Top-k 空间关键字查询的新方法探究

廖溢机　刘中喆

1. 前言

空间关键字查询是信息检索领域的一个热门研究问题，特别是在如今这个信息爆炸的时代，空间关键字查询对我们的生活起着非常重要的作用。随着电脑与智能手机的普及，以及互联网时代的迅速发展，越来越多的人需要从海量信息中快速获取自己所需要的信息。与此同时，随着卫星地图等技术的发展，关于空间地理信息的查询需求也与日俱增，诸如微博、twitter 等社交应用也都提供了关于空间地理信息的搜索功能，例如“找附近的宾馆、酒店”等。给定一个包含空间地理信息（比如坐标）和文本关键字（比如酒店信息）的查询，如何快速地返回若干个高匹配度的结果，是一个值得深入研究的问题。

2. 问题定义

一个空间关键字由一个空间位置坐标和一组文本关键词词项构成。我们所研究的问题是，在一个由若干篇包含空间位置坐标的文档构成的文档库中，给出若干个空间关键字查询，要求对于每个查询，根据给定的同时考虑空间邻近度和文本相关性的评分函数，**精确地**输出前 k 篇评分最高的文档。

一个最简单的想法是，对一个查询中的每个词项，都扫描一遍整个文档库，找出每个词项在每篇文档中的评分，然后再根据坐标求出该查询与每篇文档的空间距离评分，即可得到该查询与每篇文档的总评分。最后将所有的文档按评分排序，取前 k 个评分最高的文档。然而，这样做的时间开销将取决于文档总数目多少，而在实际应用中，我们面对的经常是海量的信息，**文档总数往往十分庞大**，若对所有查询的每个属性，都去扫描整个文档库，时间开销难以承受。而用户总是希望能尽快地获取查询信息，因此，直接这样做往往是不可行的。

另外，我们在日常生活中使用搜索引擎进行查询时，大多数用户一般只会输入 2~3 个关键词，关键词包含了我们希望获得的内容的核心意思即可。关于空间关键字查询也是如此。例如，我们希望找附近的餐厅，那么查询关键词中，首先

要包含“餐厅”或“饭店”这个词；其次，我们希望搜到符合我们口味的餐厅，比如“西式”、“麻辣”、“家常口味”等；另外，我们希望能有我们喜爱的菜肴，比如“酸甜排骨”、“鱼香肉丝”等。那么根据自己的需求，只需输入 3 个关键词，再结合地理位置的远近，基本上就可以找到满意的餐厅了，输入太多的词，用户也会觉得不方便。这意味着，一个查询中包含的词项往往是很少的。

同时，对于查找出来的结果，用户一般只会对排在前面的几十个感兴趣，而不会对成千上万的结果一一浏览，因为用户往往没有那么多的时间和精力。

因此，从实用性的角度出发，我们所研究的问题，对数据范围有一定针对性：**文档总数十分庞大；每个查询包含的词项个数很少（一般就几个）；查询结果 top-k 的 k 值较小（几十左右）。**

关于空间关键字查询问题，有许多相关的研究，其中具有代表性的包括对 R-tree 和 Quadtree 及其变种的研究，它们都是良好的空间索引数据结构。而关于文本查询，常见且实用的方法是倒排索引，因此许多关于空间关键字查询问题的研究工作，都在以上二者的基础上展开。

而我们所探究的称为 RCA（Rank-aware CA）的空间关键字查询新方法，在空间和文本上都仅基于倒排索引。RCA 的思路是将位置信息也像文本信息一样建立倒排索引，并针对评分函数的特性，利用基于 CA 算法思想改进后的方法快速地解决 Top-k 空间关键字查询问题。相比于其他方法，它不需要对空间用专门的数据结构维护，因此显得更加简便。同时，它是一个十分高效的算法。

在探究 RCA 之前，首先要了解 TA（Threshold Algorithm）、NRA（No Random Access Algorithm）和 CA（Combined Algorithm），它们是处理 Top-k 聚合查询问题的算法。我们在后面会介绍这些算法。

算法中的参数、模型结构与评分函数：

文档总数：n

文档结构： $D=\langle docID, x, y, terms\rangle$

其中 docID 表示文档 id；(x, y)表示文档坐标；terms 表示该篇文档所包含的带权词项集合。

文档 D 的每一个词项 $term_i$ 在文档 D 中的评分：$\phi_t\left(D,term_i\right)=tfidf_{term_i,D}$

查询的结构：$Q=\left\langle x_q,y_q,w_1,w_2,...,w_m,k\right\rangle$

其中（x_q，y_q）表示查询 Q 的坐标，{w_1，w_2，…，w_m} 表示查询 Q 包含的词项，k 表示需要返回前 k 篇文档。

我们也称文档 D 是一个对象，一个对象有 m+1 个属性，m+1 个属性的值分别为查询 Q 的每个词项在 D 中的评分以及 D 与 Q 的空间邻近度评分。

文档 D 与查询 Q 的文本相关性评分：

$$\phi_t(\mathcal{D},Q)=\sum_{w_i\in Q}\phi_t(\mathcal{D},w_i)$$

文档 D 与查询 Q 的空间邻近度评分：

$$\phi_s(\mathcal{D},Q)=1-\frac{d(\mathcal{D},Q)}{\gamma}$$

其中 d（D，Q）是 D 与 Q 之间的欧氏距离，γ 是所有文档与查询之间距离的最大值，这样使得所有的评分都在 0 与 1 之间。

文档 D 与查询 Q 的综合评分：

$$\phi(\mathcal{D},Q)=\alpha\cdot\phi_t(\mathcal{D},Q)+(1-\alpha)\cdot\phi_s(\mathcal{D},Q)$$

其中 α 为一个定值，α 的大小表示了对文本评分和空间评分权重的分配。

算法的输入：

算法首先输入的是一个大型文档库，由许多篇带有一个坐标和一组单词的文档构成。接着是若干个空间关键字查询 Q。

算法的输出：

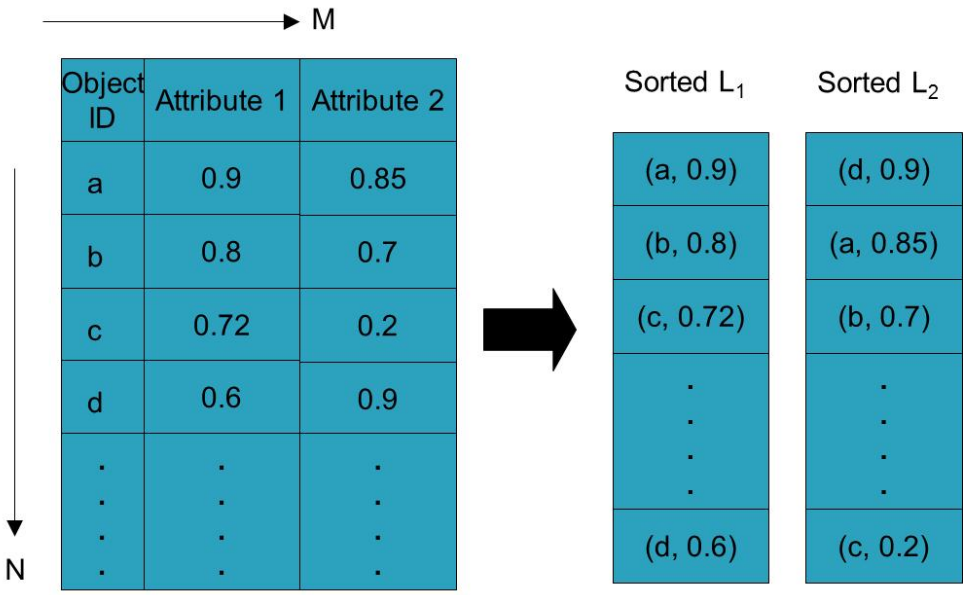
对于每个空间关键字查询 Q，精确地输出前 k 篇评分最高的文档。

3. 算法描述

因为查询是任意的，要想知道关于某个特定的查询，评分排前 k 的文档是哪些，就必须要访问文档库。而前面提到的最简单的方法之所以慢，是因为它访问了文档库中的所有文档。因此，我们希望一开始就访问那些高分的文档，并且尽量避免访问或以较小的代价访问那些低分的文档。通过从访问过的文档中获取的信息，我们可以判断某些条件是否成立，当某些条件成立时可以保证排名 top-k 的文档都已经被访问过，此时我们便可以终止访问。这样我们就不需要访问整个文档库，从而节省大量时间。本文涉及的算法便是基于这种思想。我们可以针对位置信息和文本信息在查询开始前进行一些离线预处理，从而在查询时就可以优先访问到高分的文档。

首先注意到每个词项在每篇文档中的评分是与查询无关的，也就是说词项的评分在查询开始前就可以预先计算出来。因此，我们先对每个词项建立一个倒排列表。其次，高分的对象，它的某个或某些属性的值也是很高的，因此，我们可以将每个倒排列表中的文档按照该词项在各文档中的评分从大到小排列，这样在对一个倒排列表进行访问时我们便可以按照该评分从高到低的顺序进行访问，从而有更高的可能性优先访问到总评分高分的文档。虽然对这些倒排列表排序的时间开销很大，但是它只会进行一次，并且是在查询开始前进行的，不会在查询过程中占用时间，相对于大量的查询所花费的时间而言，这一点预处理的开销是尚可接受的。

我们先仅考虑文本评分。在查询开始前，我们先对所有词项建立倒排列表，每个倒排列表按照文档评分从大到小排序。接着，对于一个由 m 个查询关键词组成的查询，我们取出这 m 个词对应的 m 个倒排列表。



在描述算法前，先要介绍几个重要概念：

顺序访问：从列表的头部第一项开始按顺序访问列表中的所有项目。因此如果一个对象 R 在第 i 个列表中有着第 h_i 高的属性值，那么访问到 R 的这个属性值需要进行 h_i 次顺序访问。

随机访问：在第 i 个列表中可通过一次随机访问得到对象 R 在第 i 个列表中的属性值。

访问代价：如果顺序访问有 s 次，随机访问有 r 次，且每次顺序访问的代价为 c_S，每次随机访问的代价为 c_R，那么总的访问代价为 sc_S+rc_R。

聚合函数：如果 x_1，…，x_m 是对象 R 的 m 个属性评分，那么 $t(x_1, \ldots, x_m)$ 是对象 R 的总体评分，我们称 t 为聚合函数。

单调聚合函数：对于一个聚合函数 t，若 $x_i \le x_i'$ 对每个 i 都成立时，有 $t(x_1,\ldots,x_m) \le t(x_1',\ldots,x_m')$，那么我们称 t 为单调聚合函数。

下面我们将分别介绍 TA、NRA、CA 与 RCA。请注意，它们的正确性的前提是：评分函数 t 为**单调聚合函数**。它们在之前所述的条件下（文档总数很大、每个查询包含的词项个数很少、top-k 的 k 值较小）被证明表现得十分优秀。

TA 算法（Threshold Algorithm）：

概念定义：

平行顺序访问：先访问每一个列表的第一层属性值，再访问每一个列表的第二层属性值，以此类推。

算法过程：

1.对 m 个已经排序好的列表L_i进行平行顺序访问，对于在一个列表的顺序访问下每一个被看到的对象 R，在其他剩余的表中对它进行随机访问找到这个对象 R 在每一个列表L_i中的属性值x_i，然后对于 R 计算出它的聚合函数值$t(R) = t(x_1, \ldots, x_m)$，如果这个函数值是当前已经看到的最高的 k 个聚合函数值之一，那么就记录对象 R 以及它的$t(R)$（对于相同$t(R)$对象顺序可以任意选取），所以在任何时候只有 k 个对象以及他们的聚合函数值需要被记录。

2.设$\underline{x}_i$（注意有下划线，与前面的x_i不同）为到目前为止，列表L_i中最后一个顺序访问到的值，定义临界值τ为$t(\underline{x}_1, \ldots, \underline{x}_m)$，一旦至少有 k 个被记录的对象并且它们的聚合函数值都大于等于τ，算法停止。此时记录列表中的 k 个对象即为聚合函数值前 k 大的对象。

TA 对当前每一个被顺序访问到的对象立刻通过随机访问获取它的全部属性值并计算出总评分，因此它的顺序访问数目是相当少的，主要时间开销在随机访问上。TA 在随机访问代价较小的系统中能够取得较好的结果。

NRA 算法（No Random Access Algorithm）：

而有的系统并不支持随机访问，对于这种系统，可以使用 NRA 算法，NRA 只进行顺序访问而不进行随机访问。

概念定义：

在顺序访问的过程中对于每一个对象 R 而言，有一个子集 $S(R) = \{i_1, i_2, \ldots, i_\ell\} \subseteq \{1, \ldots, m\}$ 为对象 R 已经被访问到的属性编号的集合，同时也就有这些属性的值 $x_{i_1}, x_{i_2}, \ldots, x_{i_\ell}$，由此可以计算出一个已经被看到的对象的聚合函数值的下界与上界。

对象 R 的聚合函数值的下界 $W_S(R)$：

$W_S(R)$ 为对象 R 在当前已知的属性值 $x_1, x_2, \ldots, x_\ell$ 的情况下可能的最小聚合函数值，它通过这个公式计算：

$$W_S(R) = t(x_1, x_2, \ldots, x_\ell, 0, \ldots, 0)$$

对象 R 的聚合函数值的上界 $B_S(R)$：

$B_S(R)$ 为对象 R 在当前已知的属性值 $x_1, x_2, \ldots, x_\ell$ 的情况下可能的最大聚合函数值，它通过这个公式计算：$B_S(R) = t(x_1, x_2, \ldots, x_\ell, \underline{x}_{\ell+1}, \ldots, \underline{x}_m)$。与在前面 TA 中的定义相同，设 $\underline{x}_i$ 为到目前为止，列表 L_i 中最后一个顺序访问到的值，我们称其为底部值。

算法过程：

1.对 m 个已经排序好的列表 L_i 进行平行顺序访问，在每一个访问深度 d（每一列表第 d 个对象都已经顺序访问过）时：

- 保存这个深度 d 的每一个列表的底部值 $\underline{x}_1^{(d)}, \underline{x}_2^{(d)}, \ldots, \underline{x}_m^{(d)}$
- 对于每一个对象 R，它在这一访问深度 d 已经被访问过的属性集合为 $S = S^{(d)}(R) \subseteq \{1, \ldots, m\}$，计算它在这一访问深度的聚合函数下界值 $W^{(d)}(R) = W_S(R)$ 与上界值 $B^{(d)}(R) = B_S(R)$（对于在访问深度 d 时还

没有被访问到的对象 R，它的下界值与上界值可以分别被计算为 $W^{(d)}(R) = t(0, \ldots, 0)$和$B^{(d)}(R) = t(x_1, x_2, \ldots, x_m)$）

- 设$T_k^{(d)}$为到目前访问深度为止，在已经被访问到的对象中下界值$W^{(d)}$前 k 大的 k 个对象所组成的集合，如果两个对象有着相同的下界值，则上界值更大的那一个排在前面，如果上界值也相同则这两个对象的顺序任意。设$M_k^{(d)}$为$T_k^{(d)}$中排名第 k 的对象的下界值。

2.如果一个对象 R 满足$B^{(d)}(R) > M_k^{(d)}$，则称这个对象为*可行的*。算法停止的条件为：至少有个 k 个对象已经被访问到（也就是$T_k^{(d)}$包含有 k 个对象），并且在$T_k^{(d)}$外没有可行的对象（也就是对于任意对象$R \notin T_k^{(d)}$，有$B^{(d)}(R) \leqslant M_k^{(d)}$）。此时$T_k^{(d)}$中的 k 个对象即是输出结果。

NRA 被证明在无法随机访问的情况下是很优秀的，但实际运用中，NRA 的效率是很低的（我们后面的实验结果中会展示），这是因为它每次顺序访问后都需要更新所有已访问过的并且属性值并未完全获取到的对象的上界值，这里造成了很大的时间开销。

CA 算法（Combined Algorithm）：

有的系统虽然既能进行顺序访问也能进行随机访问，但是随机访问的时间代价要远大于顺序访问。CA 可看作是 NRA 与 TA 的结合，也就是在 NRA 的基础上每平行顺序访问了$h = \lfloor c_R / c_S \rfloor$个对象后进行一次随机访问以取得较好的折中效果。

算法过程：

1.对 m 个已经排序好的列表L_i进行平行顺序访问，在每一个访问深度 d（每一列表第 d 个对象都已经顺序访问过）时：

- 保存这个深度 d 的每一个列表的底部值 $\underline{x}_1^{(d)}, \underline{x}_2^{(d)}, \ldots, \underline{x}_m^{(d)}$
- 对于每一个对象 R，它在这一访问深度 d 已经被访问过的属性集合为 $S = S^{(d)}(R) \subseteq \{1, \ldots, m\}$，计算它在这一访问深度的聚合函数下界值 $W^{(d)}(R) = W_S(R)$ 与上界值 $B^{(d)}(R) = B_S(R)$（对于在访问深度 d 时还没有被访问到的对象 R，它的下界值与上界值可以分别被计算为 $W^{(d)}(R) = t(0, \ldots, 0)$ 和 $B^{(d)}(R) = t(\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_m)$）
- 设 $T_k^{(d)}$ 为到目前访问深度为止，在已经被访问到的对象中下界值 $W^{(d)}$ 前 k 大的 k 个对象所组成的集合，如果两个对象有着相同的下界值，则上界值更大的那一个排在前面，如果上界值也相同则这两个对象的顺序任意。设 $M_k^{(d)}$ 为 $T_k^{(d)}$ 中排名第 k 的对象的下界值。

2.如果一个对象 R 满足 $B^{(d)}(R) > M_k^{(d)}$，则称这个对象为*可行的*。每 $h = \lfloor c_R / c_S \rfloor$ 步（也就是说访问深度 d 每增加 h 时），在所有可行且并非所有的属性都被访问过的对象中找到一个当前访问深度的上界值 $B^{(d)}$ 最大的对象 R（当有好几个符合要求的对象上界值相同时任取其中一个），对这个对象所有还没有访问过的属性值进行随机访问。如果不存在这样一个对象的话，这一步就不进行随机访问。

3.算法停止的条件为：至少有个 k 个对象已经被访问到（也就是 $T_k^{(d)}$ 包含有 k 个对象），并且在 $T_k^{(d)}$ 外没有可行的对象（也就是对于任意对象 $R \notin T_k^{(d)}$，有 $B^{(d)}(R) \leqslant M_k^{(d)}$）。此时 $T_k^{(d)}$ 中的 k 个对象即是输出结果。

CA 因为在顺序访问和随机访问之间取了一个折中的方案，因此也是相当优秀的。我们会在后面的实验结果展示，在实际运行中 CA 与 TA 难分上下。下面我们将 TA 与 CA 进行比较并进行优缺点分析。

TA 的优点：

1. 顺序访问少。TA 的顺序访问要比 CA 小得多，因为 TA 每次顺序访问后都会立刻通过随机访问来获取当前对象全部的属性，减少了后面顺序访问的量。

2. 对于已经访问过的对象，只需要记录其中最大的 k 个（用一个二叉堆维护即可），因为它们所有的属性值都已知，也即总评分已知；并且不需要每次对已经访问过的对象的信息进行修改。

TA 的缺点：

需要进行大量随机访问，因为 TA 对每个顺序访问到的对象，都需要进行随机访问以获取该对象全部属性的值。多数实际情况中，一次随机访问的代价往往比一次顺序访问的代价要大得多。

CA 的优点：

随机访问少。CA 只是选择性地进行随机访问，它把那些顺序访问到的对象储存起来并且只对其中有最大潜力的那一个进行随机访问。

CA 的缺点：

需要记录所有已经访问过的对象的信息，因此需要使用更多的储存空间；更糟糕的是，CA 每次顺序访问后都需要更新所有已访问过的并且属性值并未完全获取到的对象的上界值，因此这里造成了很大的时间开销。当对象总数 n 很大的时候，维护上界值的开销相当大，已经远远超过减少随机访问所节省下来的时间。CA 的主要运行时间几乎都花在了这里。

避免维护上界值的方法：

要想提高 CA 的速度，就必须想办法避免维护上界值。在 NRA 中这是没法做到的，因为 NRA 不能随机访问，但在 CA 中可以做到。观察可知，上界值只用在了两个地方：一是在随机访问时要在可行的且属性值未完全获取的对象中找上界值最大的对象进行访问，二是用于判断 $T_k^{(d)}$ 之外是否有可行的对象。

对于第一个地方，我们**直接对 $T_k^{(d)}$ 中的所有属性值未完全获取的对象全部都进行随机访问**。因为每个对象的上界值都大于等于下界值，每个属性值未完全

获取的对象上界值都大于下界值，因此$T_k^{(d)}$中所有属性值未完全获取的对象的上界值一定都大于$T_k^{(d)}$中最小的下界值，也就是说$T_k^{(d)}$中的对象都是可行的。并且，有着最大上界值的对象出现在$T_k^{(d)}$中的可能性很大。即使不在$T_k^{(d)}$中也没关系，因为维护上界值的开销更大，并且最大与否不会对算法正确性造成影响。另外，这样做虽然可能进行了多次随机访问，但至多也就 k 次，k 比较小还是可以接受的，并且这样可以使$T_k^{(d)}$中的最小值$M_k^{(d)}$上升得更快，从而更早地终止算法。

对于第二个地方，首先直接计算当前这一层底部值的评分，即未访问过的对象的上界值，用于判断未访问过的对象是否可行（该上界值是否大于$M_k^{(d)}$）；在该上界值小于等于$M_k^{(d)}$（即未访问过的对象都不可行的）前提下，接着扫描已经访问过的对象的列表，判断那些不在$T_k^{(d)}$中（用一个数组记录是否在$T_k^{(d)}$中）的对象是否可行（这里需要计算这些对象的上界值，但开销很小）**若可行就对它们进行随机访问**（这时候$T_k^{(d)}$也要不断更新，用堆实现）。这个操作结束之后，$T_k^{(d)}$之外就没有可行对象了，可直接终止算法，因此也只用扫描一次访问过的对象列表计算上界值，而不需要每次维护上界值。

RCA 算法（Rank-aware CA）：

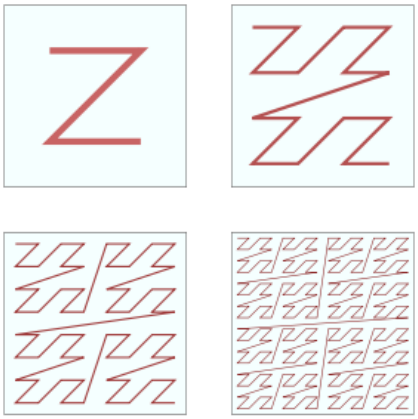
在空间关键字查询中，如果只考虑文本评分，可在建立倒排索引后使用 TA 和 CA 解决，但如果要用 TA 和 CA 处理空间评分的话，就必须对每个查询 Q 都扫描一次整个文档库，计算空间评分，排序后再与文本评分一起用 TA 和 CA 处理。虽然 TA 和 CA 不需要对每个查询 Q 都扫描整个文档库来计算文本评分，对比朴素算法而言，在文本信息处理上节省了许多时间，但是在空间信息处理上依然无法节省时间。

RCA 算法其中一个核心思想，就是让空间信息也能像文本信息一样离线预处理，即对空间信息也离线进行倒排索引。这样就要求对每一个固定的坐标，它的

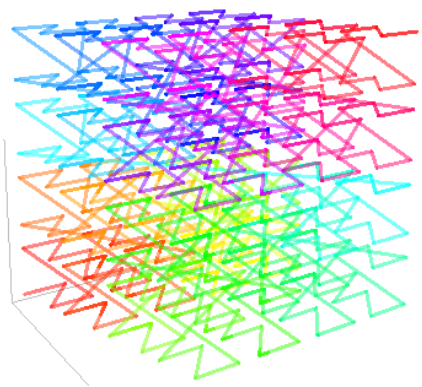
评分是固定的，与查询无关，就好像一个词项在一个确定的文档库中 tfidf 评分是固定的。因此，我们需要对每一个确定的二维坐标进行编码，之后对该列表进行顺序访问时的顺序是根据该编码的顺序进行的，这就需要该编码能近似地反映空间的邻近程度。RCA 算法采用的是 Z-order 编码，它是计算机图形学中的一种将高维空间坐标映射到一维空间上的编码方式，通过在坐标的各维的二进制表示中轮流取位交错而得。对编码按从小到大的顺序连接后可形成迭代的 Z 型线条，如下图所示。

x:	0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
y: 0 000	000000	000001	000100	000101	010000	010001	010100	010101
1 001	000010	000011	000110	000111	010010	010011	010110	010111
2 010	001000	001001	001100	001101	011000	011001	011100	011101
3 011	001010	001011	001110	001111	011010	011011	011110	011111
4 100	100000	100001	100100	100101	110000	110001	110100	110101
5 101	100010	100011	100110	100111	110010	110011	110110	110111
6 110	101000	101001	101100	101101	111000	111001	111100	111101
7 111	101010	101011	101110	101111	111010	111011	111110	111111

Z-order 编码



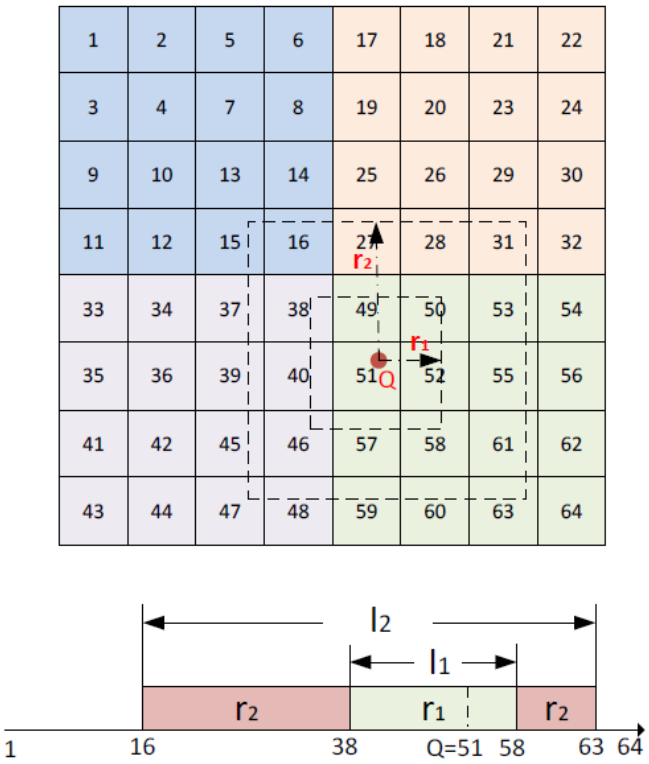
Z-order 编码按从小到大的顺序连接形成的 Z 型线条



三维空间中的 Z-order 编码线条

二维空间的 Z-order 编码具有以下特性（对于高维的情况也类似）：

1. 空间中较邻近的对象 Z-order 编码也是比较邻近的；
2. 任取一个矩形，则左上角为该矩形中的最小值，右下角为该矩形中的最大值；
3. 在编码后的一维空间上任取一段区间，对应原空间上形成一个连续的区域；
4. 在编码后的一维空间上两个区间之间的包含关系，对应原空间上两个对应区域的包含关系。



Z-order 编码的一个例子

有了 Z-order 编码后，我们对坐标进行 Z-order 编码，然后对于原先的倒排列表，再复制一份，每个倒排列表按照 Z-order 编码从小到大排序。这样我们对每个词项建立两个倒排列表（实际编写代码测试时还需要一个按 id 排序的倒排列表，用于随机访问），一个按文本评分排序（下面简称“文本列表”），用于处理文本信息时进行顺序访问；另一个按 Z-order 编码排序（下面简称“空间列表”），用于处理空间信息时进行顺序访问。在空间列表中进行顺序访问时，第一次要二分查找到当前查询在每个倒排表的空间列表中的位置，从这个位置开始往列表两端顺序访问，这样便可以优先访问近的对象，也即空间评分高的对象。

RCA 运行时是在 CA 的基础上改进后进行：每次顺序访问不是访问固定的 h 个后才进行随机访问，而是对评分落在一个固定长度的评分区间以内的所有文档顺序访问后才进行随机访问。这样每个倒排列表每次顺序访问的文档个数都不一样。

概念定义：

我们称若干次顺序访问后进行一次随机访问为一轮。

根据评分函数，文本评分和空间评分值基本都落在(0, 1]以内，总评分也落在(0, 1]以内。设 η_S 为空间列表全部属性值访问完的希望轮数，即 η_S 轮以后空间列表全部属性值都访问完，也即把空间评分值区间划分为 η_S 份。那么每一份的长度便为 $\frac{1}{\eta_s}$。

于是评分区间被划分为

$$T_1 = (1 - \frac{1}{\eta_s}, 1], T_2 = (1 - \frac{2}{\eta_s}, 1 - \frac{1}{\eta_s}], \ldots, T_{\eta_s} = (0, \frac{1}{\eta_s}],$$

故第 i 轮后空间列表中未访问对象的空间评分上界值为 $B_s(i) = 1 - \frac{i}{\eta_s}$。

对于第 i 轮，空间列表中需要访问的对象，有

$\phi_S = 1 - \frac{d(\mathcal{D},Q)}{\gamma} > 1 - \frac{i}{\eta_s}$，即 $d(\mathcal{D},Q) < \frac{i\gamma}{\eta_s}$，于是可令每一轮的半径 $r_i = \frac{i\gamma}{\eta_s}$。

同理，η_t 表示文本列表全部属性值访问完的希望轮数，把文本评分区间分成 η_t 份，每一份长度为 $\frac{1}{\eta_t}$，第 i 轮的评分区间为 $(1 - \frac{i}{\eta_t}, 1 - \frac{i-1}{\eta_t}]$，

故第 i 轮后文本列表中未访问对象的文本评分上界值为 $B_t(i) = 1 - \frac{i}{\eta_t}$。因此，第 i 轮后未访问对象的总评分的上界值为

$B_k = \alpha \cdot m \cdot B_t(i) + (1-\alpha) \cdot B_s(i)$。

η_t 与 η_s 之间满足 $\eta_t = \frac{(1-\alpha)}{\alpha} \cdot \eta_s$。

算法过程（这里在随机访问时和判断终止条件时也不维护上界值）：

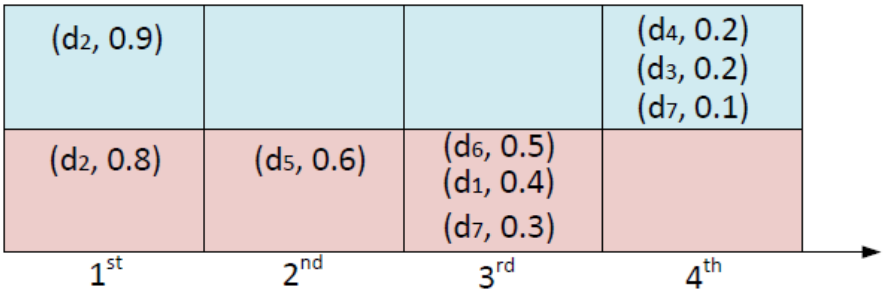
1. 设置 η_t 的值，计算 η_s 的值；
2. 计算查询 Q 的坐标的 Z-order 编码值，然后使用二分查找在每个倒排表的空间列表中找到其所在的位置，并用两个指针 pf 和 pb 指向按 Z-order 编码顺序排序下它的前一个和后一个坐标的 Z-order 编码；
3. 在每一轮中，更新 Bt、Bs、ri，计算半径 ri 以内的矩形区域所覆盖的最小和最大的 Z-order 编码，设为 zmin 和 zmax；
4. 在空间列表中对新扩展的区域内的文档进行顺序访问，即[zmin, pb]和[pf, zmax]以内的文档，然后把 pb 和 pf 分别更新为 zmin 和 zmax。由于 Z-order 并不是严格按照距离远近进行编码，因此在[zmin, pb]和[pf, zmax]区间中有一些文档与 Q 的实际距离大于 ri，这些文档我们用一个数组暂时保存，等到半径增大到包含它们时再对它们进行访问；
5. 在文本列表中对评分大于 $1 - \frac{i}{\eta_t}$ 的文档进行顺序访问；
6. 对 $T_k^{(d)}$ 中所有可行的对象进行随机访问；
7. 更新未访问文档的上界值 B_k，当 $B_k \le M_k^{(d)}$ 时，对已经访问过的且不在 $T_k^{(d)}$ 中的可行的对象进行随机访问，然后算法终止。最终 $T_k^{(d)}$ 中的 k 个对象即是输出结果。

RCA 在每一轮顺序访问时，采用对评分落在一个定长变化的评分下限以内的文档进行顺序访问的方式（下面简称为“评分限方式”），而不是对固定的 h 个文档进行访问。这样做的原因是：

1. 由于 Z-order 的只是保留了近似的空间邻近度，每次扩展的区域形状是不规则的，而且是从中间往两边扩展，若按照 CA 的方式，难以得知当前这一轮距离从近到远排第 h 的文档是哪个；

2. 采用评分限方式，每个倒排列表每次顺序访问的文档个数都不一样，取决于落在本轮评分限以内的文档个数，这个动态变化的数目其实是更合理的，使得每次顺序访问量与随机访问量之间动态平衡；

3. 采用评分限方式，能使在不同的倒排列表中顺序访问时，某个列表中评分高的属性值优先于其他列表中评分低的属性值被访问。



考虑顺序访问如图所示的列表，若按照评分限方式，上面的列表中 d4、d3、d7 在第 4 轮被访问，而下面的列表中 d6、d1、d7 在第 3 轮被访问；若按照固定个数方式，上面的列表中 d4、d3、d7 就可能会在第 2 轮被访问，优先于下面的列表中 d6、d1、d7 被访问，但是上面的列表中 d4、d3、d7 的属性值是低于下面的列表中 d6、d1、d7 的属性值的，这样可能会造成多余的访问。按照评分限方式，若在第 3 轮中已经得到 top-k 的话，就不需要再访问第 4 轮。因此按照评分限方式更合理。

4. 实验测试

我们虚拟了一个现实场景：假设我们想到附近的果蔬店购买自己想要的水果和蔬菜。每个果蔬店只包含某几种水果和蔬菜，某种水果和蔬菜与某店的文本相关性评分表面了该水果和蔬菜的质量。我们希望到尽量近的果蔬店，同时希望买质量尽量高的水果和蔬菜。

测试中所使用的数据集和查询集均为自己编写程序所生成。数据集中包含若干篇文档，每篇文档由一个坐标和一系列水果和蔬菜的单词组成。查询集中每个

查询给出一个坐标和要查询的词项。

实验开始前，需要对数据集中的数据进行预处理：计算每个词项的 tfidf 值并对每个词项建立文本评分倒排列表，分别按文档 id 排序和按评分高低排序在磁盘上保存两个版本。因为我们的随机访问的方式采用二分查找的方式，在其他倒排列表中二分查找当前文档的 id 然后获取评分，这样就需要一个按 id 排序的倒排列表。对于 RCA，还要对每个词项建立一个按 Z-order 值升序排序的倒排列表。

评价指标：

实验评价指标为查询过程的耗时，不包括预处理和将倒排列表从磁盘读入到内存的耗时。预处理相对于大量查询所花的时间是很少的，并且在实验开始前进行，不会影响查询。读入倒排列表相对于查询占用的时间相当少，并且只发生在某个新出现的词项第一次遇到时，几乎可忽略不计。

算法实现与优化简要说明：

1. Naïve 算法即为文章开始提到的简单算法，但是它也进行过优化，它是在倒排列表上进行逐个扫描，而不是在原始数据集上扫描。

2. 倒排列表按需读入，即当某个词项首次出现时才把对应的倒排列表读入到内存中，并保存在内存中，不会重复读入，从而优化读入时间。

3. 因为查询词项个数很少，因此对每个对象只需要用一个 int 型整数来记录它的某个属性是否被访问过，而不需要数组记录，若访问过则对应的二进制位标记为 1，否则为 0。这样在判断时也是通过位运算判断，不需要扫描数组，从而提升了时间和空间效率。

4. 对于 $T_k^{(d)}$（前 k 大的下界值），使用大小为 k 的小根堆进行维护，若某个元素大于堆顶元素，则删除堆顶元素，新元素入堆。同时，我们的堆还实现了修改堆中某个元素的功能，也是 O(logk)的，这样便于更新堆中对象的下界值，不需要重新建堆，大大节省时间。

5. 本问题的评分函数为

$$\phi(\mathcal{D},Q)=\alpha\cdot\phi_t(\mathcal{D},Q)+(1-\alpha)\cdot\phi_s(\mathcal{D},Q)$$

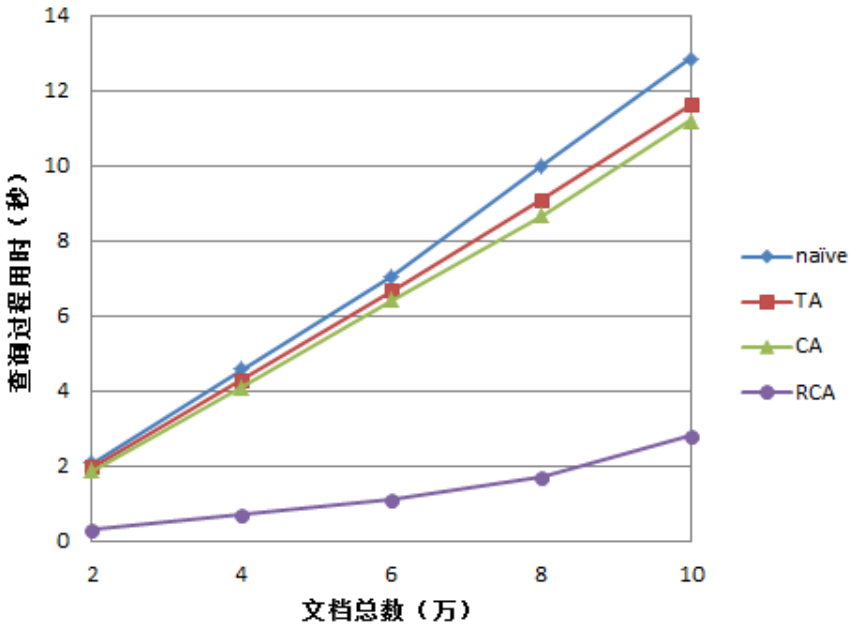
考虑一种情况：某个文档距离查询十分接近，但是不包含查询中的任何一个词项。但它的评分可能还高于那些虽然包含了查询中的词项，但是距离查询相当远的文档。这样从实际角度出发，明显不合理：我们没有理由去一个很近的果蔬店，但是里面却不包含任何一种我们想要的水果或蔬菜。因此，当文本总评分为 0 时，空间评分要特判为 0。这个特判必须在文本总评分得知后才能判断。那么在 CA 中，终止条件需要进行一些改进以适应本问题，保证最终 $T_k^{(d)}$ 中所有元素被随机访问后不会有特判为 0 分的对象才可以终止，这里有较多改进之处，包括随机访问过程中的特判，具体可见代码。

实验测试环境与参数取值：

测试环境：intel 处理器，2.60GHz，8G 内存。

参数取值为文档总数 n 从 20000 到 100000，每个查询词项个数 m=3，k=20，评分函数的 α =0.4，CA 中 h=8，RCA 中 η_t =20。另外，查询个数增大只会让时间成比例增大，不会对各算法时间增长的增长级别有影响，增长级别与 n 有关，为了能让 n 尽量大，查询集的查询个数只能设小一些，这里取 $total_Q$=50。

实验结果：



各算法查询过程用时随文档总数增加而变化的情况

n	naïve	TA	CA	RCA	NRA
20000	2.107s	2.007s	1.902s	0.337s	>1min
40000	4.589s	4.33s	4.12s	0.72s	>1min
60000	7.081s	6.705s	6.464s	1.133s	>1min
80000	10.022s	9.132s	8.714s	1.714s	>1min
100000	12.885s	11.65s	11.235s	2.836s	>1min

统计列表

我们可以看到，随着 n 的增大，naïve 效率是最低的，并且 n 如果继续增大的话，它的效率会更低，这是它访问了所有文档对象的所有属性所导致的。

TA 和 CA 用时虽比 naïve 少，但是还是不太乐观。这是因为在空间关键字查询中，它们必须每次读入查询的坐标后都要计算每个文档与查询之间的距离评分，然后对这个距离评分排序。当 n 很大时，这个排序会造成很大开销。

CA 的用时比 TA 要稍少一些，因为 TA 进行了太多的随机访问，CA 在随机访问和顺序访问之间取了一个折中。

NRA 的效率十分低下，n=20000 时也无法在 1 分钟以内出结果（实际测试要 5、6 分钟），这是因为它不进行任何随机访问所致，并且维护上界值开销太大，因此比 naïve 都要慢得多。但是它在某些特殊的系统中以及特定的聚合函数下有应用之处。

相比之下，RCA 的效率十分乐观，与其他算法差距明显。虽然它需要多维护一个空间列表，使用更多的时间进行预处理和更多的磁盘空间，但是因为查询词项数 m 很小，维护的倒排列表个数不会很多。并且预处理在实验开始前进行，不会影响查询。RCA 无需每次对距离评分排序，并且顺序访问时评分限方式比 CA 的固定个数方式要优秀得多。

5. 附录

实验代码、数据生成代码均放在“code”目录中，并且已经生成了一个 n=30000 的测试数据，并已经建立好了倒排列表。程序编译运行完后输出的结果显示在输出文件 answer_xxx.txt 中，“xxx”为对应的算法名称。

每个算法的输出文件中，对于每个查询，输出结果的第一行表示是第几个查询，后面 k 行是 top-k 的查询结果，包含文档的 id 和总评分，按照总评分从大到小排序输出。最后一行是该算法的查询过程总共用时。