第35卷 第19期 计 算 机 工 程 2009年10月
Vol.35 No.19 Computer Engineering October 2009
·软件技术与数据库· 文章编号：1000—3428(2009)19—0049—04 文献标识码：A 中图分类号：TP391

基于P2P的自适应分布式 *k* 最近邻搜索算法

余小高[1]，余小鹏[2]
(1. 湖北经济学院信息管理学院，武汉 430205；2. 武汉工程大学经济管理学院，武汉 430073)

摘 要： *k* 最近邻搜索算法无法满足数据挖掘的分布性、实时性和可扩展性要求，针对该问题提出基于P2P的自适应分布式 *k* 最近邻搜索算法(P2PAKNNs)。阐述GHT*结构，定义高维数据相似度函数 *HDSF*(*X*,*Y*)，论述GHT*中的插入算法、范围查找算法和搜索算法。给出P2PAKNNs的实现过程，通过实验证明其正确性。

关键词： *k* 最近邻搜索算法；度量空间；相似性查询

P2P-based Self-adaptive Distributed *k*-nearest Neighbor Search Algorithm

YU Xiao-gao[1], YU Xiao-peng[2]
(1. School of Information Management, Hubei University of Economics, Wuhan 430205;
2. School of Economic Management, Wuhan Institute of Technology, Wuhan 430073)

【Abstract】 *k*-nearest Neighbor search algorithm(KNNs) can not satisfy the needs of distributing, real time performance and expansibility for data mining. Aiming at this problem, a P2P-based self-adaptive distributed KNNs(P2PAKNNs) is proposed. This paper expounds GHT* structure, and gives similarity measure function *HDSF*(*X*, *Y*). Insert algorithm, range find algorithm and search algorithm in GHT* are discussed. Implementation process of P2PAKNNs is given, and its correctness is validated by experiment.

【Key words】 *k*-nearest Neighbor search algorithm(KNNs); metric space; similarity query

1 概述

k 最近邻搜索算法(*k*-nearest Neighbor search algorithm, KNNs)在数据挖掘中具有广泛用途。例如，协同过滤技术基于最近邻技术，利用客户的历史喜好信息计算客户之间的距离，目标客户对特定商品的喜好程度由其最近邻居对商品评价的加权平均值来计算[1]。随着Internet/Intranet的发展，数据趋向以分布式存在，因此，需要改进KNNs，以适合分布式数据处理。传统KNNs的效率是 $O((nd)^2)$[2]，难以符合海量数据挖掘的要求。

实用性强的距离函数应具有对比性强、简洁、易于理解和计算等特点，而欧氏距离在很多情况下不适合作为高维空间的距离函数[3]。在高维空间中用欧氏距离函数作为距离函数时，不存在点与点之间的距离对比性。解决上述问题的方法之一是对高维空间中的距离函数进行重新定义，以降低或消除高维的影响。

2 GHT*结构

基于P2P的自适应分布式 *k* 最近邻搜索算法(P2P-based self-adaptive distributed KNNs, P2PAKNNs)先将所有高维数据插入到一棵GHT*树中，然后预测一个较小的查询范围，再采用相似性度量函数 *HDSF*(*X*,*Y*)查询当前数据对象的 *k* 个最近邻居。

GHT[4]是一种集中式度量空间索引结构，为了解决范围查询问题。文献[5]提出基于GHT的GHT*结构，它是一种分布式度量索引结构。一般而言，GHT*由端点、端点内的网络节点构成，能用相似性查询函数方便地插入、存储和检索数据。GHT*结构如下：(1)端点间用消息传递机制进行通信；(2)每个端点有唯一的网络节点标识 *NNID*；(3)每个端点以桶集 *Buckets* 的形式维护数据对象，每个桶有唯一的标识 *Bid*；(4)每个数据仅存储在一个桶中。GHT*结构的核心部分是地址查找树 *AST*。在GHT*结构中，当数据对象被存储和检索时，GHT*结构用 *AST* 来导航相应的分布式数据桶。*AST* 是一个二叉树，其内部节点存储数据路由信息，叶节点存储指向数据的指针。图1描述了一棵地址查询树 *AST*。为一个新插入数据查找存储位置时，从根节点开始，计算该数据对象与关键数据的距离。如果它与第1个关键数据的距离比与第2个关键数据的距离小，则导航到左子树，否则导航到右子树，依此递归，直至到达叶节点。

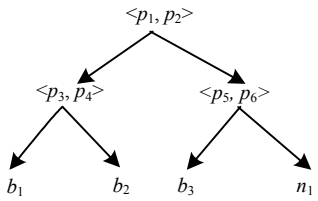

图1 地址查询树 *AST*

基金项目： 湖北省教育厅基金资助项目“群体决策的知识发现与推理模型和方法研究”(2008d095)
作者简介： 余小高(1969－)，男，副教授、博士，主研方向：数据挖掘，电子商务；余小鹏，博士后
收稿日期： 2009-04-26 **E-mail：** tecom_sam@163.com

数据对象存储在当地桶或另一个端点的桶中。*AST* 树叶节点通常有 2 种类型的指针，因此，导航到叶节点时，数据对象可以方便地存储于本地(如果找到一个 *Bid*)或异地(如果找到一个 *NNID*)。图 2 描述了一个有 3 个端点的 *AST* 结构树。从叶节点的虚线箭头表示 *NNID* 指针，实线箭头表示 *Bid* 指针。

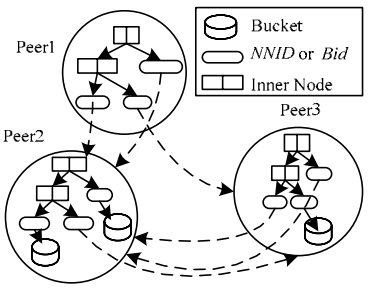

图 2　*AST* 结构树

3　高维数据的相似度函数 *HDSF*(*X*,*Y*)

对 L_k-范数度量而言，在高维空间中性能下降的主要原因之一是高维空间中点的稀疏性以及噪声的存在。在高维空间中，最相似的 2 个对象仍然会有一些不同的维，而在 L_k-距离中占主导作用的就是此类维，因此，2 个对象的相似信息被差别较大的少数几个维掩盖。相似度函数 *HDSF*(*X*,*Y*)可以较好地克服 L_k-范数等传统距离函数在高维空间中的缺点。

定义 1(相似度函数 *HDSF*(*X*,*Y*)) 设 $X=[x_1,x_2,\cdots,x_d]$和 $Y=[y_1,y_2,\cdots,y_d]$是 d 维空间中的 2 个点，相似度函数 *HDSF*(*X*,*Y*)定义为

$$HDSF(X,Y)=\sum_{i=1}^{d}\left(\frac{\sqrt{\frac{1}{1+(x_i-y_i)^2}}}{d}\right) \tag{1}$$

该函数具有以下性质：

(1)0< *HDSF*(*X*,*Y*)≤1，当且仅当 *X*=*Y* 时，*HDSF*(*X*,*Y*)=1，表示 2 个对象在 d 维空间中重合。

(2) *HDSF*(*X*,*Y*)=*HDSF*(*Y*,*X*)。

(3)它表示 2 个对象的相似程度，其值越大，2 个对象越相似。

(4)当各维的差接近无穷大时，函数值趋向于 0，2 个对象相似性最小。

可以看出，与 L_k-范数距离度量不同，该函数中占主导地位的是取值差别较小(较接近)的维。

实验结果表明，用 *HDSF*(*X*,*Y*)函数计算的相对距离差不随维数的增加而减小，与传统的欧氏距离相比，它对高维数据具有较强的适应能力。

4　GHT*中的插入和范围查找算法

建立一棵 GHT*树，将所有数据对象插入到该树中，使每个数据对象能有效地被发现。

定义 2 T_{AST}：L_{NNID} 是所有可能的 *NNID* 组成的集合，L_{BID} 是由所有可能的 *BID* 组成的集合，则数据集 *F* 所有可能的 T_{AST} 集可以根据如下规则形成：

(1)$L_{NNID}\subset T_{AST}$；

(2)$L_{BID}\subset T_{AST}$；

(3)设$<p_1,p_2>\in F\times F$, $T_l\in T_{AST}$, $T_r\in T_{AST}$，则$<<p_1,p_2>, T_l, T_r>\in T_{AST}$；

(4)集合 T_{AST} 不包含其他元素。

T_{AST} 是一棵二叉树，叶节点是 L_{BID} 或 L_{NNID} 的元素，内部节点是偶对$<p_1,p_2>$。每个内部节点有 2 个指针，一个指向左子树 T_l，另一个指向右子树 T_r。例如，给定数据对象 $p_1, p_2, p_3, p_4, p_5, p_6$，集合 *BIDs* b_1, b_2, b_3 和 *NNID* n_1，则一棵可能的 3 层地址查询树 $T(T\in T_{AST})$是$<<p_1, p_2>,<<p_3, p_4>, b_1, b_2>, <<p_5, p_6>, b_3,n_1>>$，如图 1 所示。

定义 3(偶对$<p_1, p_2>$) 假定 $\gamma(T)$是一个返回树 $T\in T_{AST}$ 根节点的函数，则被 $\gamma(T)$返回的节点是一个 *Bid* 指针或是一个 *NNID* 指针，其实质是一个偶对$<p_1, p_2>$。

定义 4(*BPATH*) 将一棵通用树 $T\in T_{AST}$ 称作 *BPATH*，其带有 n 个二进制元素$\{0,1\}$:$p=(b_1,b_2,\cdots,b_n)$。给定一个 *BPATH* p，函数 $S(T, p)$返回 p 在树 T 上所能到达的子树，如式(2)所示。

$$S(T,())=T \tag{2}$$

$$S(<<p_1,p_2>,T_l,T_r>,(b_1,b_2,\cdots,b_n))=\begin{cases}S(T_l,(b_2,b_3,\cdots,b_n)) & \text{if } b_1=0\\ S(T_r,(b_2,b_3,\cdots,b_n)) & \text{if } b_1=1\end{cases} \tag{3}$$

设 $p=(p_1, p_2,\cdots, p_n)$, $s=(s_1, s_2,\cdots, s_n)$是 2 棵 *BPATH*，连接符“ + ”定义为 $p+s=(p_1, p_2,\cdots, p_n, s_1, s_2,\cdots, s_n)$。设 $Q=\{q_1, q_2,\cdots, q_n\}$是 *BPATH* 集合，则 $p+Q=\{p+q_1, p+q_2,\cdots, p+q_n\}$。为了查找一棵 $T\in T_{AST}$ 树，需要一个算法在树中执行查询功能。为了实现此功能，本文定义遍历运算符 ψ，用于检查每一对内部节点并决定查找哪个子树。

定义 5($\psi(T, q, \rho)$) 设数据对象为 q，给定一棵 $T\in T_{AST}$ 树，对于非负实数 ρ，遍历操作 $\psi(T, q, \rho)$返回一个 *BPATH* 集合，如式(4)~式(6)所示。

$$\psi(l_{bid}, q, \rho)=\{()\} \tag{4}$$

$$\psi(l_{nnid}, q, \rho)=\{()\} \tag{5}$$

$$\psi(<<p_1,p_2>,T_l,T_r>,q,\rho)=\begin{cases}(0)+\psi(T_l,q,\rho) & \text{if } d(p_1,q)-\rho\le d(p_2,q)+\rho\\ (1)+\psi(T_r,q,\rho) & \text{if } d(p_1,q)+\rho>d(p_2,q)-\rho\\ (0)+\psi(T_l,q,\rho)\cup(1)+\psi(T_r,q,\rho) & \\ \quad \text{if } d(p_1,q)-\rho\le d(p_2,q)+\rho & \\ \quad \text{and } d(p_1,q)+\rho>d(p_2,q)-\rho & \end{cases} \tag{6}$$

如果 T 仅由一个叶节点组成，则 $\psi(T, q, \rho)$返回一个空路径 *BPATH*()。对于其他情况，该算法根据查询范围参数递归地遍历树 T，即对根节点$<p_1, p_2>$来说，如果条件 $d(p_1,q)-\rho\le d(p_2,q)+\rho$，则连接 $\psi(T_l, q, \rho)$所有的 *BPATH* 路径到简单的 *BPATH*(0)；如果 $d(p_1,q)+\rho>d(p_2,q)-\rho$，则连接到 *BPATH*(1)。当半径 ρ 等于 0 时，进行精确匹配查询或插入处理，返回一个简单的 *BPATH* 路径。

插入操作从请求插入的端点开始，用函数 $\psi(T,q,0)$遍历它的 *AST* 树。如果发现一个 *Bid* 指针，则被插入对象存储于该桶中，否则根据被发现的 *NNID* 指针所指向的端点递归地进行插入操作，直到发现带有 *Bid* 指针的 *AST* 树叶节点为止。插入算法描述如下：

```
PROCEDURE Insert(x,p)
S_p=S(T,p);
{p_1}=ψ(S_p,x,0);
n=γ(S(S_p,p_1));
IF n∈L_NNID THEN
Send a request for Insert(x,p_1) to peer with NNID n;
ENDIF
IF n∈L_BID THEN
Insert x in local bucket with BID n;
ENDIF
```

上述算法形式化地描述了插入过程，其中，x 是插入对象；p 表示 *AST* 树遍历路径，初始化为空，即 p=()。如果查找前移到另外一个端点，则参数 p 将包含已经遍历的 *BPATH*

路径传送到该端点。范围查找算法描述如下：

```
PROCEDURE RangeSearch(q,ρ,p)
S_p=S(T,p);
p=ψ(S_p,q,ρ);
FOR EACH p_i ∈ p
n=r(S(S_p,p_i));
IF n∈L_NNID THEN
Send a request for RangeSearch(q,ρ,p_i) to peer with NNID n;
ENDIF
IF n∈L_BID THEN
Search(q,ρ) in local bucket with BID n;
ENDIF
END FOR EACH
```

与插入操作类似，范围查找从查询端点开始用函数 ψ 遍历本地 *AST* 树。函数 ψ 可以很方便地找出以 q 为中心、以 ρ 为查询半径的超球内的所有数据，即 q 的 k 个最近邻居。对于在叶节点有 *NNID* 指针且符合条件的所有路径中，查询请求递归地前移到各自端点，直到在每个叶节点中出现一个 *Bid* 指针为止，该桶被置为目标对象的查找范围。

5 搜索算法

5.1 算法思路

定义 6(超球 $HB(q,\rho)$) 以 q 为中心、以 ρ 为半径的一个超球体空间。

定义 7($\rho_{effective}$) 设 N 是样本总数，超球 $HB(q,\rho)$中的样本数目为 k'。如果 $1\le k'/k, k'<<N$，则有效半径 $\rho_{effective}$ 为 ρ。

以 3D 空间为例说明 P2PAKNNs 算法的思路，具体如下：

(1)设 $q(x,y,z)$为位于点集合中的任意查询点，设 N 是样本总数，建立集合 S'的方法如下：找出所有在以 q 为中心、以 ρ_q(ρ_q 为初始值)为半径的超球 $HB(q,\ \rho_q)$内的点的集合 $S'=\{p_i|1\le i\le k'\}, q\notin S'$，$k'$为 S'中元素的数目，$k'<<N$。

(2)若 $k'\ge k$，则用穷尽法求出 S'中所有元素到 q 的欧氏距离$\{d_i|d_i\le d_j, 1\le i\le j\le k'\}$，$\{p_i|1\le i\le k\}$是 q 的 k 个最近邻居组成的集合 S，其中，p_1 是 q 的最近邻居。

(3)若 $k'<k$，则根据规则和事例确定步长$\Delta\rho$，$\rho_q=\rho_q+\Delta\rho$，扩大半径 ρ_q，依此递归调整，确定$\Delta\rho$ 并扩大 ρ_q 直至 k'大于等于 k。此时，ρ_q 是 q 的有效半径 $\rho_{effective}$，同时可以确定 $\rho^q_{\min}$。

(4)将 p_1(q 的最近邻居)作为查询点，用 $\rho^q_{\min}$的修正值 ρ' 作为 p_1 的查询超球半径的初始估计值 ρ_{init}，进行深度检索，寻找 p_1 的 k 个最近邻居。依此递归，直至搜索出所有样本的 k 个最近邻居。

5.2 ρ 的确定

$V_\rho=K_{p,d}\rho^d$ 为需要搜索的超球的体积，其中，p=2；$K_{p,d}$ 由 p 和 d 决定。因此，$K_{2,d}=\pi^{d/2}/\Gamma(d/2+1)$。

通过上述分析可知，P2PAKNNs 算法的搜索效率依赖半径 ρ。虽然传统 KNNs 算法允许逐步扩充 ρ，但若不对 ρ 进行有效预测，或不能有效指导 ρ 的扩充，就会增加不必要的搜索次数，从而增加 I/O 访问量，影响其效率。

在 P2PAKNNs 算法中，ρ 的确定分为 2 种情况：

(1)起始查询样本或半径估计不准确样本的 ρ 的确定

具体思路如下：由于 $\rho_{init}<\rho_{effective}$，因此 P2PAKNNs 算法先从 ρ 的初始值开始，用初始步长进行扩充试探，然后根据规则和事例学习经验，逐步调整步长，扩大 ρ。各变量说明如下：

k 为需要查找的邻居数目。

k'是半径为 ρ 时超球 $HB(q,\rho)$内 q 的邻居数。

q 为起始查询样本。

δ 用于衡量 k'向 k 的靠近程度$\{\delta_1,\delta_2\}$，δ_1 表示 k'离 k 较远(即 k'/k 的比值较小)，δ_2 表示 k'远大于 k；在本文中，只定义 δ_1 和 δ_2 即可，δ_1 一般定义为 0.5，δ_2 参考当前计算机性能和 k 值来确定(在本文中，当 k=20 时，δ_2 可以小于等于 5)。

$\Delta k'$为半径增长后 q 邻居的增长数。

$\Delta\rho$ 为半径的增长步长。

在上述情况下，$\Delta\rho$ 根据规则 1 和规则 2 进行扩充(ω_i 为权重)。经过具体查询取得事例后，权重参数根据规则 3 进行修正。

规则 1：

```
IF k'/k =< δ_1 THEN
  IFΔk'=0   THEN
Δρ=w_1*Δρ
   ELSE
     Δρ =w_2*Δρ
   ENDIF
    ENDIF
```

规则 2：

```
IF k'/k>δ_1 AND k'/k<1 THEN
   IFΔk'=0 THEN
      Δρ =w_3*Δρ
   ELSE
      Δρ =w_4*Δρ
ENDIF
    ENDIF
```

规则 3：

```
IF k'/k <1 THEN
w_i= w_i+1
w_i+1= w_i+1+1
ELSE IF k'/k >= δ_2 THEN
w_i= w_i-1
w_i+1= w_i+1-1
ENDIF
```

(2)深度查询样本的 ρ 确定

由于查询样本 q 与其最近邻居 p_1 并非总是连续分布的，因此不合适始终从单位距离开始扩充查询样本的超球半径。

本文提出如下假设：给定任意一个查询样本 q，其 k 个最近邻居$\{p_i|1\le I\le k\}$的超球半径取为 $\rho^q_{\min}$。若将 $\rho^q_{\min}$的修正值作为 p_1 的超球半径，则在该超球 $HB(p_1,\rho^q_{\min})$内 p_1 的邻居数目 k'接近 k 的概率较大，即概率 $p\{k'\in[k-\varepsilon_1,k+\varepsilon_2]\}$较大，$\varepsilon_2$ 可以比 ε_1 稍大，ε_1 是一个较小的数。

根据上述假设，在实际应用中可以进行如下处理：p_1 超球的初始半径定为 $\rho^q_{\min}$，若取得的邻居数 $k'<k$，则将其作为第(1)种情况来确定 p_1 的有效半径。若 $k'\ge k$，则采用穷尽法找出 p_1 的 k 个最近邻居。由于 $\rho^q_{\min}$较小，因此 k'远小于样本总数。

证明：设空间数据集为 DB，其样本总数为 N，DB 具有聚类性质，设包含的聚类为 $c_1,c_2,\cdots,c_n$。c_i 中点的个数为 m_i，$Q=x_{i0}, j=0,1,\cdots,m_{i-1}$时，从 x_{ij} 起遍历 c_i，且 $d(x_{ij}, x_{ij+1})\le d(x_{ij}, x_{ij+l})$，$l\ge 2$。覆盖住 $x_{ij}(j=0,1,\cdots,m_{i-1})$的 k 个最近邻居的超球半径为 r_{ij}。

将 r_{ij} 覆盖 r_{ij+1}，得到邻居数接近 k，即 $k-\varepsilon_1\le k_{ij+1}\le k+\varepsilon_2$。$k_{ij+1}$ 表示用 r_{ij} 覆盖 r_{ij+1} 得到邻居数。令 x_{ij+1} 的特征函数为

$$\phi_{ij+1}=\begin{cases}1 & k-\varepsilon_1\le k_{ij+1}\le k+\varepsilon_2\\ 0 & k_{ij+1}<k-\varepsilon_1 \text{或} k_{ij+1}>k+\varepsilon_2\end{cases}$$

$\sum_{j=0}^{m_{i-1}}\phi_{ij+1}$ 即频数，$\hat{p}=\frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{j=0}^{m_{i-1}}\phi_{ij+1}}{m_i-1}$ 可以作为 p 的估计。

实验中随机采样 30 次，取得估计量 $\hat{p}$ 的值如下：0.963, 0.958, 0.870, 0.879, 0.997, 0.847, 0.853, 0.985, 0.991, 0.915, 0.959, 0.919, 0.954, 0.867, 0.894, 0.946, 0.990, 0.880, 0.864, 0.908, 0.879, 0.953, 0.999, 0.890, 0.851, 0.937, 0.865, 0.967, 0.974, 0.978, 0.887, 0.829, 0.864, 0.850, 0.959, 0.880, 0.941, 0.968, 0.866, 0.947。

假设 H_0:$P'\geq 0.96$，则 H_1:$P'<0.96$。

令随机变量 $X=\begin{cases}1 & \text{若}p\text{的估计值}\geq 0.85\\ 0 & \text{若}p\text{的估计值}<0.85\end{cases}$，$X\sim B(1,p')$。取 30 个随机样本，$X_1,X_2,\cdots,X_{30}$ 都服从 $B(1,p')$且独立。根据中心极限定理有：当 n 较大时，$U=\frac{\overline{X}-p_0}{\sqrt{\frac{p_0(1-p_0)}{n}}}\approx N(0,1)$。此检验的拒绝域是 $u\leq -u_{1-\alpha}$，取 $\alpha=0.05$，则 $-u_{1-\alpha}=-1.65$。计算 $u=\frac{\frac{28}{30}-0.96}{\sqrt{\frac{0.96\times 0.04}{30}}}=-0.745$，没有落入拒绝域，接受 H_0:$P'\geq 0.96$。

因此，假设成立，证毕。

5.3 P2PAKNNs 算法

根据上述分析，P2PAKNNs 算法描述如下：

```
创建 GHT*树，存储样本数据; //用 Insert(x,p)将样本数据插入
//GHT*树中
并行初始化各本地全局样本数组 q[1…N];
Indicator[1…N]=1,i=j=1; //初始化指示器
各端点并行执行如下操作：
WHILE i≤N
{IF Indicator[j]==1 THEN
{ Indicator[j]置 0;
Neighbor[1…k]=NULL;
ρ=Estimate_Initial_Radius(q[j]);
Δρ=Estimate_Initial_Step(q[j]);
RangeSearch(q,ρ+Δρ,p);
//查找以 q 为中心、以 ρ+Δρ 为半径的邻居，路径存放在 p 里
k'=Length(p);   //求 p 的节点数目
WHILE(k'<k)
{ Δρ=Enlarge_Radius(rule);
RangeSearch(q,ρ+Δρ,p);
k'=Length(p);//求 p 的节点数目
}
Output(q[j], p);
j=Select_Depth_Search_Sample(q,Neighbor);
//深度查询 q 下一个邻居
IF j==-1 THEN j=i+1;
}
ELSE
j=i+1;
i=i+1;}
```

6 实验分析

实验所用机型为 CPU PIV 2.0，256 MB 内存，使用 Eachmovie 数据集。Eachmovie 数据集由 72 916 个数据对象组成，每个数据对象对 1 628 个项进行评价，共包含 2 811 983 个 0~5 的不同评分。为了提高实验速度，仅抽取部分数据。

图 3 比较了 *HDSF*(*X*,*Y*)与欧氏距离函数应用于 k 最近邻搜索中的效率。可以看出，2 种方法消耗的 CPU 资源随着数据维数的增加而增加，但前者的增长幅度较小，表明 *HDSF*(*X*,*Y*)函数具有更好的性能。对不同 k 值，P2PAKNNs、SR-tree KNNs、传统 KNNs 的距离计算次数变化各不相同。图 4 描述了它们的对比情况，可以看出，随着 k 值的增加，P2PAKNNs 算法的计算次数增幅最小。

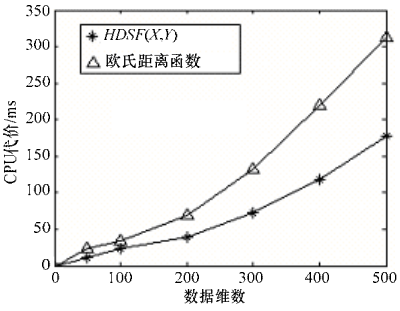

图 3 *HDSF*(*X*,*Y*)与欧氏距离函数的性能对比

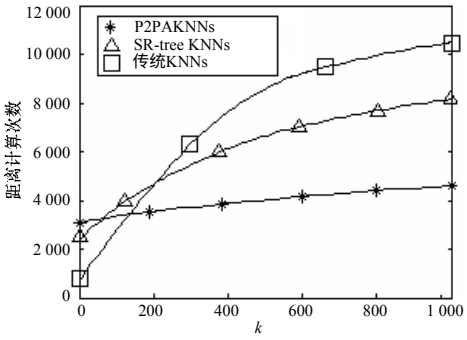

图 4 k 值对 3 种算法性能的影响

常见 k 最近邻搜索算法的性能通常随着数据集的增加而线形下降[3]，P2PAKNNs 则具有较好的可扩展性。图 5 显示了 P2PAKNNs 作用于不同大小数据集时的性能，其中，NN 表示最近邻居。可以发现，当数据集大小为 4 000 时，其距离计算次数与数据集大小为 7 000 时近似。

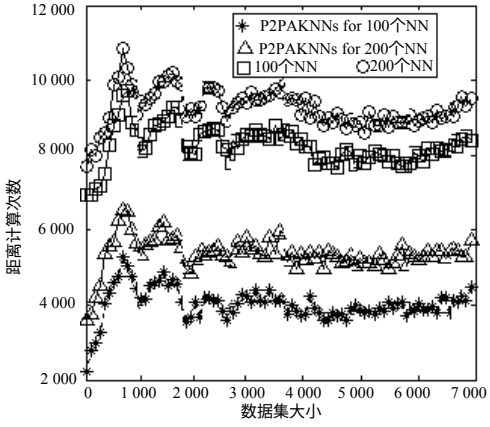

图 5 数据集大小对 P2PAKNNs 算法性能的影响

在数据集很大的情况下，由于能较好地确定样本的查询超球半径，因此可以较准确地确定搜索范围，从而减少大量 I/O 操作，提高效率。

7 结束语

针对传统 k 最近邻搜索算法的不足，本文提出 P2PAKNNs 算法。该算法将搜索范围限制在一个以查询点为中心的超球内，不必假定数据集的样本分布，无须预先设置、估算查询超球的半径范围。它能自适应估计并预测查询超球的有效半径，从而有效确定查询范围，极大提高了各数据对象的 k 个邻居的查询效率。

（下转第 55 页）