高维主存 kNN 连接索引结构的核心算法

刘 艳[1,2] 郝忠孝[1,3]

(哈尔滨理工大学计算机科学与技术学院 哈尔滨 150080)[1]

(长春大学计算机科学技术学院 长春 130022)[2]

(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)[3]

摘 要 kNN(k 最近邻)连接是高维数据库中的一种重要但代价昂贵的基本操作。随着 RAM 容量越来越大且价格逐渐低廉,更多的数据集能够被装入主存。如何实现快速主存 kNN 连接,引起人们的关注。索引 Δ-tree-R 和 Δ-tree-S 是根据 kNN 连接的特点专门为主存 kNN 连接设计的索引。结合编码、节点中心重合技术,给出了构建 Δ-tree-R 和 Δ-tree-S 的核心算法及相关证明,实验表明,基于该索引的主存 kNN 连接算法 Δ-tree-KNN-Join 明显优于目前已存在的可用于主存的 kNN 连接算法 Gorder。

关键词 kNN 连接,高维空间,主存,索引结构,kNN 搜索

中图法分类号 TP311.13 **文献标识码** A

Core Algorithm of High-dimensional Main Memory kNN-Join Index Structure

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

(College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China)[1]

(College of Computer Science Technology, Changchun University, Changchun 130022, China)[2]

(College of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)[3]

Abstract kNN-Join is an important but costly primitive operation of high-dimensional databases. As RAM gets cheaper and larger, more and more datasets can fit into the main memory, how to realize the kNN-Join efficiently brings people's interests. Δ-tree-R and Δ-tree-S were designed especially for main-memory kNN-Join according to the properties of it. The core algorithms and relevant certificates of building them were presented combining with coding and node center coincidence technologies. Experiments show that the algorithmΔ-tree-kNN-Join based on Δ-tree-R and Δ-tree-S is superior to the existing kNN-Join algorithm of Gorder that can be used in main memory.

Keywords kNN-Join, High-dimensional space, Main-memory, Index structure, kNN search

1 引言

许多新出现的数据库应用如图像、时间序列和科学数据库等,都是对高维数据的操作。在这些应用中,一个重要但非常昂贵的基元操作是 kNN 连接[1],该操作将一个数据集中的每个点和其在另一个数据集中的 kNN 连接在一起,能够比范围相似连接提供更有意义的查询结果,支持涉及多维数据的各种应用。

到目前为止,一些研究人员对 kNN 连接及 kNN 查询的有效计算进行了研究。MuX kNN-join[1]是专门为 kNN 连接设计的算法,该算法使用双层索引结构分别优化 CPU 和 I/O 代价,本质上是一种基于 R-tree 的设计用来解决 CPU 和 I/O 优化要求冲突的方法。文献[2]中提出的 Gorder 算法是一种块嵌套循环连接算法,它使用排序、连接调度和距离计算过滤来减少 CPU 和 I/O 代价。Cui 等[3]将 iDistance 作为基础的索引结构,提出一种基于索引的 kNN 连接算法 iJoin,以及它的两个加强版本 iJoinAC 和 iJoinDR。何洪辉等[4]提出的 pgi-distance 是一种并行 kNN 连接处理方法,它采用双层结构来达到 CPU 和 I/O 同时优化的目的。梁俊杰等[5]利用高维数据空间合理划分,提出一种简单有效的 kNN 检索算法 LBD。最近,Emrich 等[6]应用基于球近似的 SS-Tree 索引,引入三角修剪法(trigonometric pruning)减小搜索空间,加速 AkNN 处理(与 kNN 连接同类型的查询)。Wang 等[7]研究了极高维稀疏数据的 kNN 连接问题,提出了 3 种 kNN 连接算法:蛮力算法 BF,基于倒排索引的算法 IIB,改进的基于倒排索引的算法 IIIB。Cui 等[8]提出 kNN Join+技术,通过维护 kNN 连接表处理具有更新操作的动态高维数据的 kNN 连接查询。

现有的 kNN 连接算法主要是基于磁盘的,其着力解决 CPU 和 I/O 同时优化的问题。目前的计算机可用主存容量不断增大,将高维数据及索引结构全部装入主存成为现实,本文对文献[9]在主存索引 Δ-tree[10]的基础上提出的主存 kNN

到稿日期:2011-01-14 返修日期:2011-05-06 本文受黑龙江省自然科学基金(F200601)资助。

刘 艳(1981-),女,博士生,主要研究方向为高维相似连接和相似搜索等,E-mail:liuyan6374@yahoo.com.cn;郝忠孝(1940-),男,教授,博士生导师,主要研究方向为数据库系统与理论。

连接的索引结构 Δ-tree-R 和 Δ-tree-S 进行了扩展研究，给出了构建该索引的核心算法及相关证明，实现了使用编码、位置对应节点具有相同聚类中心等技术解决 kNN 无先验知识的难题，使得在连接过程中能够迅速缩小 kNN 的修剪距离，达到快速剪枝的目的。实验表明，基于 Δ-tree-R 和 Δ-tree-S 的主存 kNN 连接算法 Δ-tree-KNN-Join[9] 具有较优的连接效率。

2 Δ-tree-R 的核心算法 R_insertR

Δ-tree-R 和 Δ-tree-S 的建树算法 BuildTreeR 和 BuildTreeS 在文献[9]中已初步给出，本文作为文献[9]的补充，将在本节和第 3 节中分别给出它们的核心算法 R_insertR 和 R_insertS。

算法 R_insertR 的符号说明：*nodeR* 为 Δ-tree-R 中的节点，pC 为 PCA 空间中的点集，*lev* 为 *nodeR* 在 Δ-tree-R 中所处的层数，*code*[]和 *codeLen* 为 *nodeR* 的编码及编码长度；Δ-tree-R'为以 *nodeR* 为根的 Δ-tree-R 的子树，*QueueLeaf* 为装叶子节点的队列，*QueueBigLeaf* 为装大叶子节点的队列。

算法 1 R_insertR(决定 *lev* 层上 *nodeR* 中各项的内容)

输入：*nodeR*，pC，*lev*，*code*[]，*codeLen*；

输出：Δ-tree-R'，*QueueLeaf*，*QueueBigLeaf*；

begin

(1) *QueueLeaf*=NewPriorityQueue()；

(2) *QueueBigLeaf*=NewPriorityQueue()；

(3) *pClusters*=LevCluster(pC, f, m_{lev})；

(4) for 每一个子聚类 $pC_j \in pClusters$ do

nodeR. *center*[j]=$pCenter_j$；

nodeR. *radius*[j]=$pRadius_j$；

if sizeof(pC_j) < f then

// sizeof(pC_j)表示子聚类 pC_j 中点的数目

leafR=New(*leaf*)；

//构建新的叶子节点 *leafR*

leafR=Insert_Leaf(pC_j)；

//将 pC_j 中的点装入叶子节点 *leafR*；

for 变量 i 从 0 到 *codeLen*−1 do

leafR. *code*[i]=*code*[i]；

//*leafR*. *code*[]表示 *leafR* 的编码

leafR. *code*[*codeLen*]=j；

leafR. *codeNum*=*codeLen*+1；

//*leafR*. *codeNum* 表示 *leafR* 的编码长度

nodeR. *children*[j]=*leafR*；

//将 *leafR* 作为 *nodeR* 的第 j 个孩子节点

Enqueue(*QueueLeaf*, *leafR*)；

else if sizeof(pC_j)=sizeof(pC)||*lev*+1=L then

bigLeafR=New(*bigLeaf*)；

//构建新的大叶子节点 *bigLeafR*；

bigLeafR=Insert_BigLeaf(pC_j)；

//将 pC_j 中的点装入大叶子节点 *bigLeafR*；

for 变量 i 从 0 到 *codeLen*−1 do

bigLeafR. *code*[i]=*code*[i]；

//*bigLeafR*. *code*[]表示 *bigLeafR* 的编码

bigLeafR. *code*[*codeLen*]=j；

bigLeafR. *codeNum*= *codeLen*+1；

//*bigLeafR*. *codeNum* 表示 *bigLeafR* 的编码长度

nodeR. *children*[j]=*bigLeafR*；

Enqueue(*QueueBigLeaf*, *bigLeafR*)；

else //sizeof(pC_j)≠sizeof(pC) && *lev*+1 < L

interNodeR=New(*inter_node*)；

//构建新的内部节点 *interNodeR*；

for 变量 i 从 0 到 *codeLen*−1 do

interNodeR. *code*[i]=*code*[i]；

//*interNodeR*. *code*[]表示 *interNodeR* 的编码

interNodeR. *code*[*codeLen*]=j；

interNodeR. *codeNum*=*codeLen*+1；

//*interNodeR*. *codeNum* 表示 *interNodeR* 的编码长度

nodeR. *children*[j]=*interNodeR*；

以 *interNodeR* 为根的子树=R_insertR(*interNodeR*, pC_j, *lev*+1, *code*, *codeLen*+1)；

(5) return(Δ-tree-R')；

end

定理 1 算法 R_insertR 正确地决定了 Δ-tree-R 中 *lev* 层上 *nodeR* 中各项的内容，递归调用该算法能够生成以 *nodeR* 为根、各节点具有编码、高度不超过指定层数的 Δ-tree-R'，且该算法是可终止的，其时间复杂度为 $O(f^{L-lev+2}\, d\, \frac{(L+lev-1)(L-lev)}{4L} + \frac{f-f^{L-lev+1}}{1-f} + f^{L-lev+1} + (L-lev)\, f^{L-lev})$，其中，$d$ 为数据点的维数，*lev* 为 *nodeR* 在 Δ-tree-R 中所在层数，f 是 Δ-tree-R 中节点的扇出，L 为 Δ-tree-R 的层数。

证明(正确性)：算法 R_insertR 的执行步骤(1)，(2)分别初始化装叶子节点和大叶子节点的队列；执行步骤(3)，使用 m_{lev} 维将聚类划分为 f 个子聚类(m_{lev} 维为 Δ-tree-R 中第 *lev* 层所使用的维数)；执行步骤(4)，将每个子聚类的中心和半径信息存入节点 *nodeR* 的各孩子节点对应的项中，根据 Δ-tree-R 中各类节点的生成条件对每个子聚类进行判断。若子聚类中数据点的数目小于节点扇出，则构建新的叶子节点，将子聚类中包含的数据点装入该叶子节点，为该叶子节点编码，并延长编码长度，将该叶子节点作为 *nodeR* 的孩子节点；当聚类算法不起作用或 *lev*+1 已经达到 Δ-tree-R 的层数 L 时，将子聚类作为大叶子节点，生成过程及编码方法与叶子节点类似；否则，构建内部节点，对其进行编码，延长编码长度，将该内部节点作为 *nodeR* 的孩子节点，继续调用算法 R_insertR 决定该内部节点中的各项内容。当 *nodeR* 中没有满足聚类分解条件的子孙节点时停止递归，最终生成一棵以节点 *nodeR* 为根、各节点具有编码、高度不超过指定层数的 Δ-tree-R'，故该算法是正确的。

(可终止性)：执行步骤(3)中使用 k 均值法进行聚类分解，是可终止的；执行步骤(4)中的双重 for 循环可自动终止，因为 Δ-tree-R 的层数是固定的，算法 R_insertR 被调用的次数是有限的，故算法 R_insertR 是可终止的。

(时间复杂度分析)：该算法执行步骤(3)中的算法 LevCluster 的时间复杂度为 k 均值法的时间复杂度 O(sizeof(pC)×f×m_{lev})；设 Δ-tree-R 第 i 层中使用的维数为 $\frac{id}{2L}$，则算法 R_insertR 从第 *lev* 层递归到 L 层，执行步骤(3)中算法 LevCluster 的时间复杂度为 $O(\frac{(L+lev-1)(L-lev)}{4L} d f^{L-lev+2})$。执行步骤(4)中 for 循环的时间复杂度为 $O(\frac{f-f^{L-lev+1}}{1-f})$，数据点装入叶子节点(或大叶子节点)的时间复杂度为 $O(f^{L-lev+1})$，对以 *nodeR* 节点为根的子树中所有节点进行编码的时间复杂度为 $O((L-lev)\, f^{L-lev})$。故算法 R_insertR 的时间复杂度为 $O(f^{L-lev+2}\, d\, \frac{(L+lev-1)(L-lev)}{4L} +$

$\frac{f-f^{L-lev+1}}{1-f}+f^{L-lev+1}+(L-lev)f^{L-lev}$)。证毕。

3 Δ-tree-S 的核心算法 R_insertS

算法 R_insertS 的符号说明：*nodeS* 为 Δ-tree-*S* 中的节点，*pC* 为 PCA 空间中的点集，*lev* 为 *nodeS* 在 Δ-tree-*S* 中所处的层数，Δ-tree-*S*′为以 *nodeS* 为根的 Δ-tree-*S* 的子树，*nodeR* 为 Δ-tree-*R* 中与 *nodeS* 位置对应的节点。

算法 2 R_insertS(决定 *lev* 层上的 *nodeS* 中各项的内容)

输入：*nodeS*，*pC*，*lev*，*nodeR*；
输出：Δ-tree-*S*′；
begin
(1) *pClusters*=LevClusterS(*pC*, *f*, m_{lev}, *nodeR*)；//m_{lev}维为 Δ-tree-*S* 中第 *lev* 层所使用的维数
(2) for 每一个子聚类 $pC_j \in pClusters$ do
　　nodeS. *center*[*j*]=*pCenter*$_j$；
　　nodeS. *radius*[*j*]=*pRadius*$_j$；
　　if sizeof(pC_j) < *f* then
　　　leafS=New(*leaf*)；
　　　leafS=Insert_Leaf(pC_j)；
　　　nodeS. *children*[*j*]=*leafS*；
　　else if *nodeR*. *children*[*j*]不为内部节点 then
　　　bigLeafS=New(*bigLeaf*)；
　　　bigLeafS=Insert_BigLeaf(pC_j)；
　　　nodeS. *children*[*j*]=*bigLeafS*；
　　else // *nodeR*. *children*[*j*]为内部节点
　　　interNodeS=New(*inter_node*)；
　　　nodeS. *children*[*j*]=*interNodeS*；
　　　以 *interNodeS* 为根的子树=R_insertS(*interNodeS*, pC_j, *lev*+1, *nodeR*. *children*[*j*])；
(3) return(Δ-tree-*S*′)；
end

定理 2 算法 R_insertS 正确地决定了 Δ-tree-*S* 中 *lev* 层上 *nodeS* 中各项的内容，该算法的递归调用能够生成以 *nodeS* 为根且各层节点与 Δ-tree-*R* 中位置对应节点具有相同聚类中心的 Δ-tree-*S*′，时间复杂度为 $O(f^{L-lev-1}\,d\frac{(L+lev-1)(L-lev)}{4L}+\frac{f-f^{L-lev+1}}{1-f})+f^{L-lev+1})$，其中 *d* 为数据点的维数，*lev* 为 *nodeS* 在 Δ-tree-S 中所在层数，*f* 是 Δ-tree-*S* 中节点的扇出，*L* 为 Δ-tree-*S* 层数。

证明(正确性)：算法 R_insertS 的执行步骤(1)，以 *nodeR* 中各孩子节点的中心为聚类中心，使用 m_{lev} 维将聚类划分为 *f* 个子聚类。执行步骤(2)，将每个子聚类的中心和半径信息存入 *nodeS* 中各孩子节点对应的项中，并根据 Δ-tree-S 中节点的生成条件对各子聚类进行判断，若子聚类中包含的数据点的数目小于叶子节点扇出，则创建新的叶子节点，将子聚类中包含的数据点装入该叶子节点时，且该叶子节点作为 *nodeS* 的孩子节点；若子聚类中包含的数据点的数目大于叶子节点扇出，当 *nodeR* 中该位置对应节点不为内部节点时，则将子聚类作为大叶子节点，具体处理方法与生成叶子节点类似，当 *nodeR* 中该位置对应节点为内部节点时，则创建新的内部节点，继续调用算法 R_insertS 为 *nodeS* 中内部节点的子孙节点生成孩子节点，并最终生成一棵以 *nodeS* 为根且各层节点与 Δ-tree-*R* 中位置对应节点具有相同聚类中心的 Δ-tree-*S*′。

(可终止性)：执行步骤(1)是可终止的；执行步骤(2)中的 for 循环可自动终止，由于 Δ-tree-*R* 中内部节点是有限的，因此该算法被递归调用的次数是有限的。故算法可自动终止。

(时间复杂度分析)：设 Δ-tree-S 第 *i* 层中使用的维数为 $\frac{id}{2L}$，则算法 R_insertS 从第 *lev* 层递归到 *L* 层步骤(1)中算法 LevClusterS 的时间复杂度为 $O(\sum_{i=lev}^{L-1} f^{i-lev}(f^{Li+1} f\frac{id}{2L}))$，整理得 $O(\frac{(L+lev-1)(L-lev)}{4L}df^{L-lev+2})$；执行步骤(2)中 for 循环的时间复杂度为 $O(\frac{f-f^{L-lev+1}}{1-f})$，数据点的信息装入叶子节点(或大叶子节点)的时间复杂度为 $O(f^{L-lev+1})$。故该算法的时间复杂度为 $O(\frac{(L+lev-1)(L-lev)}{4L}df^{L-lev+2}+\frac{f-f^{L-lev+1}}{1-f}+f^{L-lev+1})$。证毕。

使用算法 R_insertS 和 R_insertS 构建的 Δ-tree-*R* 和 Δ-tree-*S* 具有以下特性。

(1) 构建 Δ-tree-*S* 的过程中，对每层进行聚类分解时，使用与 Δ-tree-*R* 中位置对应节点相同的聚类中心进行聚类，从而使 Δ-tree-*R* 和 Δ-tree-*S* 中位置对应节点尽可能重合，能够在对应节点中找到大部分 *k*NN 连接匹配。

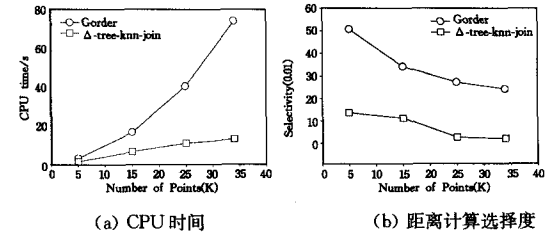
(2) 为了能够快速定位 Δ-tree-*R* 中节点在 Δ-tree-S 中位置对应的节点，在构建 Δ-tree-*R* 时对生成的每个节点进行编码，在连接过程中先对位置对应节点进行连接，从而能够尽快缩小 *k*NN 的修剪距离。

(3) 为了省略连接过程中对 Δ-tree-*R* 的遍历过程，将 Δ-tree-*R* 建树过程中生成的叶子节点(或大叶子节点)存储在队列中。

4 实验设计与性能评价

本实验在真实和合成数据上对基于 Δ-tree-*R* 和 Δ-tree-*S* 的连接算法 Δ-tree-KNN-Join 和算法 Gorder 进行了比较。实验的硬件环境为 AMD 4000+ 处理器、512×2kB 二级缓存和 2GB 内存，算法用 C++实现、VC 6.0 编译。

真实数据集来自 UCI KDD+ 数据仓库，使用 Corel dataset 中的 32 维数据 ColorHistogram。实验结果如图 1 所示，数据集的大小从 5000 变化到 34000，Δ-tree-KNN-Join 比 Gorder 的 CPU 代价提高 2.0～5.6 倍，距离选择度提高 3.7～11.8 倍。



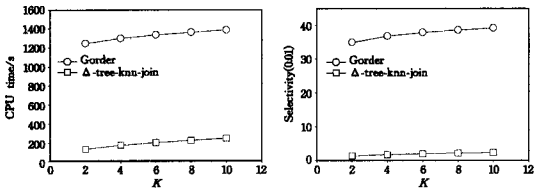
(a) CPU 时间　　(b) 距离计算选择度

图 1　32 维真实数据集上进行 *k*NN 连接

图 2 中显示在大小为 100000 的 16 维合成聚类数据上执行 *k*NN 连接，最近邻的数目 *k* 从 2 变化到 10，Δ-tree-KNN-Join

+ http://kdd.ics.uci.edu/

的 CPU 性能提高 5.9～9.3 倍，距离计算选择度优化 17.2～26.9 倍。



(a) CPU 时间　　(b) 距离计算选择度

图 2　16 维真实数据集上进行 kNN 连接

Goder 采用无索引块嵌套循环连接技术，为寻找 kNN 需要反复对数据块进行排序和扫描，CPU 消耗相当可观；Δ-tree-KNN-Join 的索引结构 Δ-tree-R 和 Δ-tree-S 的核心算法使用编码和节点聚类中心重合等技术，使得该算法在 kNN 连接过程中能够快速定位位置对应节点，并迅速找到较优的 kNN 候选、快速缩小 kNN 修剪距离，从而提高剪枝能力、大幅减少计算量。Δ-tree-KNN-Join 进行连接时使用较少的维数计算节点之间的距离，所以 CPU 性能较好。

结束语　本文针对 kNN-Join 的特征，给出了构建主存 kNN 连接索引结构的核心算法，实现了编码、定位技术，使得基于该索引结构的连接算法快速定位距离查询点较近的被查询节点，从而提高连接效率。通过实验对比分析了索引 Δ-tree-R 和 Δ-tree-S 的有效性。在后续工作中将继续研究使用较优的降维技术为 kNN 和 RkNN 连接设计索引及查询算法。

参 考 文 献

[1] Böhm C, Krebs F. The k-nearest neighbor join: turbo charging the kdd process [J]. Knowledge and Information Systems (KAIS), 2004, 6(6): 728-749

[2] Xia C, Lu J, Ooi B C, et al. GORDER: An efficient method for KNN join processing[C]// Proceedings of the 30th International Conference on Very Large Data Bases(VLDB'04). San Francisco, CA: Morgan Kaufmann, 2004: 756-767

[3] Cui Y, Cui B, Wang S, et al. Efficient index-based KNN join processing for high-dimensional data[J]. Information and Software Technology, 2007, 49(4): 332-344

[4] 何洪辉，王丽珍，周丽华. pgi-distance：一种高效的并行 KNN-join 处理方法[J]. 计算机研究与发展，2007，44(10)：1774-1781

[5] 梁俊杰，冯玉才. LBD：基于局部位码比较的高维空间 KNN 搜索算法[J]. 计算机科学，2007，6(34)：145-159

[6] Emrich T, Graf F, Kriegel H-P, et al. Optimizing All-Nearest-Neighbor Queries with Trigonometric Pruning [C]// Proceedings of the 22nd International Conference on Scientific and Statistical Database Management(SSDBM). Heidelberg, Germany: Springer, 2010: 501-518

[7] Wang J J, Lin L, Huang T, et al. Effcient K-Nearest Neighbor Join Algorithms for High Dimensional Sparse Data. Computing Research Repository(CoRR)[R]. cs. DB/1011. 2807. 2010-11

[8] Yu C, Zhang R, Huang Y, et al. High-dimensional k NN Joins with Incremental Updates[J]. GeoInformatica, 2010, 14(1): 55-82

[9] 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

[10] Cui B, Ooi B C, Su J W, et al. Indexing high-dimensional data for efficient in-memory similarity search[J]. IEEE Trans on Knowledge and Data Engineering, 2005, 17(3): 339-353

(上接第 134 页)

$(Loc_5, v) \xrightarrow{reply} (Loc_6, v[\lambda:=0])$迁移上的动作 *reply* 触发；当 $z \leqslant 10$ 且 $t \geqslant 3$ 时，状态 5 迁移到状态 8(*reply* 延迟状态)，使得$(Loc_5, (v_z, v_t)) \xrightarrow{reply} (Loc_8, (v_z[\lambda:=0], v_t[\lambda:=0]))$迁移上的 *reply* 动作触发；另外状态 7 表示发送成功。

结束语　Web 应用程序抽取模型可用于形式化验证。如何抽取基于时间约束的安全模型，也是一个挑战性研究课题。本文应用逆向工程的思想，通过解析带有时间约束的 Web 应用的 XML 源代码，提取和生成时间自动机模型，并且时间自动机模型是可并行组合的。最后通过一个实例阐述了时间约束的抽取过程，而抽取出的模型方便于后续的模型检验以及性质抽取。

本方法可以有效提取 Web 应用的 XML 文档中的时间约束等信息，而且可以通过形式化验证工具 UPPAAL 验证生成的时间自动机模型的正确性。

参 考 文 献

[1] For technical reports, work drafts, recommendations, and specifications of XML and related technologies [EB/OL]. http://www.w3.org

[2] 邓小鹏，邢春晓，蔡莲红. Web 应用测试技术进展[J]. 计算机研究与发展，2007，44(8)：1273-1283

[3] Ricca F, Tonella P. Web site analysis: structure and evolution [C]// International Conference on Software Maintenance. California, 2000: 76-86

[4] Tramontant P. Reverse engineering Web applications[C]// Proceedings of the 21st IEEE International Conference on Software Maintenance. Budapest, 2005: 705-708

[5] Di Lucca G A, Fasolino A R, Pace F, et al. Ware: a tool for the reverse engineering of Web applications [C]// Proceedings of the Sixth European Conference on Software Maintenance and Reengineering. Budapest, 2002: 241-250

[6] 苏杭，严建援. 一种新的 Web 链接提取模型[J]. 清华大学学报，2006，46(1)：975-982

[7] 胡蓉，缪淮扣，刘焕洲. 一种基于 Web 软件集成测试的建模方法[J]. 计算机科学，2007，34(6)：253-257

[8] Tsang T. Using XML-based Real-time Model for Distributed Real-time Multimedia Systems[C]// the 9th IEEE International Conference on Networks (IEEE-ICON2001). Bangkok, Thailand, 10-12October 2001

[9] 方明科，缪淮扣. 一种用于模型验证的 Web 应用模型抽取方法[J]. 应用科学学报，27(1)：90-96

[10] Aitken P G. XML—The Microsoft Way [M]. 谢君英，译. 北京：中国电力出版社，2003. 146-155

[11] Diaz G. Automatic Translation of WS-CDL Choreographies to Timed Automata[M]. Springer Berlin/Heidelberg, 2005: 230-242

[12] Kung D, Liu CH, Hsia P. An object-oriented Web test model for testing Web applications[C]// Proc. of IEEE 24th Annual International Computer Software and Applications Conference (COMPSAC2000). Taipei, 2000: 537-542

[13] Alur R, Dill D L. A theory of timed automata[J]. Theor. Comput. Sci., 1994, 126: 183-235

[14] Alur A R, Dill D. Automata for modeling real-time systems[C]// Proceedings of the 17th International Colloquium on Automata, Languages and Programming. Warwick University, England, 1990: 322-335