空间数据库中的关键词搜索

Ian De Felipe Vagelis Hristidis Naphtali Rishe

计算机与信息科学学院

佛罗里达国际大学

迈阿密，佛罗里达州 33199

{ian.de.felipe, vagelis, [rishen}@cis.fiu.edu](mailto:rishen%7d@cis.fiu.edu)

摘要：

许多应用需要寻找最接近包含有一系列关键词的位置的对象。比如，在线黄页允许用户指定一个地址和一系列关键词。然后，用户可以获取到符合条件的公司列表，它们按到用户指定地点的距离从近到远排序。空间数据的最近邻搜索问题和文本数据的关键词搜索问题已经被广泛地单独研究。然而，就我们所知，还没有一种方法能够有效地解决空间关键词搜索问题，即根据指定的位置和关键词来联合查询。

在这个研究中，我们提出了一种有效的方法，来解决top-k关键词查询问题。为此，我们引入了一个索引结构，IR2树（Information Retrieval R-Tree），它结合了R树和叠加文本签名。我们提出的算法，通过建立和维持一个IR2树，用来解决top-k空间关键词查询问题。通过实验比较，我们的算法与现有的方法相比，具有更优越的性能和良好的可扩展性。

1. 介绍

越来越多的应用需要高效地执行被空间对象的属性限制的最近邻查询。理想的结果是，按照到查询区域的距离和描述文字与搜索关键词的相关性排序的对象。一个正在使用的简化的变体是距离优先的空间关键词查询，它将结果按距离排序，而关键词则作为筛选器来排除不包含它们的结果。

这里有一个运行中的例子，如Figure 1所示，给出了一些虚拟酒店的地理坐标和描述属性（名称、设施），一个空间关键词查询的示例是寻找离[30.5, 100.0]这个点最近，并包含关键词internet和pool的酒店。可以看到，最好的结果是酒店H7。

不幸的是，不能有效地支持top-k空间关键词查询，因为这时需要的是结果列表中的前几个。目前的系统采用ad-hoc结合最近邻和关键词搜索技术来解决这个问题。例如，R-Tree用来找到最近的邻节点，而对于每个邻节点，用倒排索引来检查是否包含查询的关键词。我们发现，这样分为两阶段处理的方法是低效的。我们提出了一个方法，能够有效地解决前k空间关键字查询问题，它是基于的紧密集成的数据结构、空间数据库搜索的算法和信息检索（IR）。需要特别说明的是，我们的方法要建立一个信息检索R-Tree（IR2-Tree），这是一个基于R-Tree的数据结构[Gut84]，查询时采用一种增量算法，使用IR2-Tree来高效地产生查询结果。

IR2-Tree是一种特殊的R-Tree，它的每个节点v都有一个签名（Faloutsos and

Christodoulakis [FC84]），用来表示以v为根的子树中所以空间物件的文本内容。我们的top-k空间关键词搜索算法 ，受到了Hjaltason和Samet[HS99]的工作的启发，运用他们的研究成果，我们通过访问很小的一部分IR2-Tree就可以找出前面的查询结果。我们的研究有如下成果：

* 定义了top-k空间关键词查询问题
* 提出了IR2-Tree——一个高效的索引结构，用来存储一系列对象的空间和文本信息。同时提出了IR2-Tree的维护算法，即插入和删除对象的算法。
* 提出了使用IR2-Tree的高效的增量算法，用来解决top-k空间关键词查询问题。我们用真实的数据进行试验，将它的性能与现有的方法相比较，在执行时间上有明显的提升。请注意，我们的方法可以用于任意形状和多维对象，为了使结果更加清晰，我们在实验中不仅使用了这两种维度上的点。

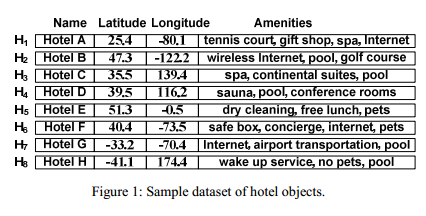
这篇论文按如下结构组织：第2节正式定义了tok-k空间关键词搜索问题；第3节介绍背景知识；第4节介绍IR2-Tree及其维护算法；第5节介绍增量搜索算法和其他基线算法；第6节通过实验比较我们的算法和其他基线算法；第7节讨论相关研究；第8节总结。

2. 问题定义

在这里，（空间）对象T被定义为有序对（T.p，T.t），T.p是多维空间中的位置描述符，T.t是一个文本文档（文本描述）。设D是数据库中所有对象的集合。在Figure 1中，T.p是由“纬度（latitude）”和“经度（longitude）”组成的点，而T.t是“旅馆名（name）”和“设施（amenities）”属性的级联。

top-k空间查询Qs，是指搜索多维空间，来找到离指定点p最近的k个对象。这些空间对象按距离排序，使得离p越近的点排位越靠前。特别地，

score(T) = distance(T.p, p)。例如，在Figure 1中，对于给定的p[30.5, 100.0]，对象H4排在最前面。



关键词查询Qw，是一系列关键词的集合w1,…wm。Qw的返回值对象的列表，按它们的文本描述和关键词的相关度排序，其中相关度IRscore(T.t, Qw)通过一个IR排序函数[Sin01]计算。

在运行实验中使用的一个特殊的例子，是一个布尔关键词查询，返回文本文档中包含有全部关键词w1,…wm的对象，即：例如，在Figure1中，对象H2和H7是布尔关键词查询{“internet”, “pool”}的结果。

Top-k空间关键词查询Q是top-k空间查询和关键词查询的结合，它被定义为需要的结果数Q.k，指定点Q.p，一个关键词的集合Q.t={w1,…,wm}和一个排序函数f(distance(T.p, Q.p), IRscore(T.t, Q.t))。查询Q的结果是一个top-k对象的列表T，这k个对象根据排序函数f排序。

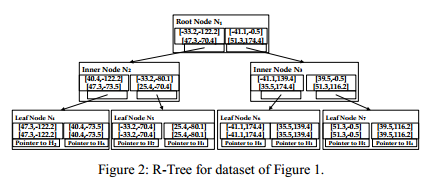
一种特殊的情况是距离优先tok-k空间关键词查询Q，在我们的实验中也使用了，它返回包含w1,…,wm且最接近Q.p的k个对象。也就是说，距离优先tok-k空间关键词查询是top-k空间关键词查询和布尔关键词查询的组合。即：



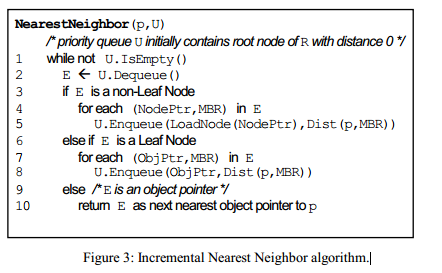
比如，在Figure1中，对象H7和H2是Q.k=2，Q.p=[30.5,100]，和Q.t={“internet”, “pool”}距离优先tok-k空间关键词查询的结果。我们的工作解决了高效解答top-k空间关键词查询的问题。

3. 增量最近邻法的背景知识

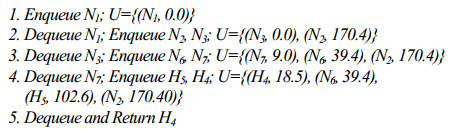
Figure2所示的是R-Tree的一个例子，使用的是Figure1中的旅馆数据。MBR用西南和东北的点表示。R-Tree通常存储在磁盘上，每个R-Tree占用一整个磁盘块，因此，访问节点需要进行磁盘I/O。节点所能指向的子节点数量，称为节点的容量。



Hjaltason and Samet [HS99]提出的增量最近邻算法，使用R-Tree连接最少的R-Tree节点和对象，并以增量的方式来获取离给定点或区域最近的对象。Figure3所示的是二维对象的增量最近邻算法。输入参数是查询点p（或者查询区域），和用R-Tree的根节点R初始化的优先队列U。第二行返回队列中离查询点距离最小的元素。如果这个元素是叶子节点，它的每个子对象通过ObjPtr引用，按距离插入到队列中；如果这个元素不是叶子节点，则它的每个子节点通过NodePtr引用，插入到队列中。最后，如果这个元素是空间对象的指针，则作为算法的下一个结果返回，如第10行所示。Dist函数计算查询点p和MBR之间的距离。我们假设R-Tree保存在磁盘中，因此，LoadNode函数从磁盘加载节点。



例1：在Figure2的R-Tree上执行增量最近邻算法，查询点是[30.5, 100.0]，执行的步骤如下：

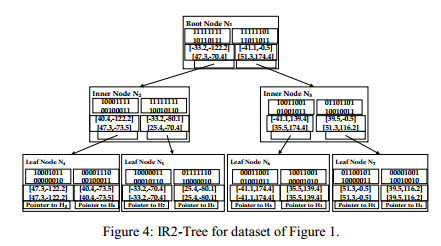


如果继续执行，接下来返回的结果就是H3, H5, H8, H6, H1, H7, H2。

4. IR2-Tree

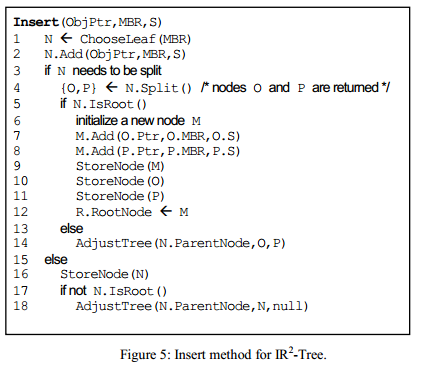
IR2-Tree是R-Tree和签名文档的结合。特别的是，IR2-Tree的每个节点都包含空间和关键词信息；前者在表中的最小边界区，后者在表中的签名中。IR2-Tree对top-k空间查询和top-k空间关键词查询都有利，这一点将在稍后说明。

更正式的说，IR2-Tree R是一种高度平衡树数据结构，其中每个叶子节点具有如下形式的条目(ObjPtr, A, S)，ObjPtr和A与R-Tree中定义的一样，S是ObjPtr引用的对象的签名。非叶子节点具有如下形式的条目(NodePtr, A, S)，NodePtr和A与R-Tree中定义的一样，而S是节点的签名。一个节点的签名是其条目的所有签名的叠加(OR-ing)。因此，一个节点的签名等效于在其子树中的所有文件的签名。Figure4所示的是Figure1中的简单数据集的IR2-Tree。为了简化下面的介绍，我们着重于二维空间。



IR2-Tree通过插入和删除操作来维护，它们都是在对应的R-Tree操作基础上修改的。Figure5和Figure6分别展示了IR2-Tree的插入和删除算法。

插入算法使用了一个标准的R-Tree方法ChooseLeaf，在[Gut84]中能够找到。使用标准Quadratic Split技术[Gut84]来做节点拆分。我们修改了标准的AdjustTree方法，使他能维护修改后的包含签名的节点。即，节点N的一个新的位被设为1时，它的祖先也必须设为1。最后，假定所有的树相关的算法，能够隐式地访问IR2-Tree R的根节点。



Insert算法的输入是对象T的指针，MBR和签名。算法的第一行根据T的MBR获取一个最合适的叶子节点。然后T的指针，MBR和签名存储到N中。如果N已经达到了它的最大容易，它将会被拆分。如第四行所写，如果N被拆分为节点O和P，且N为跟节点，则将创建一个新节点M，M称为O和P的父节点，保存它们的指针、MBR和签名。最后，M被声明为新的根节点。如果N不是跟节点，则它的父节点也要更新，像14行和18行写的那样。最后，由于我们假设IR2-Tree保存在磁盘中，StoreNode函数将节点保存到相应的磁盘块。

Delete的实现中使用了FindLeaf的标准实现，而CondenseTree则经过修改以维护修改的节点中的签名，与上面的AdjustTree类似。在Figure6的第一行，执行对包含有要删除的对象T的节点N的搜索，如果存在这样的节点N，则从节点N中移除对象T，否则算法结束。如果对象T被移除，树需要重新调整。

很明显，Insert和Delete算法的复杂度和R-Tree中是一样的，因为唯一的额外操作是维护修改的节点及其祖先节点的签名。需要注意的是，整个节点及其祖先节点的签名的更新与树更新节点及其祖先的MBR同时进行。为了弥补存储IR2-Tree的节点中的签名需要的额外的空间，同时为了与R-Tree拥有相同数量的子节点，我们在需要时为IR2-Tree的节点增加磁盘块。正如6.1节所示，这个事实对IR2-Tree算法的性能有轻微的影响。

