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

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，IR^2 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 IR^2 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. 引言

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 H_7。

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（IR^2-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 IR^2-Tree 来高效地产生查询结果。

IR^2-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 IR^2-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 IR^2-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 IR^2-Tree 的维护算法，即插入和删除对象的算法。
- 提出了使用 IR^2-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 IR^2-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 Q_s，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 H_4 排在最前面。

	Name	Latitude	Longitude	Amenities
H_1	Hotel A	25.4	-80.1	tennis court, gift shop, spa, Internet
H_2	Hotel B	47.3	-122.2	wireless Internet, pool, golf course
H_3	Hotel C	35.5	139.4	spa, continental suites, pool
H_4	Hotel D	39.5	116.2	sauna, pool, conference rooms
H_5	Hotel E	51.3	-0.5	dry cleaning, free lunch, pets
H_6	Hotel F	40.4	-73.5	safe box, concierge, internet, pets
H_7	Hotel G	-33.2	-70.4	Internet, airport transportation, pool
H_8	Hotel H	-41.1	174.4	wake up service, no pets, pool

Figure 1: Sample dataset of hotel objects.

关键词查询 Q_w，是一系列关键词的集合 $w_1,\ldots w_m$。Q_w 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, Q_w)通过一个 IR 排序函数[Sin01]计算。

$$Ans(Q) = \left\{ Q.k \text{ first } T \in D \text{ ordered by distance } (T.p, Q.p) \;\middle|\; \forall w \in Q.t, w \in T.t \right\}$$

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：$Ans(Q_w) = \{T \in D \mid \forall w \in Q_w, w \in T.t\}$ 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={$w_1,\ldots,w_m$}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 $w_1,\ldots,w_m$ 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。

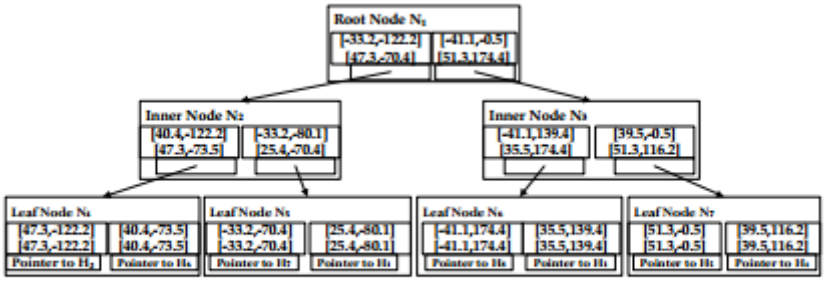


Figure 2: R-Tree for dataset of Figure 1.

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。

```
NearestNeighbor(p,U)
    /* priority queue U initially contains root node of R with distance 0 */
1   while not U.IsEmpty()
2     E ← U.Dequeue()
3     if E is a non-Leaf Node
4       for each (NodePtr,MBR) in E
5         U.Enqueue(LoadNode(NodePtr),Dist(p,MBR))
6     else if E is a Leaf Node
7       for each (ObjPtr,MBR) in E
8         U.Enqueue(ObjPtr,Dist(p,MBR))
9     else /* E is an object pointer */
10      return E as next nearest object pointer to p
```

Figure 3: Incremental Nearest Neighbor algorithm.

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：

1. Enqueue N_1; $U=\{(N_1, 0.0)\}$
2. Dequeue N_1; Enqueue N_2, N_3; $U=\{(N_3, 0.0), (N_2, 170.4)\}$
3. Dequeue N_3; Enqueue N_6, N_7; $U=\{(N_7, 9.0), (N_6, 39.4), (N_2, 170.4)\}$
4. Dequeue N_7; Enqueue H_5, H_4; $U=\{(H_4, 18.5), (N_6, 39.4),$
   $(H_5, 102.6), (N_2, 170.40)\}$
5. Dequeue and Return H_4

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

4. IR^2-Tree

IR^2-Tree 是 R-Tree 和签名文档的结合。特别的是，IR^2-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。IR^2-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，IR^2-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 IR^2-Tree。为了简化下面的介绍，我们着重于二维空间。

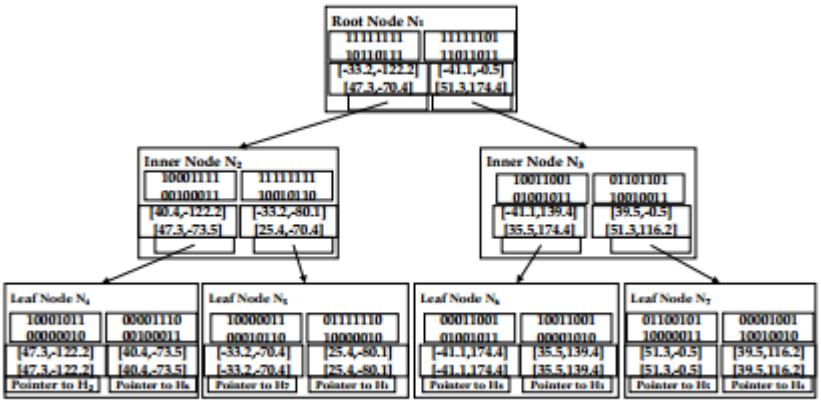


Figure 4: IR2-Tree for dataset of Figure 1.

IR^2-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 IR^2-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 IR^2-Tree R 的

根节点。

```
Insert(ObjPtr,MBR,S)
1    N ← ChooseLeaf(MBR)
2    N.Add(ObjPtr,MBR,S)
3    if N needs to be split
4       {O,P} ← N.Split() /* nodes O and P are returned */
5       if N.IsRoot()
6          initialize a new node M
7          M.Add(O.Ptr,O.MBR,O.S)
8          M.Add(P.Ptr,P.MBR,P.S)
9          StoreNode(M)
10         StoreNode(O)
11         StoreNode(P)
12         R.RootNode ← M
13      else
14         AdjustTree(N.ParentNode,O,P)
15   else
16      StoreNode(N)
17      if not N.IsRoot()
18         AdjustTree(N.ParentNode,N,null)
```

Figure 5: Insert method for IR^2-Tree.

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 IR^2-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 IR^2-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 IR^2-Tree 的节点增加磁盘块。正如 6.1 节所示，这个事实对 IR^2-Tree 算法的性能有轻微的影响。

```
Delete(ObjPtr)
1    N ← R.FindLeaf(ObjPtr)
2    if N was not found
3       return
4    else
5       N.Remove(ObjPtr)
6       CondenseTree(N)
7       if R.RootNode has only one child M
8          R.RootNode ← M
```

Figure 6: Delete method for IR2-Tree.

多级 IR2-Tree

上面描述的 IR^2-Tree 有一个缺点，每一级都使用相同的签名长度，这会在高级别上带来更多的误报（因为它们是低级别的叠加）。为了解决这个问题，我们队不同的级别使用不同的签名长度。这通过使用多级叠加编码来实现，以减少误报，特别是在非叶子节点上。在这种情况下，每个节点都使用最佳签名长度（使用最佳签名长度公式计算[MC94]），并用节点的子树中所有对象的签名来叠加，而不是之前的子节点的签名。这种变化后的方法，称为多级 IR^2-Tree（MIR^2-Tree）。它也有一个缺点，即显著地增加了维护操作（插入、删除）的复杂度。插入、删除每个对象，都必须访问相关的所有对象，来计算所有祖先节点的签名，而不是像之前那样，只需要计算子节点的签名。我们将在第 6 节中比较 IR^2-Tree 和 MIR^2-Tree 的性能。

5. 解决空间关键词查询问题的算法

在 5.1 节中，考虑两个基线算法，根据使用的底层数据结构分别命名为：R-Tree 和唯一倒排索引（IIO）。在 5.2 节中，我们介绍距离优先 IR^2 算法，它使用 IR^2-Tree 数据结构来解决距离优先空间关键词查询问题。在 5.3 节中，我们介绍一般的 IR^2 算法，它使用 IR^2-Tree 数据结构来解决一般的空间关键词查询问题。值得注意的是，最后两种算法还能够在没有修改操作的 MIR^2-Tree 使用。

5.1 目前的基线算法

为了简单起见，我们只介绍针对简化的距离优先空间关键词查询问题的 R-Tree 和 IIO 基线算法，在第 6 节的实验中也将使用。这两种算法都可以通过扩展来解决一般的空间关键词查询问题。

R-Tree 算法

第一个基线算法，R-Tree，只使用了 R-Tree 数据结构。对于给定的距离优先空间关键词查询问题，这个算法首先找出离查询点 Q.p 最近的对象。然后反向检索该对象（由于 R-Tree 只包含对象指针），比较该对象的文本描述和查询关键词。如果匹配失败，则这个对象被丢弃，继续检索下一个最近的对象。这个过程使用了 Figure 3 中的增量最近邻算法[HS99]。程序持续运行，直到找到文本内容包含有查询关键词的对象。一旦找到符号条件的对象，就返回该对象，这个过程重复执行，直到返回 k 个对象。

这个算法的缺点是，必须检索最近邻算法返回的每个对象，直到找到 top-k 结果。这可能会导致对很多无用对象的检索。最坏的情况下（当没有一个对象匹配查询关键词时），整棵树都会被遍历，每个对象都要被检查一次。

IIO 算法

IIO 基线算法使用倒排索引。首先通过对倒排索引返回的列表取交集，来找到所有文本文档中包含查询关键词的对象（通过对象 id）。设 V 为交集中对象的集合。然后检索 V 中的对象，计算每个对象与查询点 Q.p 之间的距离。随后，将这些对象排序，返回其中的 top-k 对象。IIO 算法如 Figure 7 所示。输入的参数是倒排索引表 I 和距离有限 top-k 空间关键词查询 Q。

```
IIOTopK(I,Q)
     /* I is the inverted index */
1    for each word wi in Q.t do
2      Li ← I.RetrieveObjectPointersList(wi)
3    V ← intersection of object pointers in Li's
4    initialize a list L
5    for each ObjPtr in V do
6      T ← LoadObject(ObjPtr)
7      d ← Dist(Q.p,T.p)
8      L.Add(T,d)
9    sort items in L by distance
10   return first Q.k objects in L
```

Figure 7: Inverted Index Only (IIO) algorithm.

例 2：考虑如下查询，在 Figure 1 的数据上，进行包含有关键词{“internet”, “pool”}，查询点为[30.5, 100.0]的 top-2 旅馆查询。IIO 算法的执行过程如下：

1. 关键词“internet”的倒排索引返回的结果是 H_1，H_2，H_6，H_7
2. 关键词“pool”的倒排索引范湖的结果是 H_2，H_3，H_4，H_7 ，H_8
3. 取交集，结果是 H_2，H_7
4. 访问 H_2，H_7，获取坐标

5. 添加 H_2 到列表 L={(H_2, 222.8)}
6. 添加 H_7 到列表 L={(H 7, 181.9), (H 2, 222.8)}，返回 H_2，H_7

当有很多对象都包含查询关键词时，这个算法的性能会降低很多。在这种情况下，倒排索引会返回很多对象，需要一一检索和比较。值得注意的是，IIO 算法是本文提到的算法中唯一一个不是增量的算法。也即是说，IIO 算法计算所有的查询结果，正如第 6 节将要提到的那样，它的性能取决于 k。

5.2 距离优先 IR^2-Tree 算法

在本节中，我们讨论距离优先版本的 IR^2-Tree 算法，算法输出包含所有关键词的对象，并按它们到查询点的距离排序。在 5.3 节中，我们将展示如何扩展这个算法来解决一般的 top-k 空间关键词查询问题。

距离优先 IR^2-Tree 算法利用 IR^2-Tree 结构来高效地解决 top-k 空间关键词查询问题。树的遍历基于增量最近邻算法（Figure 3）。该算法的主要优点是：如果根节点的签名与查询签名 Signature(Q.t)不匹配，将剪去整个子树。这样做的合理性在于，IR^2-Tree 的节点的签名是由其所有子节点的签名组成的。这个是传统增量最近邻法提供的修剪之外的修剪。通过这两种修剪机制的紧密结合，距离优先 IR^2-Tree 算法能够访问尽量少的 IR^2-Tree 节点和对象，来解答距离优先 top-k 空间关键词查询问题。

Figure 8 显示了距离优先 IR^2-Tree 算法(IR2TopK)。其中，最关键的方法是 IR2NearestNeighbor(.)，它基于 NearestNeighbor 算法，在其上增加了一个表示查询的签名的输入变量 w。节点和对象的签名将与 w 进行比较，如果不匹配将被跳过（即从搜索队列中移除）。需要注意的是，对 IR2NearestNeighbor(.)方法的每次访问都返回一个候选的结果，它将在稍后（21 行）进行检查，确保不会误报。

例 3：在这个例子中，我们在 Figure 4 中的 IR2-Tree 上执行上述算法，来解决如下查询：top2 旅馆查询，查询点[30.5, 100.0]，关键词{“internet”, “pool”}。

1. N_1 入队列，U={(N 1, 0.0)}；
2. N_1 出队列，N_2 入队列，U={(N 2, 170.4)}；
3. N_2 出队列，N_4、N_5 入队列，U={(N 5, 170.5), (N 4, 173.8)}；
4. N_5 出队列，H_7 入队列，U={(N 4, 173.8), (H 7, 181.9)}；
5. N_4 出队列，H_2 入队列，U={(H 7, 181.9), (H 2, 222.8)}；
6. H_7 出队列并返回；
7. H_2 出队列并返回。

请注意，IR^2-Tree 的签名修剪能力如何开始在第 2 行出现。只有一个子节点 N_1 入

队列，其他的由于签名不匹配而被丢弃了。当对象 H_1 和 H_6 的父节点被访问时，它们也被修剪了（在第 4 行和第 5 行）。

```
IR2NearestNeighbor(p,W,U)
1    while not U.IsEmpty()
2      E ← U.Dequeue()
3      if E is a non-Leaf Node
4        for each (NodePtr,MBR,S) in E
5          if S matches W
6            U.Enqueue(LoadNode(NodePtr),Dist(p,MBR))
7      else if E is a Leaf Node
8        for each (ObjPtr,MBR,S) in E
9          if S matches W
10           U.Enqueue(ObjPtr,Dist(p,MBR))
11     else /*E is an object pointer*/
12       return E as next nearest object pointer to p
IR2TopK(R,Q)
13   initialize a list L
14   Initialize a priority queue U
15   U.Enqueue(R.RootNode,0)
16   W ← Signature(Q.t)
17   c ← 0
18   while c < Q.k
19     ObjtPtr ← IR2NearestNeighbor(Q.p,W,U)
20     T ← LoadObject(ObjPtr)
21     if T.t contains all keywords in Q.t
22       c ← c + 1
23       L.add(T)
24   return L
```

Figure 8: Distance-First IR2-Tree algorithm.

5.3 IR^2-Tree 算法

在本节中，我们讨论一般的 IR^2-Tree 算法，对象通过第 2 节中定义的排序函数 f(distance(T.p, Q.p), IRscore(T.t, Q.t))来排序输出。它与距离优先版本的算法的主要区别如下：

（i）不创建单独的签名 Signature(Q.t)，而使用独特的签名 Signature(w), w∈Q.t。原因是，这里不使用 AND 语义，即，只包含有查询关键词中的一部分的对象也可能出现在结果中。

（ii）不能再因为一个对象是下一个最近的，且包含所有关键词而立即输出它，

因为，更远的对象也有可能有更高的整体分 f(.)。因此，节点 v 在队列 U 中按它们所包含的对象的最大分值排序，即，

$$Upper(v)=UpperBound_{T \in v}(f(distance(T.p, Q.p), IRscore(T.t, Q.t))$$

假设 f()随着 distance()增大而减小，随着 IRscore()增大而增大，则有：

$$Upper(v)=LowerBound_{T \in v}(f(distance(v.MBR, Q.p), UpperBound_{T\text{-has-signature-}v.S}(IRscore(T.t, Q.t)))$$

为了计算 v 的 MBR 中的对象的最大可能的 IR 分值 $UpperBound_{T\text{-has-signature-}v.S}(IRscore(T.t, Q.t))$，假设 v 中有一个假想的对象 T 包含 Q 在 v.S 的签名中指定的所有关键词一次（词频 tf=1），即，假设没有误报。因此，T.t 的文档长度（dl）就是这些关键词的数量。接着可以使用 tfidf IR 排序函数[Sin01]。这种方法有利于尽早地输出结果对象。注意，如果 IR 函数使用了先进的功能，如词库和本体论等，则不可能估计一个很准确的最大可能 IR 分值。

我们队 Figure 8 中的距离优先版本做了如下修改：

1. 替换 16 行:

```
for each keyword wi in Q.t do
  Wi ← Signature(wi)
```

2. 替换 21 到 23 行:

```
Score ← f(distance(T.p,Q.p),IRscore(T.t,Q.t))
if Score ≥ Upper(U.top())
/* check if actual score of T is greater or equal to the max possible score
of the objects in the queue */
  c ← c + 1
  L.add(T)
else
  U.Enqueue(T,Score) /*to be considered later*/
```

3. 替换 5 到 6 行:

```
Score ←UpperBoundT-has-signature-S(IRscore(T.t,Q.t))
if Score > 0
  U.Enqueue(LoadNode(NodePtr),Score)
/* check if there can be an object T with non-zero IR score. The "if"
condition can be removed if results with 0 IR score are acceptable*/
```

4.替换 9 到 10 行:

```
Score ← UpperBound_T-has-signature-s(IRscore(T.t,Q.t))
if Score > 0
   U.Enqueue(ObjPtr,Score)
/* check if T has non-zero IR score. The "if" condition can be removed if
results with 0 IR score are acceptable*/
```

6. 实验

为了衡量 IR^2-Tree、MIR^2-Tree 和其它基线算法的效率，我们用 JAVA 实现了所有的算法和底层数据结构。其中，所有的索引结构（R-Tree、IR^2-Tree、MIR^2-Tree 和倒排索引）都保存在磁盘中。实验主要针对距离优先版本的 top-k 空间关键词查询问题，因为它的结果比较容易理解和分析。空间对象保存在一个纯文本文件中，树的叶子节点中保存指向对象在文件中的位置的指针。我们根据满足查询时间和执行时间锁需要的磁盘访问来做比较。实验所用的机器配置是：CPU：Athlon 64 3400+ (NewCastle)，内存：2GB，硬盘驱动器：74GB 10000RPM。

我们使用的两个数据集均来自高性能数据研究中心（High Performance Database Research Center ，http://hpdrc.fiu.edu/）。两个数据集都是纯文本文件（制表符分割），每个空间对象占一行。第一个数据集包含的对象是一些有代表性的旅馆，作为旅馆数据集。第二个数据集作为餐馆数据集，包含餐馆数据。Table 1 展示了这两个数据集的详细信息。

TABLE 1: DATASET DETAILS

Dataset	Size (MB)	Total # of objects	Average # unique words per object	Total # unique words in dataset	Average # disk blocks per object
Hotels	55.2	129,319	349	53906	2
Restaurants	61.3	456,288	14	73855	1

在所有的实验中，磁盘块大小是 4096KB。同时，R-Tree 的一个节点的子节点数量，根据每个节点占一个磁盘块计算。也就是说，在实现时，每个节点可以容纳 113 个子节点。对于 IR^2-Tree 和 MIR^2-Tree，也使用的相同数量，因此，每个节点需要占两个磁盘块。实验表明，虽然 IR^2-Tree 和 MIR^2-Tree 需要额外的磁盘块开销，但是，

这对执行时间的影响很小。

为了比较 IR^2-Tree、MIR^2-Tree、R-Tree 和 IIO 算法的性能，我们进行了三组实验。第一组实验测试当需要的结果数 k 不同时的算法性能；第二组实验测试查询关键词的数量的影响；最后，第三组实验测试签名长度 r 的影响。

k（top-k）的取值

在这组实验中，我们把查询关键词的数量固定位 2，签名长度固定位 189bytes（对于旅馆数据集）和 8bytes（对于餐馆数据集）。需要注意的是，MIR^2-Tree 的顶层的签名长度会更长，它使用可变签名长度。这些签名长度的选取，需要使空间的需求和性能保持平衡。对旅馆和餐馆的实验结果分别在 Figure 9 和 Figure 12 中。图中 Y 轴使用对数标度，以使差异更直观。

从图中可以看出，IR^2-Tree 和 MIR^2-Tree 在 k 的所有取值下，性能都比 R-Tree 好。这是因为 R-Tree 方法需要访问更多的对象和潜在的树节点。相反，IR^2-Tree 和 MIR^2-Tree 使用签名来修剪整个子树。特别的，MIR^2-Tree 在过滤内部节点上做得更好，因为它对树的每一层采用了最佳签名长度，正如第 4 节所描述的。

Figure 9(b)和 12(b)显示了各个算法访问的磁盘块数量。柱状图表示随机磁盘块访问次数，而上面的细线表示顺序磁盘块访问次数。和预期的一样，执行时间主要和随机访问次数成正比。值得注意的是，由于修剪的效果，MIR^2-Tree 比 IR^2-Tree 进行更少的随机磁盘访问，但顺序磁盘访问更多。这主要是因为 MIR^2-Tree 的顶部节点由于签名更长而占用更多的磁盘块。IIO 算法对 k 的取值不敏感，因为它需要检查包含所有关键词的所有对象。

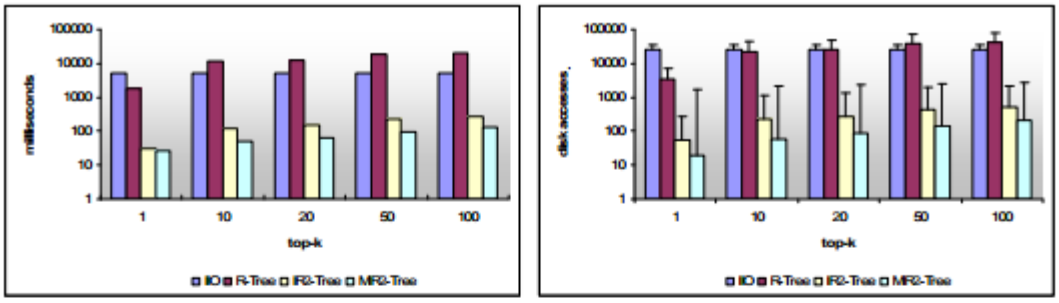
关键词数量

在这组实验中，我们把需要的对象数量 k 固定为 10，签名长度跟上一组实验相同。实验结果请参阅 Figure 10 和 Figure 13。在关键词数量增加的同时，包含全部关键词的对象也减少了（因为距离优先空间关键词查询时合取）。我们注意到，IIO 算法的性能随着关键词数量增多而变好，这是由于倒排索引表的交集变小，需要访问的对象变少。

签名长度

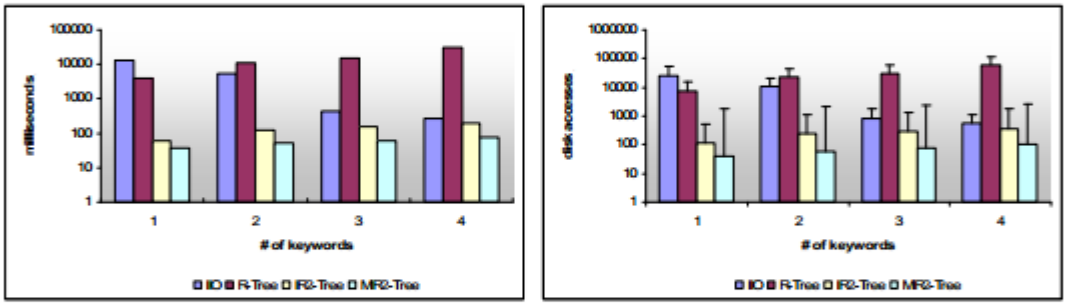
在这组实验中，我们把 k 固定为 10，关键词数量固定为 2。实验结果请参阅 Figure 11 和 Figure14。首先要注意，旅馆数据集选择的签名和餐馆数据集是不同的，这是因为一个旅馆对象包含更多的独特关键词，如 Table 1 所示。需要注意，在 MIR^2-Tree 中，签名长度是指叶子节点使用的签名长度，而顶部节点则使用更长的签名。在增加

IR^2-Tree 和 MIR^2-Tree 的签名长度时，有一个权衡。增加签名长度，能够减少误判，但是也会增加树结构占用的空间，从而增加磁盘访问。因此，签名长度变化对实验结果的影响并不明确。



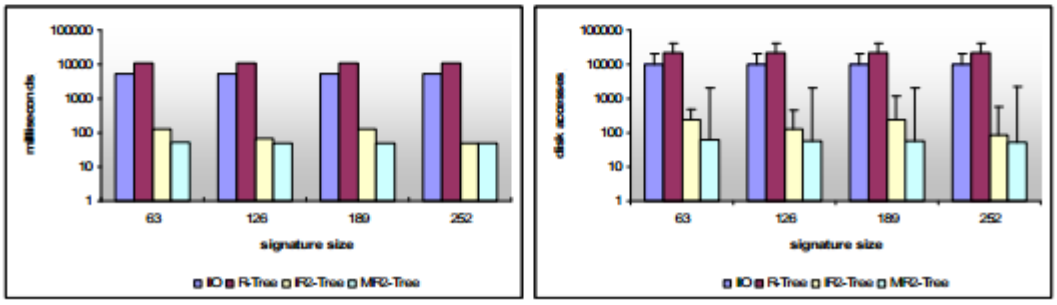
(a) Execution Time　　(b) Object Accesses

Figure 9: Results for varying k (top-k) searches for the Hotels dataset



(a) Execution Time　　(b) Object Accesses

Figure 10: Results for varying number of keywords for the Hotels dataset

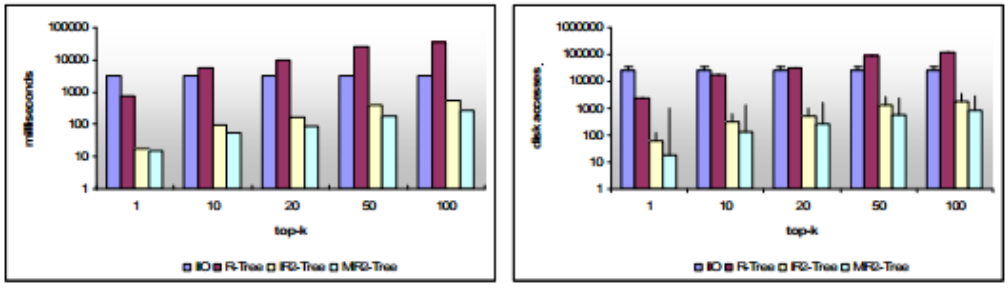


(a) Execution Time　　(b) Object Accesses

A. 需要的空间

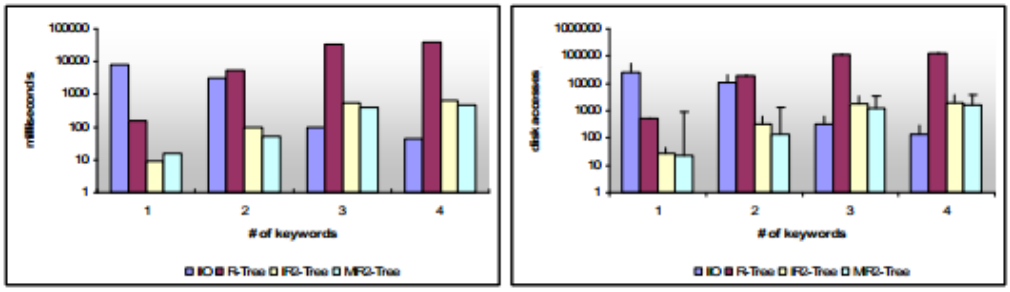
Table 2 显示了每个结构需要的总空间（以 MB 为单位）。对于 IR^2-Tree 和 MIR^2-Tree，我们考虑的实例是前面“top-k 的取值变化”和“关键词数量变化”时的实验，即，分别为 189 和 8bytes。

餐馆数据集的 IIO 结构大小显著地小于旅馆数据集。这是因为，餐馆数据集中平均每个对象包含的独特关键词更少，如 Table 1 所示。另一方面，餐馆数据集的树结构比旅馆数据集更大，因为有更多的对象。



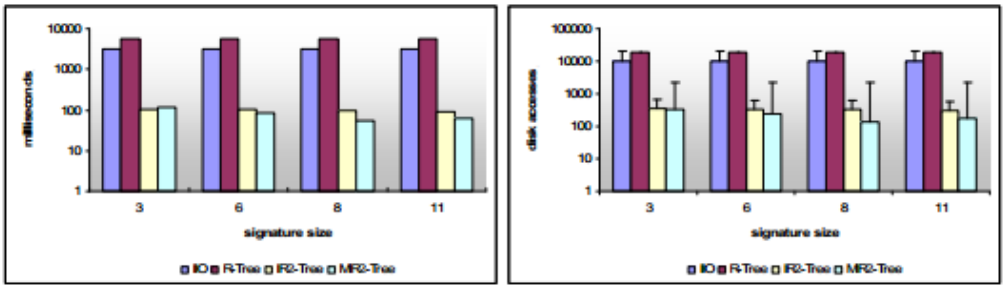
(a) Execution Time (b) Object Accesses

Figure 12: Results for varying k (top-k) searches for the Restaurants dataset



(a) Execution Time (b) Object Accesses

Figure 13: Results for varying number of keywords for the Restaurants dataset



(a) Execution Time (b) Object Accesses

B. 讨论

如“签名长度变化”的实验所示，通过增加签名长度，为了消除误判而对空间对象和节点进行的磁盘访问减少了。另一方面，长签名又增加了 IR^2-Tree 和 MIR^2-Tree 的大小。由于 IR^2-Tree 中签名长度是所有节点都一致的，因此签名的长度对 IR^2-Tree 的影响要比 MIR^2-Tree 更大。签名长度只影响 MIR^2-Tree 的叶子节点，因为内部节点会根据子树引用的对象重新计算。

TABLE 2: SIZES (MB) OF INDEXING STRUCTURES

Dataset	IIO	R-Tree	IR^2-Tree	MIR^2-Tree
Hotels	31.4	6.9	34.5	44.9
Restaurants	7.2	23.9	47.2	68.2

在极少数情况下，当查询关键词只出现在很少的对象中时，IIO 算法会快很多，因为倒排索引表很小。在另一极端下，如果查询关键词几乎出现在所有对象中，则 R-

Tree 性能比较好。总体来说，MIR^2-Tree 的性能比 IR^2-Tree 好，但是 MIR^2-Tree 维护的成本高。因此，对于经常更新的数据集来说，IR^2-Tree 是最好的选择。

7. 近期研究

最近邻查询

空间数据库上的 K 最近邻查询是一个经典的数据库问题。大多数方法使用在数据上建立的索引来协助最近邻搜索。目前最广泛使用的算法应该是分支定界算法[RKV95]，通过遍历 R-Tree[Gut84]，并在优先队列中保存 k 个可能的最近邻的列表。也有尝试用范围查询解决 K 最近邻搜索问题的，比如 Korn 等人[KSF+96]提出的。其基本思想是用范围查询来搜索潜在的 K 最近邻。这个算法的延伸有通过改善区域估计[CG99]和对区域中的 k 最近邻使用更好的搜索技术[SK98]等。

在本文中用到的，Hjaltason 和 Samet[HS99]提出的使用 R-Tree 的增量最近邻算法出现之前，增量最近邻问题的解决方法已经经历了三种不同的数据结构：[Bro90,Hen94,HS95]分别使用 k-d 树，LSD 树和 PM 四叉树。这些算法本质上都类似，主要区别在于执行时采用的数据结构。最近出现了关于持续最近邻查询的研究[TPS02,XMA05]，持续地查找查询点的 k 个最近的对象。Park 和 Kim[PK03]自主开发了与我们类似的方法来解决最近邻查询问题，他们使用 R 树和多个 S 树的组合，每一个针对一个非空间属性。

签名文件

签名文件由 Faloutsos 和 Christodoulakis[FC84,FC85,Fal85]引入，作为高效搜索文本文件集合的方法。Lee 等人[LKP95]提出了在签名文件上建议数据结构的方法。在这项工作中，我们看到文件中用符号文本块来描述空间对象，并在这些对象上建议类似的结构。特别地，我们采用了索引描述文件结构的想法[PBC80]（S-Tree[Dep86]是索引描述的一个变种），即一棵最底层有签名块组成的树。这些是从文本块获得的叠加码。第 i 层的一组签名 b 叠加在一起组成了第 i-1 层的签名。每一层的签名长度都相同。类似的，在我们的 IR^2-Tree 中，父节点的签名由子节点的签名叠加而成。

最后，在建立索引描述文件时，我们需要顶层有更多的 1，因为子树中有大量的词，但这也会导致更多的误报。多级叠加编码原理[CS89,DR83]作为这个问题的解决方案提出，其中高层的有更长的签名。这个原理通过增加空间开销而减少了误报。然而，这使得相关文件的更新维护开销变得昂贵。

Top-k 查询

Top-k 查询研究[Fag01, BGM02]在属性值属于不同来源的情况下，处理对象的属性值集。例如[BGM02]解决一系列餐馆按距离和价格排布的问题。与我们不同，他们把这些来源看作黑盒子，而我们假设可以完全访问来建立 IR^2-Tree。

Zhou 等人[ZXW+05]提出的技术组合倒排索引和 R-Tree 来解决 Web 查询问题，他们在页面上通过空间信息进行约束。他们的实验表明，倒排索引的关键词列表按 R^* 树组织时性能最佳。然而，他们的算法不是 top-k 的，即，他们需要空间区域作为输入。

此外，他们没有很好地扩展到多个关键词，因为多个 R*树必须要遍历和相交（组合算法还没有提出）。Vaid 等人[VJJS05]和 Martins 等人[MSA05]提出的技术，通过组合文本输出和空间索引来解决空间关键词查询。这些技术和我们使用的基线算法非常相似。然而，他们没有考虑到在一个简单的结构中组合这些索引，就像我们的 IR^2-Tree。进一步，Vaid 等人[VJJS05]对空间对象使用基于网格的分布。

8. 总结

在这篇论文中，我们引入了空间关键词搜索的问题，并阐述了现有方法的性能局限性。然后，提出了一个比现有方法更快的解决方案，基于 R-Tree 和签名文件的组合技术。特别地，我们引入了 IR^2-Tree，并说明了在数据更新时的维护方法。同时，也提出了一个使用 IR^2-Tree 的算法，高效地解决空间关键词查询。通过实验测试，证实了它的卓越性能。

9. 参考文献

[Bro90] A. J. Broder. Strategies for efficient incremental nearest neighbor search. In Pattern Recognition, 23(1–2):171–178, January 1990.

[BGM02] Nicolas Bruno, Luis Gravano, Amélie Marian.Evaluating Top-k Queries over Web-Accessible Databases., ICDE 2002.

[CG99] S. Chaudhuri and L. Gravano. Evaluating top-k selection queries. In VLDB, 1999.

[CS89] W. W. Chang, Hans-Jörg Schek: A Signature Access Method for the Starburst Database System. VLDB 1989: 145-153

[CSM06] Yen-Yu Chen, Torsten Suel, Alexander Markowetz. Efficient Query Processing in Geographic Web Search Engines. SIGMOD 2006

[Dep86] U. Deppisch. S-Tree: A dynamic balanced signature index for office retrieval. In Proc. of the ACM Conf. on Research and Development in Information Retrieval, Pisa, 1986.

[DR83] Ron Sacks-Davis, Kotagiri Ramamohanarao: A two level superimposed coding scheme for partial match retrieval. Inf. Syst. 8(4): 273-289 (1983)

[Fag01] Ronald Fagin, Amnon Lotem, Moni Naor: Optimal Aggregation Algorithms for Middleware. In PODS 2001

[Fal85] Christos Faloutsos: Signature files: Design and Performance Comparison of Some Signature Extraction Methods. In SIGMOD Conference 1985

[FC84] Christos Faloutsos, Stavros Christodoulakis:Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. In ACM Trans. Inf. Syst. 2(4): 267-288(1984)

[FC85] Christos Faloutsos, Stavros Christodoulakis: Design of a Signature File Method that Accounts for Non-Uniform Occurrence and Query Frequencies. In VLDB 1985: 165-170

[FO95] C. Faloutsos, D. W. Oard. A survey of information retrieval and filtering methods. Technical Report. UMI Order Number: CSTR-3514., University of Maryland at College Park, 1995

[Gut84] A. Guttman. R-Trees: a dynamic index structure for spatial searching. In SIGMOD Conference, 1984. [Hen94] A. Henrich. A distance-scan algorithmfor spatial access structures. In Proceedings of the Second ACM Workshop on Geographic Information Systems, pages 136–143, Gaithersburg,MD, December 1994.

[HS95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In Advances in Spatial Databases — Fourth International Symposium, pages 83–95, Portland, ME, August 1995. [HS99] G.R. Hjaltason and H. Samet. Distance browsing in spatial databases. In ACM Transactions on Database Systems,Vol. 24, No. 2, 1999

[KSF+96] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In VLDB, 1996. [LKP95] Dik Lun Lee, Young Man

Kim, Gaurav Patel: Efficient Signature File Methods for Text Retrieval. Pages 423-435. TKDE Vol 7, Number 3, June 1995

[MC94] Malcolm Campbell. The Design of Text Signatures for Text Retrieval Systems. Technical Reports 1994 [MSA05] B. Martins, M. Silva, and L. Andrade. Indexing and ranking in Geo-IR systems. In Proc. of the 2nd Int. Workshop on Geo-IR (GIR), November 2005.

[NMN+00] G. Navarro, E. Silva de Moura, M. S. Neubert, N. Ziviani, R. A. Baeza-Yates: Adding Compression to Block Addressing Inverted Indexes. Inf. Retrieval 3(1): 49-77 (2000), 2000 [PBC80] John L. Pfaltz, William J. Berman, Edgar M. Cagley: PartialMatch Retrieval Using Indexed Descriptor Files. In Commun. ACM 23(9): 522-528 (1980)

[PK03] D. Park, H. Kim: An Enhanced Technique for k-Nearest Neighbor Queries with Non-Spatial Selection Predicates. In Multimedia Tools and Applications archive, Volume 19 , Issue 1 (January 2003), Pages: 79 – 103 [RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In SIGMOD Conference, 1995.

[Sal97] D. Salomon. Data Compression. The Complete Reference. Springer, New York, 1997. [Sin01] A. Singhal: Modern Information Retrieval: A Brief Overview, Google, IEEE Data Eng. Bull, 2001

[SK98] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In SIGMOD Conference, 1998.

[TPS02] Yufei Tao, Dimitris Papadias, and Qiongmao Shen. Continuous Nearest Neighbor Search. In VLDB, 2002.

[VJJS05] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. SSTD 2005.

[XMA05 Xiaopeng Xiong, Mohamed F. Mokbel, Walid G. Aref: SEACNN: Scalable Processing of Continuous K-Nearest Neighbor Queries in Spatio-temporal Databases. In ICDE 2005

[ZMR98] J. Zobel, A. Moffat, K. Ramamohanarao: Inverted Files Versus Signature Files for Text Indexing. In ACM Trans. Database Syst. 23(4): 453-490 (1998)

[ZXW+05] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.Ma. Hybrid index structures for location-based web search. ACM CIKM 2005