第 37 卷 第 24 期
Vol.37 No.24

计 算 机 工 程
Computer Engineering

2011 年 12 月
December 2011

• 软件技术与数据库 •　　文章编号：1000—3428(2011)24—0022—03　　文献标识码：A　　中图分类号：TP311.13

高维主存的反向 K 最近邻查询及连接

刘 艳[1,2]，郝忠孝[1,3]

(1. 哈尔滨理工大学计算机科学与技术学院，哈尔滨 150080；2. 长春大学计算机科学技术学院，长春 130022；
3. 哈尔滨工业大学计算机科学与技术学院，哈尔滨 150001)

摘 要：对高维主存的反向 K 最近邻(KNN)查询进行研究，提出一种 Δ-RdKNN-tree 索引结构。通过在该索引结构上进行主存 KNN 自连接，预处理数据集中点的 KNN 距离信息。将这些距离扩展到索引的各层节点中，基于该索引设计高维主存的反向 KNN 查询算法以及反向 KNN 连接算法。分析结果表明，该算法在高维空间中是有效的。

关键词：高维；主存；反向 K 最近邻查询；反向 K 最近邻连接；预处理

High-dimensional Main-memory Reverse K Nearest Neighbor Query and Join

LIU Yan[1,2], HAO Zhong-xiao[1,3]

(1. College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China;
2. College of Computer Science and Technology, Changchun University, Changchun 130022, China;
3. College of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

【Abstract】The Reverse K Nearest Neighbor(RKNN) problem is a generalization of the reverse nearest neighbor problem which receives increasing attention recently, but high-dimensional RKNN problem is little explored. This paper studies on the high-dimensional main-memory RKNN queries, proposes an indexing structure called Δ-RdKNN-tree, precomputes KNN distances of points in the dataset by main-memory KNN self-join based on this index and propagates these distances to higher level index nodes. Main-memory RKNN query algorithm based on this index is proposed and main-memory RKNN join algorithm is given for set-oriented RKNN queries. Analysis shows that the two algorithms are effective in high dimension space.

【Key words】high-dimensional; main-memory; Reverse K Nearest Neighbor(RKNN) query; Reverse K Nearest Neighbor(RKNN) join; preprocessing

DOI: 10.3969/j.issn.1000-3428.2011.24.008

1 概述

反向 K 最近邻(Reverse K Nearest Neighbor, RKNN)查询在决策支持系统、市场决策、资料库文件搜索、生物资讯及地理信息系统(GIS)等领域中起着非常重要的作用。目前对 RKNN 查询的研究主要局限在 RNN 查询和低维数据。高维 RKNN 查询的研究不是很多，文献[1]只给出了求高维 RKNN 查询近似解的方法，文献[2]给出在度量空间上的 RKNN 查询算法，文献[3]中的 TPL 算法在低维空间中且 k 比较小时是有效的，但在高维空间中修剪搜索区域的计算代价非常昂贵。这些 RKNN 查询算法都是针对磁盘系统而设计的，随着主存容量越来越大，且价格逐渐低廉，使得将数据集和索引装入主存成为可能，已有算法不再是最好的选择。

目前已提出的有效处理 RKNN 查询的方法可以分为 2 类：预处理方法和空间修剪方法。当数据的维数高或 k 值大的时候，空间修剪方法的计算代价是非常昂贵的，所以在 KNN 连接可用且高效的情况下，本文采用预处理方法实现高维数据的主存 RKNN 查询和 RKNN 连接。

2 索引结构 Δ-RdKNN-tree

2.1 Δ-RdKNN-tree 的构建

为了有效地支持高维主存反向 K 最近邻查询和连接，本文在主存索引结构 Δ-tree-R[4]的基础上为包含 N 个点的数据集 R 构建 Δ-RdKNN-tree。该索引结构在 Δ-tree-R 的节点记录中加入了 K 最近邻距离信息以满足查询反向 K 最近邻的需要。Δ-RdKNN-tree 的叶子节点主要包含形如(*center*, *radius*, *num*, *pointer*, *code*, *codeLen*, *max_dknn*)的记录，其中，*center*、*radius* 和 *num* 分别为叶子节点的中心、半径及其包含的数据点的数目；*pointer* 为指向数据点的指针；*code*、*codeLen* 分别为叶子节点在 Δ-RdKNN-tree 中的编码及编码长度；*max_dknn*=max{*dknns*(*p*)}，*dknn* 表示点 p 到它的第 k 个最近邻之间的距离值，p 是该叶子节点中包含的点。

内部节点包含表示该节点自身信息的形如(*num*, *code*, *codeLen*, *max_dknn*)的记录，以及表示该内部节点中包含的子聚类的信息，形如(*center*, *radius*, *children*)的一组记录，每个子聚类(即孩子节点)对应一个记录，其中，*num*、*code*、*codeLen* 和 *max_dknn* 的含义与叶子节点中的各项类似；*center*、*radius* 分别为子聚类的中心和半径；*children* 为指向子聚类(即下一层孩子节点)的指针。

构建索引 Δ-RdKNN-tree 时，Δ-RdKNN-tree 各节点中 *max_dknn* 的初始值为−1。本文提出的索引 Δ-RdKNN-tree 既支持 KNN 查询及连接，也支持 RKNN 查询及连接。

基金项目：黑龙江省自然科学基金资助项目(F2006-01)

作者简介：刘 艳(1981－)，女，讲师、博士研究生，主研方向：高维数据处理，空间数据库；郝忠孝，教授、博士生导师

收稿日期：2011-07-15　**E-mail：**liuyan6374@yahoo.com.cn

2.2　修剪原理

定理 1(点与节点之间的修剪原理) 在转换后的低维 PCA[5]空间中，如果查询点与 Δ-RdKNN-tree 中节点之间的距离大于该节点的 *max_dknn* 值，则该节点中不包含查询点的 RKNN。

证明详见文献[6]中的定理 1。

定理 2(节点与节点之间的修剪原理) 在转换后的低维 PCA 空间中，如果 Δ-RdKNN-tree 中的节点 A 与 Δ-tree 中的节点 B 之间的距离大于节点 A 的 *max_dknn* 值，则节点 A 中不包含节点 B 中各查询点的 RKNN。

证明：设 Q 为聚类 B 中任意一点，其转换后为 Q'。PCA 转换后，聚类 A 的中心到 Q'的距离大于到聚类 B 的距离，由文献[6]中的定理 1，Q 到聚类 A 中任意一点的距离大于转换后 Q'到聚类 A 的距离，所以，转换后的低维空间中 2 个聚类之间的距离为这 2 个聚类中任意 2 点之间距离的下限，命题得证。

3　反向 K 最近邻查询及连接算法

Δ-tree-KNN-Join[4]是一种有效的高维主存 KNN 连接算法，该算法使用 Δ-tree[7]作为基础索引，采用编码解码、自底向上、深度优先遍历和剪枝等技术，解决了 KNN 连接中确定搜索半径困难的问题。本节通过在 Δ-RdKNN-tree 上执行 Δ-tree-KNN-Join 的自连接算法，求出数据集 R 中各点的 KNN 距离，再通过算法 Add-Distance-KNN 将这些 KNN 距离信息扩展到 Δ-RdKNN-tree 各层节点的 *max_dknn* 中。在 Δ-tree-KNN-Join 的自连接算法中，不需要对 Δ-RdKNN-tree 中的叶子节点进行解码，直接使用其编码找到它们的各祖先节点，然后按自底向上的顺序进行连接即可，具体算法参见文献[4]。算法 Add_Distance_KNN 通过深度递归的方式遍历 Δ-RdKNN-tree 中的各节点，对于每个节点遍历其中包含的数据点，取 KNN 距离的最大值作为该节点的 *max_dknn*，限于篇幅，具体算法省略。

3.1　基于 Δ-RdKNN-tree 的反向 K 最近邻查询

算法 RKNN_Query 的符号说明：*node* 为 Δ-RdKNN-tree 中的内部节点；*queryPoint* 为与数据集 R 中的数据点同时经过 PCA 转换的查询点；*result* 表示 *queryPoint* 在数据集 R 中的反向 K 最近邻的集合。

算法 1 RKNN_Query(基于 Δ-RdKNN-tree 的反向 K 最近邻查询算法)

输入 *node*，*queryPoint*

输出 *result*

```
begin
(1)for   node 中的每个孩子节点 child_i   do
   if P_dist(child_i, queryPoint, m_child_i.lev)<child_i.max_dknn   then;
    if child_i 为内部节点 then
        RKNN_Query(child_i, queryPoint);
    else   //为叶子节点
        for child_i 中的每个数据点 point   do
            if dist(point, queryPoint)<point.dknn   then
    //point.dknn 表示点 point 到它的第 k 个最近邻之间的距离
                result = result ∪ point;
(2)return result;
end
```

定理 3 算法 1 对 Δ-RdKNN-tree 正确地进行了 RKNN 查询。时间复杂度为 $O((f^{L-1}-1)m_{child_i.lev}\alpha/(f-1)+f^L\alpha d)$。其中，$f$ 是 Δ-RdKNN-tree 中节点的扇出；$m_{child_i.lev}$ 为孩子节点 $child_i$ 所在层的维数；α为步骤(1)中第 1 个 if 语句的选择度；d 为数据点的维数。

证明：

正确性：执行步骤(1)，遍历 *node* 中的每个孩子节点 $child_i$，由定理 1，如果 $child_i$ 到 *queryPoint* 之间的距离小于 $child_i$ 中 *max_dknn* 的值，则根据节点类型分 2 种情况进行处理：如果 $child_i$ 为内部节点，则递归调用算法 RKNN_Query，对以 $child_i$ 为根的子树进行 RKNN 查询，如果 $child_i$ 为叶子节点，则遍历 $child_i$ 中的每个 *point*，计算 *point* 与 *queryPoint* 之间的距离，将小于 KNN 距离的点加入结果集。当递归调用停止时，该算法使用修剪策略完成对 Δ-RdKNN-tree 的 RKNN 查询。

可终止性：算法 RKNN_Query 被调用的次数是有限的，步骤(1)中 for 循环可自动终止，故该算法可终止。

时间复杂度分析：执行步骤(1)，Δ-RdKNN-tree 中节点执行 P_dist()的时间复杂度为 $O((f^{L-1}-1)m_{child_i.lev}\alpha/(f-1))$；Δ-RdKNN-tree 中叶子节点执行 dist()的时间复杂度为 $O(f^L\alpha d)$。综上，该算法总的时间复杂度为 $O((f^{L-1}-1)m_{child_i.lev}\alpha/(f-1)+f^L\alpha d)$。

证毕。

3.2　基于 Δ-RdKNN-tree 的反向 K 最近邻连接

在使用 RKNN 的数据挖掘任务中往往需要为数据集中的每个点执行 RKNN 查询(即面向集合的 RKNN 查询)，本节给出基于 Δ-RdKNN-tree 的 RKNN 连接算法。对于进行主存 RKNN 连接的 2 个数据集 R 和 S，首先将它们同时进行 PCA 转换，然后分别为转换后的数据集 R 和 S 构建索引 Δ-RdKNN-tree 和 Δ-tree[7]。

算法 RKNN_Join 及其子算法 Leaf_RKNN_Join 的符号说明：*leafR*、*nodeR* 分别为 Δ-RdKNN-tree 中的叶子节点和内部节点，*leafS*、*nodeS* 分别为 Δ-tree 中的叶子节点和内部节点，*result* 表示数据集 R 中每个查询点的反向 K 最近邻的集合。

算法 2 Leaf_RKNN_Join(*leafR* 和 *leafS* 进行反向 k 最近邻连接)

输入 *leafR*，*leafS*

输出 *result*

```
begin
(1)for   leafS 中的每个查询点 queryPoint   do
      if  (P_dist(leafR,  queryPoint,  min(m_leafR.lev,  m_leafS.lev)) <
      leafR.max_dknn)   then
          for leafR 中的每个数据点 point   do
              if dist(point, queryPoint) < point.dknn;
                result = result ∪ point;
(2)return result;
end
```

定理 4 算法 Leaf_RKNN_Join 对 *leafR* 和 *leafS* 中的数据点对正确地进行了 RKNN 连接。时间复杂度为 $O(f(m_{\min}+\sigma' fd))$级。其中，$m_{\min}$ 为 $m_{leafR.lev}$ 和 $m_{leafS.lev}$ 中较小的维，在满足 Δ-RdKNN-tree 的情况下，$m_{\min}$ 取值为 d，$m_{leaf.lev}$ 为节点 *leaf* 所在层的维数，σ'为步骤(1)中第 1 个 if 条件的索引选择度。

证明：

正确性：执行步骤(1)，根据定理 1 将 *leafS* 中距离 *leafR* 大于 *leafR.max_dknn* 的 *queryPoint* 过滤掉，然后使用 dist()计算 *leafS* 中剩余查询点与 *leafR* 中每个数据点之间的真实距离，将距离小于 KNN 距离的数据点对加入结果集。

可终止性：该算法中只有一个双重 for 循环，故可自动

终止。

时间复杂度分析：执行步骤(1)中内、外层 for 循环都为 $O(f)$级，P_dist()为 $O(m_{\min})$级，dist()为 $O(d)$级，故算法的时间复杂度为 $O(f(m_{\min}+\sigma' fd))$。

证毕。

算法 3 RKNN_Join(基于 Δ-RdKNN-tree 的反向 K 最近邻连接算法)

输入 *nodeR*，*nodeS*

输出 *result*

```
begin
(1) Queue = NewPriorityQueue(); result = ∅;
(2) for  nodeR 中的每个孩子节点 child_i  do
      for  nodeS 中的每个孩子节点 child_j  do
        if (P_dist(child_i, child_j, min(m_child_i.lev, m_child_j.lev)) <
         child_i.max_dknn)  then
            if child_i 和 child_j 都为内部节点  then
               RKNN_Join(child_i, child_j);
            else if child_i 和 child_j 都为叶子节点  then
               result = Leaf_RKNN_Join (child_i, child_j);
            else if child_i 为叶子节点且 child_j 为内部节点  then
          Enqueue (Queue, child_j);
              While Queue 非空  do
                node = RemoveFirst(Queue);
                for node 中的每个孩子节点 child_k  do
                  if  (P_dist(child_i,  child_k,  min(m_child_i.lev,
                  m_child_k.lev)) < child_i. max_dknn)  then
                    if child_k 是内部节点  then
                      Enqueue (Queue, child_k);
                    else  // child_k 是叶子节点
                      result = Leaf_RKNN_Join
                      (child_i, child_k);
            else   //child_i 为内部节点且 child_j 为叶子节点
              处理方法与第 3 种情况类似，故省略
(3) return result;
 end
```

定理 5 算法 RKNN_Join 正确地对 Δ-RdKNN-tree 和 Δ-tree 进行 RKNN 连接，是可终止的，时间复杂度为 $O((f^2-f^{2L-2})f^2 m_{lev}\sigma/(1-f^2)+f^{2L-1}\sigma d(1+f\sigma'))$，其中，$\sigma$为步骤(2)中第 1 个 if 条件的索引选择度；m_{lev} 表示 Δ-RdKNN-tree 中第 *lev* 层使用的维数。

证明：

正确性：执行步骤(1)，初始化优先队列 *Queue* 和 *result*。执行步骤(2)，由定理 2，保留 *nodeR* 和 *nodeS* 中满足节点与节点之间修剪条件的孩子节点对，然后根据节点类型分 4 种情况进行处理。当对该算法的递归调用停止且 *Queue* 为空时，该算法对 Δ-RdKNN-tree 和 Δ-tree 中所有满足修剪条件的叶子节点对进行了 RKNN 连接，由定理 4，该算法对 Δ-RdKNN-tree 中和 Δ-tree 中所有点对正确地进行了 RKNN 连接处理。

可终止性：在步骤(2)中，双重 for 循环及单重 for 循环均可自动终止，算法 RKNN_Join 被递归调用的次数及 *Queue* 的容量都是有限的，算法 Leaf_RKNN_Join 可终止，故该算法可终止。

时间复杂度分析：满 Δ-RdKNN-tree 和满 Δ-tree 的情况下，执行步骤(2)，Δ-RdKNN-tree 和 Δ-tree 中的节点对执行 P_dist()的时间复杂度共为 $O((1+(f^2-f^{2L-2})\sigma/(1-f^2))f^2 m_{lev})$，叶子节点对执行算法 Leaf_RKNN_Join 的时间复杂度共为 $O(f^{2L-1}\sigma d(1+f\sigma'))$，故算法的时间复杂度为 $O((1+(f^2-f^{2L-2})\sigma/(1-f^2))f^2 m_{lev}+f^{2L-1}\sigma d(1+f\sigma'))$。

证毕。

4 算法性能分析

目前还没有专门为高维主存 RKNN 查询和连接设计的算法，在为磁盘而设计的高维 RKNN 查询算法中，或是不能提供准确的解[1]，或是计算代价昂贵[3]。本文采用预处理的方法求解主存 RKNN 查询，在高维情况下，使用“一次计算的操作”KNN 连接代替多个 KNN 查询是非常好的选择，文献[4]已证明 KNN 连接算法 Δ-tree-KNN-Join 的高效性，所以本文使用它的自连接算法计算出数据集 R 中每个点的 KNN 及 KNN 距离；文献[7]已对算法 RangeQuery 的优越性进行了证明，所以基于该算法的 RKNN 查询算法 RKNN_Query 具有较高的效率；文献[8]已对 Δ-tree-Join 算法的高效性进行了证明，故本文基于该算法的 RKNN_Join 算法具有较优的性能。另外，本文提出的 RKNN 查询及连接算法也适用于灵活的 k，当 k 变化时，只需在索引 Δ-RdKNN-tree 上重新执行 Δ-tree-KNN-Join 的自连接算法和算法 Add_Distance_KNN 以更新索引节点中的 *max_dknn* 值后，就可以对新的 k 进行 RKNN 查询及连接。

5 结束语

本文为基于主存的高维数据设计了索引 Δ-RdKNN-tree，根据该索引提出了主存 RKNN 查询算法 RKNN_Query 和 RKNN 连接算法 RKNN_Join。算法性能分析表明，这 2 个算法是高效的和可扩展的。高维主存 K 最近对连接、KNN 连接和 RKNN 连接的具体应用问题有待于进一步研究。

参考文献

[1] Singh A, Ferhatsomanoglu H, Tosun A S. High Dimensional Reverse Nearest Neighbor Queries[C]//Proc. of the l2th International Conference on Information and Knowledge Management. New Orleans, Louisiana, USA: ACM Press, 2003: 91-98.

[2] Achtert E, Bohm C, Kroger P, et al. Efficient Reverse k-Nearest Neighbor Search in Arbitrary Metric Spaces[C]//Proc. of the ACM International Conference on Management of Data. Chicago, Illinois, USA: ACM Press, 2006: 515-526.

[3] Tao Yufei, Papadias D, Lian Xiang, et al. Multidimensional Reverse kNN Search[J]. International Journal on Very Large Data Bases, 2007, 16(3): 293-316.

[4] 刘 艳, 郝忠孝. 一种基于主存 Δ-tree 的高维数据 KNN 连接算法[J]. 计算机研究与发展, 2010, 47(7): 1234-1243.

[5] Jolliffe I T. Principal Component Analysis[M]. New York, USA: Springer-Verlag, 1986.

[6] 刘 艳, 郝忠孝. 基于Δ-tree 的递归深度优先KNN查询算法[J]. 计算机工程, 2011, 37(22): 48-50.

[7] Cui Bin, Chin O B, Su Jianwen, et al. Contorting High Dimensional Data for Efficient Main Memory KNN Processing[C]//Proc. of the ACM SIGMOD International Conference on Management of Data. New York, USA: ACM Press, 2003: 479-490.

[8] 郝忠孝. 时空数据库查询与推理[M]. 北京: 科学出版社, 2010.

编辑 任吉慧