如果对于每次的查询请求仅构造单一关键字树进行查询优化，往往会漏掉一些关键的信息，在这种情况下，若将每次的查询结果都进行整理、分析来构造成多个关键字树，在新的查询请求到来的时候，根据多个关键字树多系统进行优化，使系统能够得到更高的查全率和查准率。

将每次查询结果生成的关键字树，动态地插入到一个包含有多个不同分类子树的森林中，再将每次生成的单一子树中的关键字在森林中搜索，得到更多的相关关键字，利用这些关键字对原系统实现二次查询。

首先，分析当前的查询结果，构造对应的关键字树，并在关键字树中使用单一关键字树的方法提取出关键字序列，在森林中遍历森林中的每一个子树，若找到与之相同的关键字，则以这两个关键字作为根结点，临时构造两棵新树，分析两个关键字结点下面的子树，计算这两个关键字树的相似度，通过设定一个阈值，若相似度大于阈值，则将两个关键字树合并成一个关键字树，将原本不相邻的子树都作为同一根结点的子树；若相似度小于阈值，则重新对这两棵关键字树下面的所有文档重新进行分析，利用前面叙述的方法，建立相似矩阵，生成连通图，构造一棵新的关键字树。将新生成的关键字树插回到原关键字树在森林中的位置，更新森林，得到新的森林。查询重写规则如下：

设由查询条件 q 返回结果集生成的关键字树为 $T_q(N_q,D_q,K_q)$，森林为 $T_f(N_f,D_f,K_f)$，在 T_i 中搜索 T_q 中 $l<2$的关键字，设搜索到的子树为 $T_i(N_i,D_i,K_i)$，分两种情况讨论：

当不存在 T_i 时新的查询序列为

$q'_2=q'_{11}:-\bigcup_n k_n(k_n\in k_c)$。

当存在 T_i 时首先计算 $Sim(T_q,T_i)$，若相似度小于 Sim_{min} 则从当前结点向下从新构造关键字树 T_{new}，$q'_{22}:-\bigcup_i k_i(k_i\in k_{new})$ 其中当结点 $n(k_n,l,d_n)$ 中 $l>l_{min}$ 或 $d<\lambda$ 时 $k_n\in K_{new}$；若相似度大于 Sim_{min} 则 $q'_{22}:-\bigcup_i k_i(k_i\in k_i)$ 其中当结点 $n(k_n,l,d_n)$ 中 $l>l_{min}$ 或 $d<\lambda$ 时 $k_n\in K_i$。新的查询序列为：$q'_2:-q'_{11}\cup q'_{22}$。

6 实验分析

我们实现了对于目录索引查询结果的关键字树的构建，从而生成新的关键字查询，对数据库进行二次查询。

6.1 实验设置

实验测试用的数据库使用的是一个基于目录索引系统的数据库（http://www.sses.com.cn），其中包含了2000篇 IT 领域的相关文档。

实验用服务器配置为 Cpu：Intel Pentium4 1.4Ghz，系统主存256MB，操作系统为 WindowsXP，网络服务器采用 Weblogic，数据库使用 SQLServer2000。

6.2 单一关键字树的数据分析

首先，应用单一的关键字树技术系统进行优化，下表对比地列出了优化前后查询结果的数据统计。

其中引入两个衡量系统效率的指标：

·查准率（Precision）：正确查询结果数（用 C 表示）占通过接口获得的查询结果数（用 I 表示）的百分比

$$P=\frac{C}{I}*100\% \quad (4)$$

·查全率（Recall）：正确查询结果数（用 C 表示）占正确查询结果数（用 D 表示）的百分比

$$R=\frac{C}{D}*100\% \quad (5)$$

表1 优化前后的查全率与查准率

Pq	Rq	Pq′	Rq′
100%	20%	97%	26.20%
100%	6.70%	87.70%	37.04%
100%	27.03%	75.68%	51.85%
90.32%	19.31%	92.68%	26.21%
80%	2.96%	95.65%	16.30%
87.50%	4.83%	95%	26.21%
92.86%	9.63%	97.62%	30.37%
85.71%	4.44%	97.14%	25.19%
86.78%	4.54%	87.18%	25.18%
80.16%	2.76%	87.50%	9.67%
75%	2.22%	95.24%	14.81%
88.89%	5.52%	96.00%	33.10%
59.41%	44.44%	65.25%	57.04%
63.64%	5.19%	90.48%	28.15%

表1中前一部分数据显示了在原系统查准率很高的情况下查询结果的查全率得到了增加，但查准率有所下降，分析原因是由于在应用关键字树技术进行二次查询的过程中，同时也扩大了查询的范围，导致查准率的下降；后一部分数据显示在原系统查准率不高的情况下，不但查全率得到了提高，查准率也得到了一定的提高，其原因在于分析查询结果的过程中过滤掉了错误的文档信息，因此，查准率并不受到之前的错误文档信息的影响，而查全率的上升也同时带动了查准率的上升，产生了查准率上升的现象。

6.3 多树查询

在查询次数很多情况下，采用构造多个关键字树的技术，对系统重新进行了一系列的查询测试。在下表列出了采用多树技术的测试结果，并同时列出了在同一查询条件下采用单一关键字树技术测得的数据结果。

表2 单树与多树的查全率、查准率比较

Ps	Rs	Pm	Rm
97.40%	26.20%	91.23%	35.86%
87.70%	37.04%	81.58%	45.93%
95.65%	16.30%	94.12%	23.70%
97.14%	25.19%	82.95%	54.07%
87.18%	25.18%	81.32%	54.81%
87.50%	9.67%	94.00%	32.41%

表2中 P_s、R_s 表示应用单一关键字树查询所得到的查准率、查全率，P_m、R_m 表示应用多关键字树查询得到的查准率、查全率。数据显示了在其他条件相同的情况下，应用多关键字树往往能够带来更高的查全率，但却是在牺牲了查准率的基础上换来的。产生这种结果的主要原因在于利用多关键字树进行查询重构相当于增加了一定数量的查询条件，也就意味着对查询范围的扩大，这样查询结果数量势必会进一步得到增加，这样也就会出现查全率的增加，同时也可能导致错误查询结果的增加，查准率的下降。对于不同需要，可以选择不同的方法对系统进行查询优化。

6.4 参数设置

前面实验对于关键字树的生成过程中，设置了一个阈值来过滤树型模型中的关键字，前面的实验结果是基于阈值为100得到的，下面是在相同的条件下，设置不同阈值所得到的实验结果。

通过观察图2，可以发现当阈值设置在300的时候，尽管查准率为95.00%，但由于阈值设置过大导致了从关键字树中提取出来的关键字数量有限，影响到了查全率，查全率仅为13.10%；在其他条件不变的情况下将阈值设置为100，查准率有了进一步的提高，达到96.00%，同时查全率有了显著的提高，上升为33.10%，表现出了优秀的性能；在其他条件不变的情况下，继续缩小阈值到30，此时，由于对于文档的拆分过细，导致了提取出的关键字对主题微量的偏离导致查准率下降到了81.33%，尽管其查全率得到了进一步的提高42.07%。分析测试得到的数据，若阈值数值设置得过大，会导致数据分类的泛化，提取出的信息并不具备代表性，过小则可能会对数据的分类过细，提取出的关键字过多而导致提取出的信息与大环境相异，相对来说综合三种情况，我们选择在各方面性能都表现很好的优化设置，将阈值设定在100上完成了大量的实验。

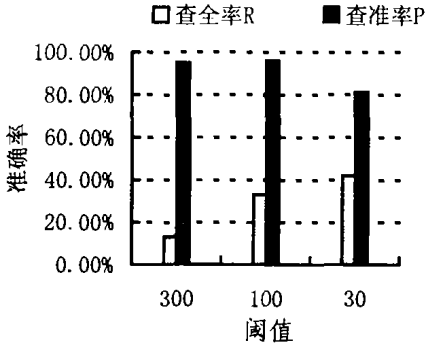

图2 不同阈值性能对比

6.5 性能分析

优化后系统的运行时间可以分成两个部分，一是原系统查询所用掉的时间，二是进行查询重写所用掉的时间，其中在第二部分中，时间基本上都耗费在了分析原查询结果生成关键字树的过程中。下面仅就这一过程，对不同的文档数量的运行时间进行了统计：

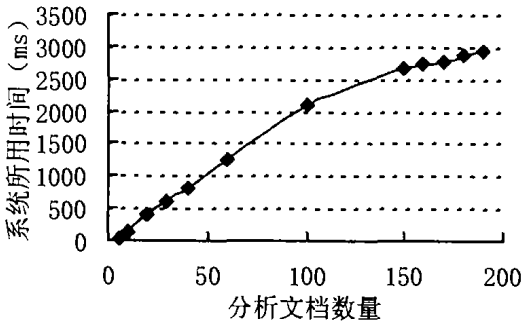

图3 系统性能表

图3中横坐标表示分析文档数量，纵坐标表示所需要的时间，单位为毫秒，实验结果表明在采用了关键字树技术后随着分析文档数量的增加，所需要的时间也随着增加，其性能优于线性函数。

结论与未来工作 本文引入了一种作用在目录索引查询接口上的查询优化技术，同过应用关键字树的技术，对数据库进行二次检索。重点讨论了关键字树的构建和在关键字树的基础上对查询条件进行重写。大量的实验结果显示，这种查询优化技术对提高系统的查询效率（查全率、查准率）有着良好的表现。

目前，这种基于关键字树的方法，还比较简单，能够满足较小的数据库集。在对大数据库方面，需要进一步完善相对复杂度较高的多关键字树理论，在构造多关键字树上及在其基

础上的查询重构、查询条件的重写规则上进一步优化现在的实现算法，在不过多损失系统总体性能的基础上，达到更高的查询效率(查全率、查准率)。

参 考 文 献

1 Yu H-J, Yang Jiong, Han Jia-Wei. Research track: Classifying large date sets Using SVMs with hierarchical clusters. In: Proc. 9th ACM SIGKDD, Aug. 2003

2 Kleinberg J. Authoritative sources in a hyperlinked environment. In: Proc. of the 9th ACM SIAM New Orleans: ACM Press, 1997. 668～677

3 王晓宇，熊方，凌波，周傲英. 一种基于相似度分析的主题提取和发现算法. 软件学报，2003，14 (9)：1578～1585

4 biteboul S A, Segoufin L, Vianu V. Representing and querying XML with incomplete information. In: Proc. of PODS, 2001. 40～50

5 Fernandez M, Suciu D, Tan W C. SilkRoute: Trading between relations and XML. In: Proc. of 9th Intl. Conf. on World Wide Web, 2000

6 Popa L, Deutsch A, Sahuguet A, tannen V. A chase too far? In: Proc. of SIGMOD, 2000

7 Calvanese D, Giacomo G D, Lenzerini M, Vardi M. View-Based Query Processing for Regular Path Queries with Inverse In: proc. of PODS, 2000. 58～66

8 Pottinger R, Levy A R. A scalable algorithm for answering queries using views. In: Proc. of the 26th Intl. Conf. on VLDB, 2000. 484～485

9 Ullman J D. Information Integration Using Logical Views. In: Proc. of ICDT'97, 1997. 19～40

10 Raeza-Yates R, Ribeiro-Neto B. Modern Information Retrial. ACM, 1999

(上接第120页)

隔划分法分成十个区间：[0，20]，0；[21，40]，1；[41，60]，2；[61，80]，3；[81，100]，4；[101，120]，5；[121，140]，6；[141，160]，7；[161，180]，8；[181，201]，9。这样便可以得到决策表。

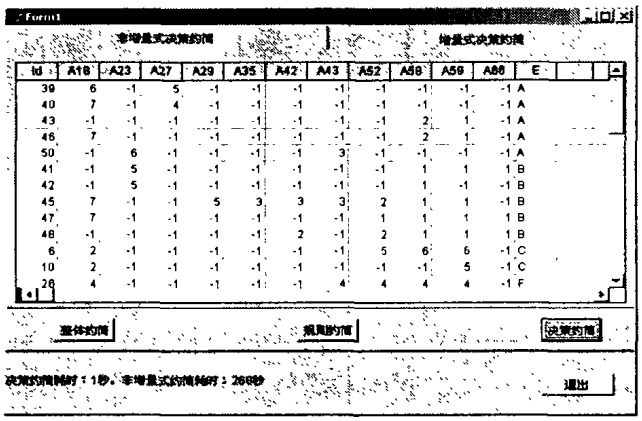

图4 分类结果1

该系统主要有两项功能：静态属性约简与增量式属性约简。其中在静态属性约简模块中又分三块：整体约简模块、规则约简模块、求极小决策算法模块。在增量式属性约简模块中也分了三块：添加模块、删除模块、求极小决策算法模块。具体形式及运行结果如图4和图5所示。

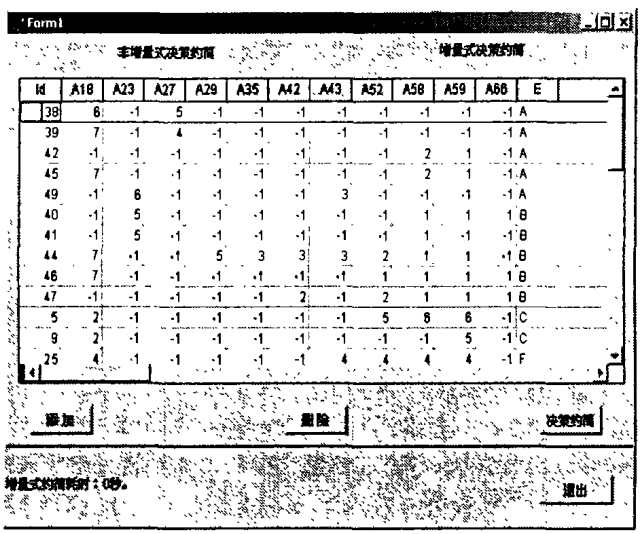

图5 分类结果2

上述运行结果表明：经过整体约简后70个条件属性中仅有11个必要属性 A18，A23，A27，A29，A35，A42，A43，A52，A58，A59，A66，经过规则约简与决策约简后得到极小决策算法。上述结果中的每一行是一条分类规则，如第一行 $A18_6A23_{-1}A27_5A29_{-1}A35_{-1}A42_{-1}A43_{-1}A52_{-1}A58_{-1}A59_{-1}A66_{-1}EA$ 表示：属性 A18、A27分别取6和5即在波长为4700、5300处流量值分别处于区间[141，160]与[121，140]时该恒星所属类别为 A。属性值为－1表示该属性值可以约简掉。对于其他行的结果意义相同。

结束语 本文提出的基于粗糙逻辑的增量式属性约简算法，其特点是在原有的极小决策算法的基础上，并依据第3节中给出的理论求得新系统的约简与极小决策算法，从而实现了对原极小决策算法的更新与维护。以此为基础，采用 VC++ 和 Oracle9i 为开发工具，设计与实现了基于属性约简的恒星光谱数据分类规则挖掘系统，从而为恒星光谱数据的自动分类实现提供了一种有效的途径。但由于人类对天体认识还很有限，以及天体光谱的多样性和信噪影响，要想真正解决天体光谱的识别和分类问题，必须有足够的能用以准确的分类的天文光谱专家知识。如何将天文光谱专家知识进行规则化并融入相应的分类算法中，以尽可能地实现计算机自动较准确地进行分类，是天体光谱数据挖掘分类中的一个重要研究问题。

参 考 文 献

1 刘宗田. 属性最小约简的增量式算法. 电子学报，1999，27(11)：96～98

2 陈云化，叶东毅. 基于粗糙集理论的一个增量算法. 计算机科学，2002，29(9)：53～55

3 Bang W-C, Bien Z. New incremental inductive learning algorithm in the framework of rough set theory. International Journal of Fuzzy Systems, 1999, 1 (1): 25～36

4 Pawlak Z. Rough sets: Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dordrecht, 1991

5 覃冬梅. 天体光谱信号的自动识别方法研究：[博士论文]. 中国科学院自动化研究所，2003

6 Bonikowski Z. Algebraic Structures of Rough Sets. In: Ziarko W, ed. Rough Set, Fuzzy Sets and Knowledge Discovery. Springer-Verlay, 1994. 242～247