第 36 卷 第 4 期 计 算 机 工 程 2010 年 2 月
Vol.36 No.4 Computer Engineering February 2010
·软件技术与数据库· 文章编号：1000—3428(2010)04—0047—03 文献标识码：A 中图分类号：TP311

基于 Buddy*-Hash 的移动对象时空查询方法

刘 玥，郝忠孝
(哈尔滨理工大学计算机科学与技术学院，哈尔滨 150080)

摘 要：索引技术可以提高数据检索和查询效率，为了实现对时空数据库中移动对象的查询操作，需要引入时空索引技术。在传统 Buddy-树的基础上提出 Buddy*-Hash 索引结构，根据扩展查询窗口策略给出范围查询算法。实验结果表明，基于 BH 索引结构的范围查询算法具有良好性能。

关键词：移动对象；索引结构；Hash 辅助索引表；范围查询

Spatio-temporal Query Method of Moving Object Based on Buddy*-Hash

LIU Yue, HAO Zhong-xiao
(College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080)

【Abstract】 Index technique can improve the efficiency of data searching and query. In order to realize query operation of moving object in spatio-temporal database, the spatio-temporal index technique should be introduced. This paper proposes the Buddy*-Hash index structure based on Buddy-tree. The range query algorithm is given according to the expansion query window strategy. Experimental results indicate that the range query algorithm based on BH index structure has good performance.

【Key words】 moving object; index structure; Hash auxiliary index table; range query

1 概述

在时空数据库中，移动对象不断产生大量时空信息，要保持最新的信息必须及时更新数据，因此，其数据量很大。在时空数据库中如何有效管理此类动态信息一直被广泛关注。

目前，空间索引结构的构造方式[1]主要有 2 种：

(1)基于空间分割的索引结构，将多维空间分割成细小单元，如四叉索引；

(2)基于实体对象的索引结构，如 R-树索引。

现有时空索引方法主要考虑对空间索引结构进行扩展。传统空间索引方法用来快速检索静态对象，而在移动对象数据库等应用中，移动对象的频繁改变将引起索引结构的动态变化，因此，移动对象索引技术需要考虑查询效率和更新代价。

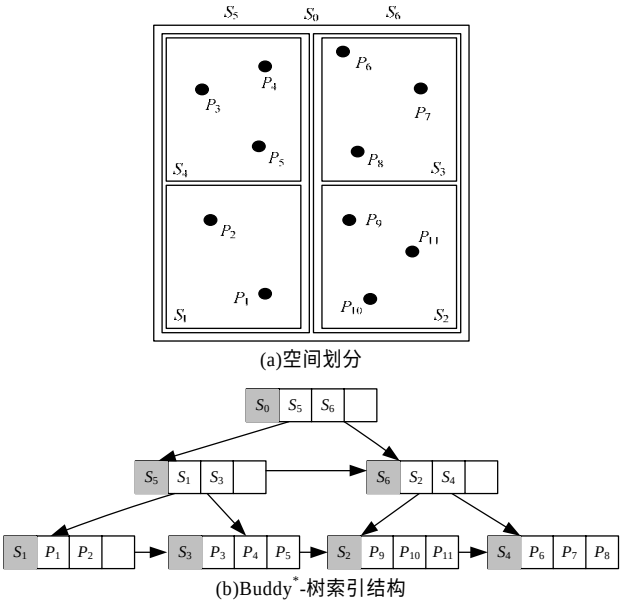
Buddy-树[2]是属于基于空间分割的时空索引结构，并兼具 R-树索引结构的特点。本文对 Buddy-树进行改进，并在此基础上提出范围查询算法。

2 索引结构

2.1 Buddy-树

Buddy-树是建立在对空间区域循环分解原则之上的一种索引结构，可以看作是 R-树和网格文件[3]的一种折中方法。Buddy-树可用于索引移动对象并支持各种查询操作，如范围查询、连接查询、最近邻查询等。

文献[4]提出 Buddy*-树，其主要思想是在原有 Buddy-树的索引结构上用指针将处于同一层次上的节点联接起来，并用 *LBS*(Loose Bounding Space)替代非叶节点的外包矩形，以提高其并行操作的性能。图 1 给出了其在二维空间的索引结构。



(a)空间划分

(b)Buddy*-树索引结构

图 1 Buddy*-树在二维空间的索引结构

2.2 BH 索引结构

本文在传统 Buddy-树的基础上提出一种新的索引结构。该索引结构通过引入 Hash 辅助索引表直接检索叶节点，极大降低了查询操作时间，从而提高了查询性能。将此索引结构记为 BH(Buddy*-Hash)索引结构。

基金项目：黑龙江省自然科学基金资助项目(F200601)
作者简介：刘 玥(1981－)，女，硕士研究生，主研方向：时空数据库；郝忠孝，教授、博士生导师
收稿日期：2009-07-29 **E-mail：**arrow626@163.com

BH 结构由 2 部分组成：(1)对移动对象位置进行索引的树形结构 Buddy*-树；(2)用来简化更新过程的 Hash 辅助索引表。

2.2.1 Buddy*-树

定义 1(Buddy*-树叶节点) Buddy*-树结构中叶节点形式为

(T, oid, $data$, p_{next})

其中，T 表示移动对象存储的时间信息；oid 为移动对象标示符；$data$ 为数据记录；p_{next} 指向在叶子节点层与其相邻的节点。

定义 2(Buddy*-树非叶节点) Buddy*-树结构中非叶子节点形式为

(LBS, p_{child}, p_{next})

其中，LBS 为包含其对应孩子节点的外包矩形；p_{child} 是指向该节点孩子节点的指针；p_{next} 指向同一层次上与其相邻的节点。

特别地，Buddy*-树结构中根节点的形式为

(LBS, p_{child1}, p_{child2}, …, p_{childN})

2.2.2 Hash 辅助索引表

在 Buddy-树中进行时空检索时，随着索引目标数量的增多，树的深度逐渐增大，而查找依旧从根节点出发，其查找性能随着空间数据量的增加而急剧下降。因此，本文增加建立在叶节点之上的 Hash 辅助索引表来支持对象的更新操作。

定义 3(Hash 辅助索引表) BH 结构中 Hash 辅助索引表中每个单元的形式为

(oid, ptr)

其中，oid 表示移动对象的标识符；ptr 表示指向移动对象最近一次发生位置更新时所处的 Buddy*-树叶节点的位置。

图 2 是 BH 结构中 Buddy*-树结构和 Hash 辅助索引表结构的示例。当在 Buddy*-树中插入一个新对象时，如果相应的叶节点已满则需进行节点分裂生成新的叶子节点，需要向 Hash 辅助索引表中插入一条记录(oid, ptr)。

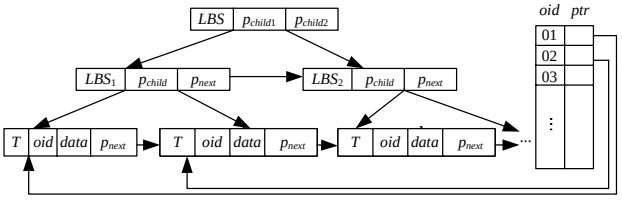


图 2 BH 索引结构

在 Buddy*-树中删除一个移动对象时，先通过 Hash 辅助索引表找到该对象所在的叶节点，然后将该对象对应的叶节点删除，若节点删除后其父节点没有子节点则对其父节点采取相应的合并操作，同时将该对象在 Hash 辅助索引表中的记录删除。

当移动对象位置发生改变时，只需要在叶子节点层通过 Hash 辅助索引表找到该对象所在叶节点，然后将该对象删除，将其新位置索引信息插入新的叶节点中，并修改 Hash 表中该对象对应记录的指针使其指向新的叶节点。

可见，由于引入了 Hash 辅助索引表，移动对象的插入、删除和移动都是发生在叶子节点层的局部更新，因此，极大提高了查询效率。

3 基于 BH 结构的移动对象范围查询算法

基于 BH 索引结构，本节在扩展查询窗口策略的基础上给出了移动对象范围查询算法。

3.1 查询窗口扩展

扩展查询窗口的基本思想是：由于处理的是移动对象，当移动对象位置发生变化时，与其对应的目录页的外包矩形也会发生变化，因此对查询窗口进行扩展，将外包矩形的变化转变为查询过程中查询窗口的变化。

定义 4(查询窗口扩展) 令查询 q 的查询窗口为[qx_i^l, qx_i^u](i =0,1,…, d −1， d 是空间的维数)，查询时间为 t_q，则扩展的查询窗口[eqx_i^l , eqx_i^u]可表示为

$$eqx_i^l = \begin{cases} qx_i^l + u_i^l \cdot (t_{ref} - t_q) & \text{if } t_q < t_{ref} \\ qx_i^l + (-u_i^u) \cdot (t_q - t_{ref}) & \text{otherwise} \end{cases}$$

$$eqx_i^u = \begin{cases} qx_i^u + u_i^u \cdot (t_{ref} - t_q) & \text{if } t_q < t_{ref} \\ qx_i^u + (-u_i^l) \cdot (t_q - t_{ref}) & \text{otherwise} \end{cases}$$

其中，u_i^l , u_i^u 是 i 维空间中移动对象在查询窗口中的最小和最大速度；t_{ref} 是标准时刻。

3.2 范围查询算法

定义 5(移动对象范围查询) 给定一个查询窗口 q，找出在指定时刻 t 包含在查询窗口内的移动对象。

基于 BH 索引结构和扩展查询窗口策略，进行范围查询的算法描述如下：

Range_query(q ,N, l)
输入 查询范围 q，查询对象节点集 N，父节点的 LBS 指针 l
输出 查询结果集 *result.set*
begin
将查询窗口 q 扩展为 q'；
for N 中 LBS 与 q' 有交迭的对象 e do
  if e 是非叶节点 then
    (N' , l')=(e.node , e.LBS)；
    call Range_query(q , N' , l')；
  else 将 e 加入到数组 L 中；
if N.LBS 不等于 l then
通过每一层节电之间的指针从 N 开始查找，找到第 1 个 LBS 不包含于 l 的节点 M；
for M do
l' :=M.LBS；
call Range_query(q ,M, l')；
  for L 中的每一个对象 o do
    if t 时刻在 q 中 then
      将对象 o 加入到结果集 result.set 中；
  return result.set；
  end

定理 算法 *Range_query* ()是可终止的，并能正确求出范围查询的结果。

证明：(1)可终止性。因为在给定时刻 t，在给定范围内仅有有限查询对象满足要求，且在算法中只包含 for 循环语句，所以算法是可终止的。(2)正确性。在时刻 t 进行范围查询，首先对查询范围 q 进行扩展，然后在扩展的查询范围 q' 内找出符合查询条件的对象 e 并暂存入数组 L 中，在确定该对象为最终结果后将其放入结果集 *result.set* 进行输出。在查询过程中对其非叶节点先判断是否在扩展后的查询范围 q' 内，将符合条件的节点的(*e.node* , *e.LBS*)赋给(N' , l')进行递归。每次调用 *Range-query*()，N 就被指向当前节点的孩子节点，因此，递归调用会在叶子节点处停止，并将结果放入 L 中。当 *N.LBS* 与 l 代表的 LBS 不相等时，通过链表依次对其他分支进行查找，若查找到符合条件的节点，则进行相应递归处理。该算法由于采用了扩展查询范围的策略，对其原来的查询范围进行了扩展，因此不会丢失查询结果。且通过对范围的限制，充分保证了其结果的有效性和正确性。证毕。

时间复杂度分析如下：该算法是一个递归算法，其查询是按由根到叶子的方向进行的，因此，节点数目是逐层递增的。对于深度为 h、度为 c 的 Buddy*-树而言，其每一层节点

数至多为 c^{i-1},$1\leq i\leq h$，且每层节点的最大数目不会超过叶子节点层的最大数目。设叶子节点层的最大节点数目为 n，则 $n=c^{h-1}$。算法中共有 3 个循环步骤，第 1 个步骤的时间复杂度是 $O(hn)$，第 2 个步骤与第 1 个类同，在第 3 个循环步骤中的实体个数不会超过 n 个。综上所述，算法的时间复杂度为 $O(hn)$。

4 性能分析

为了验证 BH 索引结构和查询算法的性能，本文采用 TPR-树作为对比实验。实验数据是通过时空数据产生器 GSTD[5]生成的虚拟数据。数据集大小为 50 KB，树形索引结构每个节点的大小与一个页面大小相等，为 2 KB，最大更新时间间隔为 120 时间戳。将查询窗口从 10~100 进行划分。实验目的是通过改变查询窗口的大小来比较基于 BH 索引结构和基于 TPR-树索引结构的查询算法的性能。如图 3 所示，查询代价随着查询窗口的增大而增加，原因是较大的查询窗口包含更多移动对象，查询时要访问更多索引节点。TPR-树的查询性能下降幅度比基于 BH 索引结构的大，原因是 TPR-树存在 MBR 重叠问题，而在 BH 索引中采用扩大查询窗口的策略解决了 MBR 的扩展问题。

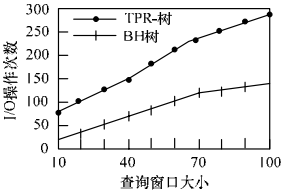


图 3 查询性能对比

5 结束语

本文提出一种基于 BH 索引结构的移动对象时空查询方法，通过引入 Hash 辅助索引表，极大提高了系统的更新性能。提出扩展查询窗口的策略，解决了节点重叠问题。

参考文献

[1] Abdelguerfi M, Givaudan J, Shaw K, et al. The 2-3 TR-tree: A Trajectory-oriented Index Structure for Fully Evolving Valid-time Spatio-temporal Datasets[C]//Proceedings of the ACM Workshop on ADV in Geographic Information System. Virginia, USA: [s. n.], 2002: 29-34.

[2] Seeger B, Krieger H P. The Buddy-tree: An Efficient and Robust Access Method for Spatial Data Base Systems[C]//Proceedings of the 16th VLDB Conference. Brisbane, Australia: [s. n.], 1990: 590-601.

[3] Nievergelt J, Hinterberger H, Sevcik K C. The Grid File: An Adaptable, Symmetric Multikey File Structure[J]. ACM TODS, 1984, 9(1): 38-71.

[4] Guo Shuqiao, Huang Zhiyong, Jagadish H V, et al. Relaxed Space Bounding for Moving Objects: A Case for the Buddy Tree[J]. SIGMOD Record, 2006, 35(4): 24-29.

[5] Theodoridis Y, Silva R, Nascimento M. On the Generation of Spatiotemporal Datasets[C]//Proc. of the 6th Int'l Symp. on Spatial Databases. Hong Kong, China: [s. n.], 1999: 147-164.

编辑 陈 晖

(上接第 46 页)

根据运行后所得数据及分析图总结如下：

(1)属性相似度值整体偏高，结构相似度分布较均匀。虽然 Halstead 属性对于那些规模相近的程序度量值偏高，影响了属性度量的精度，但结合结构度量，可以实现优势互补。在本测试中在属性相似度偏高的情况下，结合结构相似度可以做出正确的度量，提高度量的精度和效率。

(2)由于 Halstead 属性相似度偏高，考察 Halstead 属性的前 15 个相似度值(≥0.999 999 7)，有 7 个在结构相似度的前 10 位(≥98.43%)中，说明这些程序对极其相似，可能存在作业的拷贝现象。

(3)属性相似度值总体偏高，这是由于该班在学习 C 语言时已经接触过栈操作，在数据结构中再一次学习，又是同一位教师授课，导致学生作业的程序主体基本相同，结果使相似度值偏高。但其中也不乏程序的完全或经过简单修改的拷贝现象。

根据本研究运行的结果结合教师对程序的比较判断，对学生进行调查，结果发现 50%以上的学生进行了作业的借鉴或拷贝。通过本研究在教学中的应用，起到了督促学生学习、保证教学质量、提高学生学习兴趣的作用。

实验发现，通过采用之前提到的更改方式(1),(2),(4)~(7)以及更改空白区域和简单数据类型更改等更改方式的一种或少部分组合进行修改后得到的程序对具有较高的 Halstead 属性相似度。而对于无关程序对属性相似度碰巧偏高的情况，必须结合结构相似度正确考察两程序的相似程度。如果综合运用了每一种更改手段，则程序对的相似度较低。

属性相似度和结构相似度结合度量，如果两者都偏高，可以确定地说 2 个程序为相似程序；两者都偏低，2 个程序为无关程序；如果属性相似度偏高，结构相似度适中，值得作进一步考察，以决定程序对是否为相似程序。

对于初学程序设计的学生所编制的较小的程序，本研究的度量精度不高，这种情况下任何人编出的程序几乎大体相同，又由于 C 语言是结构化的程序设计语言，精度不高也是可以理解的。如果教师预先提供了要编制程序的部分模块供全体学生使用，也会使相似度偏高。

5 结束语

本文的研究实现了对程序代码度量理论验证，构建并验证了程序代码相似度的度量方法。能够实现对汉字的精确处理，这是国外的任何系统做不到的，也可以考虑将其应用在作业批改的自动化或试卷评阅的自动化上，这是今后的研究方向。

参考文献

[1] Edward J L. Metrics-based Plagiarism Monitoring[J]. Journal of Computing Sciences in Colleges, 2001, 16(4): 253-261.

[2] Gitchell D, Sim N T. A Utility for Detecting Similarity in Computer Programs[C]//Proc. of the ACM SIGCSE'99. New Orleans, LA, USA: [s. n.], 1999.

[3] Halstead M. Elements of Software Science[M]. New York, USA: Elsevier North Holland Press, 1977.

[4] Eppstein D. Design and Analysis of Algorithms Lecture Notes[Z]. (2006-02-09). http://www.ics.uci.edu/~eppstein/161/960229.html.

编辑 陈 文