空间对象的反最近邻查询*)

郝忠孝[1,3]　刘永山[1,2]
（哈尔滨理工大学计算机与控制学院　哈尔滨 150080）（燕山大学信息科学与工程学院　秦皇岛 066004）
（齐齐哈尔大学计算机学院　齐齐哈尔 161006）

摘　要　本文在对现有反最近邻查询方法研究的基础上，提出了一种新的索引结构—SRdnn-树；在此基础上提出了基于SRdnn-树的反最近邻查询方法，并给出了该结构上的最近邻查询方法，以及插入和删除方法，第5节实验表明，基于SRdnn-树的反最近邻查询在性能上优于以往查询方法。

关键词　最近邻，反最近邻，SRdnn-树

Reverse Nearest Neighbor Search in Spatial Database

HAO Zhong-Xiao[1,3]　LIU Yong-Shan[1,2]
(Harbin University of Science & Technology, Harbin 150080)[1]　(Yanshan University, Qinhuangdao 066004)[2]
(Qiqiha University, Qiqiha 161006)[3]

Abstract　In this paper we have studied the reverse nearest neighbor and existing algorithms. We present a new structure—SRdnn-tree and a new algorithm based on this structure. Also, the nearest neighbor search algorithm, the deletion and the insertion algorithms are provided. According to the experiments, the performance of reverse nearest neighbor search based on the SRdnn-tree has more advantages over the existing one.

Keywords　Nearest neighbor, Reverse nearest neighbor, SR-tree

1　引言

反最近邻查询(Reverse Nearest Neighbor, RNN)是在最近邻查询(Nearest Neighbor, NN)基础上提出的一种新的查询类型[1]。反最近邻查询问题是和人们平常所遇到的影响集概念相一致的。比如一家百货公司想在某一地点设一个新的分公司，这必然会对附近的同类公司或商店(假设人们一般选择就近购物)带来商业上的影响，而受其影响最大的同类公司或商店，就是反最近邻查询所要的结果。另外，在相似性查询问题中，存在反向最相似问题。例如，某公司为一新产品做广告，以电子邮件的形式发到用户的信箱里，但不能随意发到每个人的信箱，而要发给那些对本产品相当感兴趣的人，这就不会使用户的信箱总是被一些垃圾信息所充满，也能使产品的广告更加有效。发给哪些用户？实际上就是反最近邻查询。

目前对反最近邻所作的研究还很有限，所提出的查询方法的性能也都不尽如人意。据我们所知，对反最近邻查询研究的文献主要有两篇[1,2]，一篇是 Korn 和 Muthukrishnan 提出的基于 RNN-树的查询方法，另一篇是由 C. Yang 和 K.-I. Lin 提出的基于 Rdnn-树的查询方法。由于它们都是从 R*-树演化而来的，高维情况下，查询性能随着维数的增加急剧下降，为此本文提出一种新的索引结构，即 SRdnn-树。实验结果表明，基于此结构的反最近邻查询，在性能上优于其它查询方法。

本文第2节对反最近邻的定义作了说明，第3节简要介绍了当前的两种查询方法，并指出了它们的主要不足，第4节提出了 SRdnn-树索引结构，并给出反最近邻查询、最近邻查询以及插入和删除方法，第5节给出了实验结果，最后是结论。

2　反最近邻的定义

先给出 RNN 查询的形式化定义，这里假设 $D(p,q)$ 为两点 p 和 q 之间的距离。

定义 1　假设有一 d 维数据集 S 和一查询点 q，RNN 查询就是找出 S 的子集 $RNN_s(q)$，即：

$$RNN_s(q)=\{r\in S \mid \forall\, p\in S: D(q,r)\leqslant D(r,q)\}$$

本文以符号 $RNN_s(q)$ 表示对查询点 q 进行反最近邻查询的返回结果，以 $NN_s(q)$ 表示对 q 进行最近邻查询的结果，$dnns(p)$ 表示 p 和它的最近邻之间的距离 $D(p, NN_s(p))$。$RNN_s(q)$ 和 $NN_s(q)$ 中的数据点都是所给数据集合或数据库中的点，而查询点 q 可以是数据集合中的点，也可以不是。

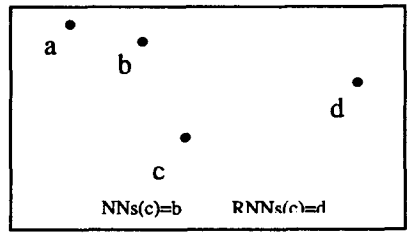

图1　最近邻和反最近邻

一般来讲，$RNN_s(q)$ 和 $NN_s(q)$ 没有必然的关系，即由 $r\in RNN_s(q)$ 不能得出 $r\in NN_s(q)$，反之亦然，如图1所示。

3　相关研究

目前，人们提出的 RNN 查询算法有两种，一种是 Korn

*)黑龙江省自然科学基金 F00-06。郝忠孝　教授，博士生导师，主要从事数据库理论及应用研究。刘永山　教授，主要从事数据理论及计算机应用研究。

和 Muthukrishnan 提出的基于 RNN-树的查询方法[1]，另一种是由 C. Yang 和 K. I. Lin 提出的基于 Rdnn-树的查询方法[2]。前者是 RNN 查询的基本方法，后者的查询性能比前者有所提高，但是，由于它们都是基于 R* 树的，仅仅对低维数据有效，在处理高维数据时(5 维以上)，查询性能急剧下降。

RNN-树在本质上是一棵 R*-树，只是对于每一点 p，叶子节点存储了以 p 为圆心，以 $dnns(p)$ 为半径的圆的最小包围矩形，这样，RNN 查询就归结为点查询问题。对于任意查询点 q，p 是 q 的 RNN 当且仅当 q 在以 p 为圆心，$dnns(p)$ 为半径的圆周内，当然也落在圆的 MBR 中。因此，只要确定那些包含点 q 的圆，并返回其圆心 p，p 就是满足条件的查询结果。基于 RNN-树的反最近邻查询方法的缺点主要有两个：第一是叶子节点存储的是区域而不是点，重叠增加，使得父节点的 MBR 变大，影响 RNN 查询性能。第二是要求有第二棵树来实现 NN 查询。

Rdnn-树克服了 RNN-树的缺点，在节点中存储了最近邻的距离值，这样既不用存储区域值，也不用第二棵树来实现 NN 查询。

Rdnn-树的叶子节点包含形如($ptid$, dnn)的记录，其中 $ptid$ 表示数据集中的一 n 维数据点，dnn 表示该点到其最近邻的距离值；非叶子节点包含形如(ptr, Rect, max_dnn)的一组记录，其中 ptr 是孩子节点的地址，如果 ptr 指向的是叶子节点，Rect 是该叶子节点内所有点的最小包围矩形，如果 ptr 指向的是中间节点，Rect 是其指向的孩子节点内所有最小包围矩形的最小包围矩形；max_dnn = max{$dnns(p)$}，p 是以 ptr 指向的节点为根的子树内的点。

在 Rdnn-树上进行 RNN 查询，类似于点定位查询，设查询点为 q，若当前节点是叶子节点，则对节点中每一点 p，计算其到查询点的距离 $D(q,p)$，如果 $D(q,p) \leqslant dnns(p)$，那么 p 就是 q 的一个反最近邻；若当前节点不是叶子，则对每个记录(ptr, Rect, max_dnn)，计算查询点到每个分枝的距离 $D(q,$ Rect)，如果 $D(q,$ Rect$) >$ max_dnn，那么该分枝就被剪除，不被访问，因为 max_dnn 是该分枝内所有点的最近邻距离的最大值，该分枝内一定没有查询点的反最近邻，否则，递归调用本查询算法，这里 $D(q,$ Rect$)$ 表示 q 和 Rect 的最小距离。

与 RNN-树相比，Rdnn-树省掉了第二棵树的应用，而且避免了叶子节点中存储区域，这必然使其 RNN 查询更加有效。然而，这两种树都是基于 R*-树的，受 R*-树结构的限制，在维数增加时，其性能急剧恶化[3]。因此有必要研究新的索引结构，以改进其性能。

4 基于 SRdnn-树的反最近邻查询算法

鉴于 R*-树在高维数据空间查询的局限性，加上 SR-树[4]的优越性，用 SRdnn-树来实现 RNN 查询，以提高其查询性能。SR-树的最突出特点就是节点记录中存储了下层节点或数据点的最小包围矩形(MBR)和最小包围圆(MBS)，也就是说，下层的数据存放在其 MBR 和 MBS 的相交区域，这也正是集中了 R-树和 SS-树[5]的优点。

4.1 SRdnn-树索引结构

这里提出的 RNN 查询方法，是在 SRdnn-树上进行的，与标准 SR-树不同的是，在节点的记录中，加入了最近邻信息，称这样的树结构为 SRdnn-树。SRdnn-树的叶子节点记录形式为(p, dnn)，其中 dnn 是 p 和它的最近邻之间的距离值，非叶子节点记录形如(S, R, w, $childptr$, max_dnn)，其中 max_dnn = max{$dnns(p)$}，p 是以 $childptr$ 指向的节点为根的子树内的点，其他记录项和 SR-树相同。这种结构在设计思想上和 Rdnn-树一样，但它是建立在 SR-树这样一个性能优越的索引结构基础上的。

4.2 基于 SRdnn-树的反最近邻查询算法

由于 SRdnn-树结构存储了最近邻信息，这就使 RNN 查询像点查询一样。假设 q 是查询点，如果当前节点是叶子，则对叶子中每个记录(p, dnn)，计算该点 p 和查询点 q 的距离值 $D(p,q)$，如果 $D(p,q) \leqslant dnn$，那么 p 就是 q 的一个 RNN；如果当前节点是非叶子节点，则计算 q 和每个记录(S, R, w, $childptr$, max_dnn)中的 S 的距离 $D(q,S)$，如果 $D(q,S) >$ max_dnn，那么该记录中 $childptr$ 指向的子树就被剪除，因为其中的任何点和 q 的距离都不会小于该点的 dnn 值，即不可能是 q 的 RNN，否则，递归调用本查询算法。这里的距离值 $D(q,S)$ 的计算要比 Rdnn-树中的 $D(q,$ Rect$)$ 的计算简单得多，它只要计算两点之间的欧几里得距离就行了，即点 q 到圆心的距离，再减去圆的半径值，由于该子树中的数据点都包含在该圆周内，因此这个计算结果就是查询点到该子树中点的距离的下限值，可以作为剪枝的标准。基于 SRdnn-树的 RNN 查询算法用伪代码描述如下。

```
RNN_Search(Node n,Point q){
  //n 为 SRdnn-树节点类型数据,q 为查询点,该程序输出 q 的反最近邻
  if( n is a leaf node)
    for(all entry(p,dnn)in n )
      if(D(q,p)≤dnn)
        printf("p is one of the RNN of q");
  else
    if(n is a internode node)
      for(all entry(S,R,w,childptr,max_dnn)in n )
        if(D(q,S)≤max_dnn)
          RNN_Search(*childptr,q);
}
```

4.3 基于 SRdnn-树的最近邻查询算法

在 SRdnn-树中存储了最近邻距离信息，因此有下面的定理：

定理 1 设 q 是一查询点，p 是数据集 S 中的任一点，如果满足不等式 $D(p,q) \leqslant dnns(p)/2$，那么 p 就是 q 的最近邻。

该定理的证明很容易，由 $dnns(p)$ 的定义，圆 $C(P, dnns(p))$ 的圆周内不包含数据集中任何点，如果 $d(p,q) \leqslant dnns(p)/2$，则对于任一点 $x \notin C(p, dnns(p))$，必有 $D(x,q) \geqslant dnns(p)/2$，所以对圆周外的任一点，到查询点的距离都大于 $D(q,p)$，因此 p 就是 q 的最近邻。

根据这个定理可以对 SRdnn-树进行修剪，减少访问节点次数，提高查询效率，以下是最近邻查询的伪代码描述。

```
NN_Search(Node n, Point q){
  //n 为 SRdnn-树节点类型数据,q 为查询点,该程序输出 q 的最近邻,初始化最近邻为 c
  if(n is a leaf node)
    for(all entry(p,dnn)in n )
      if(D(q,p)≤(q,c)
        replace c by p;
      if(D(q,p)≤dnns(p)/2
        printf("p is the NN");
    else
      if(n is a internode node){
        for(all entry Bi=(S,R,w,childptr,max_dnn) in n)
          di=D(q,s);
        Sort Bi according di;
        for(all Bi in the sorted order)
          if di≤(q,c)
            NN_Search(childptr,q);
      }
}
```

基于 SRdnn-树的最近邻查询算法，由于利用包围圆的最近邻查询的优越性，其查询性能要比 Rdnn-树好，从上面伪代码描述中也可以看出，计算查询点到节点的距离可以节省大

量的计算量，这也是 SRdnn-树的优势。

4.4 基于 SRdnn-树的插入算法

SRdnn-树的插入算法需要更新最小包围圆和最小包围矩形，因为 SRdnn-树存储了最近邻距离信息，所以在插入新的数据点时，对由插入操作引起的相关点的最近邻距离信息的改变要进行更新操作，同时也要计算出被插入点的最近邻距离来和该点一起作为一个记录插入到叶子节点中。插入一点要引起它的反最近邻的相关记录的改变，所以在插入操作过程中要先进行反最近邻查询，找出其影响集，并更新其中的记录的 *dnn* 信息，由此引起的父节点的 max_*dnn* 信息的变化也要向上作调整，把这个调整过程叫预插入，伪代码描述如下。

```
Pre_Insert(Node n, Point q){
  //n 为 SRdnn-树节点类型数据，初始调用时为该树的根节点，q 为查询点，该程序对树进行调整，并输出 q 的最近邻，初始化最近邻为 c
  if (n is a leaf node)
    for(all entryin (p,dnn)in n)
      if(D(q,p))≤D(q,c)
        c=p;
      if(D(q,p))≤dnn
        printf("p is one of thf RNN",p,D(q,p));
  else
    for(all entry Bi=(S,R,w,childotr,max_dnn)in n)
      if((D(q,S)≤max_dnn) or D(q,S)<(q,c))
        Pre_Isert(childptr,q);
      if(childptr was adjusted)
        adjust max_dnn for Bi and return max_dnn;
}
```

经过上面的预插入过程之后，在 SRdnn-树中将插入点 q 和预插入过程中得出的 q 的最近邻距离值一起作为一个记录插入 SRdnn-树叶子节点中。

4.5 基于 SRdnn-树的删除算法

删除也和插入一样要进行更新删除点 q 的反最近邻的记录信息，删除一点就是删除叶子节点中的记录 (q,dnn)，q 的父节点中的 max_*dnn* 同时也做相应调整，另外 q 的每个反最近邻相应记录中的 *dnn* 域是它们到 q 的距离值，因此要对每个点进行最近邻查询操作，更新 *dnn* 值以及相应父节点中的 max_*dnn* 域值。除了更新 SRdnn-树中的最近邻信息之外，删除操作和 SR-树中的删除操作算法相同，下面是伪代码描述的这一过程。

```
Delete(Node n, Point q ){
  //从以 n 为根节点的 SRdnn-树中删除点 q
  Delete_SR-tree(Node n,Point q ); //调用 SR-树的删除算法
  RNN_Search(n,q); //调用反最近邻查询方法
  for(all p in RNNs of q ){
    NN_Search(n,p);
    Adjust the dnn of p;
    Propagate the change of p up to the root;
  }
}
```

SRdnn-树的查询方法具有 Rdnn-树的所有优点，此外，由于 SRdnn-树的区域是包围圆和包围矩形的公共区域，在建树过程减少了其节点之间的重叠，加快了查询的速度，尤其是在高维空间效果更加明显，从实验部分可以容易看出该结论。

5 实验

实验是在 Intel Celeron 1 GHz CPU，128 MB RAM，Windows 2000 平台上用 C 程序实现的，采用 GSTD 程序人工随机生成均匀分布、高斯分布和群分布的实验数据，根据数据集大小输出不同的节点访问次数。

5.1 反最近邻查询

将数据维数固定为 2 和 11 维，分别对不同大小的数据集进行测验，并与 Rdnn-树的查询结果进行比较。

首先测试二维数据的实验结果，如图 2、图 3、图 4 分别是均匀、高斯和群分布的数据集，横坐标是数据集的大小值，纵坐标是节点访问的次数，实线是 SRdnn-树的 RNN 查询结果曲线，虚线是 Rdnn-树的结果曲线。

由以上实验结果可以看出，在二维空间数据集的反最近邻查询，SRdnn-树的性能比 Rdnn-树上有所提高，而且两者都能在数据集的变换下有比较稳定的性能。图 5、图 6、图 7 分别是 11 维的均匀分布、高斯分布和群分布的数据集。

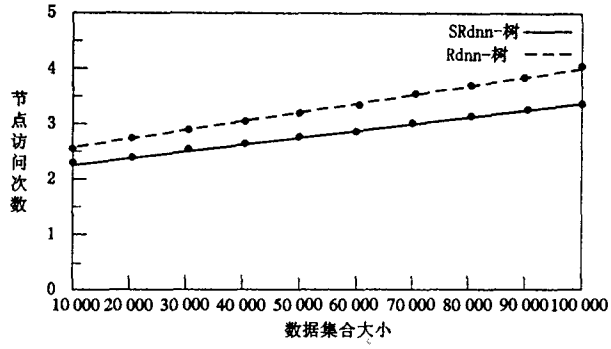

图 2 二维均匀数据集的查询性能

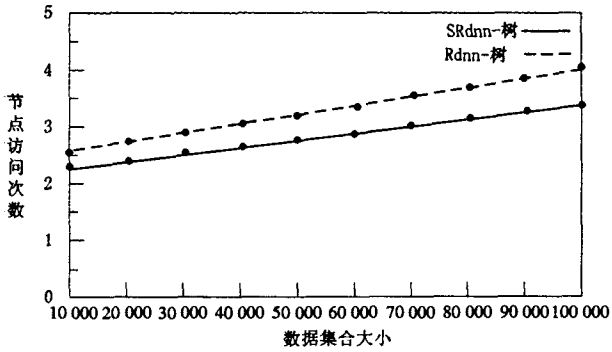

图 3 二维高斯分布数据集的查询性能

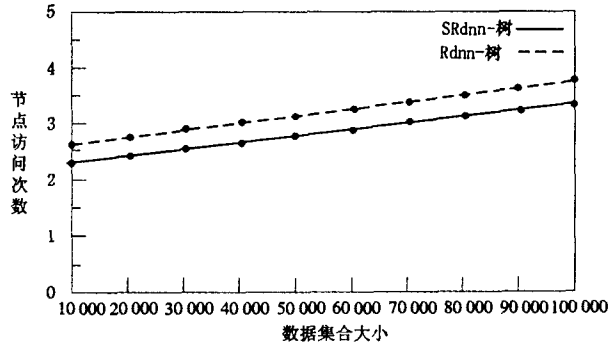

图 4 二维群分布数据集的查询性能

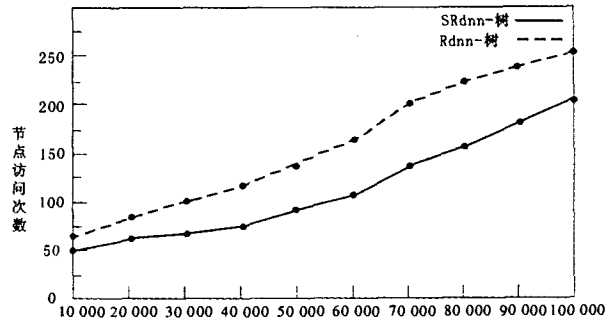

图 5 11 维均匀分布数据集的查询性能

在数据的维数为 11 时，SRdnn-树的反最近邻查询性能比 Rdnn-树有较大的提高，这是因为前者优越的结构，这在很大程度上突破了 R*-树系列索引结构的限制，使得在高维空间的反最近邻查询也能收到好的效果。而 Rdnn-树是基于 R*-树结构的，在维数大于 5 的时候由于性能受到很大限制，使得在此基础之上设计的 Rdnn-树的性能也很快恶化，因此对于高维空间的反最近邻查询，SRdnn-树具有明显优势。

5.2 最近邻查询

由前节中给出的最近邻和反最近邻查询算法，两者的查询过程基本是一致的，实验中查询所访问的节点数也大致相当，在此只给出 4 维群分布数据集合的节点访问次数和数据集大小的关系曲线，并在同等条件下与 Rdnn-树作了比较，如图 8 所示。

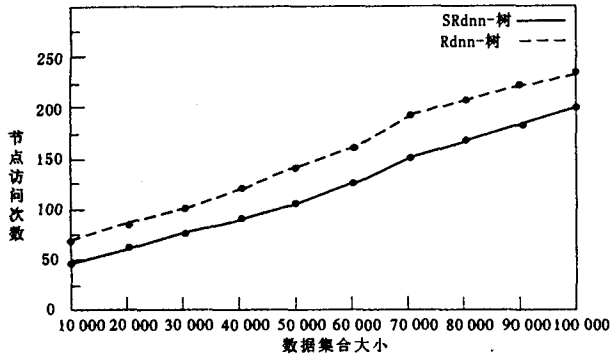

图 6　11 维均匀分布数据集的查询性能

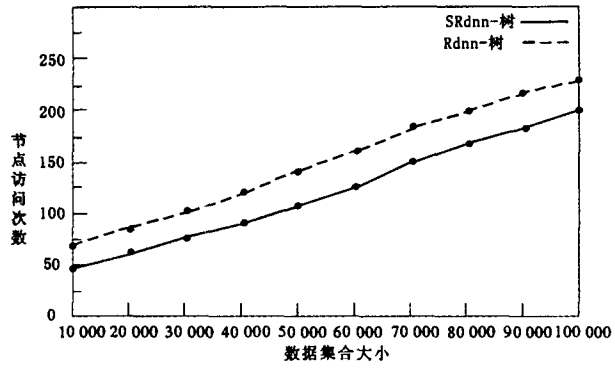

图 7　11 维均匀分布数据集的查询性能

5.3 插入和删除

由 SRdnn-树的结构，插入和删除过程中要更新最近邻距离信息，这就要求进行最近邻和反最近邻查询操作，这两个查询操作可以在同一过程进行，而且访问节点的次数主要花费在这两个查询上面，因此插入和删除操作的性能关键取决于最近邻和反最近邻查询的性能，这里不再给出实验结果，可参考查询实验。

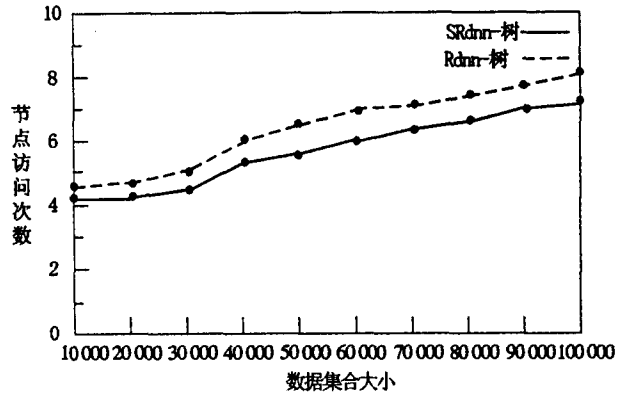

图 8　4 维群分布数据集的归近邻查询性能

结论　本文介绍了目前的两种主要的反最近邻查询方法，即基于 RNN-树和基于 Rdnn-树的查询方法。这两种方法都是从 R*-树演化而来的，在高维情况下，查询性能随着维数的增加急剧下降。本文提出了 SRdnn-树索引结构，并在这种新的结构上进行反最近邻查询、最近邻查询及动态情况下的操作。实验结果表明，此方法在高维空间与以往的查询方法相比有明显的优势。

本文是在高维索引结构下进行反最近邻查询，然而应用一维索引结构，如 B-树来实现此类查询，可以得到近似的查询结果，而且效率会更高，这也是今后的研究工作。

参 考 文 献

1　Korn F, Muthukrishnan S. Influence Sets Based on Reverse Nearest Neighbor Queries. In: Weidong Chen, Jeffrey F. Naughton and Philip A. Bernstein, eds. Proc. of the 2000 ACM SIGMOD Intl. Conf. on Management of Data, Dallas, Texas, USA, 2000. New York, NY, USA, ACM Press, 2000. 201～212

2　Yang C, Lin K I. An Index Structure for Efficient Reverse Nearest Neighbor Queries. In: George Kollios, ed. Proc. of the IEEE Intl. Conf. onData Engineering, Heidelberg, Germany, 2001. Washington, IEEE Computer Society, 2001. 485～492

3　Berchtold S, Bohm C. On Optimizing Nearest Neighbor Queries in High-Dimensional Data Spaces. In: Jan Van den Bussche, Victor Vianu, eds Database Theory-ICDT 2001, 8th Intl. Conf London, UK, 2001. Springer, 2001. 435～449

4　Katayama N, Satoh S. The SR-tree: An Index Structure for High-dimens ional Nearest Neighbor Queries. Transactions of the Institute of Electronics, Information and Communication Engineers, 1997. 703～717

5　White D A, Jain R. Similarity Indexing with the SS-tree. Stanley Y. W. Su. ICDE, New Orleans, Louisiana, 1996. Washington, IEEE Computer Society, 1996. 46～52