计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20120645
51(4):918-929, 2014

基于关系数据库的 top-k 聚合关键词查询

张东站　苏志锋　林子雨　薛永生
(厦门大学计算机科学系　福建厦门　361005)
(zdz@xmu.edu.cn)

top-k Aggregation Keyword Search over Relational Databases

Zhang Dongzhan, Su Zhifeng, Lin Ziyu, and Xue Yongsheng
(*Department of Computer Science, Xiamen University, Xiamen, Fujian* 361005)

Abstract　Structured query language (SQL) is a classical approach to performing query over relational databases. However, it is difficult to search information for ordinary users who are unfamiliar with the underlying schema of the database and SQL. While keyword search technology used in information retrieval (IR) systems allows users to just simply input a set of keywords to get the required results. Therefore, it is desirable to integrate DB and IR, which allows users to search relational databases without any knowledge of database schema and query languages. Given a keyword query, the existing approaches find individual tuples which match a set of query keywords based on primary-foreign-key relationships in databases. However, it is more useful for users to get the aggregation result of tuples in many real applications, and those existing methods cannot be used to deal with such issue. Therefore, this paper focuses on the problem of top-k aggregation keyword search over relational databases. Here recursion-based full search algorithm, i.e., RFS, is proposed to get all aggregation cells. To achieve high performance, new ranking techniques, keyword-tuple-based two dimensional index and quick search algorithm, i.e., OQS, are developed for effectively identifying top-k aggregation cells. A large number of experiments have been implemented upon two large real datasets, and the experimental results show the benefits of our approach.

Key words　aggregation keyword search; relational database; two dimensional index; aggregation cell; ranking

摘　要　基于关系数据库的关键词查询,使得用户在不需要掌握结构化查询语言和数据库模式的情况下,可以方便地进行关系数据库查询.给定一个关键词查询,已有的方法通过数据库中的主外键关联,查询得到包含关键词的元组集合.但是,在很多实际应用中,元组集合的聚合结果对用户更有价值;研究了基于关系数据库的 top-k 聚合关键词查询,提出了基于递归的聚合单元枚举算法——基于递归的完全搜索(recursion-based full search, RFS).为了获得更好的查询性能,设计了新的排序方法、二维索引和快速搜索算法——基于输出的快速搜索(output-based quick search, OQS),从而可以高效地枚举 top-k

收稿日期:2012-07-31;**修回日期**:2012-12-05
基金项目:中央高校基本科研业务费专项资金项目(2011121049);国家自然科学基金项目(61102136,61202012);福建省自然科学基金项目(2011J05156,2011J05158);国家自然科学基金项目(61303004);福建省自然科学基金项目(2013J05099)
通信作者:林子雨(ziyulin@xmu.edu.cn)

个聚合单元;在不同的数据集上进行了大量的实验,实验结果表明 OQS 算法具有良好的查询性能.

关键词 聚合关键词查询;关系数据库;二维索引;聚合单元;排序

中图法分类号 TP311.132.3

关键词查询作为一种有效的信息获取方式,已经广泛应用于信息检索系统和万维网.近年来,许多研究人员将关键词查询应用于关系数据库[1-6]和 XML 数据库[7-8].基于关系数据库的关键词查询方式,使得用户只需要输入关键词就可以找到包含关键词的相关结果.与传统的结构化查询方式相比,这种方式不要求用户了解数据库模式和结构化查询语言.

对于输入的关键词查询,基于关系数据库的查询方法返回与关键词相关的元组集合,已经存在的算法可以分为 3 种类型:基于数据图的方法[1,9-11]、基于模式图的方法[2,12]和基于元组的方法[13-14].这 3 种方法都利用关系数据库的主外键关联,输出与关键词查询相关的、具有一定结构的元组集合.然而,在实际的企业应用中,元组集合中包含大量的结果,让用户自己从中寻找有用的信息是非常困难的.此外,文献[15]通过大量的实际应用调研表明,元组集合的聚合结果对用户更有价值.因此,研究聚合关键词查询具有重要的应用价值.下面用一个实例说明聚合关键词查询的意义.

例 1. 表 1 是电影数据库 IMDB[16]的部分数据.假设 Scott 是一个电影爱好者,他想查找一部主题与"love"相关、类型为"action"、语言为"Japanese"的电影.因此,他输入的查询关键词是{"love","action","Japanese"}.但是,在表 1 中,这 3 个关键词分别出现在不同的元组中.因此,已经存在的方法只能返回 5 个结果,其中,每个结果是包含一个关键词的元组.事实上,因为"Quentin Tarantino"执导的电影"Kill Bill: Vol. 1(2003)"是一部有关"love"的电影,并且这部电影的类型是"action",语言也有"Japanese".因此,表 1 中前 3 行元组的聚合结果(Kill Bill: Vol. 1(2003),Quentin Tarantino,*,*,*)完全满足 Scott 的需求,并且它更有价值,其中符号"*"表示属性"Language","Genres"和"Keyword"有多个不同的值.

Table 1 Movies Table

表 1 电影数据表

Puple No.	Title	Director	Language	Genres	Keyword
t_1	Kill Bill: Vol. 1(2003)	Quentin Tarantino	English	Action	17-year-old, action-heroine, airplane
t_2	Kill Bill: Vol. 1(2003)	Quentin Tarantino	Japanese	Crime	Baseball, beaten-to-death, blood
t_3	Kill Bill: Vol. 1(2003)	Quentin Tarantino	French	Drama	Chinese, danger, love
t_4	Titanic(1997)	James Cameron	Japanese	Romance	Artist, iceberg
t_5	Titanic(1997)	James Cameron	Japanese	Romance	love, heroine

因此,本文研究基于关系数据库的 top-k 聚合关键词查询.与通常的基于关系数据库的关键词查询不同,对于给定的关键词集合,聚合关键词查询返回聚合单元,并且聚合单元中的元组包含了所有的关键词,这样的元组被称为内容元组.我们采用数据库评分和信息检索(information retrieval, IR)评分相结合的方式,对聚合单元进行评分并排序,从而输出满足用户要求的 top-k 个结果.此外,提出了两种优化策略,可以大幅提升查询效率.首先,设计了基于关键词元组的二维索引(keyword-tuple-based two dimensional index, KTTD),用来快速获取内容元组的评分.并且对 KTTD 索引进行物化,以加速元组排序的过程.其次,提出了基于输出的快速搜索(output-based quick search, OQS)算法,可以大大缩减聚合单元的搜索空间.最后,在不同的数据集上进行了大量的实验,实验结果表明 OQS 算法具有良好的查询性能.

1 问题描述

定义 1. 聚合单元[15].$T=(A_1,A_2,\cdots,A_n)$是关系数据库中的一个表.表 T 上的一个聚合单元是一个元组 $c=(x_1,x_2,\cdots,x_n)$,其中,$x_i\in A_i$ 或者 $x_i=*(1\leqslant i\leqslant n)$,"*"是一种元符号,它表示对应属性的泛化.聚合单元 c 的覆盖 $Cov(c)$是表 T 中的元组集合,并且这些元组和 c 在所有非 * 属性上具有相同

的属性值,其定义如下:

$$Cov(c)=\{(v_1,v_2,\cdots,v_n)\in T \mid v_i=x_i,\text{如果}\ x_i\neq *,1\leqslant i\leqslant n\}.$$

如果 c 中有 $n_0(n_0\leqslant n)$个值不是“*”,则 c 是 n_0 维聚合单元. 如果 $n_0=n$,那么 c 是基本聚合单元;如果 $n_0=0$,则称 c 为完全聚合单元.

例 2. 在表 1 中,元组 t_1,t_2 和 t_3 在“Title”和“Director”属性上具有相同的值,因此,这两个属性保留原来的属性值;而在“Language”,“Genres”和“Keyword”属性上值不完全相同,所以这 3 个属性的值被泛化成“*”. 由此可以得到元组 t_1,t_2 和 t_3 的聚合单元为 c_0=(Kill Bill:Vol. 1(2003),Quentin Tarantino,*,*,*),c_0 是一个二维聚合单元. 根据定义 1 可以得到 c_0 的覆盖 $Cov(c_0)=\{t_1,t_2,t_3\}$. 这是因为元组 t_1,t_2,t_3 和 c_0 在非 * 属性上(即“Title”和“Director”)具有相同的属性值. 同理可得,元组 t_3 和 t_4 的聚合单元为 c_1=(*,*,*,*,*),并且 $Cov(c_1)=\{t_3,t_4\}$,c_1 是一个完全聚合单元.

在实际的求解过程中,为了得到一个聚合单元 c,可以先求得它的覆盖 $Cov(c)$,然后由 $Cov(c)$得到 c,具体过程如下:对于 $Cov(c)$中元组,如果在某个属性上具有相同的值,则在该属性上保留原来的值;如果在某个属性上值不完全相同,则该属性的值被泛化成“*”.

问题描述:给定关键词查询 $K=\{k_1,k_2,\cdots,k_m\}$,关系数据库的 top-k 聚合关键词查询输出 top-k 个具有最高评分的聚合单元.

例 3. 例 1 中的关键词查询 K = {“love”,“action”,“Japanese”},表 1 中元组 t_1 包含关键词“action”,t_2 包含关键词“Japanese”,t_3 包含关键词“love”,因此 t_1,t_2 和 t_3 的聚合单元(Kill Bill:Vol. 1(2003),Quentin Tarantino,*,*,*)是一个满足 K 的查询结果. 同时,由于 t_4 包含关键词“Japanese”,所以,t_1,t_3 和 t_4 的聚合单元(*,*,*,*,*)也是一个满足 K 的查询结果,它是完全聚合单元. 由于完全聚合单元不包含任何对用户有用的信息,因此这里不把它作为查询结果输出,尽管完全聚合单元的评分可能很高(评分机制参见 2.1.2 节).

2 基于关系数据库的 top-k 聚合关键词查询

本节首先提出一个解决 top-k 聚合关键词查询的朴素算法——基于递归的完全搜索(recursion-based full search, RFS),该算法采用递归查询的方式,遍历所有与关键词查询相关的聚合单元,然后对其进行排序,从而得到 top-k 聚合单元. 在此基础上,进一步提出了 RFS 的改进算法——基于输出的快速搜索(output-based quick search, OQS),它可以大幅减少搜索空间,提高查询效率.

2.1 朴素算法 RFS

假设输入的关键词查询为 $K=\{k_1,k_2,\cdots,k_m\}$,对于每一个关键词 k_i,由倒排索引可以很容易找到其对应的元组集合 S_i. 从每一个 S_i 中取出一个元组,就可以组成一个聚合单元的覆盖. 遍历所有的聚合单元,并对它们进行评分和排序就可以得到 top-k 个聚合单元.

2.1.1 算法描述

算法 1. RFS(K,l).

输入:查询关键词集合 $K=\{k_1,k_2,\cdots,k_m\}$、返回查询结果数 l;

输出:top-k 个聚合单元.

① initialize an array L; /* L 是一个数组,第 i 个元素存储关键词 k_i 对应的内容元组的集合 S_i */
② initialize an array U; /* U 是一个包含 m 个元素的数组 */
③ $A\leftarrow\phi$; /* A 是一个集合,用来存储所有聚合单元 */
④ for each $k_i\in K$ do
⑤ $S_i\leftarrow$get the content tuples of k_i from the inverted index;
⑥ $L[i]\leftarrow S_i$;
⑦ endfor
⑧ $Traversal(A,L,U,1)$;
⑨ $Sort(A)$;
⑩ Output the top-k elements of A;
⑪ Procedure $Traversal(A,L,U,j)$ begin
⑫ if $j=m+1$ then
⑬ $c\leftarrow$get the aggregate cell of U;
⑭ $A\leftarrow A\cup c$;
⑮ else
⑯ for each $t_i\in L[j]$ do /* t_i 表示一个元组 */
⑰ $U[j]\leftarrow t_i$;
⑱ $Traversal(A,L,U,j+1)$;
⑲ endfor
⑳ endif
㉑ end /* $Traversal$ */

如算法 1 所示,RFS 算法主要分为两个部分:

数据定义部分(行①～⑦)和递归遍历部分(行⑪～㉑).在遍历的过程中,算法生成的所有聚合单元被添加到集合 A 中,然后,对 A 排序就可以得到 top-k 个聚合单元(行⑨～⑩).下面结合例 1 的数据对算法 1 进行详细说明.

例 4. 例 1 中的关键词查询为 $K=\{$"love","action","Japanese"$\}$,经过 RFS 算法的"数据定义部分"之后,数组 $L=[\{t_3,t_5\},\{t_1\},\{t_2,t_4,t_5\}]$,接下来算法执行"递归遍历部分".首次执行函数 *Traversal* 时 $j=1(m=3,j\neq m+1)$,算法执行行⑯,此时 t_i 的值是 t_3,其中 t_3 是 $L[1]$中的第 1 个元组,从而得到 $U[1]=t_3$,算法递归调用函数 *Traversal*.此时 $j=2(j\neq m+1)$,算法执行到行⑰时,t_i 的值是 t_1,$U[2]=t_1$,其中 t_1 是 $L[2]$的第 1 个元组,紧接着第 3 次递归调用函数 *Traversal*.此时 $j=3(j\neq m+1)$,同理得到 t_i 为 t_2,$U[3]=t_2$,其中 t_2 是 $L[3]$的第 1 个元组,算法第 4 次递归调用函数 *Traversal*.此时 $j=4(j=m+1)$,因此算法执行行⑫～⑭,从而得到 $U=(t_3,t_1,t_2)$,由覆盖 U 可以求出 c_1,并把 c_1 添加到 A 中,此时算法产生第 1 个聚合单元.然后算法回退到第 3 次函数 *Traversal* 的递归调用过程,执行行⑯语句,此时 t_i 为 t_4,$U[3]=t_4$,其中 t_4 是 $L[3]$的第 2 个元组,算法第 5 次递归调用函数 *Traversal*.由于 $j=4(j=m+1)$,因此,算法执行行⑫～⑭,从而得到 $U=(t_3,t_1,t_4)$,由 U 可以得到第 2 个聚合单元 c_2.同理可知,算法依次产生的覆盖为(t_3,t_1,t_2),(t_3,t_1,t_4),(t_3,t_1,t_5),(t_5,t_1,t_2),(t_5,t_1,t_4),(t_5,t_1,t_5),并且 A 保存了对应的聚合单元.由此,算法对 A 排序后就可以产生 top-k 个聚合单元.

在算法 RFS 中,递归遍历部分(行⑪～㉑)将所有的聚合单元添加到集合 A 中,因此 A 的大小为 $\prod_{i=1}^{m}|S_i|$.而在表 T 中,内容元组集合 S_i 的最大值为表 T 中元组的总个数 n,因此,$|S_i|$的最大值为 n,$\prod_{i=1}^{m}|S_i|$的最大值为 n^m,从而可以得到算法的空间复杂度为 $O(n^m)$.RFS 算法的时间复杂度取决于 3 个方面:1)算法数据定义部分(行①～⑦)的时间复杂度,它与关键词的个数 m 相关,即 $O(m)$;2)递归遍历部分(行⑪～㉑)的时间复杂度,即 $O(\prod_{i=1}^{m}|S_i|)$,由于$|S_i|$的最大值为 n,因此遍历聚合单元的时间复杂度为 $O(n^m)$;3)对集合 A 进行排序的时间复杂度,由于 RFS 算法采用快速排序算法对聚合单元进行排序,因此其时间复杂度为 $O(n^m\log(n^m))$.综上所述,RFS 算法的时间复杂度为 $O(m+n^m+n^m\log(n^m))$,即 $O(n^m\log(n^m))$.

2.1.2 评分机制

由例 4 可知,给定一个关键词查询 K,可以得到与其相关的所有聚合单元.但是对用户来说,并不是所有的聚合单元都具有同样的价值.因此需要采用相关评分机制对聚合单元进行评分并排序,从而输出满足用户需求的 top-k 个结果.

传统的基于 $TF\times IDF$ 的评分方法可以直接用来对元组进行评分,目前已经有不少文献[3,17]采用 $TF\times ID$ 或者其变种对查询结果进行评分,本文也采用 $TF\times IDF$ 评分方法对直接包含关键词的元组进行评分.给定一个数据库表 T,T 中的每一个元组 t 可以看作是一个文档,这里假设,ts 表示表 T 中所有元组构成的集合,ks 表示表 T 中所有关键词的集合,ts_k 表示包含关键词 k 的元组的集合.对于给定的元组 $t\in ts$ 和关键词 $k\in ks$,令 $FREQ(t,k)$表示关键词 k 在元组 t 中出现的次数,定义 $TF(t,k)$为关键词 k 在元组 t 中的相对词频,即

$$TF(t,k)=\begin{cases}0, & FREQ(t,k)=0;\\ 1+\ln(1+FREQ(t,k)), & \text{其他.}\end{cases}\tag{1}$$

同时,定义 $IDF(k)$为关键词 k 的逆文档频率,即

$$IDF(k)=\ln\frac{1+|ts|}{|ts_k|},\tag{2}$$

其中,$|ts|$表示集合 ts 的元素个数,$|ts_k|$是集合 ts_k 的元素个数.给定的关键词查询 $K=\{k_1,k_2,\cdots,k_m\}$,元组 t 和关键词 k_i 的相关度定义为

$$TF\times IDF(t,k_i)=TF(t,k_i)\times IDF(k_i),\tag{3}$$

则关键词查询 K 与内容元组 t 的相关度为

$$TF\times IDF(t,K)=\sum_{i=1}^{m}TF\times IDF(t,k_i),\tag{4}$$

$TF\times IDF$ 评分能够表示关键词和元组之间的相关程度,但是它并不能反映关系数据库中丰富的结构信息.因此,受到文献[13]的启发,根据一对关键词$\langle k_i,k_j\rangle$与元组 t 的不同结构关系,我们对其赋予不同的评分,其结构关系分为以下几种情况:

情况 1. k_i 和 k_j 都不在元组 t 中;

情况 2. k_i 和 k_j 一个在元组 t 中,另一个不在元组 t 中,但是,它们在同一个属性列;

情况 3. k_i 和 k_j 在同一个元组 t 的不同属性列;

情况 4. k_i 和 k_j 在同一个元组 t 的相同属性列.

对于以上的 4 种情况,定义$\langle k_i,k_j\rangle$与元组 t 的相对距离为

$$DIST(t,\langle k_i,k_j\rangle)=\begin{cases}0, & \text{情况 1;}\\ 1, & \text{情况 2;}\\ 2, & \text{情况 3;}\\ 4, & \text{情况 4.}\end{cases}\tag{5}$$

从而,可以得到$\langle k_i,k_j\rangle$与元组 t 的相关度为

$$REL(t,\langle k_i,k_j\rangle)=\ln(1+DIST(t,\langle k_i,k_j\rangle)),\quad(6)$$

继而可以得到关键词查询 $K=\{k_1,k_2,\cdots,k_m\}$与元组 t 的相关度为

$$SCORE(t,K)=TF\times IDF(t,K)+\sum_{1\leqslant i\leqslant j\leqslant m}REL(t,\langle k_i,k_j\rangle).\quad(7)$$

最终可以得到聚合单元 c 与关键词查询 $K=\{k_1,k_2,\cdots,k_m\}$的相关度为

$$SCORE(c,K)=\frac{1}{|Cov(c)|}\sum_{t\in Cov(c)}SCORE(t,K),\quad(8)$$

其中,$|Cov(c)|$表示 $Cov(c)$中元组个数.

2.2 改进算法 OQS

朴素算法虽然可以得到相关查询结果,但是,采用朴素算法求 top-k 个聚合单元的时间复杂度是 $O(n^m\log(n^m))$,空间复杂度为 $O(n^m)$. 因此,这种方法的效率低,可扩展性差. 本节介绍针对朴素算法的一种改进算法 OQS,它可以快速得到 top-k 个聚合单元.

2.2.1 改进算法的理论基础

通过研究发现,不需要搜索所有聚合单元也可以得到和朴素算法同样的 top-k 个结果. 下面用实例说明如何减少需要搜索的聚合单元的数量.

例 5. 假设存在关键词查询 $K=\{k_1,k_2,k_3\}$,每个关键词对应的降序倒排索引集合分别为 $S_1=\{t_1,t_2,t_3\}$,$S_2=\{t_5,t_1\}$,$S_3=\{t_6,t_2,t_1\}$. 从而可以得到,所有聚合单元的覆盖是空间 $V_{all}=S_1\times S_2\times S_3$. S_1,S_2 和 S_3 的第 1 个元组组成的覆盖是 $Cov(c_1)=\{t_1,t_5,t_6\}$,其中 $t_1=S_1[1]$,$t_5=S_2[1]$,$t_6=S_3[1]$,因为集合 S_1,S_2,和 S_3 降序排列,所以 c_1 是空间 V_{all}中评分最大的聚合单元. 为了得到下一个最大聚合单元,可以将整个搜索空间 V_{all}分为 4 个子集空间:

V_1: $(S_1-\{t_1\})\times S_2\times S_3$;

V_2: $\{t_1\}\times(S_2-\{t_5\})\times S_3$;

V_3: $\{t_1\}\times\{t_5\}\times(S_3-\{t_6\})$;

V_4: $\{t_1\}\times\{t_5\}\times\{t_6\}$.

其中,4 个子空间的并集为整个搜索空间,同时它们两两之间没有交集,而子空间 V_4 恰好是 c_1 的覆盖. 并且,对于 S_1,S_2 和 S_3 而言,把每个集合的首元素都剔除后得到的 3 个新的集合 $S_1'=(S_1-\{t_1\})$,$S_2'=(S_2-\{t_5\})$和 $S_3'=(S_3-\{t_6\})$,也是按照降序排列的. 在子空间 V_1 中,由分别来自 S_1',S_2 和 S_3 的第 1 个元组所构成的覆盖是 $Cov(c_1^1)=\{t_2,t_5,t_6\}$,因为它们都是降序排列的集合,所以 c_1^1 是子空间 V_1 的评分最大的聚合单元,同理,子空间 V_2 和 V_3 的最大聚合单元的覆盖依次是 $Cov(c_1^2)=\{t_1,t_1,t_6\}$和 $Cov(c_1^3)=\{t_1,t_5,t_2\}$. 从而可以得到下一个评分最大的聚合单元 $c_2=\max(c_1^1,c_1^2,c_1^3)$,其中,max 函数表示取评分最大的聚合单元. 这里假设 $\max(c_1^1,c_1^2,c_1^3)$的结果为 c_1^2,即 $c_2=c_1^2$. 采用类似的方法,根据 c_2 将搜索空间分为 4 个子空间,同时,也可以获得除 c_2 之外的 3 个子空间的最大聚合单元. 假设为 c_2^1,c_2^2,c_2^3,则第 3 个评分最高的聚合单元是 $c_3=\max(c_1^2,c_1^3,c_2^1,c_2^2,c_2^3)$. 同理,可以依次得到 top-$k$ 个评分最高的聚合单元.

下面将给出关于改进算法理论基础的正确性证明.

定理 1. 给定关键词查询 $K=\{k_i\mid 1\leqslant i\leqslant m\}$,对于任意 $k_i\in K$,其对应的内容元组集合为 $S_i=\{t_j\mid 1\leqslant j\leqslant l_i\}$,其中,$l_i$ 表示集合 S_i 的元组个数. 则评分最大的聚合单元 $c_{\max}$的覆盖为 $Cov(c_{\max})=\{t_i\mid 1\leqslant i\leqslant m$ 并且 $t_i=\max S_i\}$,其中,$\max S_i$ 表示集合 S_i 中评分最高的元组.

证明. 这里采用反证法证明. 假设还存在另外一个评分比 $c_{\max}$更高的聚合单元 c',即 $SCORE(c',K)>SCORE(c_{\max},K)$,$c'$ 的覆盖 $Cov(c')=\{t_i'\mid 1\leqslant k\leqslant m$ 其中 $t_i'\in S_i\}$. 从而可以得到 $\frac{1}{|Cov(c')|}\sum_{t_i'\in Cov(c')}SCORE(t_i',K)>\frac{1}{|Cov(c_{\max})|}\sum_{t_i\in Cov(c_{\max})}SCORE(t_i,K)$,由于$|Cov(c')|=|Cov(c_{\max})|=m$,因此,整理可得 $\frac{1}{m}\sum_{i=1}^{m}(SCORE(t_i',K)-SCORE(t_i,K))>0$,其中,$t_i'\in Cov(c')$,$t_i\in Cov(c)$,并且 $t_i'\in S_i$,$t_i\in S_i$. 这就意味着至少存在某个 i 值,使得 $SCORE(t_i',K)-SCORE(t_i,K)>0$,这说明 t_i'的评分比 t_i 大,而在已知条件中 t_i 是评分最高的元组,二者存在矛盾,因此,定理得证. 证毕.

定理 1 说明,每一个集合 S_i 的最大评分元组构成的集合是评分最大的聚合单元的覆盖,从而可以得到最大聚合单元. 因此,对于按照降序排列的 S_i,最大聚合单元的覆盖是每一个 S_i 的第 1 个元组的集合. 由此可以得到推论 1.

推论 1. 给定关键词查询 $K=\{k_i\mid 1\leqslant i\leqslant m\}$,对于任意 $k_i\in K$,其对应的降序排列的内容元组集合为 $S_i=\{t_j\mid 1\leqslant j\leqslant l_i\}$,其中,$l_i$ 表示集合 S_i 的元组个数,给定一个关于 $\prod_{i=1}^{m}S_i$ 的任意子空间 $V=\prod_{i=1}^{m}S_i'$,其中 $S_i'\subseteq S_i$,并且 S_i' 中的元组也是按照评分降序排序,则子空间 V 的最大聚合单元 c 的覆盖为 $Cov(c)=\{t_i\mid 1\leqslant i\leqslant m$,并且 t_i 是 S_i' 的第 1 个元组$\}$.

由推论 1 可知,任意子空间 V 的评分最大的聚合单元的覆盖是由每一个 S_i' 的第 1 个元组组成的.因此,只需要访问 m 个元组就可以得到子空间 V 的最大聚合单元.同时,根据已经找到的聚合单元,将搜索空间划分为不相交子空间,从而可以快速产生每个子空间的最大聚合单元.所以,只需要搜索与子空间相同个数的聚合单元就可以产生下一个评分最大的聚合单元,从而大大缩小了需要搜索的聚合单元的数量.

2.2.2 算法描述

以上述理论为基础,本文设计了朴素算法的一种改进算法 OQS(如算法 2 所示),后者在时间复杂度和空间复杂方面都优于前者.为了方便描述算法,OQS 算法采用数组下标的方式表示内容元组集合中的一个元组,例如 $S_1[2]$表示 S_1 中的第 2 个元组.

在例 5 中,对于子空间 $V_2=\{t_1\}\times(S_2-\{t_5\})\times S_3$,因为 $S_1'=\{t_1\}$只包含一个元组,所以,根据 V_2 的最大聚合单元 $Cov(c_1^2)=\{t_1,t_1,t_6\}$,对 V_2 进行划分时,集合$\{t_1\}$不变,所以只需划分为 3 个子空间,其中一个子空间正是 $Cov(c_1^2)$.同理,在子空间 $V_3=\{t_1\}\times\{t_5\}\times(S_3-\{t_6\})$中,由于 $S_1'=\{t_1\}$和 $S_2'=\{t_5\}$都只包含一个元组,因此,根据 V_3 的最大聚合单元 $Cov(c_1^3)=\{t_1,t_5,t_2\}$,只需划分 2 个子空间,其中一个子空间正是 $Cov(c_1^3)$.所以,在 OQS 算法中,结构 $q(idx,start,score)$控制子空间的划分,其中,idx 是一个聚合单元的覆盖对应的在 S_i 中的下标,$score$ 是对应聚合单元的评分,$start$ 是根据 q 划分子空间时,第 1 个元组个数大于 1 的 S_i'的下标.

算法 2. OQS(K,l).

输入:关键词查询 $K=\{k_1,k_2,\cdots,k_m\}$、返回查询结果数 l;

输出:top-k 聚合单元.

① for each $k_i\in K$ do
② $S_i\leftarrow$get the content tuples of k_i from KTTD;
③ for each $t\in S_i$ do
④ compute the score of t according to equation (7);
⑤ endfor
⑥ $Sort(S_i)$;
⑦ endfor
⑧ $q_0\leftarrow(idx_0,begin_0,score_0)$; /* 获取 top-1 聚合单元 */
⑨ $Q\leftarrow Q\cup q_0$; /* Q 是一个集合,它保存算法遍历到的 q */
⑩ while Q!=null do
⑪ $q'\leftarrow$pop max from Q; /* 获取 $score$ 最大的 q */
⑫ if $q'.idx$ not exist in L then /* 判断对应覆盖是否已经被输出 */
⑬ output the cover according to q';
⑭ $l\leftarrow l-1$;
⑮ if $l=0$ then break;endif
⑯ $L\leftarrow L\cup q'.idx$;
⑰ endif
⑱ $Next(q')$;
⑲ endwhile
⑳ Procedure $Next(q')$ begin
㉑ for each $i\in[q'.start,m]$ do
㉒ $q''\leftarrow q'$;
㉓ $q''.idx[i]\leftarrow q''.idx[i]+1$;
㉔ if $q''.idx[i]\leqslant S_i.size$ then
㉕ $q''.start\leftarrow i$;
㉖ $Q\leftarrow Q\cup q''$;
㉗ endif
㉘ endfor
㉙ end /* $Next$ */

OQS 算法主要分为 3 个部分:

1) 对于每一个关键词 k_i,通过 KTTD 索引(索引方法参见 2.2.3 节)找到其对应的内容元组的集合 S_i(行②),根据式(7)计算每一个内容元组与查询关键词的评分并排序(行③~⑦);

2) 根据定理 1 获取最大聚合单元(行⑧),接下来根据已经搜索到的聚合单元逐个寻找下一个最大聚合单元(行⑩~⑲);

3) 根据当前 q',将搜索空间化分为 $m-q'.start+1$ 个不相交的子空间.根据推论 1,获得比 q'小的 $m-q'.start$ 个聚合单元(行㉑~㉙).

需要特别指出的是,OQS 算法中可能生成重复的聚合单元.在例 5 中,子空间 V_3 包含覆盖 $Cov(c_1)=\{S_1[1],S_2[1],S_3[2]\}=\{t_1,t_5,t_2\}$,子空间 V_1 包含覆盖 $Cov(c_2)=\{S_1[2],S_2[1],S_3[3]\}=\{t_1,t_5,t_2\}$,因此有$\{S_1[1],S_2[1],S_3[2]\}=\{S_1[2],S_2[1],S_3[3]\}$,也就是说,不同的子空间可能包含相同的覆盖.这是因为同一个元组可能同时包含多个关键词,从而使同一个元组可能出现在不同的 S_i 中.例 5 中,t_1,t_2 同时出现在 S_1 和 S_3 中,所以导致 $Cov(c_1)=Cov(c_2)$.事实上,如果两个聚合单元的覆盖完全相同,则这两个聚合单元必定相同,因此,

OQS 算法用 L 保存已经输出的聚合单元的覆盖. 对于新产生的聚合单元 c,只有与其覆盖相同的聚合单元没被输出 c 才会被输出(行⑫),这样就避免输出重复结果.

在算法 OQS 中,对内容元组评分(行③～④)的时间复杂度是 $O(m^2n)$,对内容元组集合进行排序(行⑥)的时间复杂度是 $O(n\log(n))$,其中,n 为表 T 中元组的个数. 因此,OQS 算法第 1 部分(行①～⑦)的时间复杂度是 $O(m(m^2n+n\log(n)))$. OQS 算法采用堆数据结构实现优先队列,插入元素的时间复杂度是 $O(\log(w))$,取最大的元素的时间复杂度为 $O(1)$,其中,w 为已经搜索到的聚合单元数量. 因此算法第 2 部分(行⑧～⑲)的时间复杂度为 $O(w\log(w))$. OQS 算法为了得到新的评分最高聚合单元,只需要搜索 $m-q'.start+1$ 个聚合单元,因此 w 的上限是 mk. 由此可得 OQS 算法的时间复杂度为 $O(m(m^2n+n\log(n))+w\log(w))$,即 $O(m(m^2n+n\log(n)))$. 在算法执行的过程中,需要记录 KTTD 索引(空间复杂度是 $O(mn)$)以及算法搜索到的聚合单元集合(空间复杂度是 $O(w)$)和已经输出的聚合单元的集合(空间复杂度是 $O(k)$),由于 $k\leqslant w$,并且 $w\leqslant mk$,因此算法的空间复杂度为 $O(mn+w)$,即 $O(mn)$. 由于 $m(m^2n+n\log(n))$ 远小于 $n^m\log(n^m)$,并且 mn 远小于 n^m,所以,OQS 算法的时间复杂度和空间复杂度都优于朴素算法.

2.2.3 构建 KTTD 索引

为了加快在线处理,提供高效的查询,我们设计了一种新的索引方式——KTTD. 对于每一个关键词 k_i,建立一个散列表 H,H 的键为关键词 k_i,H 的值也是一个 Hash 表 H',其中,H'的键为包含关键词 k_i 的元组 t,H'的值是元组 t 与 k_i 的相关度,即 $TF\times IDF(t,k_i)$(参见式(3)). 对于给定的关键词 k_i,这里用 KTTD(k_i)表示关键词 k_i 的内容元组的集合,KTTD(k_i,t)表示 $TF\times IDF(t,k_i)$. 采用该索引后,对于给定的关键词 k_i,获取其对应的内容元组 S_i 的时间复杂度为 $O(1)$,获取 k_i 与元组 t 的相关度的时间复杂度是 $O(1)$. KTTD 的结构如图 1 所示.

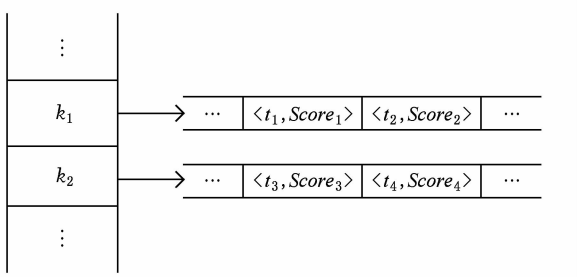


Fig. 1 A KTTD indexing example.

图 1 一个 KTTD 索引实例

KTTD 索引的创建可以采用一次扫描表的方式,具体方法是依次扫描表 T 中的每一个元组 t,对于 t 中的关键词 k_i,按照式(3)计算 k_i 和 t 的相关度评分,即 $TF\times IDF(t,k_i)$.

2.2.4 OQS 算法定理

下面给出算法 OQS 的正确性证明.

定理 2. 给定关键词查询 $K=\{k_i\mid 1\leqslant i\leqslant m\}$,OQS 算法可以输出与朴素算法相同的 top-$k$ 个具有最高评分的聚合单元.

证明. 这里采用数学归纳法证明. 由定理 1 可知,OQS 算法可以输出 top-1 个与朴素算法相同的具有最高评分的聚合单元. 假设对于任意 $k(k\geqslant 1)$,OQS 算法可以输出 top-k 个和朴素算法相同的具有最高评分的聚合单元,则 OQS 算法可以根据第 k 个聚合单元 c_k 对应的结构 q_k,将搜索空间分为 $m-q_k.start+1$ 个子空间 $V=\{V_i\mid 1\leqslant i\leqslant m-q_k.start+1\}$. 对于任意的 V_i,由推论 1 可知,OQS 算法可以得到 V_i 的最大聚合单元 c_i,并将 c_i 对应的 q_k^i 添加到 Q 中. 因此,OQS 算法的第 $k+1$ 个聚合单元 c_{k+1} 对应的结构 $q_{k+1}=\max Q$,由于 OQS 算法和朴素算法搜索空间都是 $S_1\times S_2\times S_3$,并且,除去已经找到的 top-k 个聚合单元,两种算法的剩余搜索空间 V' 完全相同,因此,c_{k+1} 也是朴素算法 V' 的最大聚合单元,也就是说,c_{k+1} 也是朴素算法的第 $k+1$ 个聚合单元. 综上可知,OQS 算法输出与朴素算法相同的 top-k 个具有最高评分的聚合单元. 证毕.

3 实验设计和结果

我们在两个真实数据集上进行了一系列实验. 实验硬件环境为 3.1 GHz Intel® Core™ i3-2100 CPU;2 GB 内存以及 500 GB 硬盘. 实验所有程序采用 Java 语言编写;数据库为 MYSQL;操作系统为 Windows Server 2003. 首先,下载了完整的 IMDB 数据库[16],并抽取其中的 8 个属性构成 IMDB 表,其结构如表 2 所示. 该表总共包含 1 181 024 个元组,提取得到 117 655 个关键词,建立 KTTD 索引的时间为 86.2 s,索引中元素个数为 16 715 072. 其次,从 newegg.com 网站上抓取所有手机信息,抽取其中 8 个属性构成 Phone 表,其结构如表 3 所示. 该表总共包含 227 295 个元组和 21 390 个关键词,其中,每个元组平均包含 21.4 个关键词,建立 KTTD 索引的时间为 32.9 s,索引中元素个数为 4 864 113.

Table 2 The Schema of IMDB

表 2 IMDB 表结构

Attribute	Description
Movie	Movie title
Director	Director of the movie
Country	Producing country of the movie
Language	Language of the movie
Year	Producing year of the movie
Genre	Genres of the movie
Keyword	Keywords of the movie
Technical	Technical of the movie

Table 3 The Schema of Phone

表 3 Phone 表结构

Attribute	Description
Form Factor	Form factor of the phone
Brand	Brand of the phone
Color	Color of the phone
OS	Operating System of the phone
Processor	Processor of the phone
Memory	Internal memory of the phone
Specifications	Specifications of the phone
Reviews	Reviews of the phone

3.1 算法效率分析

我们在 IMDB 和 Phone 数据集上测试算法 RFS 和 OQS 的 CPU 时间和内存消耗,为了便于比较,本文用算法搜索到的聚合单元数代表算法消耗的内存.对于每一组实验,从每一个数据集中随机选择 100 个关键词查询.因为已经存在的基于关系数据库关键词查询方法不能直接用于解决聚合关键词查询问题[15],所以,本文与文献[15]中的 Fast 算法作比较.但是文献[15]并没有对结果进行排序,因此,实验采用 2.1.2 节的评分机制对结果进行评分,从而得到 top-k 个聚合单元.评分过程并不影响文献[15]的算法效率,所以,对其评分只改变结果的排列顺序.

3.1.1 算法时间复杂度分析

我们首先比较 3 种算法的时间需求与查询关键词个数和输出结果数量之间的关系.图 2 和图 3 分别是 IMDB 数据集和 Phone 数据集上的实验结果.由图 2 和图 3 可知,RFS 算法枚举同样数量的聚合单元需要最多的时间,这是因为 RFS 算法需要遍历全部聚合单元,而其余 2 种算法则都只需要遍历部分聚合单元.例如,在 IMDB 数据集中(如图 2(b)所示).当输入 3 个查询关键词时,OQS 算法枚举 top-10 聚合单元的时间是 38.4 ms,Fast 算法需要 18 907 ms,而 RFS 算法需要 872 390 ms.也可以看出,Fast 算法枚举同样个数的结果比 RFS 算法更快,这是因为 Fast 算法采用 3 种策略对搜索空间进行剪枝.但是,Fast 算法和 RFS 算法都需要较长的时间,主要原因是为了得到 top-k 个结果,它们需要搜索大量的聚合单元,而 OQS 算法只需要根据已经找到的 k 个结果,最多搜索 m 个聚合单元就可以产生第 $k+1$ 个结果,因此 OQS 算法效率最高.

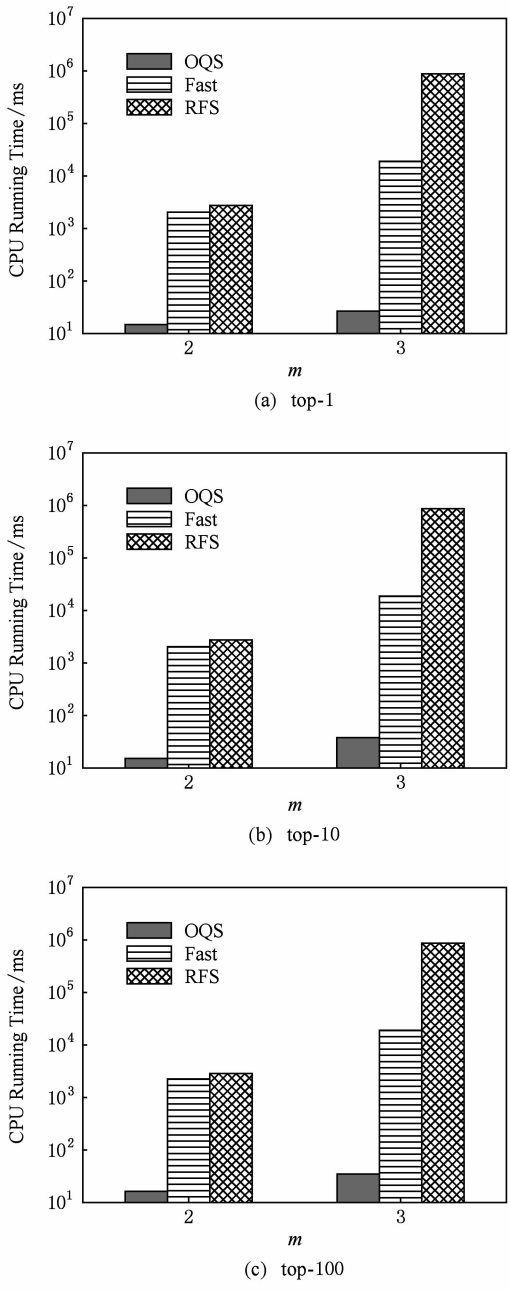
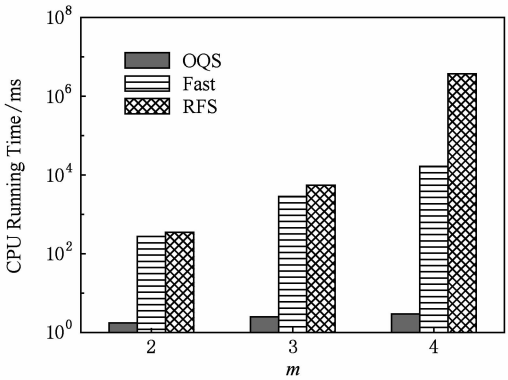
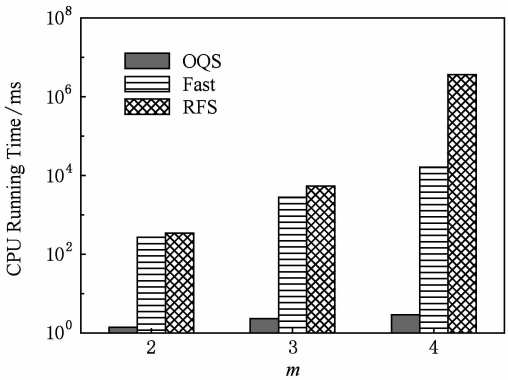


Fig. 2 Search efficiency on IMDB.

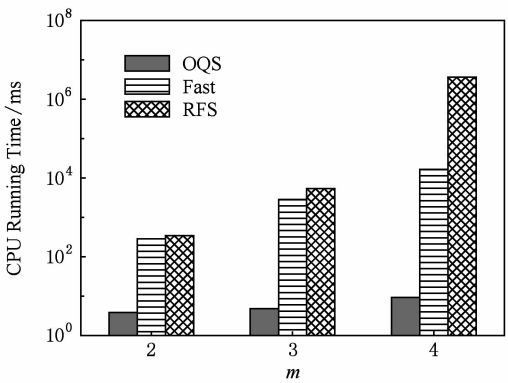
图 2 IMDB 数据集的查询效率



(a) top-1



(b) top-10



(c) top-100

Fig. 3 Search efficiency on Phone.

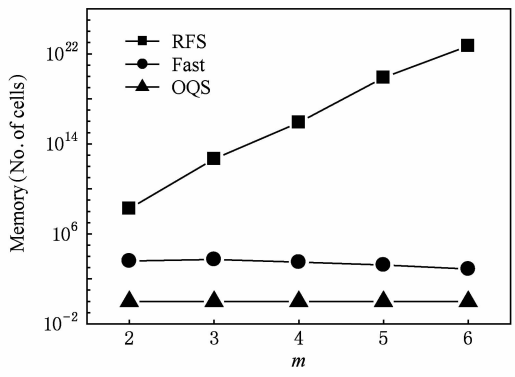
图 3 Phone 数据集的查询效率

同时,我们发现随着关键词查询个数的变化,Fast 算法和 RFS 算法消耗的时间迅速增加,而 OQS 算法则变化不大. 例如,在 Phone 数据集上(如图 3(b)所示),关键词个数从 2 增大到 3 时,Fast 算法产生 top-10 个结果消耗的时间,从 274 ms 增大到 2 820 ms,RFS 算法消耗的时间从 350 ms 增大到 5 443 ms. 这是因为 Fast 算法和 RFS 算法随着查询关键词个数的增加,其时间复杂度呈指数增长,而 OQS 算法则是多项式时间的增长. 这也与 OQS 算法分析中的时间复杂度吻合.

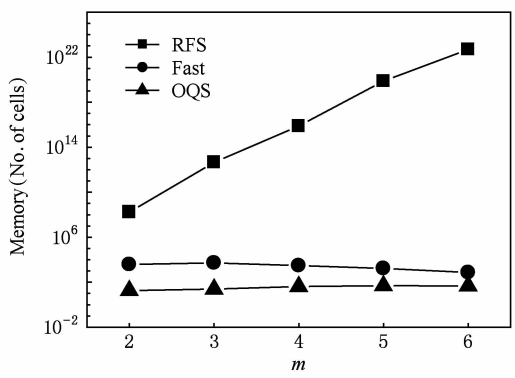
需要说明的是,由于 Fast 算法和 RFS 算法的时间瓶颈主要在搜索聚合单元的空间,因此我们看到随着输出结果的增加,它们消耗的时间并没有明显的变化. 例如,在 Phone 数据集中,当关键词个数为 2 时,产生 top-1,top-10 和 top-100 个结果,Fast 算法消耗的时间依次是 274 ms,274 ms 和 175 ms,RFS 算法消耗的时间依次是 350 ms,351 ms 和 351 ms.

3.1.2 算法空间复杂度分析

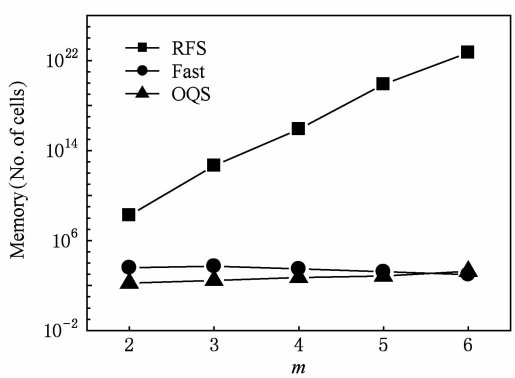
为了观察 3 种算法的空间需求,我们把查询关键词的个数从 2 连续变化到 6,比较枚举 top-k 个聚合单元的内存使用情况. 图 4 和图 5 分别显示了在



(a) top-1



(b) top-10



(c) top-100

Fig. 4 Memory usage for IMDB.

图 4 IMDB 数据集的内存使用情况

IMDB 数据集和 Phone 数据集上 3 种算法的内存使用情况. 从图 4 和图 5 可以看出,对于相同个数的查询关键词,RFS 算法消耗的内存空间最大,主要原因是 RFS 算法需要遍历全部的聚合单元. 例如,在 IMDB 数据集中,当输入 2 个关键词时产生 top-10 个结果,OQS 算法需要 18.1 个聚合单元空间,Fast 算法需要 3 876 个,而 RFS 算法则需要 1.7×10^8 个. 随着查询关键词数量的增加,RFS 算法和 OQS 算法的都需要更多的内存,这是因为 RFS 算法的搜索空间迅速扩大,OQS 算法遇到的重复结果也会增加. 同时,OQS 算法的增长速度远小于 RFS 算法.

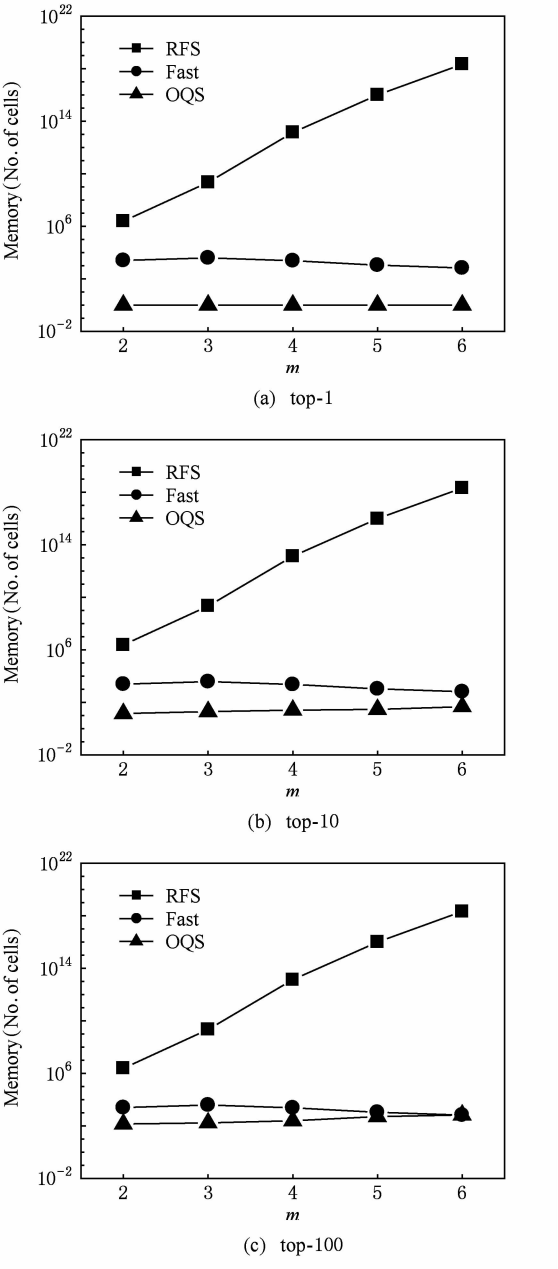


Fig. 5 Memory usage for Phone.

图 5 Phone 数据集的内存使用情况

这是由于 RFS 算法只有遍历所有的聚合单元才能输出 top-k. 例如,在图 4(b)中,当关键词个数为 3 时,OQS 算法需要 24.8 个聚合单元的空间,而 RFS 算法则需要 4×10^{12} 个. 当关键词个数为 4 时,RFS 算法的需要的内存空间迅速上升到 7×10^{15} 个,此时 OQS 则仅仅需要 42 个. 我们也发现,Fast 算法的剪枝策略与关键词个数和输出结果个数相关,因此,只是关键词个数变化时,其对应的经过剪枝后的聚合单元数量并不是有规律的变化.

3.2 输出结果数量对算法的影响

为了更好地分析 OQS 算法与输出结果的关系,我们查看其枚举 top-1,top-10 和 top-100 所消耗的 CPU 时间. 图 6 显示了 OQS 算法在两个数据集上输出不同数量结果的查询效率. 从图 6 可以看出,随着关键词个数的增加,OQS 算法需要更长的时间枚举 top-k 个聚合单元. 因为,每增加一个关键词 k_i,算法需要多维护一个包含 k_i 的内容元组集合,并且需要对其进行评分和排序. 例如,在 IMDB 数据集上查询 top-100 个结果,当输入 2 个查询关键词时,OQS 算法消耗 10.6 ms,当关键词个数增大

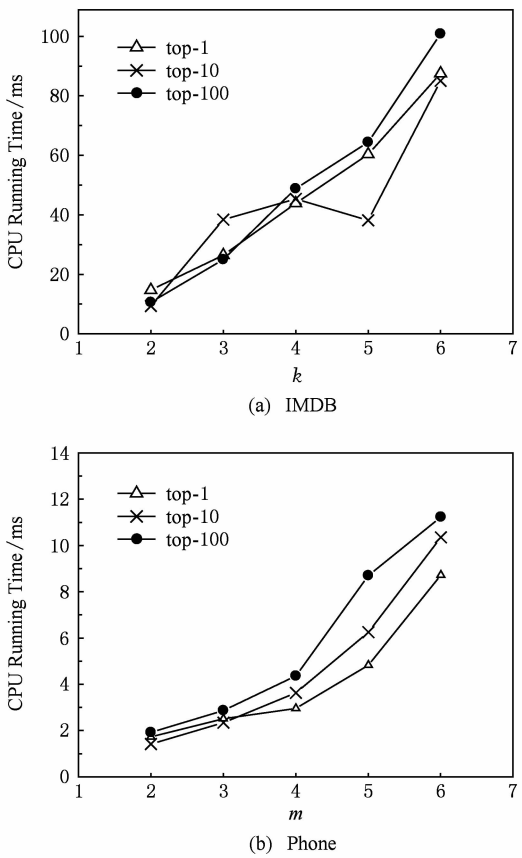


Fig. 6 The influence of output number upon algorithm efficiency.

图 6 输出结果数量对算法效率的影响

到 3 时,OQS 算法消耗的时间增大到 24.9 ms. 同时,从图 6 也可以看出,当查询相同个数的关键词时,OQS 算法生成不同数量聚合单元所耗费的时间并没有显著的不同,主要原因是,OQS 算法的瓶颈是对每个关键词对应的内容元组集合的排序和评分,而与输出结果数量关系不大. 例如,在 IMDB 数据集上,当查询 4 个输入关键词时,OQS 算法产生 top-1,top-10 和 top-100 的时间依次是 43.88 ms, 45.35 ms 和 48.82 ms.

4 相关工作

已经存在的基于关系数据库的关键词查询算法整体可以分为 3 类:基于数据图的方法[1,9-11]、基于模式图的方法[2,12]和基于元组的方法[13-14]. 基于数据图的方法把数据库看成一个带权重的数据图,数据库元组作为图的节点,元组之间的主外键关联作为边,给定关键词查询,返回包含关键词的元组连接树. 基于模式图的方法首先生成候选元组连接树,然后对候选结果进行评估,进而得到 top-k 个结果. 基于元组的方法利用关系数据本身的功能,利用视图得到元组单元,并且对元组单元进行物化和索引,采用简单的 SQL 语句就可以得到包含关键词的查询结果. 其中元组单元是元组的集合,它是一个完整的信息单元,因此它比单个元组更加有意义.

BANKS[1]采用反向扩展搜索算法对数据图进行搜索,查询包含全部关键词的 Steiner 树. BLINKS[9]提出基于代价均衡扩展的反向搜索算法,并且构建双层索引,加快了搜索速度. 由于 Steiner 树无法适应生物数据库、环路等复杂数据图,因此 EASE[3]搜索“R-半径 Steiner 树”,把关键词查询问题转化为“R-半径 Steiner 树”问题. 由于 Steiner 树可能丢失部分相关信息,并且给定关键词查询,用户很难从大量结果中找到真正需要的信息,因此文献[5]提出“社区”的概念,查询更加有意义的子图结构. DBXplorer[2],DISCOVER[12]等文献采用了基于模式图的方法,它们很好地利用了数据库模式,提高了算法效率,但是忽略了用户的需求. 已有算法并不能用来解决聚合关键词查询问题. 首先,已有方法的依赖表之间的关联关系,而不能充分挖掘单个数据库表的内部信息. 其次,已有方法搜索包含关键词具有一定结构的元组集合,不能获得元组集合的聚合结果.

文献[15]研究聚合关键词的查询,它与本文非常相似. 对于输入的关键词集合,它查询覆盖所有关键词的最小分组. 但是文献[15]并没有对查询结果进行评分和排序,而是输出所有相关的结果,因此当结果较多时它并没有时间复杂度的上限,并且没有排序的大量结果对用户也是没有意义的. 基于关键词的分析处理 KDAP[18]采用两阶段框架方法来解决基于关键词的 OLAP 查询,但是并没有考虑效率问题.

与此同时,有许多研究[7-8]也关注基于 XML 的关键词查询问题. 已有方法用树结构描述 XML 文档,给定关键词查询,它们查询最小代价子树(LCAs). 而本文的方法也可以扩展,用以支持基于 XML 的 top-k 聚合关键词查询:首先,对于一个 XML 文档,通过扫描建立二维索引;其次,给定关键词查询 $K=\{k_1,k_2,\cdots,k_m\}$,由索引即可以得到每一个关键词对应的内容节点的集合;最后,采用 OQS 算法的思路获得 top-k 聚合单元. 也就是说,本文的方法同样适用基于 XML 数据库的 top-k 聚合关键词查询.

5 结 论

本文研究基于关系数据库的 top-k 聚合关键词查询. 考虑到关系数据库具有丰富结构性的特点,采用关系数据库评分和 IR 评分相结合的方式,对聚合单元进行评分,从而使查询结果能够更好地满足用户需求. 同时,提出了 KTTD 索引,并设计了具有多项式时间复杂度的算法 OQS,用以枚举 top-k 个聚合单元,在不同数据集上的大量实验表明,本文的算法可以取得很好的性能. 在未来的工作中,我们将继续改进算法,研究更加高效的解决方法和更友好的结果呈现方式.

参 考 文 献

[1] Bhalotia G, Hulgeri A, Nakhe C, et al. Keyword searching and browsing in databases using BANKS [C] //Proc of the 18th Int Conf on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2002: 431-440

[2] Agrawal S, Chaudhuri S, Das G. DBXplorer: A system for keyword-based search over relational databases [C] //Proc of the 18th Int Conf on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2002: 5-16

[3] Li Guoliang, Ooi B C, Feng Jianhua, et al. EASE: An effective 3-in-1 keyword search method for unstructured, semi-structu red and structured data [C] //Proc of the 2008 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2008: 903-914

[4] Wen Jijun, Wang Shan. SEEKER: Keyword-based information retrieval over relational databases [J]. Journal of Software, 2005, 16(7): 1270-1281 (in Chinese)
(文继军, 王珊. SEEKER: 基于关键词的关系数据库信息检索[J]. 软件学报, 2005, 16(7): 1270-1281)

[5] Qin Lu, Yu J X, Chang Lijun, et al. Querying communities in relational databases [C] //Proc of the 25th Int Conf on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2009: 724-735

[6] Tao Yue, He Zhenying, Zhang Jiaqi. Keyword queries over relational databases based on tuple combination [J]. Journal of Computer Research and Development, 2011, 48(10): 1890-1898 (in Chinese)
(陶岳, 何震瀛, 张家琪. 关系数据库上基于元组组合的关键字查询[J]. 计算机研究与发展, 2011, 48(10): 1890-1898)

[7] Guo Lin, Shao Feng, Botev C, et al. Xrank : Ranked keyword search over XML documents [C] //Proc of the 2003 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2003: 16-27

[8] Xu Yu, Papakonstantinou Y. Efficient LCA based keyword search in XML data [C] //Proc of the 11th Int Conf on Extending Database Technology. New York: ACM, 2008: 535-546

[9] He Hao, Wang Haixun, Yang Jun, et al. Blinks: Ranked keyword searches on graphs [C] //Proc of 2007 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2007: 305-316

[10] Ding Bolin, Yu JX, Wang Shan, et al. Finding top-k min-cost connected trees in databases [C] //Proc of the 23rd Int Conf on Data Engineering. Los Alamitos, CA: IEEE Computer Society, 2007: 836-845

[11] Wang Shan, Peng Zhaohui, Zhang Jun, et al. NUITS: A novel user interface for efficient keyword search over databases [C] //Proc of the 32nd Int Conf on Very Large Data Bases. New York: ACM, 2006: 1143-1146

[12] Hristidis V, Papakonstantinou Y. DISCOVER: Keyword search in relational databases [C] //Proc of the 28th Int Conf on Very Large Data Bases. New York : ACM, 2002: 670-681

[13] Li Guoliang, Feng Jianhua, Zhou Lizhu. Retune: Retrieving and materializing tuple units for effective keyword search over relational databases [C] //Proc of the 27th Int Conf on Conceptual Modeling. Berlin: Springer, 2008: 469-483

[14] Feng Jianhua, Li Guoliang, Wang Jianyong. Finding top-k answers in keyword search over relational databases using tuple units [J]. IEEE Trans on Knowledge and Data Engineering, 2011, 23(12): 1781-1794

[15] Zhou Bin, Pei Jian. Aggregate keyword search on large relational databases [J]. Knowledge and Information Systems, 2012, 30(2): 283-318

[16] IMDB. IMDB Dataset [OL]. [2012-01-07]. http://www.imdb.com/interfaces

[17] Liu Fang, Yu C, Meng Weiyi, et al. Effective keyword search in relational databases [C] //Proc of the 2006 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2006: 563-574

[18] Wu Ping, Yannis S, Berthold R. Towards keyword-driven analytical processing [C] //Proc of the 2007 ACM SIGMOD Int Conf on Management of Data. New York: ACM, 2007: 617-628

Zhang Dongzhan, born in 1974. Received his PhD degree in computer software and theory from Beijing Institute of Technology in 2003. Now he is associate professor and master supervisor at Xiamen University. Member of China Computer Federation. His main research interests include data warehouse, OLAP, data mining (zdz@xmu.edu.cn).

Su Zhifeng, born in 1988. Postgraduate in computer science in Xiamen University, Fujian, China. His current research interests include relational database (rocsky2010@gmail.com).

Lin Ziyu, born in 1978. Received his PhD degree in computer software and theory from Peking University in 2009. Now he is assistant professor and master supervisor at Xiamen University. Member of China Computer Federation. His main research interests include data warehouse, OLAP, data mining, etc.

Xue Yongsheng, born in 1946. Now he is professor at Xiamen University. Member of China Computer Federation. His main research interests include data warehouse and data mining (ysxue@xmu.edu.cn).