

文章编号:1001-9081(2010)01-0022-03

一种基于 B^X 树的移动对象混合索引方法

张辉,刘永山,高云

(燕山大学 信息科学与工程学院,河北 秦皇岛 066004)

(wing333@sina.com)

摘要:为实现移动对象当前及未来位置索引,提出了一种自适应 SAB^X 树,考虑到移动对象在现实世界中分布密度不同的特点,利用时间划分和空间填充曲线技术计算移动对象位置信息,并引进了一个以秩的范围为标识的 Hash 辅助索引表,给出了 SAB^X 树的插入、删除、更新方法以及范围查询算法。实验表明,该索引结构的动态更新性能和查询效率优于 B^X 树和传统的 TPR 树。

关键词:移动对象索引; B^X 树; Hilbert 曲线; 秩

中图分类号: TP392 **文献标志码:** A

Hybrid indexing of moving objects based on B^X -tree

ZHANG Hui, LIU Yong-shan, GAO Yun

(School of Information Science and Engineering, Yanshan University, Qinhuangdao Hebei 066004, China)

Abstract: A Self-Adaptive B^X -tree (SAB^X) was presented for indexing the current and future position of moving objects. Taking account of the different density of moving objects in reality, SAB^X -tree utilized the technology of temporal partitioning and space-filling curves to calculate the objects' position, then a supplemented Hash index was introduced to store and index the forest of B^X -trees. Besides, the insertion, deletion, update methods and range query algorithm were developed for SAB^X -tree. Experimental results show that SAB^X -tree's dynamic update and query performances outperform those of B^X -tree and traditional TPR-tree.

Key words: moving objects Index; B^X -tree; Hilbert curve; order

0 引言

随着无线通信技术和定位技术的迅速发展,基于位置的服务(Location-Based Service, LBS)在交通监控、智能导航、电子商务等领域发挥着越来越显著的作用,移动对象数据库^[1](Moving Object Database, MOD)的概念也应运而生。与静态数据库系统相比 MOD 具有频繁更新的特点,而传统的空间索引方法无法对存储的海量移动对象信息进行实时更新。为了满足用户对移动对象准确、快速地存储和检索需求,目前国内外多种移动对象索引技术大量涌现,但总体来说该领域的研究与发展仍然处于未成熟阶段。因此,建立一种高效的索引机制成为了一个极富挑战性的研究热点。

近来,对移动对象未来位置的索引需求愈演愈烈,已经逐渐深入到人们的日常生活中,因此关于未来索引方法的研究成为备受国内外技术人员关注的前沿课题。其中,由 Saltenis 等人提出的基于 R^* ^[2] 树的 TPR 树^[3]是目前使用较为广泛的索引结构,它利用时间参数化最小边界矩形对移动对象的位置信息实行动态且“保守”的存储方法,TPR 树能有效预测移动对象的未来信息,但随着时间延伸,移动对象 MBR 之间大量重叠以及节点频繁分裂造成查询性能急剧下降。Tao 等人提出的 TPR⁺ 树^[4]与 TPR 树结构相同,但对动态更新算法进行了优化。Patel 等人提出的 STRIPES^[5]采用二元转换的方法,该方法可以保证较高的查询和更新效率,其代价是占用双倍的磁盘存储空间。Dan 等人提出的 B^X 树^[6]是基于 B^+ 树的索引结构,很方便整合到数据库中且大大节省了存储空间。

本文在 B^X 树的基础上创建了自适应 B^X 树混合索引结构(Self-Adapt B^X -tree,简称 SAB^X -tree),首先根据移动对象的分布密度对整个索引空间定义填充秩(order)不同的填充曲线,并建立相应的 B^X 树,用 Hash 表对所有 B^X 树的根节点进行索引。已有理论证实 Hilbert 曲线^[7]的性能优于 Z-排序曲线,因此本文选择前者构造 SAB^X 树。

1 Hilbert 曲线

1.1 问题描述

在 B^X 树中,为实现移动对象二维位置信息向一维数据的转换,必须先确定 Hilbert 曲线的秩,也就是说在整个索引结构建立和更新的过程中,秩是静态不变的。若选择的秩较大,则索引结构需占用庞大的存储空间,对于分布稀疏区域索引结构利用率很低;反之,若秩较小,在移动对象分布稠密区域会出现多个移动对象拥有相同 Hilbert 值的情况,导致查询无效。

1.2 Hilbert 秩

基于上述问题,本文提出了 SAB^X 树,其自适应性的思想体现在两个“动态”上:随着移动分布情况和位置变动,动态生成 Hilbert 曲线的秩;动态生成与秩相对应的 B^X 树,并由 Hash 表中的 pt 域指向该树的根节点。

图1说明了如何动态确定移动对象的秩。由 $order = 2(2^2 \times 2^2 = 16$ 个子区域)的 Hilbert 曲线填充的空间索引区域,00区域的移动对象分布稀疏, O_1 、 O_2 、 O_3 可用唯一的曲线值(Hilbert curve value,简称 cv)表示,分别为 0001_b 、 0010_b 、

收稿日期:2009-07-23;修回日期:2009-09-08。 基金项目:河北省自然科学基金资助项目(F2009000473)。

作者简介:张辉(1985-),女,山东嘉祥人,硕士研究生,主要研究方向:时空数据库; 刘永山(1963-),男,河北张家口人,教授,博士,主要研究方向:数据库理论、计算机应用技术; 高云(1983-),女,河北昌黎人,硕士研究生,主要研究方向:空间数据库。

0011_b。10区域的移动对象分布密集,例如 O_4, O_5, O_6, O_7 的 cv 都为1001_b,依据B^x树的思想,要实现对象准确存储,即一个 cv 唯一确定一个移动对象,就要对整个空间进行 $order = 4$ 的曲线填充。而SAB^x树只对稠密区域1001用 $order = 2$ 的曲线填充, O_4, O_5, O_6, O_7 的 cv 可以唯一确定(1001X_b),分别为10011111_b、10010100_b、10011000_b、10010110_b,实际上,1001区域的 $order = 2 + 2 = 4$,其他区域的 $order$ 不变。

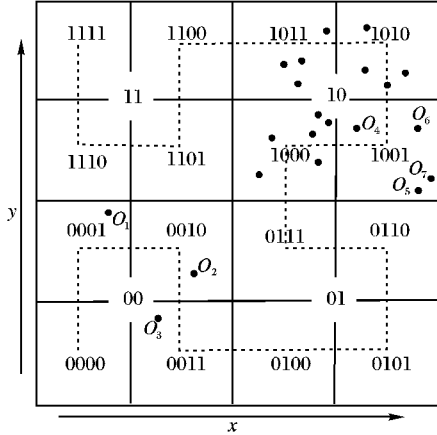


图1 Hilbert curve($order = 2$)

2 自适应B^x树(SAB^x树)

2.1 创建SAB^x树索引结构

图2为SAB^x树索引结构,其创建过程并不复杂。图中右上角为一个以秩的范围为标识的Hash索引表,每个单元包括数据域和指针域,数据域存储秩的范围,我们取四个连续数值为一个范围,max即秩的最大值应依据实际需求而定,pt指针指向与该秩的范围相对应的B^x树的根节点。所有生成的B^x树建立在同一时间轴上,首先把时间轴按 Δt_{mu} 大小等间隔划分, Δt_{mu} 为预测的移动对象发生两次更新的时间间隔的最大值,再将每个划分为 n 个长度相等的子间隔,称为阶段, n 取决于移动对象发生两次更新的时间间隔的最小值。每个阶段末尾时刻称为标记时间戳 t_{lab} ,用 t_{lab} 作为移动对象位置信息的前缀,若移动对象在 t_{up} 时刻发生更新,则 $t_{lab} = \lceil t_{lab} + \frac{\Delta t_{mu}}{n} \rceil$, $\lceil x \rceil$ 返回未来距离 x 最近的标记时间戳,例如,在图3中,B^x树的 $n = 2$,若 $t_{up} = 0$,则 $t_{lab} = \Delta t_{mu}/2$;若 $0 < t_{up} \leq \Delta t_{mu}/2$,则 $t_{lab} = \Delta t_{mu}$ 。我们把在同一阶段内发生更新的移动对象标记为同一 t_{lab} 。

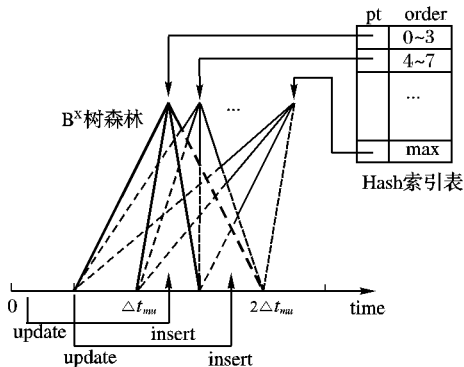


图2 SAB^x树索引结构

B⁺树仅存储线性有序的数据,因此SAB^x树也只能存储线性变化的移动对象信息。假设移动对象 o 在 t_{up} 时刻发生

位置更新, $o = (\vec{x}, \vec{v})$, \vec{x} 为二维位置信息, \vec{v} 为速度矢量, t_{up} 对应的标记时间戳为 t_{lab} ,则该移动对象在B^x树中对应的叶子节点数据项为更新时刻的阶段号与一维位置信息的串联值,即:

$$B^x value(o, t_{up}) = [phase]_b + [x_rep]_b \quad (1)$$

$phase$ 值和 x_rep 值的计算公式为:

$$phase = \left(\left(\frac{t_{lab}}{\Delta t_{mu}} - 1 \right) \mod (n + 1) \right) \quad (2)$$

$$x_rep = x_value(\vec{x} + \vec{v}(t_{lab} - t_{up})) \quad (3)$$

其中 $phase$ 为更新时刻所在的阶段号, x_rep 是由 (\vec{x}, \vec{v}) 通过空间填充曲线转换而来, $[x]_b$ 返回 \vec{x} 的二进制串。

n 的取值对于索引结构的查询性能和存储效率非常关键,当 $n > 2$ 时,SAB^x树所需存储空间略大于TPR树, $n = 1$ 时,扩展的查询窗口大于TPR树中的MBR,导致查询窗口重叠。因此,本文取 $n = 2$ 。

举例说明如何计算与存储移动对象更新时的相关信息。取 $n = 2, \Delta t_{mu} = 120$,在0时刻01区域插入移动对象 $O_p = ((7, 4), (-0.1, -0.05))$;在10时刻1001区域插入 $O_q = ((3, 11), (0.2, -0.1))$ 。下面计算移动对象在插入时刻的一维数据。

步骤1 确定移动对象的秩。

由图1可知,01区域的秩为1, O_p 应插入第一棵B^x树中;1001区域的秩为4, O_q 应插入第二棵B^x树中。

步骤2 计算标记时间戳 t_{lab} 和阶段号 $phase$ 。

$$t_{lab}^p = \lceil 0 + \frac{120}{2} \rceil_i = 60, phase_p = 0 = 00_b$$

$$t_{lab}^q = \lceil 20 + \frac{120}{2} \rceil_i = 120, phase_q = 1 = 01_b$$

步骤3 由式(3)计算 O_p, O_q 分别在 $t_{lab}(O_p), t_{lab}(O_q)$ 时刻的位置 \vec{x}_p', \vec{x}_q' 。

$$\vec{x}_p' = (1, 1), \vec{x}_q' = (5, 10)$$

步骤4 按生成Hilbert曲线的算法把位置向量转换为 cv 。

$$cv_p = 0101_b, cv_q = 01100110_b$$

步骤5 由式(1)计算 $B^x value$ 。

$$B^x value(O_p, 0) = 000101_b = 5$$

$$B^x value(O_q, 10) = 10001100110_b = 1127$$

即在0时刻, $order$ 为1的B^x树中存储的移动对象 O_p 的关键词为5;在10时刻, $order$ 为4的B^x树中存储的移动对象 O_q 的关键词为1127。

同一时刻SAB^x树只在三个阶段有效,随着时间增长,第一个阶段(图2中第一棵B^x树的左边三角区)就会消失,一个新的阶段产生(图2中第一棵B^x树的右边三角区)。

2.2 索引结构的插入、删除及更新

空间填充曲线技术确保了在二维空间中邻近的移动对象映射到一维空间的数值也是邻近的,时间是线性增长的因素,因此SAB^x树的叶子节点中存储的关键词是单调递增的。

SAB^x树的插入与删除算法与B⁺树相似。在 T 时刻插入一个新的移动对象 O ,根据 O 的秩所在的Hash表单元找到 pt 所指向的B^x树,由创建SAB^x树的过程中的公式计算出 $B^x value(O, T)$ 作为叶子节点的关键字插入到该B^x树中合适的位置。若在插入过程中发生节点上溢,节点分裂方法与B⁺树相同,使得整个索引树保持平衡状态。从SAB^x树中一个

删除移动对象时,找到该对象所在的叶子节点把关键字删除即可。若插入或删除操作引起了整个 B^x 树秩的变化,则要更改 Hash 表中 pt 指针的指向。

与 B^+ 树的更新算法不同, SAB^x 树的更新只发生在 t_{lab} 时刻。例如在图3中,在 $t_0 \sim t_1$ 发生更新的移动对象存储在 T_0 阶段,在 $t_1 \sim t_2$ 发生更新的移动对象存储在 T_1 阶段。在 t_3 时刻之前, T_0, T_1, T_2 阶段共存,在 $t_3 \sim t_4$, T_0 阶段消失, T_1, T_2, T_3 阶段共存,在 T_0 内没有发生更新的移动对象都被重新存储到 T_2 阶段。

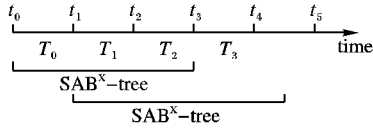


图3 SAB^x 树更新过程

2.3 范围查询算法

SAB^x 树支持范围查询,能有效预测移动对象未来的位置信息。对于一个给定的查询窗口, SAB^x 树采用扩展查询窗口的方法来预测移动对象在将来某一刻的位置,索引的未来有效时间不能超过 $t_{ref} + \Delta t_{mu}$,即扩展窗口大小不能超过 Δt_{mu} 。这种设计可以满足用户的实际要求,因为移动对象将在 $(t_{ref}, t_{ref} + \Delta t_{mu})$ 范围内发生更新,对于大于 $t_{ref} + \Delta t_{mu}$ 时刻的索引意义较小。图4所示的时间轴上 t_Q 为未来某一查询时刻, t_{ref} 为移动对象 O_1, O_2 发生最近一次更新的时刻, Q 为 t_Q 时刻的查询窗口, Q' 为扩展后的查询窗口。移动对象 O' 与 O 之间的位置关系可以用等式 $\vec{x}' = \vec{x} + \vec{v} \times (t_Q - t_{ref})$ 表示。

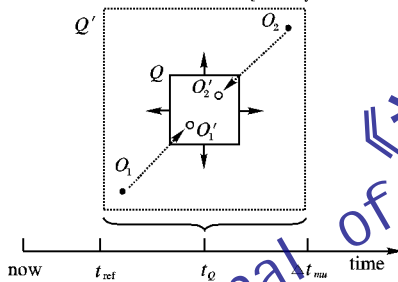


图4 SAB^x 树扩展窗口查询

给出 SAB^x 树的范围查询算法:

Input query window Q , time t_Q , t_{ref}

- 1) calculate order of Q by Hilbert curve
// 确定 Q 所属区域及 Q 的秩
- 2) for $i \leftarrow 0$ to n // n 为 Hash 索引单元个数
- 3) if order of $Q \in \text{Hash}$ then // 找到 Q 所属的 Hash 单元
- 4) $B^x\text{-tree} \leftarrow i, \text{pt}$ // 沿 pt 指针找到对应的 B^x 树
- 5) for $j \leftarrow 0$ to m // m 为阶段的个数
- 6) if phase T_j is valid at t_Q then
- 7) $Q' \leftarrow \text{TimeParameterizedRegion}(Q, t_Q)$ // 扩展窗口到 Q'
- 8) calculate starts and ends $i_1 \rightarrow i_2$ for Q'
// 计算所有阶段 T 的起始位置
- 9) for $k \leftarrow 1$ to s do
- 10) locate leaf node containing point i_{2k-1}
// 找到包括 i_{2k-1} 的叶子节点
- 11) repeat
- 12) store candidate objects in L
// 把移动对象存在链表 L 中
- 13) follow the right pointer to sibling node
// 沿叶子节点右指针找到兄弟节点
- 14) until node with the point i_{2k} is reached
- 15) for object in L do
- 16) if object's position at t_Q is included in Q then
- 17) store the object to L' // 符合条件的结果返回链表 L' 中
- 18) return L'

3 实验结果与分析

本节通过实验与 B^x 树、TPR 树对比来评估 SAB^x 树的动态更新性能及查询效率。实验运行环境为 1.6 GHz 的 CPU、1 GB 内存、Windows XP 操作系统。实验模拟数据与文献[6]提供的相似,移动对象的位置随机生成,移动对象速度大小在 0 到 3 之间变化,速度方向随机变化。 SAB^x 树的秩随着移动对象分布密度变化,因此本文采用较小的数据集进行模拟与测试(50 K ~ 1 000 K)。

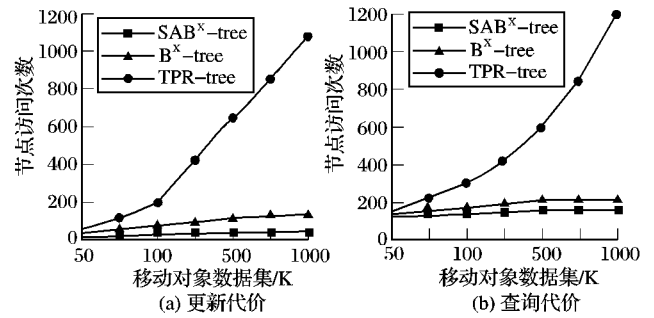


图5 数据集大小对更新性能及查询性能的影响

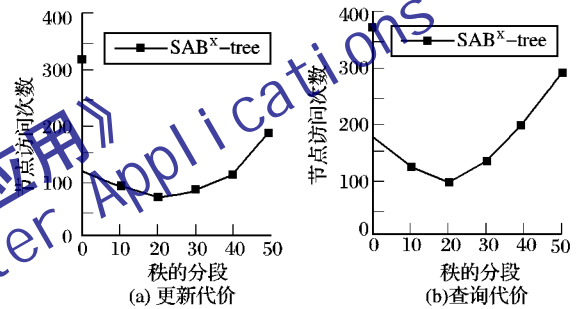


图6 秩的分段范围对更新性能及查询性能的影响

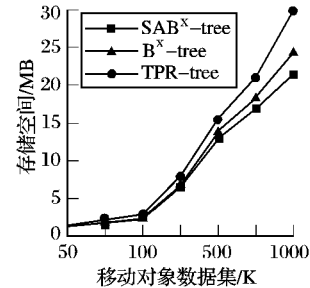


图7 所需存储空间

首先锁定 $\Delta t_{mu} = 120, n = 2$ 。从图5可以看出,当插入或删除一个移动对象时,随着数据集增大,TPR 树的更新代价和查询代价是 SAB^x 树和 B^x 树的四倍多,因为 TPR 树中 MBRs 的重叠,插入算法需要遍历多条路径,而 SAB^x 树和 B^x 树只与树的高度有关,仅需遍历一条路径,而且数据集越大 SAB^x 树比 B^x 树的优越性越突出,表现出更好的稳定性。再锁定数据集为 500 K,查询窗口为 50 K,如图6所示,秩的分段范围对索引结构更新及查询性能的影响呈“U”型变化。当分段大小为 0 时整个索引区域由秩唯一的 Hilbert 曲线填充, SAB^x 树则演变为 B^x 树,更新性能与 B^x 树相同,随着分段增大, SAB^x 树的更新代价明显减小并在 20 左右时达到最小,但分段继续增大时更新代价开始回涨,因为秩的分段很大导致每棵 B^x 树结构庞大,这种现象在移动对象分布稠密区域更为显著。存储空间利用率是体现一个移动对象索引方法优劣的关键因素,从图7可以看出, SAB^x 树比 B^x 树和 TPR 树占用

(下转第28页)

有突变性的增加,而本文方法有效地应用了粗糙集的相关性质,减少了频繁项集数量以及数据集扫描次数,执行时间变化相对平稳而且平均约减少3~4倍时间。

表5 Apriori 算法与本文算法挖掘规则数和用时

数据集	Apriori 算法		本文算法	
	规则数	用时/s	规则数	用时/s
Zoo	11	0.33	9	0.12
breast-cancer	5	2.34	3	0.45
Tic-Tac-Toe	32	4.46	18	1.13
Solar Flare	19	5.32	12	1.98
mushroom	37	56.80	28	9.30

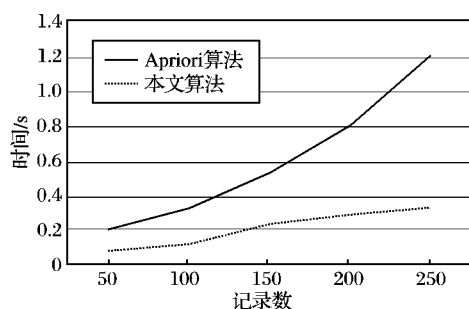


图1 breast-cancer 数据集测试结果

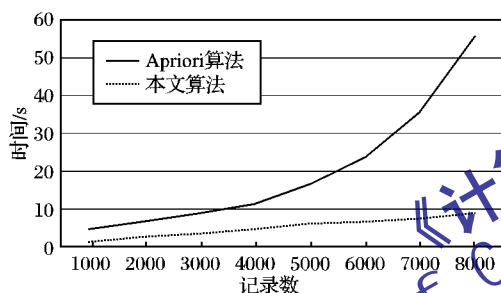


图2 mushroom 数据集测试结果

5 结语

本文给出的关联规则挖掘方法基于粗糙集理论,对传统的Apriori算法进行了改进,解决了该算法需多次扫描数据库

及产生庞大的候选项集的问题。理论及实验证明该方法可有效进行属性约简以及关联规则的挖掘,在运行时间复杂度、空间复杂度上都有一定程度上的改善。但在实际应用中最终参与决策规则挖掘的条件属性及决策属性的选择仍然依靠人为判断,所以研究如何建立相关分析模型自动确定条件属性、决策属性有待进一步的工作。

参考文献:

- [1] HAN JIAWEI, KAMBER M. Data mining concepts and techniques [M]. San Francisco: Morgan Kaufmann Publishers, 2005.
- [2] SAVASERE A, OMIECINSKI E, NAVATHE S. An efficient algorithm for mining association rules in large database[C]// Proceedings of the 21th International Conference on Very Large DataBase. San Francisco: Morgan Kaufmann Publishers, 1995: 432 - 443.
- [3] PASQUIER N, BASTIDE Y. Discovering frequent closed item sets for association rules[C]// Proceedings of the 7th International Conference on Database Theory. London: Springer-Verlag, 1999: 398 - 416.
- [4] HAN JIAWEI, PEI JIAN. Mining frequent patterns without candidate generation[C]// Proceedings of the 20th ACM SIGMOD International Conference on Management of Data. New York: ACM, 2000: 1 - 12.
- [5] BERZAL F, CUBERO J, MARIN N. TBAR: An efficient method for association rule mining in relational databases[J]. Data and Knowledge Engineering, 2001, 37(1): 47 - 64.
- [6] 叶东毅. 基于近似精度递归计算的一个属性约简算法[J]. 小型微型计算机系统, 2003, 24(12): 2272 - 2274.
- [7] 刘振华, 刘三阳, 王珏. 基于信息量的一种属性约简算法[J]. 西安电子科技大学学报, 2003, 30(6): 835 - 838.
- [8] 王国胤, 于洪, 杨大春. 基于条件信息熵的决策表约简[J]. 计算机学报, 2002, 25(7): 759 - 766.
- [9] 唐建国, 谭明术. 粗糙集理论中的求核和约简[J]. 控制与决策, 2003, 7(4): 449 - 452.
- [10] PAWLAK Z. Rough sets[J]. International Journal of Computer Information Science, 1982, 11(5): 341 - 356.
- [11] PAWLAK Z. Rough sets: Theoretical aspects of reasoning about data [M]. Netherlands: Kluwer Academic Publishers, 1991.

(上接第24页)

更少的存储空间,因为 SAB^x 树节点的数据项为一维数值,并且采用动态的曲线填充的方法使得二进制串长度随秩的变化而变化,可以在一定程度上节省存储空间。

4 结语

针对移动对象当前及未来信息的索引方法,本文在 B^x 树的基础上提出了一种自适应 SAB^x 树。该方法考虑到移动对象在空间中分布密度不同,引入了 Hilbert 曲线动态地对整个索引空间进行填充,根据秩的范围大小划分 Hash 辅助索引表的各个单元,各单元通过 pt 指针域对相应的 B^x 树进行存储管理。实验结果表明 SAB^x 树比 TPR 树节省了很多存储空间,比现有的 B^x 树在范围查询效率方面有所提高。本文将继续研究 SAB^x 索引结构的 K 近邻查询和连续查询算法。

参考文献:

- [1] TRAJCEVSKI G, WOLFSON O, HINRICHS K. Managing uncertainty in moving objects databases[J]. ACM Transactions on Database Systems, 2004, 29(3): 463 - 507.
- [2] BECKMANN N, KRIEGER H, SCHNEIDER R, et al. The R*-

Tree: An efficient and robust access method for points and rectangles[C]// Proceedings of the ACM SIGMOD on Management of Data. New York: ACM, 1990: 322 - 331.

- [3] SALTENIS S, JENSEN C S, LEUTENEGGER S T, et al. Indexing the position of continuously moving objects[C]// Proceedings of the ACM SIGMOD on Management of Data. New York: ACM, 2000: 331 - 342.
- [4] TAO Y, PAPADIAS D, SUN J. The TPR*-tree: An optimized spatio-temporal access method for predictive queries[C]// Proceedings of the VLDB. Berlin: Morgan Kaufmann, 2003: 790 - 801.
- [5] PATEL J M, ARBOR A, CHEN Y, et al. STRIPES: An efficient index for predicted trajectories[C]// Proceedings of the ACM SIGMOD on Management of Data. New York: ACM, 2004: 635 - 646.
- [6] JENSEN C S, LIN D, OOI B C. Query and update efficient B^+ -Tree based indexing of moving objects[C]// Proceedings of the VLDB. Berlin: Morgan Kaufmann, 2004: 768 - 779.
- [7] MOON B, JAGADISH H V, FALOUTSOS C, et al. Analysis of the clustering properties of the Hilbert space-filling curve[J]. IEEE Transactions on Knowledge and Data Engineering, 2001, 13(1): 124 - 141.