软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2014,25(3):528−546 [doi: 10.13328/j.cnki.jos.004384]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

关系数据库中的关键词查询结果动态优化*

林子雨[1], 邹 权[1], 赖永炫[2], 林 琛[1]

[1](厦门大学 计算机科学系,福建 厦门 361005)

[2](厦门大学 软件学院,福建 厦门 361005)

通讯作者: 林子雨, E-mail: ziyulin@xmu.edu.cn

摘 要: 关键词查询可以帮助用户从数据库中快速获取感兴趣的内容,它不需要用户掌握专业的数据库结构化查询语言,降低了使用门槛.针对基于关键词的数据库查询,基于数据图的方法是一种比较常见的方法,它把数据库转换成数据图,然后从数据图中计算最小Steiner树.但是,已有的方法无法根据不断变化的用户查询兴趣而动态优化查询结果.提出采用蚁群优化算法解决数据库中的关键词查询问题,并提出了基于概念漂移理论的用户查询兴趣突变探查方法,可以及时发现用户兴趣的突变.在此基础上,提出了基于概念漂移理论和蚁群优化算法的查询结果动态优化算法 ACOKS*,可以根据突变的用户兴趣,动态地优化查询结果,使其更加符合用户查询预期.在原型系统上得到的大量实验结果表明,该方法具有很好的可扩展性,并且可以比已有的方法取得更好的性能.

关键词: 关键词查询;关系数据库;数据图;蚁群优化;Steiner树

中图法分类号: TP311 **文献标识码**: A

中文引用格式: 林子雨,邹权,赖永炫,林琛.关系数据库中的关键词查询结果动态优化.软件学报,2014,25(3):528−546. http://www.jos.org.cn/1000-9825/4384.htm

英文引用格式: Lin ZY, Zou Q, Lai YX, Lin C. Dynamic result optimization for keyword search over relational databases. Ruan Jian Xue Bao/Journal of Software, 2014,25(3):528−546 (in Chinese). http://www.jos.org.cn/1000-9825/4384.htm

Dynamic Result Optimization for Keyword Search over Relational Databases

LIN Zi-Yu[1], ZOU Quan[1], LAI Yong-Xuan[2], LIN Chen[1]

[1](Department of Computer Science, Xiamen University, Xiamen 361005, China)

[2](School of Software, Xiamen University, Xiamen 361005, China)

Corresponding author: LIN Zi-Yu, E-mail: ziyulin@xmu.edu.cn

Abstract: Keyword search helps users to efficiently get interested information from relational databases, and users are exempted from learning the professional structural query language for relational databases, which greatly reduces the usabilitye threshold. Keyword search over relational databases commonly employs data-graph-based methods which first models a database into a graph and then uses it to identify the minimum Steiner tree. However, the available methods are not able to dynamically optimize query results according to the dynamically changing user interest. In this paper, an ant-colony-optimization-based algorithm is proposed to achieve the task of keyword search over relational databases. Furthermore, a novel approach based on the theory of concept drift is presented to capture the mutation of user interest. In addition, based on concept drift theory and ant colony optimization algorithm, a new algorithm called ACOKS* is proposed to dynamically optimize the search results according to the time-changing user interest, so as to achieve the results in more accordance with user interest. Finally, a prototype is developed to carry out extensive experiments, and the results show that our method can achieve high scalability and perform better than other state-of-the-art methods.

Key words: keyword search; relational database; data graph; ant colony optimization; Steiner tree

* 基金项目: 国家自然科学基金(61303004, 61370010, 61102136, 61202012); 福建省自然科学基金(2013J05099, 2011J05156, 2011J05158); 厦门大学基础创新科研基金(中央高校基本科研业务费专项资金)(2011121049)

收稿时间: 2011-12-09; 修改时间: 2012-11-06; 定稿时间: 2013-02-05

关系数据库在经历多年的发展后,已经是一种非常成熟的数据存储和管理技术,在各个领域得到广泛的应用.在关系数据库中,数据存储在“关系”中,具有规则的数据结构.因此,这类数据通常被称为结构化数据.针对存在于关系数据库中的结构化数据,可以借助于功能强大的结构化查询语言(比如 SQL 语句)来查找满足特定要求的记录集合.由于已经存在很多的针对 SQL 查询的性能优化算法,因此,利用 SQL 语句进行查询可以获得很高的性能.但是,SQL 查询也有一个明显的缺陷,那就是要求用户必须具有一定的数据库专业知识,能够熟练掌握 SQL 语句的书写方法,并且需要了解底层的数据结构.很显然,这对于普通用户而言,是一个不小的障碍,因而也就无法灵活地为普通用户提供个性化的查询服务.与此同时,为了适应激烈的市场竞争,企业越来越需要普通员工能够灵活访问企业数据库中的各种数据,提升服务质量和工作效率,因而需要开发面向普通用户的关系数据库数据查询技术.

关键词查询可以很好地解决 SQL 查询面临的这个问题.关键词查询是互联网中普遍采用的信息检索方式(比如 Google 和百度),它不需要用户具备任何专业知识,只需要用户提供查找的关键词,系统就可以为用户返回相关的查询结果.受此启发,研究人员开始采用关键词进行关系数据库查询的各种探索工作.

对于关系数据库的关键词查询,绝大多数研究都可以归为两大类,即基于数据图的方法(比如文献[1–4])和基于模式图的方法(比如 DBXplorer[5]、DISCOVER[6]以及文献[7]等).前者把数据库建模成数据图,然后从数据图中寻找包含查询关键词的多棵 Steiner 树[8],经过评分排序后,输出 top-k 棵 Steiner 树作为查询结果;后者利用数据库模式创建连接表达式,然后在 DBMS 上执行连接表达式对应的 SQL 语句得到结果.基于模式图的方法只能应用于关系型数据的搜索;而基于数据图的方法则可以用于关系型、XML 和 HTML 数据,因为这些数据都可以用数据图的形式表示.

但是已有的研究存在一个明显的不足:无法根据不断变化的用户查询兴趣而动态地优化查询结果.在日常的应用中,用户查询的兴趣有时候会发生突然变化,比如发生了突发事件,会迅速转移大量用户的查询兴趣.针对这种突变的用户兴趣,不能使用基于长期的概率统计来衡量.比如用户的查询兴趣突然从 A 转到 B,即在查询结果中突然更多选择 B 而很少选择 A,导致 B 的访问量急剧增加.但是由于历史上有很多用户曾经访问过 A,使得 A 的绝对访问次数远远高于 B.因此,如果仍然采用基于长期概率统计的方法,就会给出错误的查询结果.因此,必须研究一种可以有效衡量用户查询兴趣突变的方法,并设计相应的查询结果动态优化算法.对于已有的研究方法,每次查询都是根据事先确定的评分排序(score and rank)方法对大量候选结果进行排序,生成 top-k 个结果推荐给用户,用户的访问情况不会对这些算法产生动态的影响,自然也就无法实现根据用户查询兴趣的动态查询优化.因此,本文提出一种高效的、基于概念漂移和蚁群优化算法的关键词查询新方法,可以根据突变用户兴趣而不断地动态优化查询结果.

需要指出的是,本文的研究重点在于采用概念漂移理论解决查询结果优化问题,而不在于如何采用蚁群优化算法提高查询响应速度.本文的关键是发现了蚁群算法和概念漂移理论的巧妙结合,可以较好地解决用户查询结果动态优化问题.首先,本文使用蚁群算法解决从数据图中搜索 Steiner 树(查询结果)的问题;其次,采用概念漂移理论跟踪用户兴趣变化,指导蚁群优化算法的结果生成过程,使得查询结果更加符合用户预期.实际上,采用蚁群算法得到查询结果的过程,就是数据图中各个节点上的蚂蚁爬向目标节点(包含查询关键词)的过程,而目标节点就是用户的查询兴趣所在,因为包含了用户查询关键词.用户查询兴趣的变化就会导致目标节点的变化,概念漂移理论可以有效跟踪用户的当前兴趣节点,从而改变蚂蚁的爬行目标,动态给出不同的、更加满足用户需求的查询结果.因此在本文中,蚁群优化算法是基础,采用概念漂移理论探测用户兴趣变化和指导蚁群算法执行过程是核心内容.

总的来说,本文的主要贡献是:

- 提出了基于蚁群优化算法的、关系数据库关键词查询新算法 ACOKS(ant-colony-optimization-based keyword search),为基于概念漂移理论的查询结果动态优化提供基础;
- 提出了基于概念漂移理论的用户查询兴趣突变探查方法,可以及时发现用户兴趣的突变;
- 提出了基于概念漂移理论和 ACOKS 算法的查询结果动态优化算法 ACOKS*,可以根据突变的用户兴

趣,动态优化查询结果,使其更加符合用户查询预期;

- 构建了原型系统并进行了大量实验,结果表明,本文的方法可以比已有的方法取得更好的性能.

本文第 1 节给出问题描述.第 2 节详细阐述如何用蚁群优化算法解决关键词查询问题,描述了 ACOKS 算法.第 3 节介绍基于概念漂移和 ACOKS 算法的查询结果动态优化算法 ACOKS*.第 4 节展示大量实验结果,从而证明本文方法的良好性能.第 5 节讨论本研究领域的主要相关工作.最后,第 6 节总结并展望未来的研究方向.

1 问题描述

关系数据库中的关键词查询的核心思想,就是从数据库中枚举简化子树.枚举简化子树的算法主要包括两类:基于数据图的方法和基于模式图的方法.由于本文采用基于数据图的方法,因此这里将描述采用这类方法如何解决关系数据库中的关键词查询问题.下面首先给出数据图[8]的定义.

定义 1(数据图). 数据图 G 包含一个节点集合和一个边的集合.图 G 中存在两种类型的节点,即结构化节点和关键词节点.关键词节点只有入射边,而结构化节点既有入射边也有出射边.因此,一条边不可以连接两个关键词.图 G 中存在两种类型的边:一种是前向边(u,v),表示节点 u 和 v 之间存在主外键关联;一种是后向边(v,u),它与边(u,v)相对应,并且只有图中存在前向边(u,v)时才存在对应的后向边(v,u).一个数据图 G 的边可以具有权重,权重函数 w_G 为每条边 e 分配一个正的权重 $w_G(e)$.数据图 G 的权重用 $w(G)$表示,就是图 G 中所有边的权重之和.一个数据图 G 是有根的,如果它包含某个节点 r,并且从图 G 中的任何节点都可以通过一条有向路径到达节点 r,这个节点 r 就被称为图 G 的根.

数据图的边具有方向性,可以反映不同方向上连接的强弱,因为两个节点之间的连接在不同方向上的连接强度不是对称的.比如在反映主外键关联的边中,外键到主键方向的边和其反方向的边是具有不同的重要性的.数据图中的节点和边都可以具有权重,这些权重都是预先被赋值的,从而可以更好地支持关键词查询.本文采用了 BANKS[1]中的方法确定权重值.节点 u 的权重 $w(u)$是一个关于节点 u 的入度的函数,边(u,v)的权重是根据边的类型来确定的,比如对于一个反映主外键关系的前向边(u,v),权重 $w(u,v)$为 1;而对于一个后向边(v,u)而言,其权重 $w(v,u)=w(u,v)\times\log_2(1+D_{in}(v))$,其中,$D_{in}(v)$表示节点 v 的入度.

例 1:如图 1 所示,数据库中包含 4 个表,分别是 Author,Paper,Citation 和 Paper-Author.其中:Author 表记录了作者的标识(AID)和姓名(name);Paper 表记录了论文的标识(PID)和标题(title);Paper-Author 表记录了论文(PID)和作者(AID)之间的对应关系,PID 和 AID 都是外键,分别引用了 Paper 表的 PID 属性和 Author 表的 AID 属性;Citation 表记录了引用论文(cite)和被引用论文(cited)之间的对应关系,Cite 和 Cited 也都是外键,都引用了 Paper 表的 PID 属性.

Author

AID	Name
a_1	Jim
a_2	Kate

Citation

Cite	Cited
t_1	t_2
t_2	t_3
t_4	t_3
t_6	t_5

Paper

PID	Title
t_1	Spatial database
t_2	Structural query language
t_3	XML database
t_4	Steiner tree problem
t_5	Simple query interface
t_6	Web database schema

Paper-Author

PID	AID
t_1	a_1
t_2	a_1
t_3	a_1
t_5	a_1
t_5	a_2
t_6	a_2

Fig.1 An example of database

图 1 一个数据库实例

图 2 显示了图 1 中的数据库所对应的数据图,这里不包括边的方向和权重.

目前,大多数基于数据图的方法[1,4]都把简化子树枚举问题看成 Steiner 树问题,即从数据图中寻找包含了所有关键词的 Steiner 树.这里给出一个关于 Steiner 树的简单例子.假设有 4 个查询关键词 k_1=database,k_2=XML,k_3=Jim,k_4=Steiner,我们使用这 4 个关键词在图 2 的数据库中进行搜索,就可以得到如图 3 所示的一棵可能的

Steiner 树,它包含了全部的 4 个关键词,从图 3 中可以看出,树中每个节点所包含的关键词的具体情况.

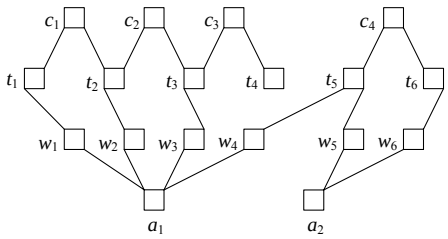


Fig.2 An example of data graph

图 2 一个数据图的实例

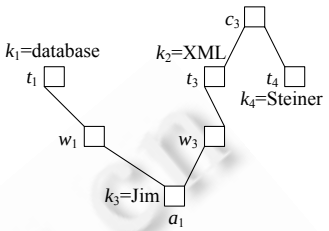


Fig.3 An example of Steiner tree

图 3 Steiner 树实例

在关系数据库中,关键词查询的一个最优结果,就是数据库对应的数据图中包含关键词的一棵最小 Steiner 树.最小 Steiner 树问题的任务如下:对于给定的顶点集合 V,使用一个最短的路径把这些顶点连接起来,其中,路径的总长度是各条边的路径的长度之和.Steiner 树问题和最小生成树问题看起来十分相似,但是二者有本质的不同,根本区别在于:在 Steiner 树问题中,为了减少生成树的路径总长度,可以在图中增加额外的顶点(不属于 V)和边,这些不属于集合 V 的顶点,通常被称为 Steiner 点.下面给出最小 Steiner 树问题[4,9]的形式化定义.

定义 2(最小 Steiner 树). 给定一个图 $G(V,E)$以及 $V'\subseteq V$,如果 T 是图 G 中的一棵连通子树,并且 T 包含了 V' 中的所有节点,那么我们就说 T 是图 G 中关于 V'的一棵 Steiner 树.假设用 $c(T)=\sum_{e\in E(T)} w(e)$ 表示 T 的代价,其中,$E(T)$表示树 T 中边的集合,$w(e)$表示边 e 的权重.如果在所有图 G 中关于 V'的 Steiner 树中 $c(T)$最小,我们就说 T 是一棵最小 Steiner 树.

最小 Steiner 树的一种扩展形式是最小分组 Steiner 树,关键词查询问题也可以看成最小分组 Steiner 树问题.

定义 3(最小分组 Steiner 树). 给定一个图 $G(V,E)$和分组 $V_1,V_2,\ldots,V_n\subseteq V$,如果 T 是一棵最小 Steiner 树,并且 T 中包含了分组 $V_i(1\leqslant i\leqslant n)$中至少一个节点,则称 T 是图 G 中关于这些给定分组的最小分组 Steiner 树.

下面我们来解释关键词查询问题为何可以看成是最小分组 Steiner 树问题.假定一个查询包含 m 个关键词 $k_1,k_2,\ldots,k_m$.为了回答这个查询,第一步工作就是定位与查询关键词匹配的节点,这个可以通过符号表技术或者数据库内嵌的全文索引功能来获得.对于查询中的每个关键词 k_i,都可以找到和 k_i 相关的节点集合 S_i.一个查询的答案是一棵有根有向树,并且树中包含了来自每个 S_i 的至少一个节点.很显然,这里的集合 S_i 就对应定义 3 中的分组 V_i,因此,关键词查询的最优结果就是一棵最小分组 Steiner 树.

最小分组 Steiner 树问题可以给出代价最小(与用户给定的关键词集合 K 相关性最大)的简化子树,当需要返回 top-k 个代价最小的简化子树时,这个问题被称为 top-k 分组 Steiner 树问题,具体定义如下:

定义 4(top-k 分组 Steiner 树问题). 给定一个关键词查询 $k_1,\ldots,k_m$,为该查询寻找 top-k 个代价最小的分组 Steiner 树 $T_1,T_2,\ldots,T_k$,这些树之间根据代价函数 c 进行排序,并且有 $c(T_1)\leqslant c(T_2)\leqslant\ldots\leqslant c(T_k)$.

2 用蚁群优化算法求解关键词查询问题

本节内容将首先介绍蚁群优化算法的基本原理,然后介绍如何用蚁群优化算法求解关键词查询问题,并详细阐述基本算法以及其中所涉及的蚂蚁的移动规则、信息素的更新等问题,最后给出了算法的空间代价分析.需要指出的是,本文采用了与 BANKS 一样的评分方法,因此这里不再介绍评分方法.

2.1 蚁群优化算法的基本原理

蚁群算法[10]是意大利学者 Dorigo 等人于 20 世纪 90 年代提出来的,是一种模仿自然界蚂蚁搜索食物行为的启发式智能进化算法,可以用来在图中寻找优化路径.自然界中的蚂蚁在寻找食物的时候,事先没有任何关于食物位置的信息,每只蚂蚁在开始寻找食物的过程中,对路径的选择会表现出随机的行为.一旦一只蚂蚁找到食

物以后,它就会向环境中释放信息素,其他蚂蚁遇到信息素以后,就会沿着信息素指引的方向寻找食物,最终会有越来越多的蚂蚁找到食物.由于不同蚂蚁会发现不同的食物寻找路径,并且会分别释放出信息素,信息素的浓度与路径长度相关,路径越短,信息素浓度越大.蚂蚁在前进时,会优先选择信息素浓度大的方向移动.因此经过一段时间以后,就可能会出现一条最短路径被大多数蚂蚁重复着.所以总的来说,单只蚂蚁的行为都非常简单,但是不同蚂蚁之间通过信息素进行沟通合作,却可以快速发现全局最优解.

蚁群优化算法最初主要用于解决旅行商问题(traveling salesman problem,简称 TSP),并且取得了很好的效果.此后,经过研究人员的努力,产生了大量的研究成果[11],蚂蚁算法也进一步发展成为通用的优化技术,即蚁群优化(ant colony optimization,简称 ACO)算法,适合解决传统搜索算法难以解决的一些复杂问题.蚁群优化算法具有正反馈性,个体蚂蚁通过信息素把路径信息传递给环境,而周围环境反过来又通过信息素指导蚂蚁进行最优的路径选择.这个性质意味着,问题的解越优,通过调整相关参数值,就可以增强它被找到的概率.蚁群算法的另一个特性是适合于分布式环境.

2.2 用蚁群优化算法求解关键词查询问题

关系数据库中的关键词查询问题,可以转化为 Steiner 树问题.已有的研究证明,Steiner 树问题是一个 NP-hard 问题[1],因此,已有的研究通常都采用启发方法来减少搜索空间.研究人员通过大量实验发现,蚁群算法对大量的 NP-complete 和 NP-hard 问题进行求解都表现出了高效性.因此,本文引入蚁群优化算法求解关键词查询问题,核心思想就是利用蚁群算法求解 Steiner 树问题.目前已经有一些利用蚁群优化算法求解 Steiner 树问题的研究[12−14],不同研究采用的策略也不尽相同,但是这些研究都只是针对 Steiner 树这个理论问题本身或者其他应用场景进行探讨,并没有考虑关系数据库中的关键词查询这个具体应用环境,因而,直接把这些方法应用到本文的研究问题上将无法取得好的效果.比如:文献[12]研究的问题有一个确定的目标点作为蚂蚁前进的目的地,而在本文中则无法给定一个明确的蚂蚁前进目标点;文献[14]研究的图只能严格包含水平和竖直两个方向上的边,与本文研究的数据图有很大的区别;文献[13]需要首先采用特定分区方法对图进行分区,然后采用蚁群优化算法寻找最优路径,最后对不同分区的结果进行合并,这种方法无法在本文的关键词查询问题中使用.

2.3 用蚁群优化算法求解关键词查询问题的基本算法

用蚁群算法求解关系数据库中的关键词查询问题的基本思路是,通过多只蚂蚁的不断沟通与合作,找到包含了所有关键词的 Steiner 树.如果要给出关键词查询的最优答案,就找出一颗最小 Steiner 树;如果是 top-k 关键词查询,就找出代价最小的前 k 棵 Steiner 树.蚂蚁求解 Steiner 树的过程,采用常见的覆盖清除法(spanning and cleanup),即从分组 v_i 中的任意一个节点出发,一步步向周围扩展,直到覆盖了每个分组 v_i 中的至少一个节点,最后清除结果中多余的节点.

具体来说,假定一个查询 K 包含 m 个关键词 $k_1,k_2,\ldots,k_m$,可以通过倒排索引找到数据图 G 中和每个关键词 k_i 相关的节点集合 S_i,S_i 中的节点包含了关键词 k_i.在每次迭代时,从每个 S_i 中都随机选取一个节点,并为每个节点放置 p 只蚂蚁,所有蚂蚁构成集合 *S_ant*.每只蚂蚁放出后,会根据转移概率在数据图中寻找前进的路径.这里为每只蚂蚁设置一个集合变量 *S_visited*,里面存放这个蚂蚁已经访问过的节点,为了避免产生回路,蚂蚁不会再次访问 *S_visited* 中的节点;同时,也为每只蚂蚁设置了集合变量 *S_spanned*,表示这个蚂蚁爬过的节点已经包含的查询关键词.蚂蚁 a 前进一步以后,就把新到达的节点 v 加入到已访问节点集合 *S_visited* 中,这样,蚂蚁 a 对应的树又增加了一个树枝.如果蚂蚁 a 到达的点 v,发现节点 v 的属性 *ant_visited* 不为空,例如,*ant_visited* 记录了蚂蚁 b 的信息,说明节点 v 已经被蚂蚁 b 访问过,则销毁蚂蚁 a,并把蚂蚁 a 的 *S_visited* 的所有元素都加入到蚂蚁 b 的 *S_visited* 中,同时把蚂蚁 a 的覆盖集合 *S_spanned* 加入到蚂蚁 b 的覆盖集合 *S_spanned*.如果蚂蚁 b 的覆盖集合 *S_spanned* 已经包含了所有查询关键词,就说明找到一个结果,那么就输出包含蚂蚁 b 的 *S_visited* 中元素的 Steiner 树 *s_tree*,并把 *s_tree* 放入结果堆 *resultHeap* 中.如果 *resultHeap* 中已经有 k 棵 Steiner 树,则把当前的 *s_tree* 和 *resultHeap* 中的其他树进行比较,丢弃代价最大的树,然后销毁蚂蚁 b.蚂蚁自身有个属性 *stepNum*,在前进的过程中会记录自己已经走过的步数,当 *stepNum* 值超过门槛值 *maxStepNum* 时,查询的结果就没有意义.因此,蚂蚁

就不用继续搜索,被销毁.随着蚂蚁的销毁,蚂蚁数量会逐渐减少,最终当蚂蚁数量为 0 时,就停止本次迭代,开始下一次迭代.当迭代次数达到允许的最大值 *maxIterlationTime* 时,就停止程序执行,输出堆 *resultHeap* 中的查询结果.

算法 1. ACOKS 算法.

Input: 1. 数据图 G,倒排索引 I,迭代次数最大值 *maxIterlationTime*,蚂蚁最大前进步数 *maxStepNum*;

2. 查询关键词集合 $K=\{k_1,k_2,\ldots,k_m\}$;

Output: 1. 前 k 个最好的查询结果.

Begin

对于所有 $i\in[1,m]$,使用倒排索引 I 查找与关键词 k_i 相关的节点集合 S_i;

iteration_time←0; //初始化迭代次数

初始化信息素矩阵 M;

t=0; //记录当前时刻

while (*iteration_time*<*maxIterlationTime*) **do**

从每个 S_i 中分别随机选取一个节点,并为每个节点放置 p 只蚂蚁,所有蚂蚁构成集合 *S_ant*;

ant_num←*m***p*; //求出系统中所有蚂蚁的数量

把数据图 G 中的所有节点的属性 *ant_visited* 初始化为 *NULL*;

//*ant_visited* 表示已经访问过该节点的蚂蚁

for each *ant*∈*S_ant* **do**

stepNum←0; //初始化为 0,属性 *stepNum* 表示蚂蚁爬过的步数;

S_visited(*ant*)←∅; //表示蚂蚁 *ant* 已经访问的节点集合

S_spanned(*ant*)←$\{k_i\}$; //表示蚂蚁 *ant* 爬过的节点已经包含的查询关键词集合

endfor

while (*ant_num*>0) **do**

for each *ant*∈*S_ant* **do** //每只蚂蚁都独立并行执行

if (*ant.stepNum*>*maxStepNum*) then //经过 *maxStepNum* 步,再往前走即使得到结果意义不大

销毁蚂蚁 *ant*;

对于 *S_visited*(*ant*)中所有节点的属性 *ant_visited*,删除 *ant* 的信息;

else

ant.stepNum←*ant.stepNum*+1; //蚂蚁往前走一步

t←t+1; //当蚂蚁往前走一步,更新时刻 t

ant 根据 t–1 时刻的转移概率确定即将移动到的下一个节点 v,并且使得 $v\notin$*S_visited*(*ant*);

if (*v.ant_visited*=*NULL*) **then** //该节点 v 未被其他蚂蚁访问过

S_visited (*ant*)←*S_visited* (*ant*)∪v;

v.ant_visited←*ant*;

else

S_visited(*ant_visited*)←*S_visited*(*ant_visited*)∪*S_visited*(*ant*);

S_spanned(*ant_visited*)←*S_spanned*(*ant_visited*)∪*S_spanned*(*ant*);

销毁蚂蚁 *ant*;

if (*S_spanned*(*ant_visited*)=*K*) **then** //蚂蚁爬过的节点已经包含了所有查询关键词

对于 *S_visited*(*ant_visited*)中所有节点的属性 *ant_visited* 为 *NULL*;

输出由 *S_visited*(*ant_visited*)中所有节点构成的 Steiner 树 *s_tree*; //删除多余节点

计算 *s_tree* 的评分 *score*(*s_tree*);

```
            if (score(s_tree)>score(s_tree_min)) then   //s_tree_min 表示堆 resultHeap 中评分最小的树
              从堆 resultHeap 中删除 s_tree_min,并把 s_tree 加入堆 resultHeap 中;
            else
              丢弃 s_tree;
            endif
            销毁蚂蚁 ant_visited;
          endif
          endif
          更新 t 时刻的信息素矩阵 M;
        endif
      endfor
    endwhile
    iteration_time←iteration_time+1;
  endwhile
  输出堆 resultHeap 中的结果;
end
```

2.4 蚂蚁的移动规则

当算法 ACOKS 每次迭代执行开始时,t=0.此后,每当 t 增加 1,每只蚂蚁都同时往前移动 1 步,即同时从一个节点移动到下一个节点.由于每只蚂蚁的最大允许移动步数是 *maxStepNum*,因此,算法每次迭代最终都会在 t=*maxStepNum* 时刻停止.

在 t 时刻,当一只蚂蚁 *ant* 从一个节点 u 移动到下一个节点 v 时,为了避免产生回路,首先要保证下一个节点 $v\notin S\_visited(ant)$,其中,$v\notin S_visited(ant)$表示 *ant* 已经访问过的节点集合.然后,蚂蚁从节点 u 的所有满足上述禁忌条件的相邻节点中选择一个转移概率最大节点 v 作为下一个访问目标.蚂蚁从节点 u 爬向节点 v 的概率计算方法如公式(1)所示:

$$P_{uv}^{ant}(t)=\begin{cases}\dfrac{[\tau_{uv}(t)]^{\alpha}[\eta_{uv}(t)]^{\beta}}{\sum_{k\notin S\_visited(ant),(u,k)\in G}[\tau_{uk}(t)]^{\alpha}[\eta_{uk}(t)]^{\beta}}, & v\notin S\_visited(ant)\\ 0, & v\in S\_visited(ant)\end{cases} \tag{1}$$

其中,$\tau_{uv}(t)$表示在 t 时刻节点 u 和 v 之间的信息素浓度;α是信息启发式因子,表示轨迹的相对重要性;$\eta_{uv}(t)$表示 t 时刻节点 u 和 v 之间的能见度,$\eta_{uv}(t)=1/d_{uv}$,d_{uv} 表示节点 u 到 v 的代价;β是期望启发式因子,表示能见度的相对重要性.

2.5 信息素的更新

在每次迭代开始后,每条边上的信息素就随着时间的流逝逐步挥发,浓度逐渐降低.因此,蚂蚁每向前移动一步,就需要更新一次信息素矩阵 M.每次更新时,要考虑两个方面的内容:第一,每条边上的信息素会随着时间挥发;第二,对于有助于找到结果的路径,要增强它们的信息素浓度.

假设蚂蚁 a 包含关键词 k_a,已经走过的节点构成的树为 T_a;蚂蚁 b 包含关键词 k_b,已经走过的节点构成的树为 T_b,蚂蚁 a 即将到达的下一个节点 v 已经被蚂蚁 b 访问过.对于这种情形,算法 ACOKS 就会把 T_a 合并到 T_b 中,得到新的树 $T=T_a\cup T_b$.T 中包含了查询关键词 k_a 和 k_b,很显然,这对于找到最终结果是有很大的帮助的.如果一个查询包含 3 个关键词 k_a,k_b 和 k_c,那么从 T 上的节点出发,只要再找到包含 k_c 的树 T_c 即可(T_c 由另一只蚂蚁 c 负责构造).因此,每当出现两个蚂蚁构造的树存在交集的情形时,就应该为合并得到的新树 T 中的所有边增强信息素的浓度.信息素的计算公式见公式(2)、公式(3).

$$\tau_{uv}(t+1)=\begin{cases}\rho\times\tau_{uv}(t)+\Delta\tau_{uv}(t), & \text{当边}(u,v)\in T\\ \rho\times\tau_{uv}(t), & \text{当边}(u,v)\notin T\end{cases} \tag{2}$$

$$\Delta\tau_{uv}(t)=\sum_{i\in[1,m],(u,v)\in T_i}\Delta\tau_{uv}^{i}(t) \tag{3}$$

其中,$\rho\in(0,1)$,表示信息素的挥发程度;$\Delta\tau_{uv}(t)$表示信息素的增加量.

两棵树之间的合并会发生多次,每次合并都会对树中的边的信息素进行增强操作,这里假设一共发生了 m 次合并,在第 i 次合并中($i\in[1,m]$),需要对合并得到的结果树 T_i 中的边(u,v)增加信息素,这个信息素的增加量在公式(3)中用 $\Delta\tau_{uv}^{i}(t)$ 表示,其计算方法见公式(4).

$$\Delta\tau_{uv}^{i}(t)=1/|T_i| \tag{4}$$

其中,$|T_i|$表示树 T_i 的边的代价之和.

这里需要指出的是,每次合并得到的不同的结果树,其优劣程度是不同的.假设第 i 次合并得到的结果树为 T_i,它的评分为 $Score(T_i)$,第 j 次合并得到的结果树为 T_j,它的评分为 $Score(T_j)$,如果有 $Score(T_i)>Score(T_j)$,很显然,T_i 要优于 T_j.为了让后来的蚂蚁更多地靠近 T_i 中的节点,应该让 T_i 中的节点比 T_j 中的节点获得更大的信息素增量,即给 T_i 一个额外的奖励σ.

本文确定的奖励方法是:假设在一次迭代结束后,堆 *resultHeap* 中包含了 k 棵 Steiner 树 $T_1,T_2,\ldots,T_k$,那么对于所有$(u,v)\in T_i(1\leqslant i\leqslant k)$,都执行公式(5):

$$\tau_{uv}(t+1)=\tau_{uv}(t)+\sigma \tag{5}$$

2.6 空间代价分析

在理论上,蚁群算法需要为所有查询关键词对存储信息素矩阵,空间复杂度是 $O(m^2n^2)$,其中,m 为数据库中所有关键词总数,n 为数据图中节点的个数.然而实际上,蚁群算法并不需要维护如此大的信息素矩阵空间.这是因为:首先,并不是任何两两关键词之间都存在联系;其次,因为评分过低的结果树并没有太大意义,所以也不需要维护距离关键词太远的节点;最后,由于大部分路径都不会有蚂蚁经过,因此这些路径的信息素是相同的,所以可以采用一个值代替.为了说明实际空间消耗要远远小于理论情况,我们在 DBLP 数据集上进行了实验分析,数据图的节点数是 n=7270404.理论上,一个关键词查询的信息素矩阵的大小为 7270404^2 个矩阵元素,但是实际上,算法在运行过程中生成的信息素矩阵只有 90 735 个矩阵元素,即压缩率达到了 1.71×10^{-9}.

3 基于概念漂移的查询结果动态优化

本文前面提出,采用基于蚁群优化算法的 ACOKS 算法来解决关键词查询问题可以取得很好的性能,但是 ACOKS 算法却无法解决查询结果的动态优化问题.因此,在前面 ACOKS 算法的基础上,本文进一步提出结合概念漂移理论的优化算法 ACOKS*,实现查询结果的动态优化,提高查询结果的有效性.下面首先介绍已有研究在查询结果有效性方面的不足,然后介绍基于概念漂移的查询结果优化的核心思想.

3.1 已有研究在查询结果有效性方面的不足

在日常应用中,用户的查询兴趣是不断发生变化的.用户查询兴趣的变化可以包括缓慢变化的兴趣和快速变化(突变)的兴趣.本文重点研究如何针对突变的用户兴趣对查询结果进行动态优化,提高查询结果的有效性.

用户查询的兴趣有时候会发生突然变化,比如发生了突发事件,会迅速转移大量用户的查询兴趣.针对这种突变的用户兴趣,不能使用基于长期的概率统计来衡量.比如,用户的查询兴趣突然从 A 转到 B,即在查询结果中突然更多选择 B 而很少选择 A,导致 B 的访问量急剧增加.但是由于历史上有很多用户曾经访问过 A,使得 A 的绝对访问次数远远高于 B.因此,如果采用基于长期概率统计的方法,那么蚁群优化算法仍然会选择 A 而不是 B.因此,必须研究一种可以有效衡量用户查询兴趣突变的方法,并设计相应的查询结果动态优化算法,让蚁群优化算法选择 B 而不是 A.已有的方法没有关于这个方面的研究.

3.2 基于概念漂移的查询结果动态优化的核心思想

概念漂移(concept drift)是机器学习领域研究的重要问题之一.概念漂移是指数据的分布随着时间而发生变化,这些变化就使得在旧数据上建立的模型不再能适用于新的数据特性,因而需要对模型进行更新调整.目前,概念漂移理论的研究成果已经应用到许多领域中,比如在大型零售公司里,可以使用概念漂移理论对顾客的购买行为进行分析,及时发现顾客购买行为的变化,采取应对措施.用户查询兴趣变化的过程,相当于一个“概念漂移”过程.由此,我们引入机器学习领域的概念漂移方面的已有研究成果来探测用户查询兴趣的变化.

基于概念漂移的查询结果动态优化方法的核心思想是,利用概念漂移理论及时发现用户兴趣的突变,并据此动态优化查询结果,使其更加符合用户查询预期.具体而言,首先,利用概念漂移理论确定用户的兴趣节点集(由用户有兴趣访问的查询结果树中的节点构成的集合);然后,只需要对 ACOKS 算法做简单改进即可得到新的 ACOKS*算法.即,引导蚂蚁在移动过程中以较大的概率向兴趣节点集中的节点移动,以较小的概率向非兴趣节点集中的节点移动,从而使得查询结果当中出现更多的兴趣节点集中的内容,更加符合用户兴趣.

3.3 动态优化过程

图 4 显示了基于概念漂移的查询结果动态优化过程.

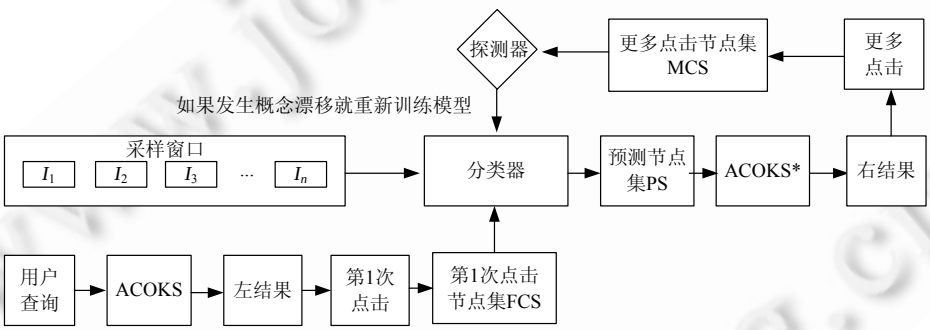


Fig.4 Process of search result dynamic optimization

图 4 查询结果动态优化过程示意图

我们采用用户兴趣节点集来反映用户的查询兴趣,具体而言,当把多个查询结果呈现给用户时,每个查询结果是一棵由多个元组连接得到的元组树,即 Steiner 树,树中的节点包含了查询关键词;当用户点击访问某个查询结果时,就表明该用户对这个结果感兴趣,因此,就可以把这个查询结果对应的元组树的所有节点都放入用户兴趣节点集 I.我们设计的系统界面包含了两个子窗口:左窗口和右窗口,前者包含了系统采用 ACOKS 算法得到的初步查询结果,后者包含了采用 ACOKS*算法得到的动态优化查询结果.系统设置了一个样本窗口,可以记录一定数量的历史用户兴趣节点集,在样本数据基础上,可以训练得到分类模型,从而预测用户未来的查询兴趣.假设样本窗口内包含了 n 个用户兴趣节点集 $I_1,I_2,\ldots,I_n$,根据上述兴趣节点集中的样本节点,可以对分类模型进行训练,得到一个分类器.假设新到达的用户为 U_{n+1},提交查询后,系统采用 ACOKS 算法得到初步查询结果(包含多条记录),这些结果在左窗口中显示,因此被称为左结果.系统将等待用户开始第 1 次点击查看自己感兴趣的一个结果.在用户第 1 次点击发生后,系统将新弹出一个窗口,并在新窗口中显示被点击的这条查询结果(即一棵包含查询关键词的 Steiner 树),同时,系统立即可以确定用户当前点击的访问节点集 FCS(first click set),即该 Steiner 树中包含的节点的集合.系统把 FCS 作为分类器的输入,判定 FCS 的分类标记,然后,系统根据该分类标记预测该用户的兴趣节点集 PS(predicting set),再以 PS 为输入,调用 ACOKS*算法,生成优化的查询结果,这些优化的结果将显示在右窗口中,因此被称为右结果.用户将根据右窗口的结果,选择自己感兴趣的结果进行点击访问.需要指出的是,如果右窗口的推荐优化结果都不是用户感兴趣的,用户将全部都不点击,而是继续在左窗口中寻找

自己感兴趣的结果.当用户发生后续的多次点击后,系统会跟踪记录这些点击所生成的访问节点集 MCS(more click set),它可以真实记录用户的访问兴趣.概念漂移探测器会对 MCS 和 PS 两个集合进行重合度计算,当 MCS 和 PS 重合度较大时,说明预测准确;当二者重合度较小时,用户系统根据历史样本预测,已经发生了较大的错误,用户查询兴趣发生了较大的变化(即发生了概念漂移),这时就需要根据新的样本对分类器的分类模型进行重新训练.

3.4 分类器的构建

用户查询提交给系统以后,系统会返回很多结果,每个查询结果都是一棵 Steiner 树,用户只访问其中感兴趣的一部分结果,这部分结果当中包含的所有节点,就构成了用户兴趣节点集合.针对每次用户查询,系统会自动跟踪记录兴趣节点集的内容,从而为分类器的构建提供样本数据.

假设当前样本窗口内包含了用户针对某个查询 $K=\{k_1,k_2,\ldots,k_m\}$的 n 次查询所对应的兴趣节点集 $I_1,I_2,\ldots,I_n$,这里令 $I=I_1\cup I_2\cup\ldots\cup I_n$.为了建立分类器所需要的类标记,我们首先使用基于距离的聚类算法对集合 I 中的节点进行聚类,从而得到关于集合 I 的多个类标记 $C_1,C_2,\ldots,C_y$.然后,就可以采用基于距离的分类算法把集合 FCS 作为分类器的输入(如图 4 所示),确定集合 FCS 中的每个节点的类标记,从而为计算集合 PS 提供依据.这里,用函数 $C=Classifier(u)$表示一个分类器,即输入一个节点 u,输出 u 对应的分类标记 C.在不产生歧义的前提下,本文以后的内容中,将把类标记 C 同时用来表示一个类集合,即由类标记为 C 的所有节点构成的集合.

3.5 预测节点集的生成

预测节点集 PS 是根据集合 FCS 来确定的.集合 FCS 中包含了用户在“左窗口”中进行第 1 次点击以后所访问的所有节点,它可以大致反映用户的当前查询兴趣.根据 FCS 生成 PS,需要首先确定预测目标类 C_p,再根据预测目标类 C_p 确定预测目标类延伸节点集 T_{C_p},最终,预测目标类 C_p 中的节点和其延伸节点集 T_{C_p} 中的节点,共同构成预测节点集,即 $PS=C_p\cup T_{C_p}$.

定义 5(预测目标类). 对于查询 $K=\{k_1,k_2,\ldots,k_m\}$,假设前 n 次查询所对应的兴趣节点集 $I_1,I_2,\ldots,I_n$,令 $I=I_1\cup I_2\cup\ldots\cup I_n$.根据 I 构建分类器后,得到的分类标记为 $C_1,C_2,\ldots,C_y$.对于第 n+1 次用户查询,已知它的第 1 次点击节点集是 $FCS=\{f_1,f_2,\ldots,f_i\}$,FCS 中的每个节点所属的分类标记为 $Classifier(f_i)$.这里,为每个分类 $C_1,C_2,\ldots,C_j$ 设置一个计数器 *counter*,当 f_i 属于某个分类 C_t 时,$Ct.counter$ 增加 1.预测目标类 C_p 是满足如下条件的类:

$$C_p.counter=\mathrm{MAX}_{i\in(1,j)}(C_i.counter).$$

定义 6(预测目标类延伸节点集). 假设有一个数据图 G,预测目标类 $C_p(\in G)$,两个节点 u 和 v,其中,$u\in G$ 且 $u\notin C_p,v\in C_p$.如果在节点 u 和 v 之间存在一条长度不大于χ的路径 $u\leftrightarrow v$,就称路径上的所有节点构成的集合为预测目标类覆盖节点集,记为 T_{C_p}.预测节点集 PS 定义为集合 $C_p\cup T_{C_p}$.

这里给出一个实例介绍预测节点集 PS 的确定方法.如图 5 所示,有一个数据图 G(用虚线空心圆圈表示 G 中的若干节点)和集合 I(图中用实线曲线表示集合边界,用 s 表示集合中的节点),I 被分成 3 个分类:C_A,C_B,C_C(图中用虚线曲线表示类的边界).用户第 1 次点击节点集 $FCS=\{f_1,f_2,f_3,f_4,f_5,f_6\}$(图中用黑色实心圆圈表示),并且有:

$Classifier(f_1)=C_A,Classifier(f_2)=C_A,Classifier(f_3)=C_B,$

$Classifier(f_4)=C_B,Classifier(f_5)=C_B,Classifier(f_6)=C_B.$

因此,$C_A.counter=2,C_B.counter=4,C_C.counter=0$.可以看出,$C_B.counter$ 最大,因此,C_B 被判定作为预测目标类 C_p.如果χ取值为 3,则预测目标类延伸节点集就是图中用 t 标记(图中用黑色实线空心圆圈表示)的节点,这些节点可以从 C_B 类中的节点出发到达,并且路径长度不大于 3.最终,预测节点集的内容就是类 C_B 中的节点和延伸节点集

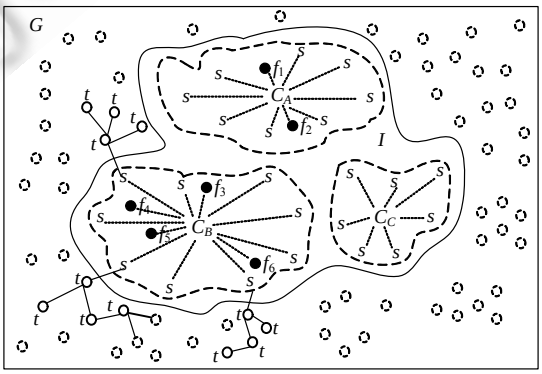


Fig.5 Identification method of predicting node set

图 5 预测节点集的确定方法

中的节点的并集.

3.6 ACOKS*算法

ACOKS*算法和ACOKS算法在很大程度上是相同的,因此这里不再给出ACOKS*算法伪代码.二者的区别在于,ACOKS*算法需要解决两个额外的问题:

- 增加节点能见度:在 ACOKS*算法中,对数据图的各个节点进行能见度计算时(见公式(1)),对于那些满足条件 $v\in PS$ 的所有节点 v,增加其他节点到节点 v 的能见度,让蚂蚁以更大的概率爬向集合 PS 中的节点,从而使得结果树中包含了更多的 PS 中的节点,更加符合用户的兴趣.根据传统的方法,$\eta_{uv}(t)=1/d_{uv}$,d_{uv} 表示节点 u 到 v 的代价.经过优化以后,从节点 u 到节点 v 的能见度计算方法见公式(6):

$$\eta_{uv}^{*}(t)=\begin{cases}(1+\omega)\eta_{uv}(t), & v\in PS\\ \eta_{uv}(t), & v\notin PS\end{cases} \tag{6}$$

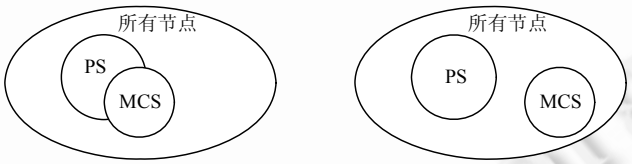
其中,$\omega>0$,表示能见度增强系数.调整后的能见度 $\eta_{uv}^{*}(t)$ 就会比原来的能见度 $\eta_{uv}(t)$ 大,这样,当节点 $v\in PS$ 时,蚂蚁就会以更大的概率向节点 v 移动;

- 增加节点评分:对一棵结果树进行评分时,对于结果树中那些满足条件 $v\in PS$ 的所有节点 v,增加节点 v 的节点分值,即让 $N_{score}(v)^{*}=(1+\varphi)N_{score}(v)$,从而使得包含节点 v 的结果树具有更高的评分,排序尽量靠前,显示给用户.

正是因为 ACOKS*算法解决了上述两个方面的问题,它得到的查询结果中就可以比 ACOKS 算法包含更多的来自集合 PS 的节点,从而更好地符合用户的查询兴趣.

3.7 概念漂移的判定

概念漂移探测器对 MCS 和 PS 两个集合进行重合度计算,当二者重合度较小或者没有重合时,就说明发生了概念漂移.图 6 描绘了在两种情形下的两个集合之间的关系.



(a) 没有概念漂移 (b) 发生概念漂移

Fig.6 Identification of concept drift

图 6 概念漂移的判定

但是我们通过大量实验发现,在少数情况下,由于分类模型误差的存在,会导致根据 FCS(第 1 次点击节点集)生成的 PS(预测节点集)存在一定的误差,这个时候,虽然 PS 和 MCS 具有一定的重合度,但是实际上已经发生了概念偏移.为了最大程度减少这种误判,我们引入了另一个指标,即集合距离.当两个集合距离大于一定程度时,就可以更加确定已经发生了概念漂移.不过,计算机集合距离需要一定的代价,因此在 PS 和 MCS 具有较大重合度时,本文没有采用集合距离,只有当 PS 和 MCS 重合度低于预设门槛值时才引入集合距离的计算,增加概念漂移判定的准确性.

对于两个兴趣结点集 $S_{K_1}=\{a_1,a_2,\ldots,a_m\}$ 和 $S_{K_2}=\{b_1,b_2,\ldots,b_n\}$,则 S_{K_1} 和 S_{K_2} 的集合距离 $D(S_{K_1},S_{K_2})$ 定义如下:

$$D(S_{K_1},S_{K_2})=\sum_{i=1}^{m}(MIN_{j=1}^{n}(Cost(a_i,b_j))) \tag{7}$$

其中,$Cost(a_i,b_j)$表示由在数据图中从节点 b_j 到 a_i 的代价,如果 $a_i=b_j$,则 $Cost(a_i,b_j)=0$;如果无法从 b_j 到 a_i,则 $Cost(a_i,b_j)=\infty$.$MIN(\cdot)$函数返回代价的最小值.

最朴素的方式是,对于一个指定的查询 K,假设当前时间窗口内包含的兴趣节点集合是 S_{cur},新到达的查询对应的兴趣结点集是 S_{new}.如果 $D(S_{cur},S_{new})\geqslant\varepsilon$($\varepsilon$是一个阈值),那么我们就说用户查询行为发生了变化.

图 7 展示了一个概念漂移的实例,这里以此为例来解释公式(7)的计算方法.假设在图 7 中的数据图上发起查询 $K=\{k_1,k_2\}$,其中,k_1="Wang",k_2="XML",节点 a_1 包含了关键词 k_1="Wang",节点 t_1,t_2 和 t_6 包含了关键词 k_2="XML",并假设所有边的权重都是 1.先后有 3 个用户发起相同的查询,但是在查询结果中,不同用户的兴趣节点集是不同的.用户 U_1 的兴趣节点集 $S_{u_1}=\{a_1,w_1,t_1\}$,用户 U_2 的兴趣节点集 $S_{u_2}=\{a_1,w_2,t_2\}$,用户 U_3 的兴趣节点集 $S_{u_3}=\{a_1,w_4,t_5,c_4,t_6\}$,这里使用公式(7)分别计算不同兴趣节点集之间的距离:

$$\begin{aligned}D(S_{u_2},S_{u_1})&=\mathrm{MIN}(Cost(a_1,a_1),Cost(a_1,w_1),Cost(a_1,t_1))+\mathrm{MIN}(Cost(w_2,a_1),Cost(w_2,w_1),Cost(w_2,t_1))+\\&\quad\mathrm{MIN}(Cost(t_2,a_1),Cost(t_2,w_1),Cost(t_2,t_1))\\&=\mathrm{MIN}(0,1,2)+\mathrm{MIN}(1,2,3)+\mathrm{MIN}(2,3,2)\\&=0+1+2=3,\\D(S_{u_3},S_{u_2})&=\mathrm{MIN}(Cost(a_1,a_1),Cost(a_1,w_2),Cost(a_1,t_2))+\mathrm{MIN}(Cost(w_4,a_1),Cost(w_4,w_2),Cost(w_4,t_2))+\\&\quad\mathrm{MIN}(Cost(t_5,a_1),Cost(t_5,w_2),Cost(t_5,t_2))+\mathrm{MIN}(Cost(c_4,a_1),Cost(c_4,w_2),Cost(c_4,t_2))+\\&\quad\mathrm{MIN}(Cost(t_6,a_1),Cost(t_6,w_2),Cost(t_6,t_2))\\&=\mathrm{MIN}(0,1,2)+\mathrm{MIN}(1,2,3)+\mathrm{MIN}(2,3,4)++\mathrm{MIN}(3,4,5)+\mathrm{MIN}(4,5,6)\\&=0+1+2+3+4=10.\end{aligned}$$

如果假设ε=5, $D(S_{u_2},S_{u_1})=3<5$,则可以认为 U_2 和 U_1 的查询兴趣大致相同;而 $D(S_{u_2},S_{u_1})=10>5$,则可以认为 U_3 和 U_2 的查询兴趣发生了突变,即二者之间存在概念漂移.

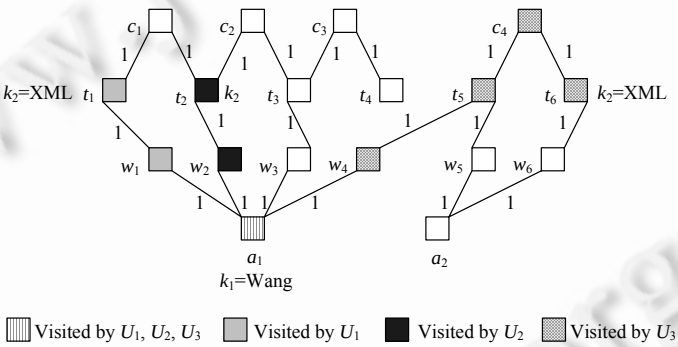


Fig.7 An example of concept drift

图 7 一个概念漂移的实例

4 实验设计与结果

本文实验全部基于我们开发的原型系统,底层采用 ORACLE11g 数据库,用 JAVA 语言编写,并以网页的形式提供对外服务.当前运行的环境是:英特尔 i7-2600 处理器(3.40GHz),16GB 内存,1TB 硬盘和 Windows Server2003 操作系统.

图 8 显示了我们设计实现的原型系统的架构.使用 RDBMS 管理测试数据集,通过调用 RDBMS 的 ODBC 接口访问底层的数据集,并转换得到数据图供概念漂移检测算法和 ACOKS*算法使用.原型系统提供了用户界面,界面包含左窗口和右窗口,用户输入查询关键词以后,系统在左窗口显示初步结果,并根据用户对左窗口结果的访问情况捕捉用户查询兴趣,在右窗口生成优化后的结果.

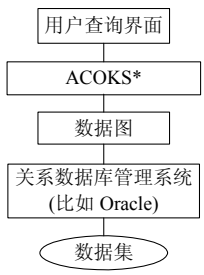


Fig.8 Prototype system architecture

图 8 原型系统架构

我们从网络上下载了 DBLP(http://dblp.uni-trier.de/xml/)数据库,这是目前大多数研究都采用的实验数据库.DBLP 数据库以 XML 文件的形式一共包含了 860MB 的数据,通过我们自己编写的数据库加载工具

把 XML 文件导入到 ORACLE 中,建立一个 DBLP 关系数据库,生成的数据库中的各个表及其所包含的元组数量请见表 1.原型系统从 ORACLE 中访问 DBLP 数据库,生成相应的数据图,存放到内存中,一共包含了 7 270 404 个节点和 9 047 382 条边.生成数据图过程耗时 89s.原型系统会从 DBLP 数据库中自动解析并抽取其中包含的词汇信息,经过解析,一共获得 534 124 个词汇,然后,基于这些词汇构建倒排索引,该过程耗时 105s.实验将对本文算法和 BANKS,BLINKS 进行各种性能指标的比较,从而证明本文算法的高效性和有效性,同时具有比其他方法更好的性能.

Table 1 Test data set

表 1 测试数据集

表	属性	元组数量
Autor	AID, Name	986 107
Paper	PID, Title	1 704 461
Citation	Cite, Cited	112 290
Paper-Author	PID, AID	4 467 546

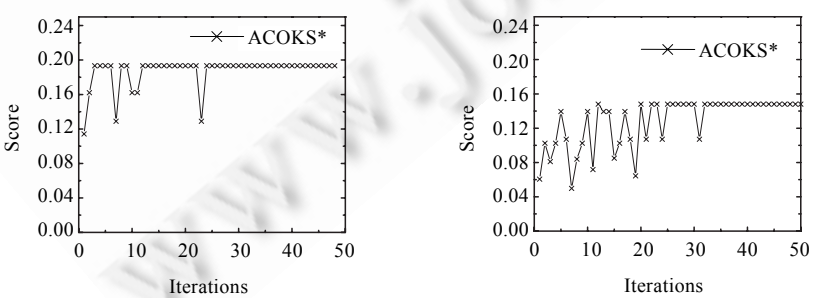
4.1 算法的收敛性

这里选取大量关键词查询对 ACOKS*算法性能进行测试,这些查询包含的关键词数量分布在 2~5 之间.从实验结果来看,ACOKS*算法在不同的关键词查询上均表现出了很好的收敛性.这里首先展示其中的两个查询的算法收敛曲线,其他查询都表现出了类似的特性.选取的 2 个关键词查询分别为 K_1={zhang,ullman}, K_2={alfred,zhang,ullman}.表 2 给出了 ACOKS*算法的参数设置,参考了已有蚁群算法中对各个参数的经验取值.图 9 显示了 ACOKS*算法在处理这两个关键词查询时生成的最优 Steiner 树的评分(采用和 BANKS 的评分方法)和迭代次数的关系,可以看出,ACOKS*算法可以取得很快收敛,最快时只需要经过 5 次迭代就可以达到收敛,找到全局最优解.另外,从图 9 中也可以看出,当 ACOKS*算法找到全局最优解以后,在少数情况下,还可能跳出最优解,然后再回归到最优解,这也恰好反映了蚁群优化算法的特性.因为蚁群优化算法中的蚂蚁在前进路径的选择上,是一种基于概率的选择,在找到全局最优路径以后,大部分蚂蚁都会以较大的概率沿着全局最优路径前进,但是仍然会有少量蚂蚁以较小的概率选择其他路径.实际上,这也是蚁群优化算法的创新机制,可以保证算法在外部环境发生变化后(比如数据图发生了更新),可以寻找到新的最优解.

Table 2 Parameter setting of ACOKS*

表 2 ACOKS*算法的参数设置

max*IterationTime*	*antNumPerNode*	max*StepNum*	Initial pheromone	α	β	ρ
50	4	20	0.01	1	2	0.75



(a) K_1 (b) K_2

Fig.9 Convergence curve of the ACOKS* algorithm

图 9 ACOKS*算法的收敛曲线

4.2 查询的效率

算法的效率(查询响应时间)是最小 Steiner 树求解算法的重要性能指标,由于 Steiner 树问题是一个 NP-hard 问题,即使采用启发式算法,通常也具有很高的代价.虽然本文的研究重点不是提高查询算法的效率,但是我们通过实验发现,在数据图不发生变化的情况下,结合了蚁群优化和概念漂移理论的 ACOKS*算法确实可以比已有的其他方法取得更好的查询效率,这主要得益于蚁群优化算法具有保留信息素的功能.需要指出的是,本文中查找速度的提升,并没有牺牲查询准确度,这种速度的提升只是信息素累积的自然结果.为了比较不同算法的性能,我们设计了 50 个查询,每个查询所包含的关键词数量 l 分布在 2 个~6 个之间.实验对 ACOKS*算法进行性能测试,在分别计算出每个查询的响应时间以后,再对这些响应时间求平均值作为评测指标.图 10 显示了不同算法的查询响应时间比较.从中可以看出,ACOKS*算法在不同关键词数量下,性能都要优于 BANKS 和 BLINKS 算法.对于关键词数量为 2 的查询,BANKS 算法平均响应时间超过 6000ms,BLINKS 算法平均响应时间超过 3000ms,而 ACOKS*算法只需要 450ms,具有明显的性能优势.同时也可以看出,ACOKS*算法具有很好的可扩展性.当关键词的数量从 2 增加到 6 时,ACOKS*算法的平均响应时间并没有大幅度明显增加,在关键词数量为 6 时,ACOKS*算法的平均响应时间也只有 1170ms,而此时,BANKS 和 BLINKS 算法分别达到了 8000ms 和 5500ms.图 11 是 l=2 时,top-k 查询的效率对比情况.可以看出 ACOKS*算法相对于 BANKS 和 BLINKS 算法的明显性能优势.ACOKS*算法能够取得上述性能优势的原因是,该算法具有很好的并发性,多只蚂蚁可以在数据图中并行执行图的遍历.

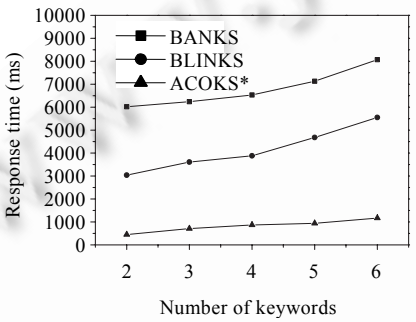


Fig.10 Search response time

图 10 查询响应时间

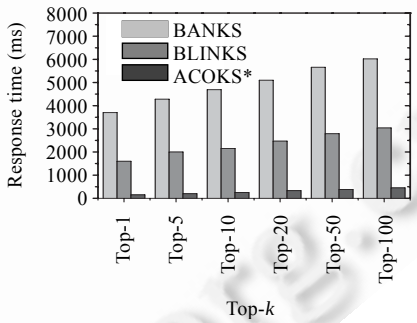


Fig.11 Top-k search response time

图 11 Top-k 查询响应时间

4.3 查询的有效性

查询结果的有效性评测指标是准确率.为了进行对比,这里设计两种情形:在第 1 种情形中,用户查询兴趣不发生变化;在第 2 种情形中,用户查询兴趣不断变化,并且包含多个兴趣突变.为了生成第 2 种情形中不断变化的用户查询兴趣,我们首先编写 JAVA 程序,使用网络上的开源代码 lucene,解析出数据集中包含的所有词汇,然后编写程序,从所有词汇中随机抽取若干词汇作为查询关键词构成若干个查询.对于每个查询,使用程序去完成搜索,如果不存在搜索结果,或者搜索结果少于 3 个,就舍弃这个查询,因为查询结果太少,用户选择结果的余地也小,无法充分反映用户的不同查询兴趣.对于包含 3 个以上结果的查询而言,我们设计一个连续的查询序列,使之能够反映用户兴趣的变化,并通过程序事先设置了用户感兴趣的结果,从而帮助确定算法查询结果的有效性.这里把查询关键词的数量 l 从 2 变化到 6,对于某个特定的 l 值,我们查询得到 top-100 结果,确定 top-100 结果中正确结果的个数,并用正确的个数除以总数 100,计算得到查询结果的准确率.图 12 显示了在第 1 种情形下,不同关键词数量时查询的准确率.图 13 显示了在第 1 种情形下,当 l=2 时,top-k 查询的准确率.从中可以看出,在查询结果的有效性方面,ACOKS*明显比 BANKS 和 BLINKS 具有更高的准确性.这是因为 BANKS 算法和 BLINKS 都是采用固定的评分机制来反映用户的查询兴趣,不能真实反映用户的查询兴趣;而 ACOKS*则是根据用户真实的近期访问数据判断用户的查询兴趣,可以取得更好的效果.图 14 和图 15 分别显示了在第 2 种情形下,不同关

键词数量时查询的准确率和当 l=2 时 top-k 查询的准确率.可以看出,当用户兴趣存在突变时, ACOKS*,BANKS 和 BLINKS 算法的准确性都下降了一些,但是 ACOKS*下降幅度最小,BLINKS 下降幅度最大.这是因为 ACOKS*算法采用的用户兴趣探查方法,可以及时有效探查到用户兴趣的突变,并生成相应的兴趣节点集,从而使得查询结果更好地符合用户预期.

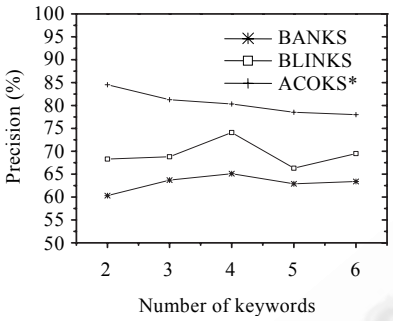


Fig.12 Search accuracy for various keyword number, without interest mutation

图 12 没有兴趣突变时,不同关键词数量时查询的准确率

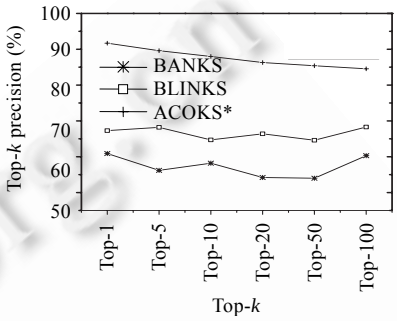


Fig.13 Search accuracy for top-k search, without interest mutation, l=2

图 13 没有兴趣突变且当 l=2 时, top-k 查询的准确率

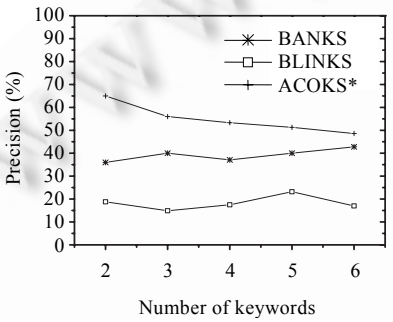


Fig.14 Search accuracy for various keyword number, with interest mutation

图 14 存在兴趣突变时,不同关键词数量时查询的准确率

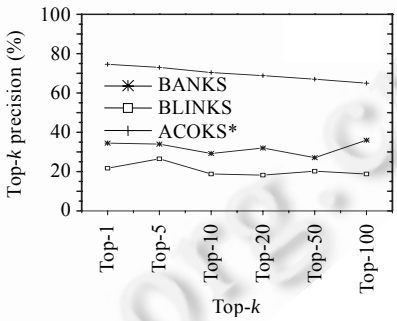


Fig.15 Search accuracy for top-k search, with interest mutation, l=2

图 15 存在兴趣突变且当 l=2 时, top-k 查询的准确率

5 相关工作

基于关键词的关系数据库查询是近年来的研究热点.DBExplorer[5],DISCOVER[6],BANKS[1]是最早一批具有代表性的研究工作,此后,不断有新的成果诞生,其中很多成果都发表在 VLDB,SIGMOD,ICDE 等学术会议和期刊(比如 TODS)上,比如文献[2−4,15−18].我们在文献[19]详细阐述了该领域的研究问题,并详细介绍了各种方法的原理以及各自的优缺点.

已有的研究方法可以分成两大类,即基于数据图的方法和基于模式图的方法.许多研究[1−4,8]都采用基于数据图的方法来生成简化子树,即直接对数据图进行处理,从中枚举简化子树.BANKS 系统使用了一种图搜索算法,即反向扩展搜索(backward expanding search)算法,直接对数据图进行搜索.但是,当一个查询关键词匹配非常大量的节点时,或者当算法遭遇一个入度非常大的节点时,反向扩展算法的性能就会非常糟糕.为此,BANKS-II[2]提出了一种新的搜索方法——双向搜索,它允许从可能的根节点出发朝着叶子节点进行前向搜索,

从而改进了后向搜索的性能.BLINKS[20]采用了一个基于代价均衡扩展(cost-balanced expansion)的反向搜索策略,并且使用了额外的双层索引来加快搜索速度.双层索引明显减少了运行最优反向搜索算法的开销,并且可以支持前向搜索,从而有效地实现类似BANKS-II中的双向搜索.与BANKS-II双向搜索方法相比,BLINKS在搜索过程当中可以实现更大的前向跳跃,加快了搜索速度.文献[8]描述了一个关于结构化数据的关键词查询的形式化框架,该形式化框架明确定义了枚举简化子树问题的3个变种,其中,有一个变种用来处理有向简化子树,另外两个变种用来处理无向简化子树.作者发现了简化子树的枚举和Steiner树优化问题这二者之间存在的复杂关系.文献[21]提出的枚举算法则可以采用任意顺序进行枚举,而且作者把枚举问题分解成几个小的子问题,从而可以提前对每个子问题进行测试,判断它的结果是否非空.最后,对子问题的结果进行组合就可以得到原问题的答案.文献[9,22]提出了一种动态编程方法,它可以近似给出top-k个元组连接树.文献[3]提出了"r-半径Steiner树"的概念,把关键词查询问题转化成为"r-半径Steiner树问题",可以从数据库中发现一些复杂且有意义的结构.文献[17]认为,简化子树还不能充分表达不同元组之间的关系,因此提出了"社区(community)"的概念,它是一个数据图中的多中心有向图,作者认为,"社区"比简化子树更能代表用户的查询兴趣.

基于模式图的方法主要包括3个步骤:第1步,利用数据库模式来枚举可能包含查询结果的所有连接表达式;第2步,根据一系列规则把连接表达式转换成SQL语句,并在数据库上执行,得到所有可能的候选结果;第3步,对结果进行排序并把相关内容返回给用户.在第2步中,执行SQL的方式可以分为两种:一种是使用SQL语句直接在RDBMS(relational database management system)上执行;另一种是采用中间件方法,在中间件上执行SQL语句,而中间件层位于RDBMS层之上.由于需要处理大量的关系代数表达式,许多现有的研究[7]都采用基于中间件的方法,而没有充分利用RDBMS的能力,只有早期的少量研究[5,6]采用直接在RDBMS上执行SQL语句.文献[16]证明了:当前的商业数据库系统是足够强大的,可以高效地支持关键词查询,而不需要增加和维护额外的索引.因此,文献[16]采用了直接在RDBMS上执行SQL语句的方法,并且用实验证明了可行性.实际上,对于一些基于数据图的方法[4],也强调采用数据库自身的强大SQL查询能力实现关键词查询.DBXplorer和DISCOVER关注的是算法的效率,而忽视了算法的有效性,算法得到的有些结果虽然也包含了所有关键词,却未必是用户感兴趣的,这些结果就是无效的.因此,文献[23]重点研究了算法的有效性,它也是第一个采用大量实验来验证算法有效性的研究;同时,作者也提出了一个新的排序策略,通过采用精炼的加权模式对排序公式进行了优化,它可以对结果进行有效排序,并返回具有基本语义的结果.文献[24–26]主要讨论在关系数据库中如何支持高效的top-k关键词查询.很多基于模式图的方法大都采用即席的方式确定元组之间的关系,由于元组之间存在大量关系,这种方式通常效率较低.为此,文献[27]提出了元组单元(tuple unit)的概念,并为关键词查询增加了数据预处理阶段,在这个离线处理阶段,需要搜索元组单元并对这些单元进行物化,在关键词查询时,就可以直接利用这些元组单元加速关键词查询的在线处理.但是,文献[27]只能使用单个元组回答查询,因此,文献[28]进一步提出了结合多个元组回答查询的新方法.

在关键词查询结果优化方面,文献[29]提出了采用CourseCloud形式展现查询结果,而不是以元组连接树的形式呈现.CourseCloud把关键词查询的灵活性和标记云的概括、可视化能力相结合,来帮助用户更好地查询数据库内容.云包含了查询结果中最重要和最具有代表性的术语(概念),被称为数据云.数据云可以引导用户的搜索,帮助用户优化查询.数据云当中的每个术语都有超链接,用户可以点击超链接来优化查询结果;同时,数据云的内容也会根据最新的查询结果进行相应的更新.不同的用户可以从数据云中选择不同的术语,从而以不同的方式来优化查询.但是,CourseCloud这种对查询结果的优化方式与本文所说的查询结果优化完全不同.CourseCloud是在前端展示结果时,通过标记的方式引导用户点击访问合适的关键词,从而获得更好的查询结果;而本文的算法是在指定关键词的前提下,生成更好的查询结果.

查询的高效性和结果的有效性是本领域研究必须重点解决的问题,没有高效性的保证,查询响应时间就会超出用户的容忍范围,查询结果不具备很好的有效性,会让用户看到很多无用的结果,降低用户对系统的使用兴趣.已有的研究侧重点各有不同,比如,文献[30]等侧重于提高算法的效率,而文献[23]等则着力于提高结果的质量.本文则同时保证了查询的高效性和有效性,前者是通过蚁群优化算法实现的,后者是通过基于概念漂移理论

的优化实现的.

6 结束语

本文把关系数据库中的元组建模成数据图,把关键词查询问题转换成等价的最小 Steiner 树问题,引入了已经被证明具有良好性能的蚁群优化算法求解该问题.此外,还提出了基于概念漂移理论的用户查询兴趣突变探查方法,能够及时发现用户兴趣的变化,并提出结合概念漂移理论和蚁群优化算法的 ACOKS*算法,可以根据用户兴趣的变化确定新到达查询的兴趣节点集,引导蚁群优化算法在生成结果的过程中,尽可能多地包含兴趣节点集中的内容,从而实现查询结果的动态优化,使得查询结果更加符合用户的预期,保证了查询结果的有效性.大量的实验结果证明了本文的算法具有很好的性能.

本文后续的研究工作将重点解决如何把算法进行拓展,应用于云计算环境中,从而可以充分利用大量的廉价工业服务器资源,并行处理多个用户的大量查询请求.

References:

[1] Bhalotia G, Hulgeri A, Nakhe C, Chakrabarti S, Sudarshan S. Keyword searching and browsing in databases using BANKS. In: Agrawal R, Dittrich KR, eds. Proc. of the 18th Int'l Conf. on Data Engineering (ICDE 2002). New York: IEEE, 2002. 431−440. [doi: 10.1109/ICDE.2002.994756]

[2] Kacholia V, Pandit S, Chakrabarti S, Sudarshan S, Desai R, Karambelkar H. Bidirectional expansion for keyword search on graph databases. In: Böhm K, Jensen CS, Haas LM, eds. Proc. of the 31st Int'l Conf. on Very Large Data Bases (VLDB 2005). New York: ACM Press, 2005. 505−516. http://www.informatik.uni-trier.de/~ley/db/conf/vldb/vldb2005.html

[3] Li GL, Ooi BC, Feng JH, Wang JY, Zhou LZ. EASE: An effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In: Tsong J, Wang L, eds. Proc. of the 2008 ACM SIGMOD Conf. on Management of Data (SIGMOD 2008). New York: ACM Press, 2008. 903−914. [doi: 10.1145/1376616.1376706]

[4] Li GL, Feng JH, Zhou XF, Wang JY. Providing built-in keyword search capabilities in RDBMS. VLDB Journal, 2011,20(1):1−19. [doi: 10.1007/s00778-010-0188-4]

[5] Agrawal S, Chaudhuri S, Das G. DBXplorer: A system for keyword-based search over relational databases. In: Agrawal R, Dittrich KR, eds. Proc. of the 18th Int'l Conf. on Data Engineering (ICDE 2002). New York: IEEE, 2002. 5−16. [doi: 10.1109/ICDE.2002.994693]

[6] Hristidis V, Papakonstantinou Y. DISCOVER: Keyword search in relational databases. In: Proc. of the 28th Int'l Conf. on Very Large Data Bases (VLDB 2002). New York: Morgan Kaufmann Publishers, 2002. 670−681. http://www.informatik.uni-trier.de/~ley/db/conf vldb/vldb2002.html

[7] Markowetz A, Yang Y, Papadias D. Keyword search on relational data streams. In: Chan CY, Ooi BC, Zhou AY, eds. Proc. of the 2007 ACM SIGMOD Conf. on Management of Data (SIGMOD 2007). New York: ACM Press, 2007. 605−616. [doi: 10.3745/KIPSTD.2009.16D.6.859]

[8] Kimelfeld B, Sagiv Y. Efficient engines for keyword proximity search. In: Doan AH, Neven F, McCann R, eds. Proc. of the 8th Int'l Workshop on the Web & Databases (WebDB 2005). New York: ACM Press, 2005. 67−72. http://www.informatik.uni-trier.de/~ley/db/conf/webdb/webdb2005.html

[9] Ding B, Yu JX, Wang S, Qin L, Zhang X, Lin XM. Finding top-*k* min-cost connected trees in databases. In: Chirkova R, Dogac A, Özsu MT, Sellis TK, eds. Proc. of the 23rd Int'l Conf. on Data Engineering (ICDE 2007). New York: IEEE, 2007. 836−845. [doi: 10.1109/ICDE.2007.367929]

[10] Colorni A, Dorigo M, Maniczzo V. Distributed optimization by ant colonies. In: Varela FJ, Bourgine P, eds. Proc. of the 1st European Conf. on Artificial Life (ECAL'91). Cambridge: MIT Press, 1991. 134−142.

[11] Gutjahr WJ. A graph-based ant system and its convergence. Future Generation Computer Systems, 2000,16(8):873−888. [doi: 10.1016/S0167-739X(00)00044-3]

[12] Yang WG, Guo TD. An ant colony optimization algorithm for the minimum Steiner tree problem and its convergence proof. Acta Mathematicae Applicatae Sinica, 2006,29(2):352−361 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-3079.2006.02.016]

[13] Prossegger M, Bouchachia A. Ant colony optimization for Steiner tree problems. In: Proc. of the 5th Int'l Conf. on Soft Computing as Transdisciplinary Science and Technology (CSTST 2008). New York: ACM Press, 331−336.

[14] Hu Y, Jing T, Hong XL, Feng Z, Hu XD, Yan GY. An efficient rectilinear Steiner minimum tree algorithm based on ant colony optimization. In: Proc. of the Int'l Conf. on Communications, Circuits and Systems (ICCCAS 2004). New York: IEEE, 2004. 1276−1280. [doi: 10.1109/ICCCAS.2004.1346406]

[15] Peng ZH, Zhang J, Wang S. S-CBR: Presenting results of keyword search over databases based on database schema. Ruan Jian Xue Bao/Journal of Software, 2008,19(2):323−337 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/323.htm [doi: 10.3724/SP.J.1001.2008.00323]

[16] Qin L, Yu JX, Chang LJ. Keyword search in databases: The power of RDBMS. In: Cetintemel U, Zdonik SB, Kossmann D, Tatbul N, eds. Proc. of the 2009 ACM SIGMOD Conf. on Management of Data (SIGMOD 2009). New York: ACM Press, 2009. 681−694. [doi: 10.1145/1559845.1559917]

[17] Qin L, Yu JX, Chang LJ, Tao YF. Querying communities in relational databases. In: Ioannidis YE, Lee DL, Ng RT, eds. Proc. of the 25th Int'l Conf. on Data Engineering (ICDE 2009). New York: IEEE, 2009. 724−735. [doi: 10.1109/ICDE.2009.67]

[18] Thein MM. Querying connected tuple trees for relational keyword search. In: Mahmod R, Abdullah R, Abdullah LN, Sembok TMT, Smeaton AF, Crestani F, Doraisamy S, Kadir RA, Ismail M, eds. Proc. of the Int'l Conf. on Information Retrieval & Knowledge Management (CAMP 2012). New York: IEEE, 2012. 285−289. [doi: 10.1109/InfRKM.2012.6204992]

[19] Lin ZY, Yang DQ, Wang TJ, Zhang DZ. Keyword search over relational databases. Ruan Jian Xue Bao/Journal of Software, 2010,21(10):2454−2476 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3913.htm [doi: 10.3724/SP.J.1001.2010.03913]

[20] He H, Wang HX, Yang J, Yu PS. BLINKS: Ranked keyword searches on graphs. In: Chan CY, Ooi BC, Zhou AY, eds. Proc. of the 2007 ACM SIGMOD Conf. on Management of Data (SIGMOD 2007). New York: ACM Press, 2007. 305−316. [doi: 10.1145/1247480.1247516]

[21] Kimelfeld B, Sagiv Y. Efficiently enumerating results of keyword search over data graphs. Information System, 2008,33(4-5):335−359. [doi: 10.1016/j.is.2008.01.002]

[22] Wang S, Peng ZH, Zhang J, Qin L, Wang S, Yu JX, Ding BL. NUITS: A novel user interface for efficient keyword search over databases. In: Dayal U, Whang KY, Lomet DB, Alonso G, Lohman GM, Kersten ML, Cha SK, Kim YK, eds. Proc. of the 30th Int'l Conf. on Very Large Data Bases (VLDB 2006). New York: Morgan Kaufmann Publishers, 2006. 1143−1146. http://www.informatik.uni-trier.de/~ley/db/conf/vldb/vldb2006.html

[23] Liu F, Yu CT, Meng WY, Chowdhury A. Effective keyword search in relational databases. In: Chaudhuri S, Hristidis V, Polyzotis N, eds. Proc. of the 2006 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2006). New York: ACM Press, 2006. 563−574. [doi: 10.1145/1142473.1142536]

[24] Luo Y, Lin XM, Wang W, Zhou XF. Spark: Top-*k* keyword query in relational databases. In: Chan CY, Ooi BC, Zhou AY, eds. Proc. of the 2007 ACM SIGMOD Conf. on Management of Data (SIGMOD 2007). New York: ACM Press, 2007. 115−126. [doi: 10.1145/1247480.1247495]

[25] Luo Y, Wang W, Lin XM, Zhou XF, Wang JM, Li KQ. SPARK2: Top-*k* keyword query in relational databases. IEEE Trans. on Knowledge and Data Engineering, 2011,23(12):1763−1780. [doi: 10.1109/TKDE.2011.60]

[26] Xu YW, Guan JH, Ishikawa Y. Scalable top-*k* keyword search in relational databases. In: Lee SG, Peng ZY, Zhou XF, Moon YS, Unland R, Yoo J, eds. Proc. of the 17th Int'l Conf. on Database Systems for Advanced Applications (DASFAA 2012). Berlin: Springer-Verlag, 2012. 65−80. http://www.informatik.uni-trier.de/~ley/db/conf/dasfaa/dasfaa2012-2.html

[27] Li GL, Feng JH, Zhou LZ. Retune: Retrieving and materializing tuple units for effective keyword search over relational databases. In: Li Q, Spaccapietra S, Yu ES, Olivé A, eds. Proc. of the 27th Int'l Conf. on Conceptual Modeling (ER 2008). Berlin: Springer-Verlag, 2008. 469−483. [doi: 10.1007/978-3-540-87877-3]

[28] Feng JH, Li GL, Wang JY. Finding top-*k* answers in keyword search over relational databases using tuple units. IEEE Trans. on Knowledge and Data Engineering, 2011,23(12):1781−1794. [doi: 10.1109/TKDE.2011.61]

[29] Koutrika G, Zadeh ZM, Garcia-Molina H. Data clouds: Summarizing keyword search results over structured data. In: Kersten ML, Novikov B, Teubner J, Polutin V, Manegold S, eds. Proc. of the 12th Int'l Conf. on Extending Database Technology (EDBT 2009). New York: ACM Press, 2009. 391−402. [doi: 10.1145/1516360.1516406]

[30] Hristidis V, Gravano L, Papakonstantinou Y. Efficient IR-style keyword search over relational databases. In: Freytag JC, Lockemann PC, Abiteboul S, Carey MJ, Selinger PG, Heuer A, eds. Proc. of the 29th Int'l Conf. on Very Large Data Bases (VLDB 2003). New York: Morgan Kaufmann Publishers, 2003. 850−861. http://www.informatik.uni-trier.de/~ley/db/conf/vldb/vldb2003.html

附中文参考文献:

[12] 杨文国,郭田德.求解最小 Steiner 树的蚁群优化算法及其收敛性.应用数学学报,2006,29(2):352−361. [doi: 10.3321/j.issn:0254-3079.2006.02.016]

[15] 彭朝晖,张俊,王珊.S-CBR:基于数据库模式展现数据库关键词检索结果.软件学报,2008,19(2):323−337. http://www.jos.org.cn/1000-9825/19/323.htm [doi: 10.3724/SP.J.1001.2008.00323]

[19] 林子雨,杨冬青,王腾蛟,张东站.基于关系数据库的关键词查询.软件学报,2010,21(10):2454−2476. http://www.jos.org.cn/1000-9825/3913.htm [doi: 10.3724/SP.J.1001.2010.03913]

林子雨(1978－),男,吉林柳河人,博士,讲师,CCF 会员,主要研究领域为数据库,实时主动数据仓库,数据挖掘.
E-mail: ziyulin@xmu.edu.cn

赖永炫(1981－),男,博士,讲师,主要研究领域为数据库,传感器网络数据管理.
E-mail: laiyx@xmu.edu.cn

邹权(1982－),男,博士,讲师,主要研究领域为生物信息学,大规模数据挖掘.
E-mail: zouquan@xmu.edu.cn

林琛(1982－),女,博士,讲师,CCF 会员,主要研究领域为数据挖掘,信息检索,社会网络分析.
E-mail: chenlin@xmu.edu.cn