# 使用锥形树的最大内积搜索算法

***ABSTRACT***

使用欧几里得距离或者余弦相似性来有效的找到最佳匹配的方法已经被广泛的研究了。但是，就我们所知，关于内积的最佳匹配算法还没有被讨论过，即使这个问题与前者密切相关。在这篇论文中，我们将探讨最大内积检索相关的问题，并且将其与此前的一些已经有结论的问题做一些对比。首先，我们将会提到一种基于单一树数据结构的分支界限算法。随后，我们会展示一种基于对偶树的在多重查询下发挥作用的算法。这些分支界限算法都基于一种新的内积界限。最后我们将提到一种新的数据结构，锥形树，来作为对偶树算法的效率优化。我们将会使用来自不同应用程序的数据集来测试我们的算法，并且向读者们展示最多五个层级的优化方案，使其在查询时间方面有提升。

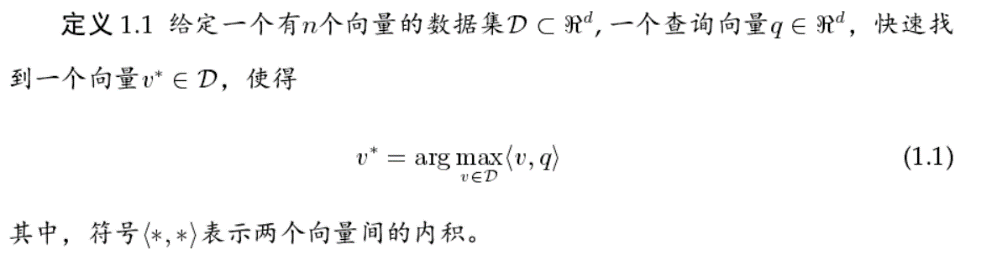
关键词：

度量树、锥形树、对偶树的分支定界

##### 介绍

在这篇论文中，我们考虑给定点集的关于内积相似度的查询的最佳匹配的效率问题。我们专注于查询的效率。一般的，我们考虑如下的问题：

最大内积搜索：



起初这个问题看起来与现有的一些问题非常相似。关于欧几里得距离的高效最佳匹配问题已经是度量空间中的最近邻居搜索这一领域中被研究的非常多的课题了。而基于cosine相似度函数的高效检索最佳匹配问题也已经在文本挖掘和信息检索领域得到了足够的讨论。但就如我们将会在下一节解释的一样，最大内积检索问题不仅与上述问题有许多的区别，并且可以说是更加难以解决的。

###### 1.1 应用

最大内积检索有一个显而易见的应用，那就是最为人熟知的推荐系统中的矩阵分解框架，其起源于著名的“Netflix推荐系统竞赛”。矩阵分解可得到按照用户向量与物品向量（例如电影或音乐）等数据产生的关于用户的偏好画像。在这个语境下，用户对于某个商品的偏好可以被描述为对应的用户向量与商品向量之间的内积，而对用户偏好的检索可以等价于一个以用户为查询并以商品为参考集的最大内积的检索。为找到最佳的推荐方案，对商品的线性搜索是最常用的方法。一个高效的搜索算法通常具有能使矩阵分解框架下的检索拓展到更大的系统的特性。

通常的文件检索任务使用cosine相似度函数来匹配文件，但是，在一些特定的情况下，文件是以向量表示的，而这些文件与其他文件之间的相似度又是由相应的内积表示的。在这种情况下，除非这些向量经过了正规化（使其具有相同的长度），使用cosine相似函数的文件匹配函数的拓展形式可能会导致不准确的解，因为内积表示的相似度与cosine相似度并不一致。（我们将会在第二节讨论这个问题。）

为了引出接下来的问题，我们先介绍一个与最大内积问题相似的问题——最大内核操作。这个问题可以被如下描述：对给定点集S，查询q与一个kernel函数K（. , .），要求找到某个点p∈S，使其满足K（q，p）在S的域上达到最大值。这个问题在机器学习的最大后验推论领域和计算机视觉的图像匹配领域内得到了广泛的应用。假设存在K（q，p） = ，如果kernel函数可以以的形式被显式的表示出来，这个问题就被化归到最大内积搜索问题，因为S中的所有点与q都被变换到了空间。

###### 1.2 这篇论文

在这篇论文中，我们引入两种基于树的分支限界算法与相应的新数据结构来解决最大内积检索问题。在第二节，我们将这个问题与其他的一些你可能更为熟悉的问题，例如度量空间中的最近邻问题与基于cosine相似度函数的最佳匹配问题等作一些比较。在第三节，我们会使用简单的分支限界算法和ball-tree，并引入一种新的界。而在第四节，我们将会处理一种新的情况，在同一数据集上有复数的查询，并引入一种关于对偶树的分支限界算法。在第五节，我们会介绍一种新的数据结构，锥形树，用以为对偶树算法编写索引。这些数据结构受益于相对于传统ball-tree更新，更紧的内积界限。在第六节，我们将会使用不同的数据集来测试以上得到算法。第七节中展示了这些算法是如何在不需要显式的表示空间中的点的情况下被应用于最大内核操作问题的。而在最后一节，我们将会做出最后的总结并且展望这个问题的前景。

##### 最大内积搜索

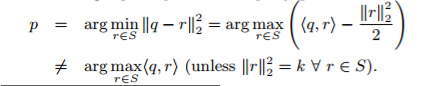
在欧几里得度量空间中有许多现有的最近邻搜索技术，在大尺度下的最佳匹配算法也有cosine相似度测度算法可以选用。最近邻搜索问题已经被现在非常常用的局部敏感性哈希算法（LSH）近似的解决了。就像cosine相似度算法一样，LSH算法也被拓展以适应其他类似的问题。最大核操作问题也能够被LSH在核方程的背景下有效率的解决。降维法已经对偶树算法也被用来高效的解决最大核操作问题。

###### 最大内积搜索是如何不同于现存的问题的？

在这一节我们会解释最大内积检索问题与其他现有的问题有哪些不同。最大的一个原因是因为可以直接应用在其他问题上的算法（例如LSH），是不能够直接应用在最大内积检索问题上的。

我们首先来看在欧几里得空间下的最近邻搜索，这个问题可以被如下描述。

现求找到一个对于查询q的点p∈S，且q满足



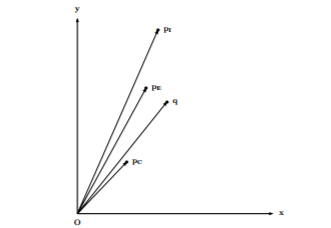


图1：

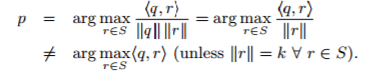
最佳匹配：对于给定的查询q，pC，pE

pI表示分别对于余弦相似度，欧几里得距离

和内积的最佳匹配候选者。

因此，如果S中的所有点的被标准化具有相同的长度，然后最大内积搜索相当于欧几里德度量中的最近邻搜索空间。 但是，如果没有这个限制，这两个问题可能有非常不同的答案（图2）。

使用cosine相似度函数的最佳匹配：这需要为查询q寻找一个点p∈S，且p满足



余弦相似度算法只有当集合S中的所有点都被归一化到相同的长度时才产生内积的最大值。（图2中的示例）。

局部敏感散列： LSH已被应用于各种相似度功能。 LSH涉及构造散列函数使得每个散列函数h满足以下对于任何一对点r，p∈S：其中，sim（r，p）∈[0,1]是感兴趣的相似度函数。对于我们的情况，我们可以

正规化我们的数据集，并假设所有数据都在第一象限（所以不存在负数内积）。 在这种情况下，是有效的相似度函数。

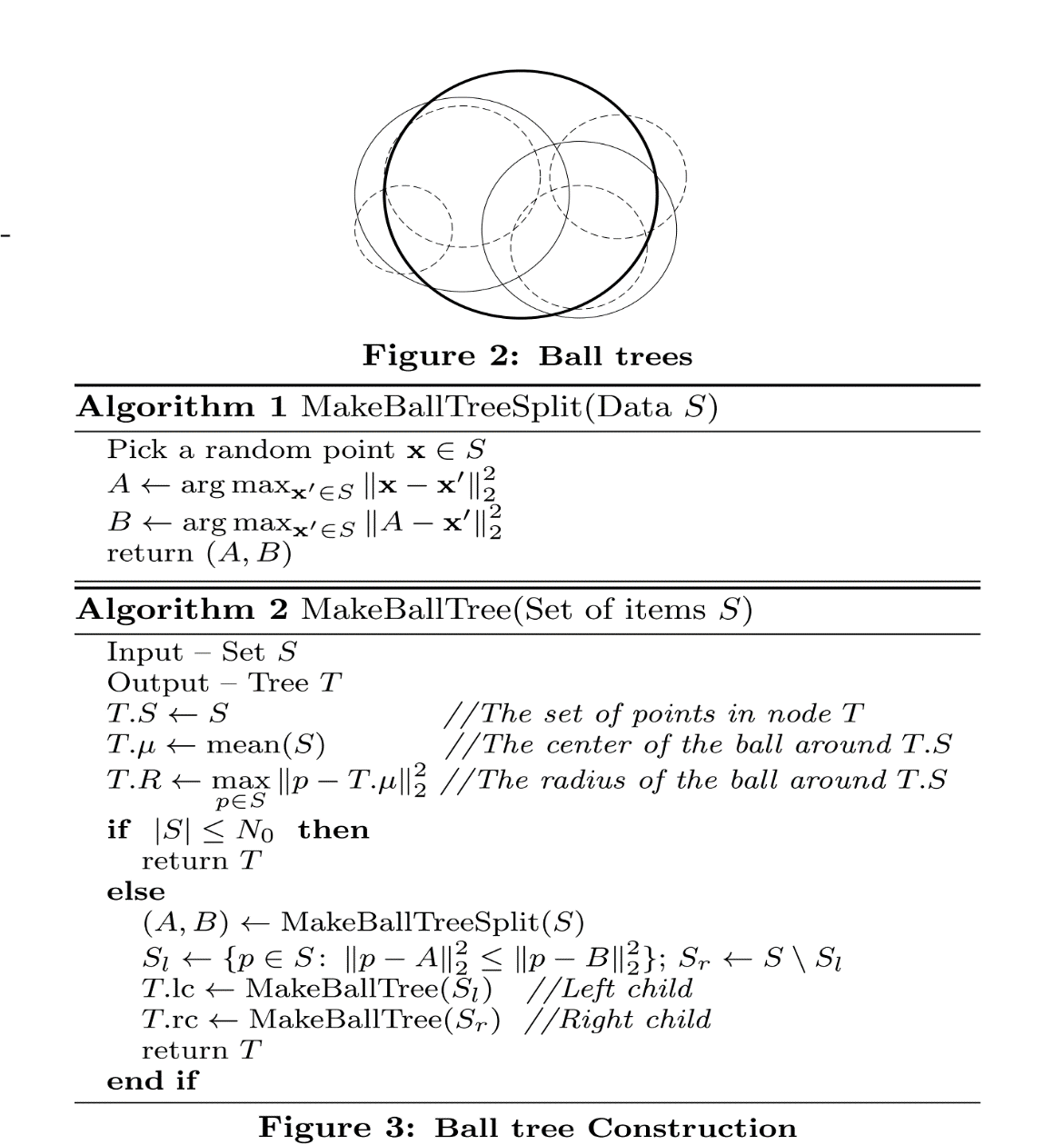
众所周知，对于任何相似性函数来定义的局部敏感哈希函数族（如方程式2定义），其距离函数d（r，p）= 1 - sim（r，p）必须满足三角不等式（[7]中的引理1）。

然而，其距离函数d（r，p）= 1-<r，p>却并不满足三角不等式（即使所有的点都被限制到第一象限）。因此LSH不能在基于内积的相似度函数中发挥作用，即使所有的数据点都在第一象限（即使这样的条件已经非常严格了）。

高效的最大内核操作：为有效地解决这个问题，已经有各种技术被提出了。对于高维的（甚至是无限维度的）核函数的显式描述，Rahimi等人在2007年提出了一种能将这些高维表达式降维的同时，仍然近似的保存了内积以提高拓展性。

###### 2 为什么最大内积搜索可能会更加困难？

内积缺少一个非常基础并普遍使用的相似度功能------巧合。例如，一个点到他自己的欧几里得距离为0；点自身的余弦相似度为1.对于一个点x∈S到它自己的内积为||x||2，可能或高或低的依赖||x||的值。另外，可能有其它如y∈S且(y, x) > ||x||2.

 有效的最近邻搜索方法通常严重依赖于这些性质（三角不等性和巧合）来实现他们的高效性。因此，没有任何额外的假设，最大内积搜索的问题本身就比先前处理过的类似问题更难。这可能就是为什么到现在我们的知识还没有解决这种问题的一般形式的原因 了。

###### 2.3 为什么尝试使用树？

树已经被广泛地使用于最近邻搜索[13, 4, 29, 8, 32]。在被广泛使用的最近邻搜索的案例中，我们认为在考虑最大内积搜索案例前审查他们是有启发性的。

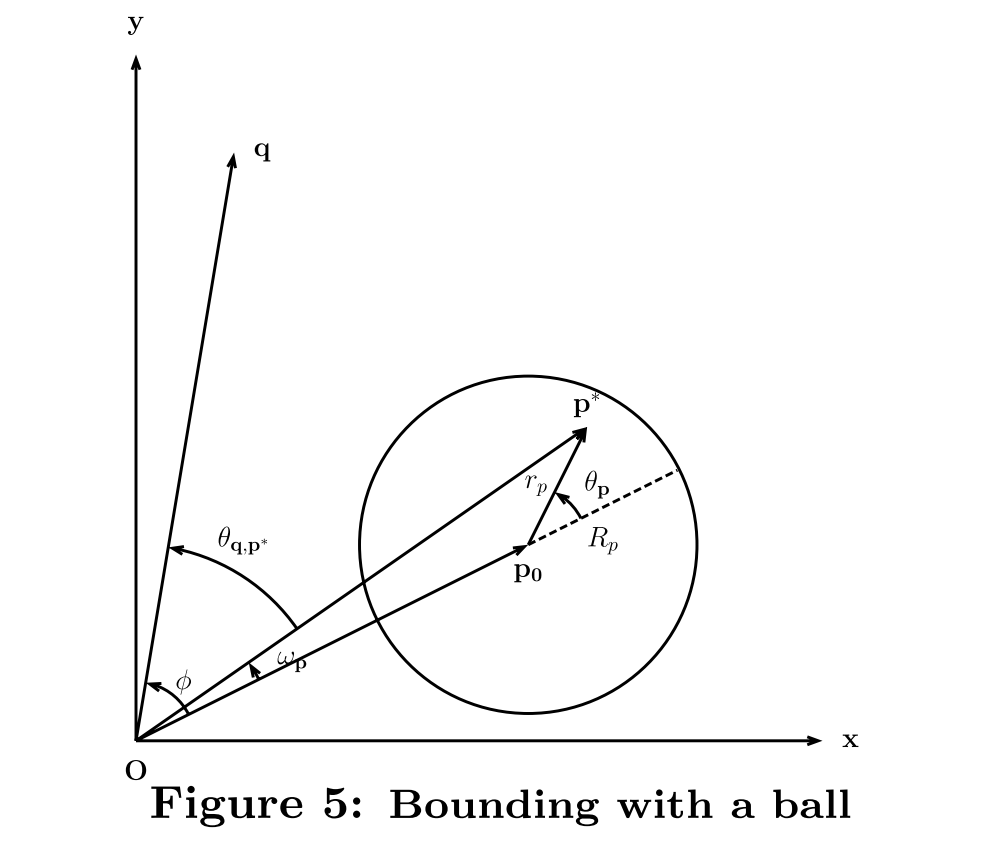
对于精确的最近邻搜索，只要存在较低的固有维树，树可以在任何地方的从低到高的的数据产生很大的加速度[4, 10]。树可以轻松的适应相似的情况，并能够保证各种错误率。这些包括等级上的相似，例如，如果实际的最佳匹配可能不会返回，树可以以一种保证结果在十大最佳匹配中的方式使用[32] –而不是提供可能不太有意义的抽象数量如距离（LSH提供；还不太清楚如何扩展LSH来提供等级保证）的保证；这在许多应用中似乎是非常有意义的，如建议。树也可以用于另一种近似搜索设置，这在实际应用中可能很重要，其中在给定用户定义的时间限制的情况下找到最佳匹配。对于基于树的分支和边界算法，这种近似是可能的，因为它们是增量算法。这是不可能的，像LSH - LSH提供理论误差界限，但是没有办法确保搜索期间的错误约束。另一个树的重要优点是树需要一个单一的结构--分支绑定算法适应不同级别的近似和/或时间限制。哈希

技术需要对不同级别的近似值进行多次散列。通常的规范是对多个近似值进行预散列。也可以通过使用机器学习[6, 25]技术从数据中学习来提供更好的准确性和效率来构建树。

最邻近设置的树型方法的显着优点激发了他们是否可以被 带到最大内积案例的问题。

##### 树型查找

球树[29, 28]是二进制空间分区树，已被广泛用于索引数据集的任务。树中的每个节点都表示一组点，每个节点都以一个中心和一个围绕节点中所有点的球进行索引。节点处的点集被分成两个不相交的集合，形成子节点，将该空间分成（可能重叠的）超球体。树是分层构建的，并且如果节点包含大小低于阈值N0的点的集合，则该节点被制成叶子。



###### 3. 1 **树的结构**

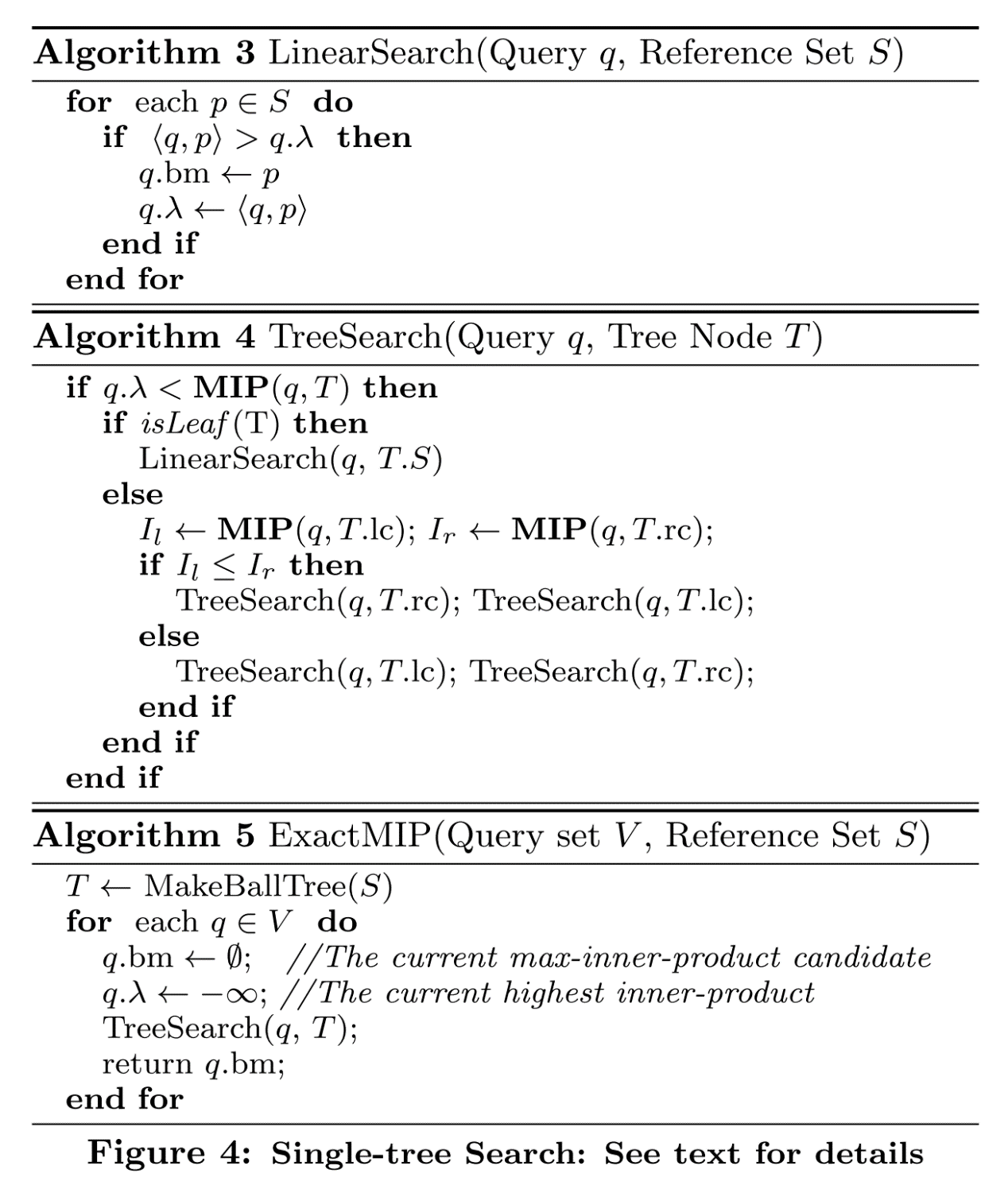
我们使用简单的球树构造启发式，大致选择一对彼此[28]最远的枢轴点，并通过将点分配给最近的枢轴来分割数据。在这个启发式背后可直观的看出这两点可能在主要方向。 分解和递归树构造算法在算法1和2中提供以获得完整性。

###### 3.2 分支绑定算法

球树被广泛地用于最近邻搜索的任务，并且已知它们具有相当可扩展性以适度的高维度[28, 26]。搜索通常采用深度优先的分支绑定算法 - 通过首先沿着靠近查询的节点向下移动树，以深度

先的方式遍历树来回答查询，并将最小可能距离限制到另一个分支，利用三角不平等。如果此绑定大于到查询的当前邻居候选者的距离，则从计算中删除该分支。

类似深度优先的贪心算法可以用于最大内积搜索。但是，不是沿着靠近查询的节点向下遍历，而是基于查询和节点的任何潜在点之间的最大可能内积进行选择。递归的深度优先的分支绑定算法在算法4中给出。查询（q）的搜索算法从树的根开始（算法5）。在每个步骤中，算法处于树节点（T），它检查查询和节点中的任何点之间的最大可能内积MIP（q，T）是否比查询（q.bm）的当前最佳匹配更好。如果检查失败，则该树的该分支不再被搜索。否则，算法递归地遍历树，以深度优先的方式查询具有更好的潜在候选的分支。如果节点是叶子，则该算法通过线性搜索找到叶内最佳匹配（算法3）。该算法确保算法结束时返回精确解（即最大内积）。

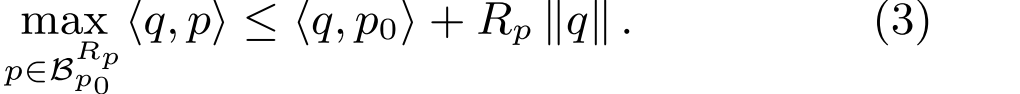


###### 3.2.1 用球绑定最大内积

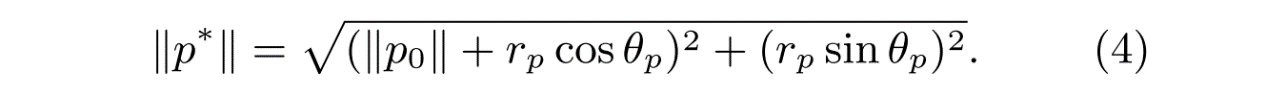
我们给出一个新的分析上界，用于给定点（在这种情况下，查询q）与球中的点的最大可能内积。重要的是要注意

关于球的信息仅限于其中心和半径。在剩下的章节中，我们使用符号||.||来表示||.||2。

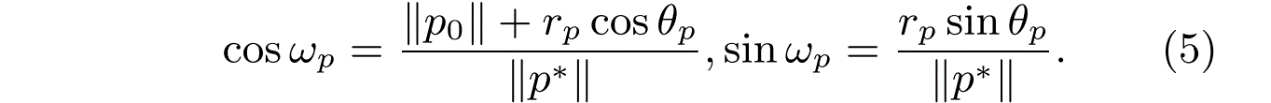
定理 3.1 给定一个球BRpP0 ,球心为点p0,半径为Rp，和（查询）点q，可能的最大内积在点q和球BRpP0之间被以下公式绑定：



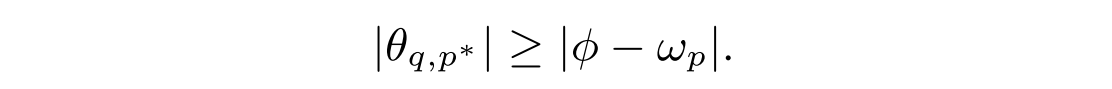
证明： 假设p\*在球BRpP0的最佳匹配的查询q和rp是球中心p0和p\*之间的欧几里得距离（通过定义rp <= Rp）。令θp为向量p0和向量p0p∗的夹角，φ 和ωp分别为原点与向量p0和向量q和p\*之间的夹角（见图5）。根据p0和θp，p\*的长度为：



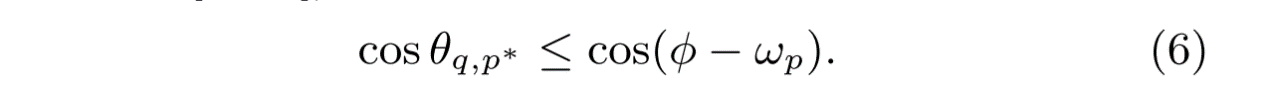
角ωp可以根据p0和θp表示为：



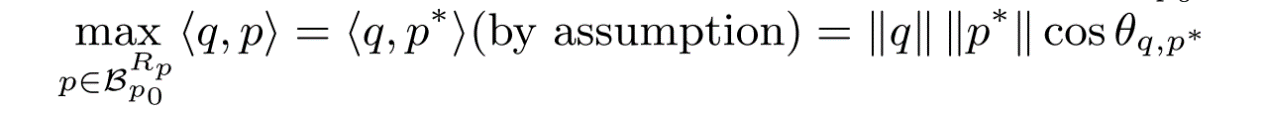
令Let θq,p∗为向量q和p\*的夹角。利用角的三角不等关系，我们有：



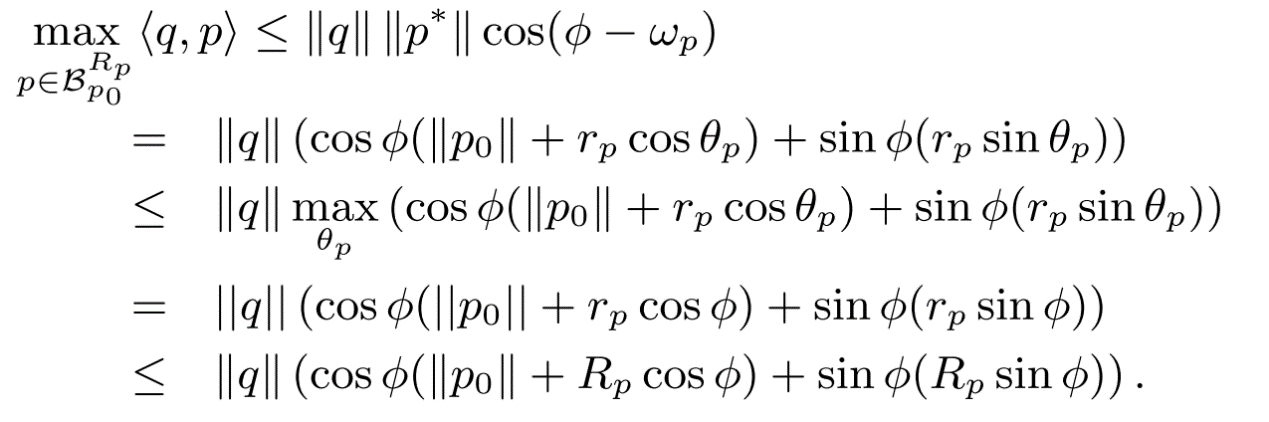
假定这个角度落在区间[-π，π]（而不是通常的[0, 2π],我们得到：



使用这些不等关系我们获得了以下q和任意p∈BRpP0的约束条件：

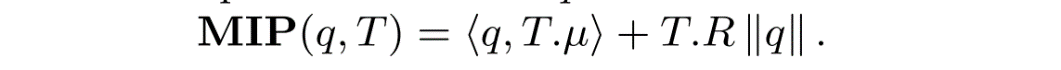


通过等式4，5和6，我们有：



第三个不等关系来自最大值的定义。以下的等式来自最大化θp。这给我们θp = φ的优化值。最后的不等式来自于事实rp ≤ Rp。化简最后一个不等式我们得到等式3.

对于树搜索算法（算法4），我们设置q和一个树节点T之间可能的最大内积为



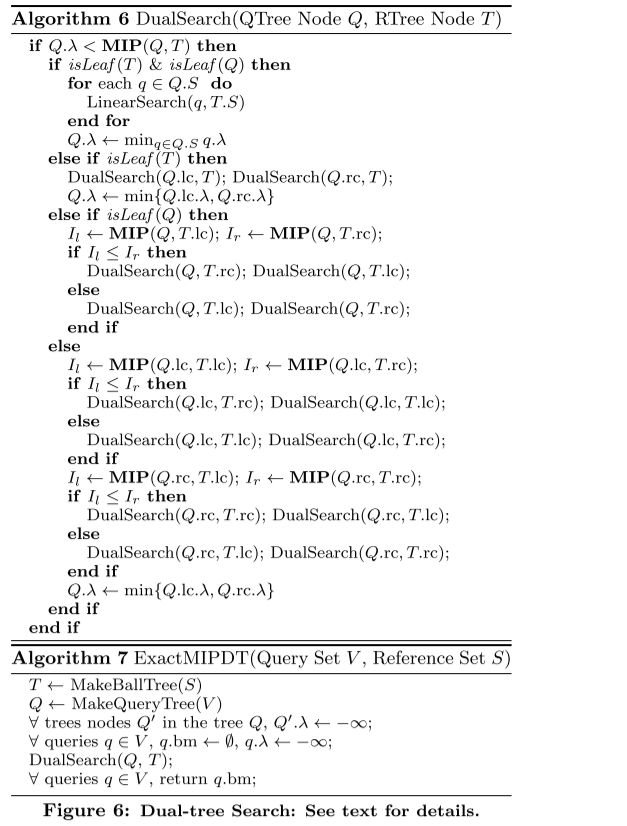
单一的内积上限可以在几乎相同的时间需求内被计算出来（由于查询的规范使得在搜索树之前可以被预先计算出来）。

##### 基于双树的搜索

对于一组查询，可以为每一个查询分别遍历树。然而，当这组查询的数目很大时，我们需要使用一个常用的技术——以树的形式为每个查询创建索引，来提高查询的效率。此时，搜索可以使用“双树算法”同时遍历两棵树。“双树算法”的基本思想是分摊一组相似的查询的“树遍历”的代价。“双树算法”可以应用于不同的基于树的算法，例如最近领域搜索、具有理论运行时间保证的核密度估计等。

###### 4.1 双树分支定界算法

通用的“双树算法”的说明在Alg.6中。和Alg.4中的算法相似，这个算法往下遍历参考集S（Rtree）上的树。然而，这个算法也往下遍历查询集V（QTree）上的树，导致四路递归。算法的每一步中，都会存在一个在QTree上的节点Q和一个在RTree上的节点T。对于每一个Q，值Q. λ代表Q中的任何查询与其当前最佳匹配候选之间的最小內积。如果这个值比“MIP(Q,T)”（即Q中的任何查询与T中的任何参考点之间的最大可能內积）还要大，则这部分的递归不再进行。当这个算法在两棵树的叶子层，则它可以通过对RTree的叶子进行线性扫描，来获得QTree叶子中的每个查询的最佳匹配。



我们现在来探索创建查询索引的两种方式——（1）通过球树为查询创建索引（在Alg.7中的MakeQueryTree是指Alg.2中的算法）；（2）通过一种新颖的数据结构“锥树”为查询创建索引（在Alg.7中的MakeQueryTree是指Alg.9中的算法）。在下面的子章节中，我们将会得到球树的MIP(Q,T)的表达式。而锥树的MIP(Q,T)的表达式则会在第五节说明。

###### 4.2 使用球树

在这个子章节中，我们将会通过以下定理提供两个球之间的內积界限。

***定理4.1*** 给定两个球和，圆心分别为和,半径分别为和，则任意一对点p∈和q∈的最大可能內积的界限由下面的公式决定：

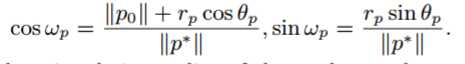
QQ截图20170425230405

*证明：* 假设有一对点（，），∈，∈，且

QQ截图20170425230430

设是和向量的夹角，是查询球中对应的夹角。设是向量和的夹角，是向量和的夹角。设是和的距离，是和的距离。设是和关于原点的夹角。

对于球存在一些事实（对于球也一样）：

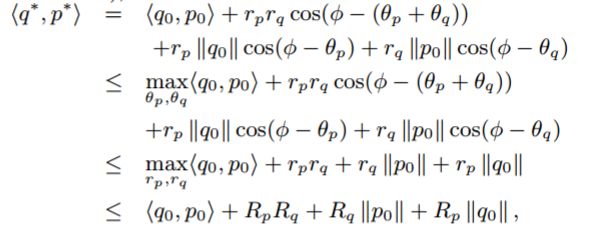
QQ截图20170425230305 

使用关于这些角的三角不等式，我们可以知道：

QQ截图20170425230628

从而可以推出下列公式：

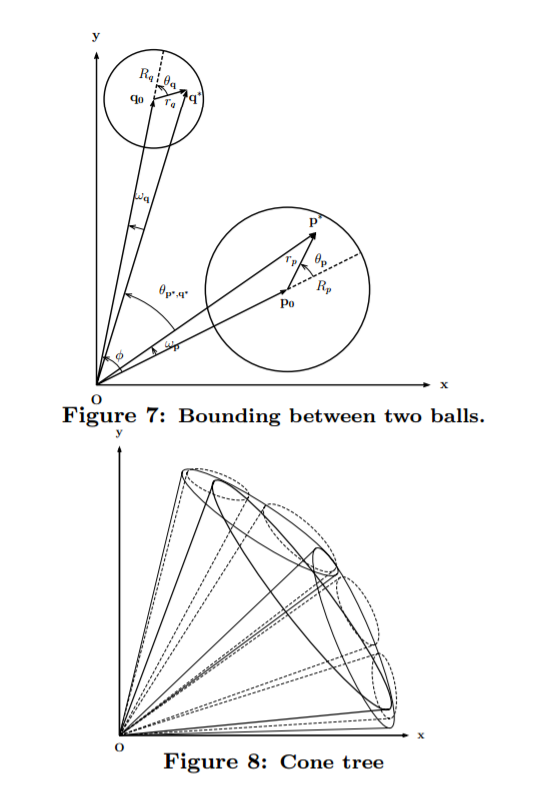
QQ截图20170425230716

用上面描述的相等性将和代替和（与证明定理3.1类似），我们可以得出：  


在这里，第一个不等式可以由最大的定义得出。第二个不等式可以由QQ截图20170425231521得出，最后一个不等式可以由QQ截图20170425231608得出。对于双树搜索算法（Alg.6），两个树节点Q和T的最大可能內积是：

QQ截图20170425231801

一个有趣的现象是，当球包含的查询缩减到一个点，即，这个上界会减少到定理3.1中的界限。



##### 锥树

在方程1中，最大值所在的点p，是和查询q的范||q||相互独立的。设为q和r关于原点的夹角，则最大內积搜索的任务等价于找到一个点p∈S满足以下条件：

QQ截图20170425233057

这条公式表明只有查询的方向可以影响结果。球提供了內积的界限，因为球限定了向量的范以及方向。因为向量的范对查询没有影响，在球中为向量的范创建索引没有意义（因此对向量的范设定界限是没有必要的）。只有向量方向的范围需要设定界限。正是由于这个原因，我们提议基于向量的方向（从原点出发）来为查询创建索引，组成一棵锥树（图8）。查询分层次地按照“开放锥”来创建索引（可能会重叠）。每一个锥代表一个向量（对应于轴）和一个角度（对应于截面）。

###### 5.1 锥树的构造

锥树的构造和球树的构造非常相似。唯一的不同就是在分裂任务上锥树使用的是余弦相似度，而不是欧几里得距离（图8的伪代码）

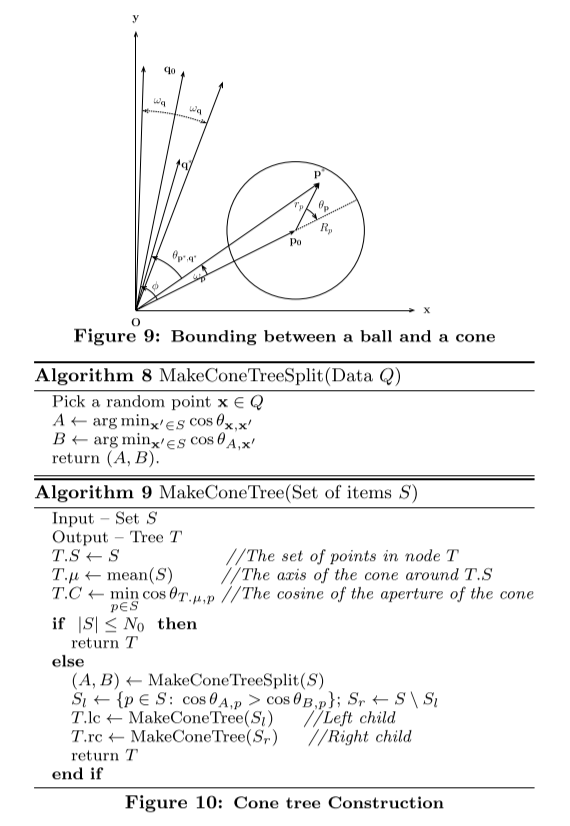
###### 5.2 锥球界限

因为在方程10中查询的范不影响结果，我们假设所有的查询的范都为单元范。

***定理5.1*** 给定一个球，圆心为，半径是，和一个查询的锥体（归一化为长度1），锥体的轴为，锥体的截面是，则任何一对点和的最大可能內积的界限由下面的公式决定：

QQ截图20170426134547

在这里，指的是和关于原点的夹角，函数。



*证明：* 这里要考虑两种情况：

QQ截图20170426135521

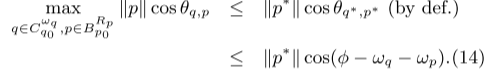
对于情况，球的圆心在锥体中，可以得出：

QQ截图20170426135803

因为在这种情况下一定存在一些查询和在同一个方向，由此可以得出最大可能內积。

对于情况，我们假设（不失一般性），则。我们继续使用定理3.1&4.1中的符号，也是图9中的符号来表示最佳的一对点，则可以得出：

QQ截图20170426140804

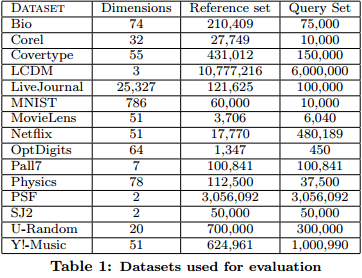
因为是固定的，我们可以得出：  
 

用，和来表示和，然后将最大化。根据

，可以得出：

QQ截图20170426141804

结合情况 和 ，我们可以得到方程（11）。



##### 实验和结果

我们来评估一下算法5（SB - 球形树）和算法7的效率。对于二元树，使用两种算法的变种：

像双球形对偶树一样查找索引的集合

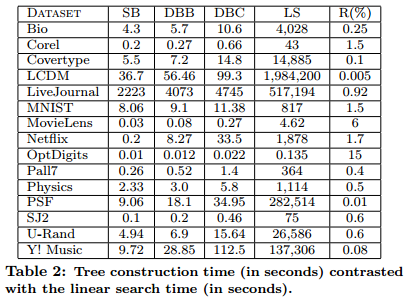
像球- 锥形对偶树算法一样查找索引的集合。

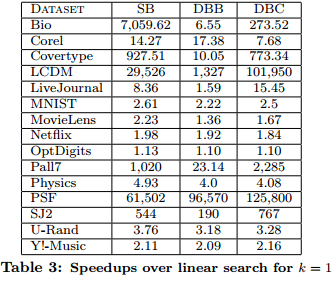
拿我们所说的这个算法和算法3（线性查找）的线性查找来比较。我们得到其他算法对线性算法的加速比。 对于树，用交叉验证选择叶子数目N0。 然而，对于我们的实验来说，我们不会

选择用昂贵代价的交叉验证，而是给所有的数据集选择一个平均值 N0 = 20，来证明效率提高。

数据集合。 我们使用数据挖掘出来的不同领域的数据集。例如以下协同过滤出来的数据集： MovieLens、Netflix、和Yahoo Music数据集。对于文本数据，使用LiveJournal的博客中的“情绪“数据集。 我们还使用MNIST的位数数据集来评估。 以及三个天文学数据集，LCDM，RSF, SJ2，也在选择之中。还有，一个含有20维的不同随机点的综合数据集也被使用了。UCI机器学习仓库余下的数据集广泛应用于机器学习。关于数据集的大小的详细信息，在表1中呈现。

建树的时间。建树的过程非常高效。表6表现了不同算法的建树时间，我们将他和线性查找算法的运行时间对比一下。 在最后一列，我们列出了建树时间相对于运行时间（算法3）的比例。对于球形算法和对偶树（双球形）算法，建树过程要分别建立一个或两个球形树。对于球-锥形对偶树算法，为了方便，查询必须以单位长度规格来进行。 随着查询的规格化，两棵树就建成了。 查询规格化所需的时间也包含在建树的总时间中，这很大程度上导致了对偶树（双球形）算法和对偶树（球- 锥形）算法建树时间的不同。





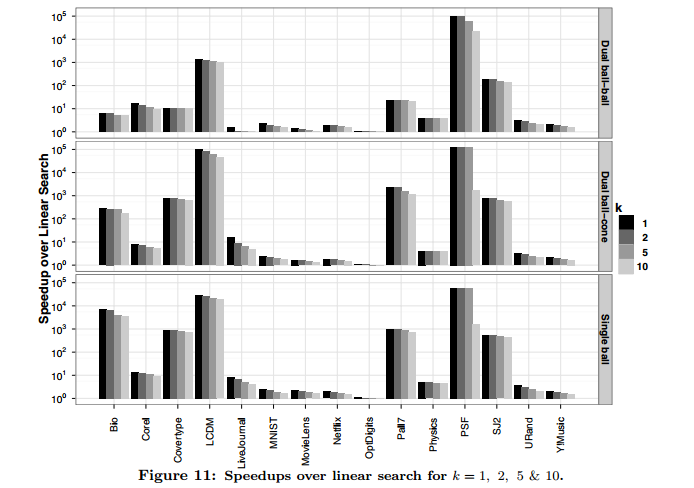


表6最后一列的数字显示，与线性搜索相比，建树时间是多么短呀！就OptDigits数据集来说，最高的比率只有0.15。这表明，只要查询有高于1.18加速比，就可弥补多出来的建树时间。对于大多数的数据集，这个比率远比这个（0.15）要小。而且，每次只需建一次树，就可以实现多次搜索数据集。

搜索效率。相对于线性搜索的加速比已经在表6中。 总的来说，加速比从最低的1.13（OptDigits数据集），到最高达到10^5（4个数量级，LCDM和PSF数据集）。这里要提醒的重要一点是，对于使用球形树算法且低加速比的数据集（低于一个数量级），这三个算法的加速比都十分相似，相当低。然而， 即使加速比为2，也能显著比绝对时间更快、更好。比如，Yahoo！Music 数据集，建树时间为120秒，加速比为2的搜索算法，

节省了19小时的计算时间！ 大多数有着高加速比的，使用球形树算法的数据集，他们使用对偶树算法时的加速比也非常高。

这里要提醒三个重要的点。 首先，若单形树算法的加速比不高，则对偶树算法（算法7）的执行也不能让人满意。这主要是因为，这棵树无法找到严格的边界值，因此必须遍历每个分支。 对偶树通过模糊边界来平衡多次查询时的遍历时间。但是，如果边界不适用于算法5，那么对偶树算法的边界将更加糟糕。因此，对偶树算法并没有呈现出任何显著提高的加速比。 第二，当检索规模很大的时候，对偶树算法（特别是对偶树）显然比单形树算法好得多。检索规模必须足够大，参考树因为平衡检查遍历提高的性能，才能超过因为遍历搜

索树的计算带来的开销。 最后，球形树算法检索通常情况下明显比有一个锥形树的对偶树算法慢。 这里有以下两个可能原因：

锥形树比球形树为检索提供了更加紧凑的索引。

一个单锥可以索引许多位于同个方向但有不同规格的球上。

方程10中MIP（Q，T）的上界相当发散。

我们还考虑了获取集合S中查询q的第k阶高的内积的点时，通常会出现的问题。这和查找第k个最近的邻居问题相似。 我们在图11中展示了我们的算法在k=1,2,5和10的时候，相对于线性搜索的加速比。

##### 具有一般核函数的最大核操作

在本节中，我们提供了一些讨论，关于如何在内部产品空间中应用所提出的算法，而没有在内部产品空间中明确表示点。内积由核函数定义 K(q, p) = ⟨φ(q), φ(p)⟩。

树形结构必须被修改为在内部产品空间中工作。对于具有一组点T.S 的树节点T,，φ空间中的平均值定义为/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 5.56.16 PM.png

然而，μ可能不具有明确的表示，但是可以使用μ计算内积，如下所示：

/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 5.58.23 PM.png

然而，这个计算在搜索期间可能是非常昂贵的。因此，我们提出在φ-空间中选择最接近于平均值μ的点作为新的中心。所以新的中心pc是由以下给出的：

/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 5.59.06 PM.png

该操作在计算时间是二次方，但是在预处理阶段完成，以在搜索阶段提供效率。给定这个新的中心PC，在φ空间中包围集合T.S的球的半径Rp由下式给出：

/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.00.22 PM.png

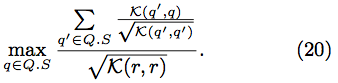
有了中心和半径的这些定义，一个球使用算法2可以在任何φ空间中构建树φ空间中的球，定理3.1中的等式3变为：

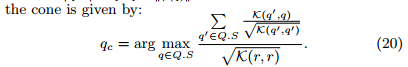
/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.01.36 PM.png

计算此上限相当于单个内核函数评估（K（q，q）在搜索树之前预先计算）。使用这个上限，可以在任何φ空间中执行树搜索算法（Alg.5）。我们将在本文的较长版本中评估此方法。

使用相同的原理，双树算法（Alg.7）也可以应用于任何φ空间。对于双树用查询的球树， 节点Q里的查询和节点T中的点之间的最大内积的上限（方程式7） 中的点被修改为：/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.02.05 PM.png

其中pc和qc分别是具有半径Rp和Rq的φ空间中选择的球中心。

/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.04.03 PM.png

对于在锥形树中索引的查询，中心轴锥体可以是φ空间中的点最小与φ空间中的集合的平均值成角度。自从查询应该在φ空间中归一化，对于a查询树节点Q，设置Q.S的平均值应该是 /Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.02.42 PM.png 。、、所以新的中心轴qc

同样，该计算在数据集的大小上是二次方，但是在搜索时间内提供了效率。圆锥孔的一半的余弦现在由下式给出：

/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.04.34 PM.png

定理5.1中的锥形树节点Q的上界参考点的查询和球树节点T变为：

/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.05.29 PM.png

其中φ定义为：

/Users/yuanyuanzhang/Desktop/Screen Shot 2017-04-26 at 6.05.11 PM.png

这个边界可以被非常有效的计算出，因为它只需要单个内核函数评估（K（pc，pc）和K（qc，qc）可以预先计算并存储在树中）。

##### 结论

我们考虑最大内积搜索的普遍问题，并提出三种有效解决这个问题的新方法。我们使用树数据结构，并提出了一种用于最大内积搜索的分支约束算法。我们也提出了一种用于多个查询的双树算法。我们用各种数据集评估所提出的算法，并展出其计算效率。

这些被提出的算法的理论分析将使我们更好地理解这些算法的计算效率。对我们的算法的运行时间进行严格的分析将是我们未来工作的一部分。

##### 引用

[1] R. Bayardo, Y. Ma, and R. Srikant. Scaling Up All Pairs Similarity Search. In Proceedings of the 16th Intl. Conf. on World Wide Web, 2007.

[2] R. M. Bell and Y. Koren. Lessons from the Netflix Prize Challenge. SIGKDD Explor. Newsl., 2007.

[3] J. Bennett and S. Lanning. The Netflix Prize. In Proc. KDD Cup and Workshop, 2007.

[4] A. Beygelzimer, S. Kakade, and J. Langford. Cover Trees for Nearest Neighbor. Proceedings of the 23rd Intl. Conf. on Machine Learning, 2006.

[5] C. L. Blake and C. J. Merz. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/, 1998.

[6] L. Cayton and S. Dasgupta. A Learning Framework

for Nearest Neighbor Search. Advances in Neural Info.

Proc. Systems 20, 2007.

[7] M. S. Charikar. Similarity Estimation Techniques from

Rounding Algorithms. In Proceedings of the 34th

annual ACM Symp. on Theory of Comp., 2002.

[8] P. Ciaccia and M. Patella. PAC Nearest Neighbor

Queries: Approximate and Controlled Search in High-dimensional and Metric spaces. Proceedings of 16th Intl. Conf. on Data Engineering, 2000.

[9] K. Clarkson. Nearest-neighbor Searching and Metric Space Dimensions. Nearest-Neighbor Methods for Learning and Vision: Theory and Practice, 2006.

[10] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In Proceedings of the 40th annual ACM Symp. on Theory of Comp., 2008.

[11] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by Latent Semantic Analysis. Journal of the American Society of Info. Science, 1990.

[12] G. Dror, N. Koenigstein, Y. Koren, and M. Weimer. The Yahoo! Music Dataset and KDD-Cup’11. Journal Of Machine Learning Research, 2011.

[13] J. H. Freidman, J. L. Bentley, and R. A. Finkel. An Algorithm for Finding Best Matches in Logarithmic Expected Time. ACM Trans. Math. Softw., 1977.

[14] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. Proceedings of the 25th Intl. Conf. on Very Large Data Bases, 1999.

[15] A. G. Gray and A. W. Moore. ‘N-Body’ Problems in Statistical Learning. In Advances in Neural Info. Proc. Systems 13, 2000.

[16] A. G. Gray and A. W. Moore. Nonparametric Density Estimation: Toward Computational Tractability. In SIAM Data Mining, 2003.

[17] GroupLens. MovieLens dataset.

[18] P. Indyk and R. Motwani. Approximate Nearest

Neighbors: Towards Removing the Curse of Dimensionality. In Proceedings of the 30th annual ACM Symp. on Theory of Comp., 1998.

[19] S. Kim, F. Li, G. Lebanon, and I. Essa. Beyond Sentiment: The Manifold of Human Emotions. Arxiv preprint arXiv:1202.1568, 2011.

[20] M. Klaas, D. Lang, and N. de Freitas. Fast Maximum-a-posteriori Inference in Monte Carlo State Spaces. In Artificial Intelligence and Statistics, 2005.

[21] Y. Koren. The BellKor solution to the Netflix Grand Prize. 2009.

[22] Y. Koren, R. M. Bell, and C. Volinsky. Matrix Factorization Techniques for Recommender Systems. IEEE Computer, 2009.

[23] B. Kulis and K. Grauman. Kernelized Locality-sensitive Hashing for Scalable Image Search. In IEEE 12th Intl. Conf. on Computer Vision, 2009.

[24] Y. LeCun. MNist dataset, 2000. http://yann.lecun.com/exdb/mnist/.

[25] Z. Li, H. Ning, L. Cao, T. Zhang, Y. Gong, and T. S. Huang. Learning to Search Efficiently in High Dimensions. In Advances in Neural Info. Proc. Systems 24. 2011.

[26] T. Liu, A. W. Moore, A. G. Gray, and K. Yang. An Investigation of Practical Approximate Nearest Neighbor Algorithms. In Advances in Neural Info. Proc. Systems 17, 2005.

[27] R. Lupton, J. Gunn, Z. Ivezic, G. Knapp, S. Kent, and N. Yasuda. The SDSS Imaging Pipelines. Arxiv preprint astro-ph/0101420, 2001.

[28] S. M. Omohundro. Five Balltree Construction Algorithms. Technical report, International Computer Science Institute, December 1989.

[29] F. P. Preparata and M. I. Shamos. Computational Geometry: An Introduction. Springer, 1985.

[30] A. Rahimi and B. Recht. Random Features for Large-scale Kernel Machines. Advances in Neural Info. Proc. Systems 20, 2007.

[31] P. Ram, D. Lee, W. March, and A. Gray. Linear-time Algorithms for Pairwise Statistical Problems. In Advances in Neural Info. Proc. Systems 22. 2009.

[32] P. Ram, D. Lee, H. Ouyang, and A. G. Gray. Rank-Approximate Nearest Neighbor Search: Retaining Meaning and Speed in High Dimensions. In Advances in Neural Info. Proc. Systems 22. 2009.