大数据环境下支持概率数据范围查询索引的研究

朱 睿 王 斌 杨晓春 王国仁
(东北大学信息科学与工程学院 沈阳 110004)

摘 要 随着数据规模的不断增长,大数据管理具有重要意义.在众多数学模型中,因为概率模型可以将海量数据抽象成少量概率数据,所以它非常适合管理大数据.因此,研究大数据环境下的概率数据管理具有重要意义.作为一种经典查询,基于概率数据的范围查询已被深入研究.然而,当前研究成果不适合在大数据环境下使用.其根本原因是这些索引的更新代价较大.该文提出了索引 HGD-Tree 解决这一问题.首先,该文提出了一系列算法降低新增数据的处理代价.它可以保证树结构平衡的前提下快速地执行插入、删除、更新等操作.其次,该文提出了一种基于划分的方法构建概率对象的概要信息.它可以根据概率密度函数的特点自适应地执行划分.此外,由于作者提出的概要是基于比特向量,上述策略可以保证索引以较低空间代价管理概率数据.最后,该文提出了一种基于位运算的方法访问 HGD-Tree.它可以用少量的位运算执行过滤操作.大量的实验验证了算法的有效性.

关键词 大数据;概率数据;索引;概率概要信息;多分辨率网格
中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.01929

Indexing Probabilistic Data for Supporting Range Query Over Big Data

ZHU Rui WANG Bin YANG Xiao-Chun WANG Guo-Ren
(*College of Information Science & Engineering, Northeastern University, Shenyang* 110004)

Abstract With the increasing of data scale, big data management is great significant. Underlying the popular mathematical models, probabilistic model is suitable for big data management since it could compress volume of data into a few probabilistic data. Therefore, it is significant for studying the problem of probabilistic data management over big data environment. As a classic query, range query over probabilistic data has been fully studied. However, the state of art efforts are not suitable since they all suffer from highly updating cost. In this paper, we propose a novel index named HGD-Tree for solving this problem. First of all, we propose a group of novel strategies for handling newly arrival objects. In this way, we could efficiently apply the insertion, deletion, and updating on the premise of balancing tree structure. In addition, we propose a novel partition-based structure to approach the probability density function of object, where the structure could self-adjust the partition resolution so as to cater for the underlying of uncertain data. Besides, our proposed structure is expressed by a few bit vectors. The above two strategies guarantee low space cost of the proposed index. Last but not least, we propose a novel algorithm for supporting the range query which could effectively apply the pruning under few bitwise operations. Theoretical analysis and extensive experimental results demonstrate the effectiveness of the proposed algorithms.

Keywords big-data; range query; index; summary; multi-resolution grid

收稿日期:2015-02-12;在线出版日期:2015-12-01.本课题得到国家"九七三"重点基础研究发展规划项目基金(2012CB316201)、国家自然科学基金(61272178,61572122,61173031,61129002,61532021,U1401256)、国家优秀青年科学基金(61322208)资助.**朱 睿**,男,1982 年生,博士研究生,主要研究方向为传感器数据管理与不确定数据管理.E-mail:neuruizhu@gmail.com.**王 斌**(通信作者),男,1972 年生,博士,副教授,主要研究方向为大数据管理.E-mail:binwang@ise.neu.edu.cn.**杨晓春**,女,1973 年生,博士,教授,博士生导师,主要研究领域为字符串数据管理与并行计算.**王国仁**,男,1966 年生,博士,教授,博士生导师,主要研究领域为数据库、大数据管理.

1 引 言

由大数据的 3V 模型可知,高效管理大数据面临两大挑战:(1)数据的高效存储;(2)事件的快速响应.例如,在环境监测系统中,通常有 MB 规模的传感器实时向服务器上报监测数据.假设传感器每隔 5s 向服务器上报一次数据,并且每条数据的净荷只有 10Byte,那么服务器每周将为这些数据分配约 1TB 的存储空间.由于环境监测系统需要长年累月地在多个区域收集数据,存储这些数据需要耗费高昂的代价.此外,服务器如果需要针对新增数据做实时计算,计算代价也是十分巨大的.当增量算法的时间复杂度高于亚线性级别(例如:$O(n^2)$)时,系统的实时性要求将很难得到满足.综上所述,在大数据背景下,高效处理大数据面临巨大的挑战.

在众多数学模型中,概率模型可以有效解决大数据管理面临的两大挑战.它可以根据实际应用对数据精度的要求自适应地将多条数据抽象成一条概率数据[1] $o\langle o_{\text{pdf}}, o_r\rangle$.其中,$o_{\text{pdf}}$ 表示概率密度函数①;o_r 表示 o 可能出现的区域.例如:在图 1 中,每个带阴影的矩形代表一个概率对象;矩形的边界刻画了概率对象可能出现的范围.显然,通过使用概率模型,数据规模可以有效被降低.相应地,以上两大难题也都迎刃而解.因此,在大数据环境下研究概率数据管理具有重要的理论和应用价值.

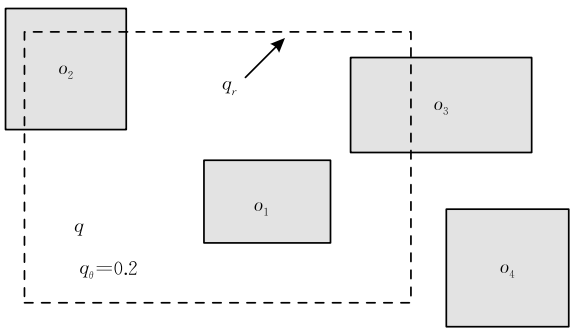


图 1 概率数据的范围查询

本文研究基于概率数据的范围查询(简称:查询).给定查询 $q\langle q_r, q_\theta\rangle$,它返回所有包含在查询范围 q_r 中概率大于阈值 q_θ 的对象.与传统范围查询相比,该查询一方面直接返回包含在 q_r 中的对象(图 1 中的对象 o_1),另一方面验证与 q_r 相交的对象(图 1 中的 o_2 和 o_3)是否满足概率阈值的要求并返回满足要求的对象.

鉴于验证操作需要耗费大量的计算代价,以往算法的研究焦点大多集中在寻找行之有效的概率剪枝策略过滤待验证对象从而降低验证规模[2-4].它们通常的做法是抽取对象概率密度函数的概要信息(记作 SUM)并用传统的索引结构管理这些 SUM.给定查询 q 和索引 $\mathcal{I}$,查询算法首先通过 $\mathcal{I}$ 找到与 q_r 相交的概率对象.接下来,算法根据对象的 SUM 计算它们包含于 q_r 的概率上界和下界.如果概率上界小于 q_θ 或者概率下界大于 q_θ,对象可以避免验证.否则,对象仍需被验证.

在众多研究成果中,Tao 等人[1]提出基于 PCR(概率限定区域)的 SUM.具体地,他们为每个对象构建一组 PCR 并将这些 PCR 当作对象的 SUM.近一步,他们提出 U-Tree 管理这些 PCR.实验证明:该索引可以有效降低对象验证规模.然而,它存在两大问题:(1)剪枝能力弱[5-7].该问题导致查询算法浪费大量的计算代价执行验证操作;(2)更新能力弱.由文献[1]可知,U-Tree 管理的 PCR 规模是对象规模的 m 倍(m 表示单个对象对应的 PCR 个数).当更新发生时,U-Tree 的更新代价也是 R-Tree 的 $O(m)$ 倍.显然,该索引不适用于管理更新频繁的数据.Zhang 等人[5-7]提出了基于对象区域划分的 SUM.近一步,他们使用 R-Tree、quad-tree 等索引管理这些划分.此类索引的概率剪枝能力强于 U-Tree.但是,它的更新代价仍然是 R-Tree 等传统索引的数倍.除此之外,基于划分构建的 SUM 需要消耗大量的存储代价.当数据规模大时,此类索引的空间代价也是难以接受的.

综上所述,由于大数据对索引性能有着较高的要求,上述索引均无法支持大数据环境下的概率数据范围查询.本文提出 HGD-Tree(Hybrid Grid Density Tree)管理大数据环境下的概率数据.为了克服上述索引存在的不足,HGD-Tree 需要面对以下挑战:

(1)快速更新.因为 SUM 的数据结构相对复杂,以 SUM 为基础构建的索引通常需要消耗较高更新代价(插入/删除/调整平衡).因此,如何在数据更新频繁的条件支持索引的快速更新是极具挑战的.

(2)空间代价小且剪枝能力强的 SUM.由于数

① 通常情况下,o_{pdf} 存在两种表示方法:连续型和离散型.其中,连续型的 o_{pdf} 用一个具体的函数表示;离散型的概率密度函数用样本点的分布刻画 o_{pdf}.由于第 2 种刻画方法更具有普遍性,本文基于第 2 种概率密度函数的刻画方法进行研究.

据更新频繁，对象的 SUM 需要快速构建. 因此，在较短时间内找到空间代价低且能准确表达对象概率密度分布特征的 SUM 是非常困难的.

针对上述问题和挑战，本文首先做出大量研究. 在此基础上，本文提出多个巧妙算法应对上述挑战. 具体贡献如下：

(1) 提出 HGD-Tree 管理概率数据. HGD-Tree 是一种基于多分辨率网格的索引. 该索引同时具备以下优点：① 快速的更新能力. 众所周知，网格结构是以空间划分为基础管理数据. 当有数据插入时，索引可以快速定位插入位置. 因此，基于网格的索引结构具有较强的动态更新能力；② 相对平衡的结构. 通过对传统的多分辨率网格进行改进，HGD-Tree 用分辨率较高网格划分数据密集的区域；用分辨率较低网格划分数据稀疏的区域. 它可以保证索引各节点维护的对象数目大致相同从而保证了索引的平衡性. 为了近一步降低 HGD-Tree 的更新代价，本文提出一种批量更新算法. 该算法通过一些事件触发机制自适应地对 HGD-Tree 进行维护. 基于上述算法，索引的更新能力得到近一步的提升.

(2) 自适应的 SUM 构建算法. 自适应性在以下方面得以体现：① 自适应于概率密度分布. 在构建对象 SUM 的过程中，本文通过挖掘对象概率密度变化的特征，找出一种新的 SUM 构建算法. 该算法可以保证在较短时间内为对象构建过滤能力更强且空间代价小的 SUM；② 自适应于查询热点. 在大数据环境下，由于数据更新频繁且计算资源有限，以不计成本的方式构建对象 SUM 是不可行的. 本文提出基于区域热度的代价模型. 该模型区别对待处于热点区域和非热点区域内的对象从而找到最优的 SUM 构建策略.

(3) 基于 bit 向量的 SUM 存储/访问算法. 根据概率数据的特点，本文提出了一种基于 bit 向量的 SUM 存储/访问算法. 由于 bit 向量可以高效利用计算资源和存储资源，HGD-Tree 不仅可以近一步提高查询效率而且可以节省空间.

本文第 2 节概述相关工作；第 3 节介绍问题定义；第 4 节详细介绍 HGD-Tree 的工作原理；第 5 节对所提方法的有效性进行实验分析；第 6 节对全文进行总结.

2　相关工作

本节的主要内容是对国内外关于概率数据范围查询的相关工作进行概述. 在介绍这些工作之前，本文首先通过表 1 定义一组符号.

表 1　符号说明

符号	名称
o	概率对象
o_r	o 可能出现的区域
o_{pdf}	o 的概率密度函数
$app(o_i)$	o 出现在 o_r 子区域 $o_i.r$ 的中概率
$ub(o,q)$	o 出现在查询区域 q_r 中的概率上界
$lb(o,q)$	o 出现在查询区域 q_r 中的概率下界

由于数据存在着不确定性，不确定数据管理近年来得到广泛关注[8-12]. 与传统数据管理相比，因为不确定数据是以区域的形式存在，所以传统多维索引已不再适合管理不确定数据. 因此，研究人员对传统索引进行适当的改造，提出了一系列经典方法管理概率数据. 多维数据管理一直以来都是数据库领域的研究热点. 以 BD-Tree[13]、BBD-Tree[14]、BV-Tree[15] 等为代表的基于空间划分建立索引；以 R-Tree[16-17]、X-Tree[18-19]、M-Tree[20]、SS-Tree[21] 等为代表的基于度量空间建立的索引都在多维数据管理领域占有重要地位. 目前为止，主流的索引包括基于 PCR 的索引和基于划分的索引.

基于 PCR 的索引. Tao 等人[1] 提出基于 PCR 的索引管理概率对象. 给定任意对象 o，它的 PCR 是根据 o_r 和特定概率阈值经收缩得到的矩形. 为了处理带不同阈值的查询，Tao 等人预先为每个对象构建一组 PCR，并以此作为概率对象的 SUM. 近一步，他们提出 U-Tree(其本质是一组 R-Tree)管理 PCR.

U-Tree 主要存在两个问题：(1) 过滤能力弱. 如文献[5-7]所示，基于 PCR 的 SUM 无法为概率对象提供紧凑的概率边界；(2) 动态更新能力弱. U-Tree 需要分配多个 R-Tree 管理基于不同阈值得到的 PCR. 当有更新发生时，算法需要同时维护多个 R-Tree. 显然，U-Tree 的更新代价远高于 R-Tree.

基于划分的索引. Zhang 等人[5-7] 提出了基于区域划分的索引管理概率对象. 这类索引(UI-Tree 和 UD-Tree)将对象区域的划分当做 SUM. 给定对象 o，查询算法根据查询区域 q_r 与 o_r 各子区域 $o_i.r$ 的拓扑信息进行概率剪枝. 具体地，如果 $o_i.r \wedge q_r \neq \varnothing$ 且 $o_i.r \nsubseteq q_r$，$ub(o,q) += app(o_i)$；如果 $o_i.r \subseteq q_r$，$ub(o,q) += app(o_i)$ 且 $lb(o,q) += app(o_i)$. 当查询算法遍历 o_r 各子区域后，当且仅当 $lb(o,q) < q_\theta < ub(o,q)$，$o$ 需要被验证. 实验结果表明，基于划分得到的 SUM 比基于 PCR 得到的 SUM 具备更强的过

滤能力.

然而,该索引存在以下问题:(1)空间代价大.由于划分没有迎合对象概率密度分布的特征,当分布倾斜时,此类索引的 SUM 需要消耗大量的空间代价;(2)更新能力差.因为这类索引需要管理所有对象的多个子区域,其管理规模远大于对象规模,所以此类索引也需花费较大的更新代价.

其他索引. Aggarwal 等人[2]研究了高维不确定对象的索引.因为该索引只能在对象概率密度函数呈无关分布时使用,所以它存在一定的局限性. Agarwal 等人[3]使用线段树管理概率对象.该索引也存在动态更新能力弱等问题. Kalashnikov 等人[4]利用网格管理[22-24]概率对象.该索引存在的问题是无法处理数据倾斜面临的问题. Zhu 等人[25]提出了 R-MRST 索引不确定对象.他们研究了当概率密度函数呈连续时的 SUM 构建.然而,在大数据环境下,新增数据的分布很难快速拟合.因此,该算法不适用于管理更新频繁的概率数据. Angiulli 等人[26]在度量空间上提出基于"轴"的索引管理不确定对象.该索引利用度量空间上的剪枝策略过滤概率对象.它的问题是空间代价远远大于其他索引.

3 问题定义

多维概率对象:给定 d 维空间下的概率对象(简称对象)o,它可以用两种方法进行描述.第 1 种方法是将概率对象抽象成二元组$\langle o_{\text{pdf}}, o_r\rangle$.其中,$o_r$ 表示 o 可能出现的区域;o_{pdf}表示关于 o 出现在 o_r 内任意点的概率密度函数.第 2 种描述方法是将概率对象表示成一组样本点$\mathcal{S}\{s_1, s_2, \cdots, s_n\}$.任意样本点 s_i可以用二元组$\langle c_i, p_i\rangle$表示.其中,$s_i.c$ 表示样本点的坐标,$s_i.p$ 表示 s_i对应的概率.

问题定义(多维概率对象范围查询).给定包含 N 个对象的集合 O 和概率范围查询 $q\langle q_\theta, q_r\rangle$,$q$ 返回所有满足不等式 $app(o,q)\geq q_\theta$的概率对象.其中,$app(o,q)$表示对象 o 出现在查询区域 q_r内的概率.

根据概率对象表示方法的不同,$app(o,q)$的计算方式也有所不同.当对象 o 的概率信息用概率密度函数表示时,o 出现在 q_r内的概率可以通过式(1)计算.其中,$q_r \bigcap o_r$表示 o_r和 q_r之间的相交面积.

$$app(o,q) = \int_{o_r \bigcap q_r} o.\text{pdf}(x)\text{d}(x) \tag{1}$$

当 o 的概率信息用一组样本点表示时,o 出现在 q_r内的概率可以通过式(2)计算.其中,n_2 等于包含在 $q_r \bigcap o_r$中 o 的样本点个数;n_1等于$|\mathcal{S}|$.

$$app(o,q) = \sum_{i=1}^{i=n_2} s_i.p \Big/ \sum_{i=1}^{i=n_1} s_i.p \tag{2}$$

在实际应用中,对象的概率密度函数很难拟合.因此,本文采用第 2 种方式表示概率数据并以此为基础研究概率数据管理.为了方便表达,本文以二维概率数据为例描述算法.

4 HGD-Tree

4.1 HGD-Tree 概述

本文提出一种新的索引结构 HGD-Tree(混网格树)管理大数据环境下的概率对象.如图 2 所示,HGD-Tree 是以多分辨率网格[26]为基础构建的两层索引结构. HGD-Tree 的第 1 层被称为 α 层,它管理概率对象的空间信息;HGD-Tree 的第 2 层被称为 β 层,它管理概率对象的 SUM.

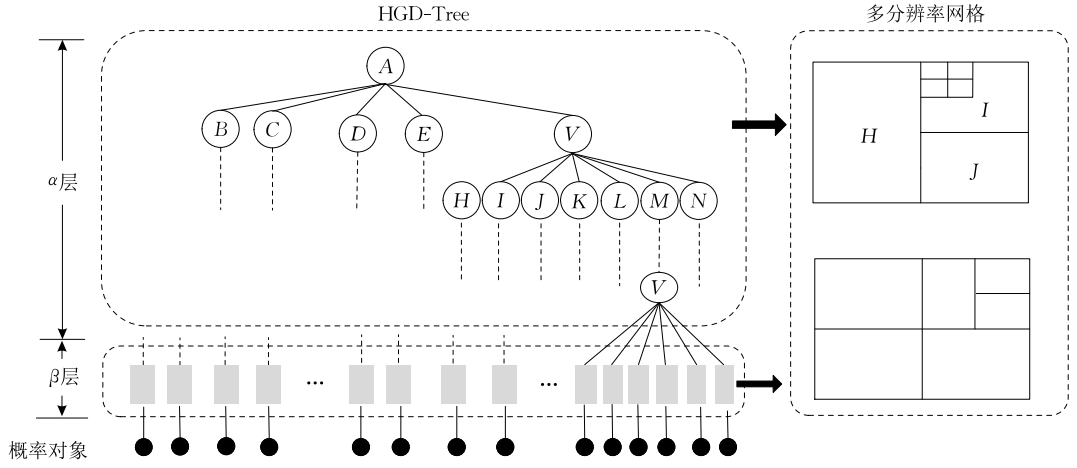


图 2 HGD-Tree 的框架

HGD-Tree 的 α 层索引(简称 α 层索引)是基于如下观察构建的：给定平衡的树状索引 T 与它的节点 e，e 的深度 $e.h$ 和它对应区域的面积 $e.s$ 可以反映 e 中数据的密度 $e.\rho$. 即：在 $e.h$ 一定时，$e.s$ 越小，$e.\rho$ 越大. 相反，$e.s$ 越大，$e.\rho$ 越小.

利用上述观察，本文提出基于对象数量密度的 α 层索引. 具体地，它以空间划分为基础，用面积相对较小的节点管理数据密度较大的区域；用面积相对较大的节点管理数据密度较小的区域. 显然，基于这种方法构建的索引可以满足索引平衡性的要求.

此外，该方法的另一个好处是索引结构可以反应对象的空间分布. 随着对象的更新，只要它们的空间分布不发生剧烈变化，即使数据更新频繁，算法也能以较少的计算代价执行更新并保持索引的相对平衡.

然而，数据的分布往往随着的时间推移逐渐变化. 针对这一问题，本文提出一种高效的批处理更新算法. 该算法利用多种数据转移策略有效地降低了更新代价.

HGD-Tree 的 β 层索引管理对象的 SUM. 该层索引为每个对象分配一棵子树. 每棵子树通过存储对象区域的划分刻画对象的 SUM. 给定对象 o，β 层索引用粒度较高的划分管理样本点分布不均匀的区域；用粒度较低的划分管理样本点分布相对均匀的区域. 基于这种方法，β 层索引可以使用较少的划分更为深刻地反映出对象概率分布特征. 因此，该方法不仅可以有效降低索引的存储空间而且可以增强对象 SUM 的概率剪枝能力.

接下来，本文将分别对 α 层和 β 层索引展开详细描述.

4.2　α 层索引描述

本节首先提出了算法 MAS(Merge And Split)初始化 HGD-Tree 的 α 层索引. 其次，本文提出 BUA 算法动态维护 α 层索引.

4.2.1　α 层索引的构建算法

在描述算法之前，本文首先定义 3 个参数 $\sigma^u_{\max}$，$\sigma^u_{\min}$ 和 σ_0. 给定 HGD-Tree 的 α 层索引 $\mathcal{I}^\alpha$，$\sigma^u_{\max}$ 和 $\sigma^u_{\min}$[①] 分别表示 $\mathcal{I}^\alpha$ 中深度为 u 的节点的最大/最小容积；σ_0 表示 $\mathcal{I}^\alpha$ 中叶子节点的容积. 此外，本文用 c^m_i 表示基于网格 $\mathcal{G}(m\times m)$ 划分下得到某一单元格；$|c^m_i|$ 表示 c^m_i 包含的对象数目.

给定 N 个概率对象，MAS 以自上而下的方式初始化 $\mathcal{I}^\alpha$. 首先，MAS 根据概率对象的值域 $\mathcal{R}$ 构建 MBR(最小包围矩形) 并对其进行 $m\times m$(本文以 $m=8$ 为例)的空间等分. 此后，MAS 递归地合并包含对象较少的单元格；分裂包含对象较多的单元格[②]. 具体地，给定单元格 c^m_i，如果 $|c^m_i|<\sigma^u_{\min}$，算法寻找可以与 c^m_i 合并的单元格；如果 $|c^m_i|>\sigma^u_{\max}$，算法分裂 c^m_i. 当合并或分裂操作完成后，MAS 重复上述操作构建新层次节点. 当所有节点包含对象数量均小于 σ_0 时，构建算法结束. 接下来，本文详细介绍合并和分裂操作.

合并操作：$\forall c^m_i$，如果 $|c^m_i|<\sigma^u_{\min}$，MAS 为 c^m_i 寻找可以合并的单元格. 这类单元格是从 c^m_i 的邻居中产生. 给定 c^n_j，当 c^m_i 与 c^n_j 满足条件(1)(2)(3)时，二者合并. 如图 3(a)所示，因为 $|A|<\sigma^i_{\min}$，算法为 A 寻找可以合并的单元格. 给定 A 的邻居 B，因为它们满足上述 3 个条件，算法将 B 合并到 A.

(1) $m=n$；

(2) $|c^m_i|+|c^n_j|<\sigma^u_{\max}$；

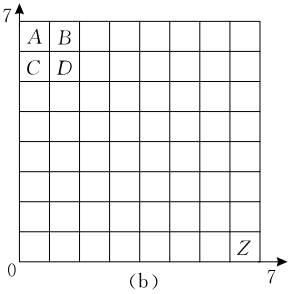
(3) $\exists c^{\frac{m}{2}}_k$，$c^m_i\bigcup c^n_j\subseteq c^{\frac{m}{2}}_k$.

此后，MAS 继续为 c^m_i 寻找可以合并的单元格. $\forall c^m_k$，如果 c^m_k 与中间结果(例如图 3 中的 $A\cup B$)之间满足条件(2)和(3)，MAS 将 c^m_k 并入中间结果. 当 MAS 无法为 c^m_i 找到可合并的单元格时，关于 c^m_i 的合并操作结束.

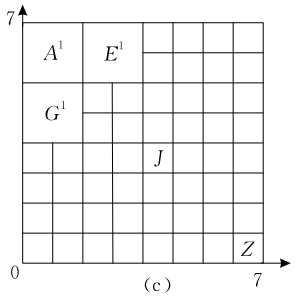
特殊地，如果 c^m_i 可以与其邻居合并成低粒度的单元格 $c^{\frac{m}{2}}_i$，MAS 以递归的形式继续为 $c^{\frac{m}{2}}_i$ 寻找可合

ID	对象数量
A	1 KB
B	4 KB
C	2 KB
D	2 KB
E	2 KB
...	...
J	93 KB
...	...
Z	28 KB

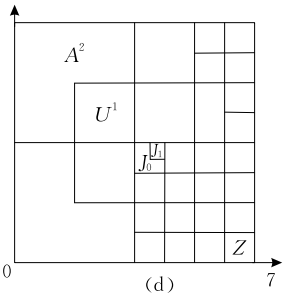
(a)



(b)



(c)



(d)

图 3　α 层节点的构造($\sigma^u_{\max}=32$ KB，$\sigma^u_{\min}=16$ KB)

① 给定第 u 层节点 e，本文将 $\sigma^u_{\max}$ 和 $\sigma^u_{\min}$ 设置为 $2|e|/64$ 和 $|e|/128$，$\sigma_0=128$.

② 为了降低维护代价，本文通过对象重心的位置统计各节点包含的对象数目.

并的单元格.例如,如图3(b)～(d)所示,因为A,B,C,D合并成A^1,算法继续为A^1寻找可合并的单元格.合并结果是将A^1,E^1,G^1合并成A^2.

分裂操作:$\forall c_i^m$如果$|c_i^m|>\sigma_{\max}^u$,MAS对c_i^m执行2×2的分裂操作.如图3(d)所示,因为$|J|>\sigma_{\max}^u$,MAS将其分裂成4个子区域.在分裂结束后,如果仍然存在c_i^m的子网格c_j^{2m}满足条件$|c_j^{2m}|>\sigma_{\max}^u$,MAS递归对这样的子网格执行分裂操作直到$c_i^m$中所有子单元格$c$都满足$|c|<\sigma_{\max}^u$.最后,MAS对$c_i^m$内包含对象数目小于$\sigma_{\min}^u$的子单元格执行合并操作.例如,本文对$J$的分裂结果做合并操作.合并结果为$J_0$和$J_1$.

为了清晰地阐明$\mathcal{I}^\alpha$的构建过程,本文给出MAS的基本流程.算法首先定义和初始化用于控制节点的构造顺序的队列dQ.接下来,算法重复执行以下操作:(1)弹出队首;(2)使用网格$\mathcal{G}(m\times m)$对队首节点进行划分;(3)对划分结果进行分裂或合并;(4)将满足条件$|e|>\sigma_0$的节点插入队尾.当队列为空时,算法结束.

算法1. MAS算法.

输入:对象集合O,值域$\mathcal{R}$

输出:索引$\mathcal{I}_\alpha$

```
1.  Queue dQ, dQ.push(R)
2.  While(dQ.empty()≠true)
3.  Set ε=partition(dQ.pop(),m×m),
4.  FOR(int i=0;i<|ε|;i++)
5.    IF |ε_i|>σ^u_max
6.      ε←split(ε_i)
7.    ENDIF
8.    IF |S_i|<σ^u_min
9.      ε←Merge(ε_i,ε);
10.   ENDIF
11. ENDFOR
12. FOR(int i=0;i<|ε|;i++)
13.   IF |ε_i|>σ_0
14.     dQ.push(ε_i)
15.   ENDIF
16. ENDFOR
17. ENDWHILE
18. RETURN;
```

根据初始化算法描述可知HGD-Tree适于管理大数据环境下的概率数据.原因如下:(1)HGD-Tree基于空间划分对概率对象进行管理,算法不必像R-Tree那样频繁更新各节点的取值范围;(2)HGD-Tree根据对象的分布密度决定节点的大小.基于这个性质,构造算法可以保证同层次节点包含对象的数目大致相同从而解决了数据倾斜带来的问题;(3)由于HGD-Tree根据概率对象的重心决定节点的维护(分裂/合并),因此,当更新操作发生时,HGD-Tree可以有效减少需要维护的节点的数量.

4.2.2 α层索引节点的存储结构描述

HGD-Tree的α层索引构建算法虽然可以增强动态更新能力,但是它可能引起查询效率降低、丢解等负面问题.接下来,本文从α层节点的数据结构入手来解决上述问题.给定HGD-Tree的α层索引$\mathcal{I}^\alpha$,本节将分别介绍$\mathcal{I}^\alpha$中间节点和叶子节点的存储结构.

(1)中间节点.如图3(d)所示,给定$\mathcal{I}^\alpha$任意中间节点e_i,其孩子节点的覆盖范围并不相同.这导致网格的优势无从发挥①从而降低了查询效率.因此,本文提出一种新的数据结构UPV(Union Protocol Vector)解决上述问题.

如图4所示,UPV表示了中间节点的数据结构.它由两部分组成:$head$和$cell$.$head$存储节点的基本信息.其中,$e_i.Id$是e_i的唯一标识符;$e_i.pr$代表划分e_i的网格分辨率;$e_i.nm$等于e_i管理的对象数目;$e_i.r$表示e_i的覆盖区域.

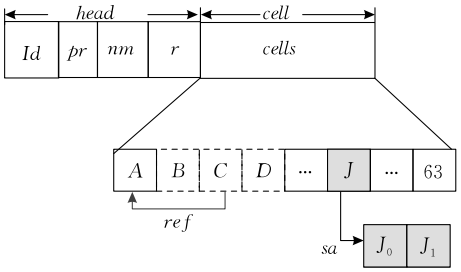


图4 $\mathcal{I}^\alpha$中间节点存储结构

$e_i.cell$是长为$e_i.r\times e_i.r$的数组.数组中每个元素对应一个基于$\mathcal{G}(e_i.r\times e_i.r)$划分得到的单元格.由于$\mathcal{I}^\alpha$在构建过程中存在着分裂、合并等操作,所以$e_i.cell$中各元素存储的内容并不相同.给定$e_i.cell[u]$和它对应的单元格$c_u$,如果$c_u$被其他单元格合并,$e_i.cell[u]$指向所有与$c_u$一起参与合并的单元格中下标最小的单元格.在图4中,$e_i.cell[2]$对应于图3(b)中的单元格C.因为它跟A,B等节点一起组成了A^2,$e_i.cell[2]=e_i.cell[0]$.如果c_u被分裂,$e_i.cell[u]$指向数组sa.该数组存储c_u的子网格.例如,$e_i.cell[36]$对应于J.因为J被分裂成两个节点(J_0和J_1),$e_i.cell[36]$

① 给定基于标准网格$\mathcal{G}(m\times m)$划分的节点e,网格访问算法可以根据网格的位置直接定位具体单元格而无需遍历所有单元格.

指向 sa. 如果 c_u 没被分裂(或合并)，$e_i.cell[u]$指向 c_u 对应的单元格.

显然，$\forall c_u$，只要它未被分裂，算法就可以利用基于网格的查询算法定位包含于 c_u 中的对象. 即使 c_u 被分裂，由于 c_u 中子网格数量往往较少，遍历范围也是可接受的.

(2) 叶子节点. 因为本文根据概率对象的重心统计各节点包含对象的数目，该策略可能导致丢解. 如图 5(a)所示，U，V，W 是 3 个叶子节点. 给定查询 q，虽然 $q_r \bigcap o_5.r \neq \varnothing$，但因为 o_5 的重心包含于 U，W 并不维护关于 o_5 的信息. 因此，o_5 可能被错误地过滤.

为了解决这个问题，本文提出 ILV(Inverted List Vector)存储叶子节点. 如图 5(b)，ILV 为每个叶子节点 e 分配了倒排列表 $list_e$. 它存储的内容是所有与 $e.r$ 存在交集的对象 ID. 在图 5(c)中，$iList_u$，$iList_v$ 是节点 U，V 的倒排表.

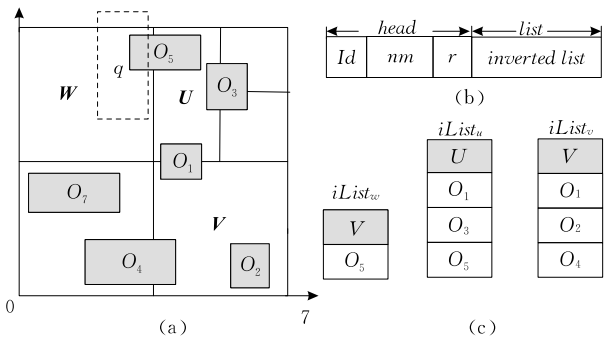


图 5　$\mathcal{I}^{\alpha}$ 叶子节点存储结构

4.2.3　增量维护算法

本文提出一种新的批量更新算法 BUA(Batch Update Algorithnm) 维护 HGD-Tree α 层索引. 如图 6 所示，该算法开辟缓冲区 $\mathcal{B}$ 临时存储更新数据. 当有特定事件发生时，BUA 将 $\mathcal{B}$ 中的一些数据批量转移到 HGD-Tree 中.

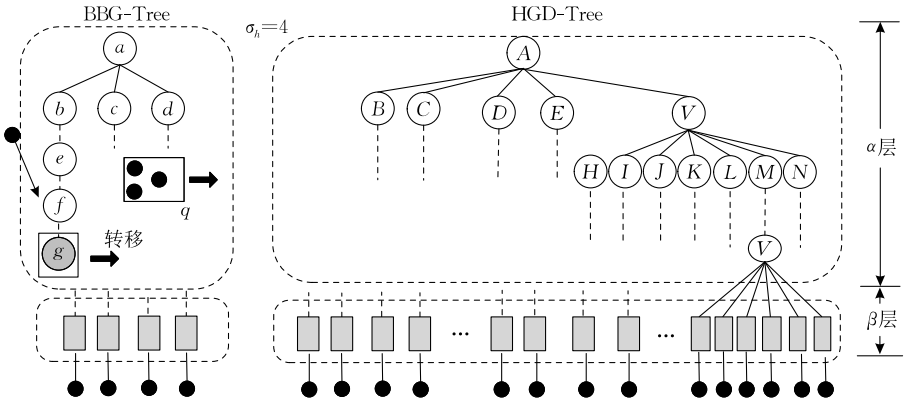


图 6　基于批处理的 HGD-Tree

为了方便管理 $\mathcal{B}$ 中数据，本文近一步提出了索引 BBG-Tree($\mathcal{B}$-Based Grid Tree). 它也是一种两层索引(α 层和 β 层). 与 HGD-Tree 不同，BBG-Tree 的 α 层索引虽然也是管理概率对象的空间信息，但是它的节点是基于等分构造的①. 这种做法的好处是可以利用网格特性实现快速插入. BBG-Tree 中的 β 层索引管理对象的 SUM. 它的结构和 HGD-Tree 的 β 层相同.

接下来，本文详细介绍 BUA. 需要强调的是：因为对象的插入与对象的删除互为逆操作；更新操作可以看成是插入和删除的组合，所以本文仅以对象的插入操作为例介绍更新算法.

给定插入对象 o_{ins} 和 BHG-Tree 的 α 层索引 $\mathcal{BI}^{\alpha}$，BUA 根据 o_{ins} 的重心将 o_{ins} 插入到 $\mathcal{BI}^{\alpha}$ 的叶子节点中. 假设 o_{ins} 插入节点 e_b，BUA 根据 $|e_b|$ 决定是否分裂 e_b. 具体地，如果 $|e_b| > 2r^2$，算法将 e_b 做 $r \times r$ 的分裂. 此时，插入操作结束.

由于 $|\mathcal{B}|$ 需要控制在一定范围内，所以 BUA 会在以 4 种触发条件下将 $\mathcal{B}$ 中的某些对象批量转移至 HGD-Tree. 它们是：查询触发；$\mathcal{B}$-倾斜触发；$\mathcal{H}$-倾斜触发；溢出触发.

(1) 查询触发. 因为本文使用两个独立的索引管理概率对象，所以查询算法需要在两棵树中分别进行搜索. 给定查询 q，$\mathcal{B}$ 中的查询结果集 $\mathcal{R}^b$ 和 HGD-Tree 中与 q_r 相交的叶子节点集 ε^q，因为 $q_r \subset \varepsilon^q$ 且 $\mathcal{R}^b \subset q_r$，所以 $\mathcal{R}^b \subset \varepsilon^q$. 因为 $\mathcal{R}^b \subset \varepsilon^q$，BUA 可以根据查询结果直接定位 $\mathcal{R}^b$ 中对象的插入位置. 显然，基于查询触发的数据转移策略可以有效降低插入代价. 如图 6 所示，BUA 将 q 在 $\mathcal{B}$ 中得到的查询结果批量插入 HGD-Tree.

① 对于任意节点 e_i，它的孩子都是基于特定分辨率划分得到的基本单元格.

(2) $\mathcal{B}$-倾斜触发. 给定 BHG-Tree 的 α 层索引 $\mathcal{BI}^{\alpha}$,因为 $\mathcal{BI}^{\alpha}$ 中节点是基于均匀划分构建的,当 $\mathcal{B}$ 中对象分布倾斜时,$\mathcal{BI}^{\alpha}$ 中某些子树的高度可能远远高于其他子树从而影响查询和插入效率. 为了解决上述问题,本文对 $\mathcal{BI}^{\alpha}$ 中高度大于阈值 σ_h 的子树执行"截断"操作,并将截断的部分转移到 HGD-Tree 中.

回顾前文,当 $|e_b|>2r^2$ 时,更新算法对 e_b 做 $r\times r$ 的划分,并将树的高度 h 加 1. 如果 $h+1>\sigma_h$,BUA 将高出的部分"截断"并转移至 HGD-Tree. 如图 6 所示,当 o_{ins} 插入 f 后,因为 $|f|>2r^2$,所以 BUA 需要创建新的网格 g. 此时,树的高度变为 5. 因为 $\sigma_h=4<5$,BUA 需要将 f 包含的对象转移至 HGD-Tree 中. 给定 HGD-Tree 的 α 层索引 $\mathcal{I}_{\alpha}$,BUA 在转移过程中首先查找 $\mathcal{I}_{\alpha}$ 中与 f_r 相交的叶子集合 ε^r(通过一次范围查询). 然后,BUA 根据查询结果进行批量插入.

(3) $\mathcal{H}$-倾斜触发. 伴随着转移操作,$\mathcal{I}^{\alpha}$ 各节点包含的对象数目也会发生变化. 给定节点 $e_i\in\mathcal{I}^{\alpha}$,如果 $|e_i|<\sigma_{\min}$,e_i 需要寻找与其合并的节点. 由于该操作花费代价较大,本文提出 $\mathcal{H}$-倾斜触发. 一方面它可以及时将 $\mathcal{B}$ 中的数据转移到 $\mathcal{I}^{\alpha}$. 另一方面它可以降低 $\mathcal{I}^{\alpha}$ 中节点的合并频率.

具体地,BUA 需要为 e_i 寻找合并节点时,BUA 发起在 $\mathcal{BI}^{\alpha}$ 中的范围查询并找到所有与 $e_i.r$ 相交对象集合 $\mathcal{R}^e$. 其次,算法将 $\mathcal{R}^e$ 转移到 e_i 并更新 $|e_i|$. 如果 $|e_i|>\sigma_{\min}$,关于 e_i 的合并操作可以避免. 否则,e_i 需要继续合并.

(4) 溢出触发. 由于 $\mathcal{B}$ 的规模相对较小,基于缓存机制的更新算法可以在总体上降低更新代价. 然而,随着 $\mathcal{B}$ 规模的变大,节点插入 $\mathcal{B}$ 的代价也会变大. 因此当 $|\mathcal{B}|>\sigma_{\mathcal{B}}$ 时,BUA 清空一些叶子节点 e_l(清空条件为 $|e_l|>\log_B N$)并将它们包含的对象以节点为单位插入 $\mathcal{I}^{\alpha}$(代价等价于多次范围查询).

参数设置. BUA 的一个疑问是关于参数 $\sigma_{\mathcal{B}}$ 的设置. 为了找到恰当的 $\sigma_{\mathcal{B}}$,本文首先对总体代价进行分析. 分析中用到的符号如下:$|\mathcal{B}|$ 为 $\mathcal{B}$ 中对象的数目;N 为总体数据规模;B 为节点的容积;F_q,F_u 分别为单位时间内查询和更新次数.

除去初始化代价外,总体计算代价由两部分组成:查询代价和更新代价. 对于查询代价,因为本文使用了两棵独立的树,所以单次查询代价为 $O(\log_B|\mathcal{B}|+\log_B(N-|\mathcal{B}|)+|\mathcal{R}|)$,其中 $|\mathcal{R}|$ 为查询结果的个数. 因此,单位时间内总的查询代价为 $O(F_q(\log_B|\mathcal{B}|+\log_B(N-|\mathcal{B}|)+|\mathcal{R}|))$.

更新代价也是由两部分组成:插入和转移. 单个对象的插入代价为 $O(\log_B|\mathcal{B}|)$,单位时间内的插入代价为 $O(F_u(\log_B|\mathcal{B}|)$;单个对象的转移代价为 $O(1)$,单位时间内的转移代价为 $O(F_u)$.

综上所述,由于引入 $\mathcal{B}$,单次插入节省的代价为 $\log_B N-\log_B|\mathcal{B}|$;总体节省的插入代价为 $(\log_B N-\log_B|\mathcal{B}|)F_u$. 单次查询额外消耗的代价为 $\log_B|\mathcal{B}|$;总体额外消耗的代价为 $(\log_B|\mathcal{B}|)F_q$. 为了使得总体节省代价大于消耗代价,不等式 $(\log_B N-\log_B|\mathcal{B}|)F_u>F_q\log_B|\mathcal{B}|$ 必须成立. 因此,本文将 $\sigma_{\mathcal{B}}$ 设置为 $N^{\frac{F_u}{F_q+F_u}}$.

4.3 β 层索引算法描述

β 层索引用来管理对象的 SUM. 与文献[5-7]一样,本文也是利用划分技术构造对象的 SUM. 不同的是,本节提出的 SUM 更能反应对象样本点的分布特征. 因此,它具备更强的过滤能力. 本节首先提出了 MMR 的概念和基于 MMR 的 SUM 概率边界. 其次,本文介绍了基于 MMR 的 SUM 构建算法和最优 SUM. 最后,本节讨论 SUM 的存储结构.

4.3.1 基于 MMR 的 SUM 概率边界

定义 1(专属矩形(记作 MR)). 给定概率对象 o,它的样本点 x_i 和包含 x_i 的矩形 r,如果 r 包含且只包含 x_i,本文称 r 为 x_i 的专属矩形. 记作:$x_i.mr$.

定义 2(最大专属矩形(记作 MMR)). 给定概率对象 o,它的样本点 x_i 和它的专属矩形集合 $\mathcal{MR}$,x_i 的 MMR 为 $\mathcal{MR}$ 中面积最大的专属矩形.

基于 MMR 的概率上下界. 我们首先以文献[5-7]中的算法为背景,讨论基于 MMR 的 SUM 能够提供的概率上下界. 给定查询 q,对象 o 和它的划分 $\mathcal{O}\{o_1,o_2,\cdots,o_n\}$,如果 $r_r\bigcap o_i.r\neq\varnothing$,本文可以通过式(3)和(4)分别计算基于 MMR 的子区域 o_i 能为对象提供的概率上界 $ub(i)$ 和下界 $lb(i)$. 式(5)表示 $\mathcal{O}$ 能为对象 o 提供的最大概率边界差. 其中,$\max R_i$ 和 $\min R_i$ 分别表示包含于 o_i 的样本点中面积最大和最小的 MMR;$app(i)$ 表示 o 出现在 $o_i.r$ 中的概率;$s_i=|q_r\bigcap o.r|$;w 为样本点的权值.

$$ub(i)=\min\left(1,\frac{ws_i}{\min R_i}\right) \tag{3}$$

$$lb(i)=\frac{ws_i}{\max R_i} \tag{4}$$

$$\max D(o)=\sum_{i<n}\min(app(i),ub(i)-lb(i)) \tag{5}$$

比起文献[5-7]中 SUM 为对象提供的概率边界,基于 MMR 的 SUM 显然能为对象提供更紧的

概率边界. 此外，从式(3)～(4)可知，$\max R_i - \min R_I$ 越小，$\mathcal{O}$ 提供的概率边界越紧. 因此，MMR 可以近一步指导 SUM 的构建.

4.3.2　基于 MMR 的 SUM 构建算法

本文将基于 MMR 构建的 SUM 称为 SUM^p. 它的构造算法与 KD-Tree 的构建算法相似；其核心思想是将 MMR 面积接近的区域尽量划分在一起；给定对象 o，算法基于 KD-Tree 的构建算法对 o_r 进行划分；用队列 sq 控制子区域的划分顺序. 算法 2 给出了 SUM^p 的构建流程.

算法首先初始化队列 sq. 此后，算法重复弹出队首并对其执行分裂操作. 给定队首节点 e 和它对应的子区域 o_i，算法对 o_i 执行等分操作. 等分之后，算法根据不等式(6)判断是否对 o_i 继续划分. 如果不等式为真，算法停止对 o_i 的划分. 否则，算法将划分结果插入队列. 算法重复上述操作直到队列为空. 最后，算法用 KD-Tree T 管理这组划分. 其中，参数 θ 是一个阈值. 本文将在 4.3.4 节讨论它的设置.

$$|\mathcal{O}|(ub(i)-lb(i))<\theta \tag{6}$$

算法 2.　SUM^p 构建算法.

输入：概率对象 o，最大边界阈值 θ

输出：o 的 SUM^p

```
1.  初始化队列 Queue sq: sq←o_r
2.  While(sq.empty()≠true)
3.    Node e←sq.pop(); {o_i1, o_i2}=split(e);
4.  Foreach(e'∈{o_i1, o_i2})
5.      IF ub(e')−lb(e')>|O|θ
6.        e←sq.push(e');
7.      ENDIF
8.  ENDFOR
9.  ENDWHILE
10. buildKDTree();
11. Return;
```

例 1(SUM^p 的构造).　图 7(a)给出了对象 o 样本点的分布. 首先，算法将 o_r 进行等分操作，结果如图 7(b)所示. 由于区域 A 满足不等式(6)，算法停止对其划分；由于区域 B 满足不等式(6)，算法将其等分为 B 和 C. 由于 B 中的样本点是均匀分布的，虽然 B 包含了大量的样本点，算法仍然停止对其划分. 接下来，算法继续对 C 进行划分. 在多次划分后，算法将 C 分成了 $\{C,D\}$，图 7(e)显示了最终划分结果. 图 7(f)表示管理划分结果的 KD-Tree[①].

与文献[5-7]中的 SUM 构造算法相比，SUM^p 的构造算法迎合了样本点分布的特性. 在大多数情况下，它只需极少次的划分就可以得到概率边界更紧的 SUM. 接下来，本文对 SUM^p 的剪枝能力和空间代价进行分析.

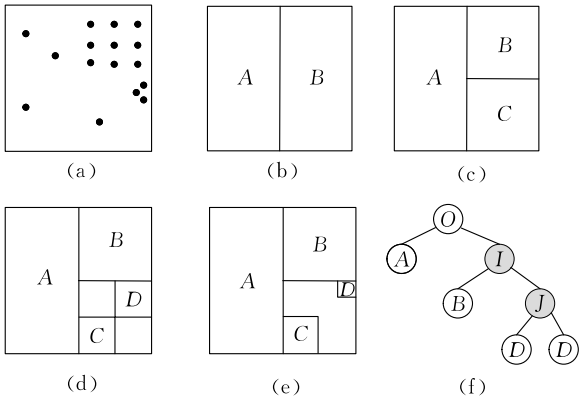


图 7　概率对象区域划分与构造

定理 1(更紧的概率边界).　给定概率对象 o 和它对应的划分 $\mathcal{O}\{o_1,o_2,\cdots,o_n\}$，如果 $\mathcal{O}$ 分别基于 UD-Tree 和 SUM^p 管理，SUM^p 的剪枝能力更强.

证明.　对于 $\forall o_i \in o_r$，当它被 UD-Tree 管理时，它提供的概率上下界分别为 $app(o,i)$ 和 0. 相反，当它被 SUM^p 管理时，它提供的概率上下界为式(3)和(4). 显然，(4)－(3)$<app(i)-0$. 因为 $ub(o)=\sum ub(i)$，$lb(o)=\sum lb(i)$，SUM^p 会为概率对象提供更紧的概率边界.　　证毕.

定理 2(更小的空间代价).　给定概率对象 o 和基于 UD-Tree 得到的划分 $O^u\{o_1^u,o_2^u,\cdots,o_n^u\}$，如果 $\max D(o)=\theta$，本文可以基于 MMR 找到一种新的划分 $\mathcal{O}^m\{o_1^m,o_2^m,\cdots,o_n^m\}$. 该划分可以同时满足不等式：(1) $\max D(o)\leq\theta$；(2) $|O^m|\leq|O^u|$. 其中，$|O^m|$ 与 $|O^u|$ 分别表示两个划分中的子区域个数.

证明.　给定概率对象 o 和它对应的划分 $\mathcal{O}^u$，SUM^p 的构建算法可以基于 MMR 将 $\mathcal{O}^u$ 中某些子区域合并且保证合并结果仍满足不等式(6). 由于合并操作减少子区域数目，不等式 $|O^m|\leq|O^u|$ 成立.　证毕.

4.3.3　MMR 构建算法

4.3.2 节中一个尚未解决的问题是如何为概率对象的样本点构建 MMR. 本文提出了一种高效算法解决上述问题.

给定概率对象 o 和它的样本点 $\mathcal{S}\{s_1,s_2,\cdots,s_n\}$，本文首先对样本点各维度的坐标排序并用倒排链表存储. 接下来，算法计算各样本点的 MMR. 具体地，算法任选一列倒排链表进行遍历（本文以选择 l_1 为

① 为了节省存储空间，KD-Tree T 只存储包含样本点的节点.

例描述算法).在遍历过程中,给定 $l_1[i]$和它对应的样本点 s_u,算法首先找到 s_u 在 l_2 中的位置①.其次,算法将 s_u 的 MMR 设置为$(l_1[i+1]-l_1[i-1])\times(l_2[k+1]-l_2[k-1])$.特殊地,对于倒排链表首部(或尾部)的元素,算法利用 $o.mbr$ 计算其 MMR.

算法总体分为两步:(1)初始化倒排链表;(2)计算样本点的 MMR.在初始化倒排列表时,由于算法需要对样本点的每一维坐标进行排序,初始化代价为 $O(dm\log m)$.计算 MMR 的代价是遍历所有样本点,它的代价是 $O(m)$.因此,总体的计算代价为 $O(dm\log m)$.由于样本点只是原始数据的一个子集,本文可以将 m 看成常数.

4.3.4 基于区域热度的 SUM^{p}构建模型

4.3.2 节遗留的另外一个问题是参数 θ 的设定.本文通过引入区域热度的概念解决这一问题.

通常情况下,给定概率对象 o,θ 越大,构造代价越小,但概率剪枝能力越差.反之,θ 越小,构造代价越大,但概率剪枝能力越强.与以往算法相比,因为概率对象只在索引中驻留一段时间,所以 SUM^{p}的构造代价是不可忽略的.本文提出基于区域热度的代价模型.它可以为对象找到最优的 SUM^{p}构建方式并计算出最优的 θ.

该模型基于以下动机,给定一组对象,某些区域内的对象经常被访问(热点区域),而另外一些区域内的对象很少被访问(非热点区域).对于具有相同 max$D$ 但处于不同区域的对象,它们被验证的频率也是不同的.具体地,处于热点区域的对象访问频率高,被验证的次数频率也高.相反,处于非热点区域内的对象访问频率低,被验证的频率也低.因此,根据区域热度来决定 SUM^{p}是十分有意义的.

在讨论代价模型之前,本文首先定义一组符号.给定对象 o,$cost_V$ 表示 o 在无法被过滤时消耗的验证代价;$cost_c$ 表示 SUM^p 的构造代价;$cost_p$ 表示 SUM^{p}的访问代价;T 表示 o 在其生存周期内被访问的次数.

如式(7)所示,给定对象 o,它消耗的计算代价包括验证代价、SUM^{p}构建和 SUM^{p}访问代价.其中,$|T|cost_p$ 为 SUM^{p}的访问代价.显然,这部分代价独立于 θ.$cost_c$ 可以拟合成关于 θ 的函数 $\varphi(\theta)$.它的含义是构建 max$D(o)=\theta$ 的 SUM^{p}所需的计算代价.显然,$\varphi(\theta)$是关于 θ 的减函数.由于拟合 $\varphi(\theta)$不是本文主要贡献,本文不对其详细叙述.$T\theta cost_V$ 是 o 总共消耗的验证代价.其中 $T\theta$ 为验证次数的数学期望.从式(7)可以看出,$cost_T$ 是一个非单调函数.本文可以根据它的拐点设置 θ.

$$cost_T = T\theta cost_V + |T|cost_p + cost_c \quad (7)$$

参数 T 的设置:给定对象 o,因为它的 SUM^{p}是在其插入时构建的.构建算法只能根据历史访问记录预估 T.具体地,给定 HGD-Tree 的 α 层索引 $\mathcal{I}_\alpha$,如果 o 的重心包含于 $\mathcal{I}_\alpha$中的叶子节点 e,本文将 T 设置为 e 的访问次数.

4.3.5 基于 bit 串的 SUM^{p}存储

虽然 SUM^{p}可以有效减少划分次数,但是它仍耗费高昂的代价存储各子区域的位置信息和概率概要信息.为了近一步降低存储代价,本文提出了基于 bit 向量的存储结构.例如,图 8 中的 bit 向量存储了图 7(f)中 KD-Tree.给定对象 o、划分 O 和它对应的 KD-Tree T,本文通过描述 T 中各节点的数据结构介绍 SUM^{p}的存储方式.

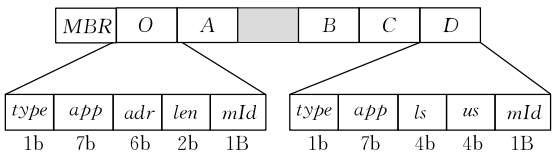


图 8 SUM^{p}的存储结构

(1)数据结构.T 包括两类节点:叶子节点和中间节点.给定叶子节点 e_f,它用五元组$\langle type, app, ub, lb, mId\rangle$表示.具体地,$type$ 表示节点类型(1 bit).app 表示对象 o 出现在 e_f 对应区域的概率.为了降低存储代价,存储算法将概率值(浮点类型)映射成一个 $rank$ 值(7 bit),其值域为[0,127].假设 o 出现在 $e_f.r$ 内的概率是 0.2,它对应的 $rank$ 值为$\lfloor 0.2\times 128\rfloor=25$.$us$ 和 ls 分别表示 $e_f.r$ 中 max$R$ 和 min$R$ 的面积.与 app 相同,本文也将面积(浮点类型)映射成 $rank$ 值(4 bit),其值域为[0,15].mID 记录了节点的位置信息,本文稍后描述其细节.

(2)中间节点的存储结构.给定中间节点 e_i,它用五元组$\langle type, app, adr, len, mId\rangle$表示.其中,$type$,$app$,$mId$ 的含义与上文相同.adr 表示其孩子节点在 bit 串中的相对地址;len 表示孩子的个数,它们分别占用的 6 bit 和 2 bit.

(3)基于 bMap 的 mId 存储方法.为了介绍 mId 的存储方法,本文首先提出 bMap 的概念.给定概率对象 o 和它的划分 $\mathcal{O}\{o_1, o_2, \cdots, o_n\}$,因为 $\forall o_i$ 都是基于不同划分得到的单元格,所以它可以用 bit 向

① 本文将同一样本点在不同链表上的坐标用指针相连.这样可以避免多次遍历倒排链表.

量表示.例如:在图 7(e)中,节点 B 是基于 2×2 划分下得到的单元格,本文可以用 bit 向量{10,01}表示它的位置;节点 A 是基于 2×1 划分下得到的单元格,本文可以用 bit 向量{01,1}表示它的位置.

然而,节点可能是基于不同网格划分得到的,它们对应的 bit 串的长度也不相同(例如:图 7(e)中的 D 和 C).因此,算法需要为长度不同的 bit 串提供一个适配器(称为 bMap)以便访问.

如图 9 所示,bMap 是由多条 bit 串列组成的静态表.它给出任意长度的 bit 串到定长 bit 串的映射关系.bMap 中每条记录都是 1 个三元组〈Id,bit,$mbit$〉.给定 $bMap[t]$,$bMap[t].bit$ 表示它在网格 $\mathcal{G}(\lfloor\log t\rfloor\times\lfloor\log t\rfloor)$划分下某一行(或列)单元格对应的 bit 串;$bMap[t].mbit$ 表示该行(或列)单元格在网格$\mathcal{G}(m\times m)$划分下包含哪些行(或列)子格.例如:如图 7(d)所示,节点 C 是基于$\mathcal{G}(4\times4)$划分下得到的单元格.它对应的 bit 向量是{0001,1000}.它包含了一组在$\mathcal{G}(8\times8)$划分下得到的单元格.这些网格的并集可以表示为{00000011,11000000}.

ID	bit	$mBit(m=8)$
0	01	00001111
1	10	11110000
2	0001	00000011
3	0010	00001100
4	0100	00110000
5	1000	11000000
…	…	…
13	10000000	10000000

图 9　bMap 的存储格式

利用 bMap,本文可以有效压缩节点位置的存储空间.具体地,本文仅存储各节点对应 bit 串在 bMap 中的下标.如图 7(e)所示,节点 C 为基于 4×4 划分下得到的单元格;其 bit 向量是{0001,0010};它们分别保存在 $bMap[2]$和 $bMap[4]$中,因此,$mId=\{2,4\}$.

在介绍节点的存储结构之后,本节额外说明以下两点:(1)bMap 只需存储少量的元组.因此,在大多数情况下,mId 只需 2 Byte 便可存储一个节点的位置信息.此外,bMap 可以被所有概率对象共享.因此,基于 bMap 的存储方式可以大大降低存储代价;(2)与传统的节点存储方式相比,基于 bit 串的存储策略可以有效压缩空间.例如,一个叶子节点只需要 3 Byte 的存储空间.结合定理 2,对象 SUM^p 的存储代价远远小于其他算法中 SUM 的存储代价.

4.4　HGD-Tree 访问算法

本文提出算法SHAB(Search on HGD-Tree and BHG-Tree)寻找 HGD-Tree 和 BHG-Tree 中满足查询条件的概率对象.为了方便描述,本文将 α 层索引中层次最低的节点称为 αleaf.

4.4.1　α 层索引的访问算法

SHAB 分为两个主要部分.第一部分是通过索引的 α 层找到所有与查询范围存在交集的概率对象.给定查询 q、HGD-Tree 的 α 层索引 $\mathcal{I}_\alpha$ 和 BHG-Tree 的 α 层索引 $\mathcal{BI}_\alpha$,SHAB 采用层次遍历的方式分别访问 $\mathcal{I}_\alpha$ 和 $\mathcal{IB}_\alpha$.本文并不对搜索过程进行详细论述.接下来,本文只强调如下两点:

(1)中间节点访问.当访问 $\mathcal{I}_\alpha$ 的中间节点 e 时,算法需要对其 UPV 进行访问.具体地,算法首先根据 $e.pr$ 和 $e.r$ 找到和 q_r 相交的网格.其次,SHAB 访问 $e.cell$ 找到与 q_r 的孩子节点.

(2)叶子节点访问.当 SHAB 找到 $\mathcal{I}_\alpha$ 和 $\mathcal{IB}_\alpha$ 中与所有 q_r 相交的 αleaf 时,SHAB 将这些 αleaf 的倒排链表执行归并操作并剔除重复对象.此后,SHAB 遍历归并后的倒排列表并找到所有与 q_r 存在交集的概率对象.

4.4.2　β 层索引访问算法

SHAB 的第 2 部分是查找符合概率阈值条件的概率对象.对于包含在查询区域内的对象,SHAB 直接将其输出.对于与查询范围相交的对象,SHAB 访问它们的 SUM^p.由上文可知,SUM^p 可以基于 KD-Tree 的搜索算法进行访问.因此,本文并不对搜索过程进行详细论述.接下来,本文只强调如下 3 点:

(1)概率边界计算.给定查询 q,概率对象 o 和它的划分$\mathcal{O}\{o_1,o_2,\cdots,o_n\}$,SHAB 根据$\mathcal{O}$中各子区域 o_i 与 q_r 的拓扑关系计算 o 包含在 q_r 中的概率上下界 $ub(o)$和 $lb(o)$.具体地:

① 当 $o_i.r\subset q_r$ 时,算法分别将 $ub(o,q)$和 $lb(o,q)$更新为 $ub(o,q)+o_i.app$ 和 $lb(o,q)+o_i.app$;

② 当 $o_i.r\not\subset q_r\wedge e.type=leaf$ 时,算法分别将 $ub(o,q)$和 $lb(o,q)$更新为 $ub(o,q)+ub(i)$和 $lb(o,q)+lb(i)$.其中,$ub(i)$和 $lb(i)$分别根据式(3)和(4)计算;

③ 当 $e.r\not\subset q_r\wedge e.type\neq leaf$ 时,算法访问 o_i 的子区域;

如图 10 所示,给定查询 q,因为 $q_r\bigcap o_r\neq\varnothing$,SHAB 遍历 o 对应的$\mathcal{O}$.因为 q_r 包含 A 区域,所以 SHAB 将 $A.app$ 分别加到 $ub(o,q)$和 $lb(o,q)$上.接下来,SHAB 访问 I 对应的子树.同理,SHAB 将

$ub(o,q)$,$lb(o,q)$更新为$ub(o,q)+C.app+ub(B)$和$lb(o,q)+C.app+lb(B)$.

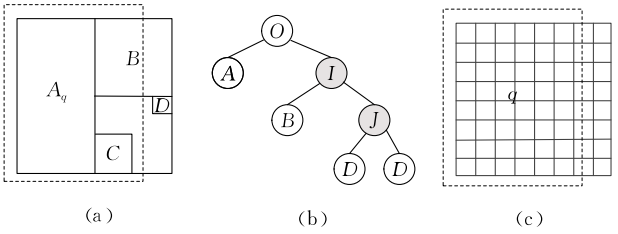


图 10 SUM^p的访问算法

(2)决策方法. 当 SHAB 结束对$\mathcal{O}$的遍历后,算法根据下列不等式决策下一步操作:① $lb(o,q)\geq q_\theta$,此时,SHAB 将 o 输入到结果集;② $ub(o,q)\leq q_\theta$,SHAB 忽略 o;③ $lb(o,q)<q_\theta<ub(o,q)$,SHAB 执行对 o 的验证操作.

(3)基于位运算的子区域位置比较算法. 给定查询 q,概率对象 o 和它的划分$\mathcal{O}\{o_1,o_2,\cdots,o_n\}$,因为$\mathcal{O}$中子区域的位置信息用 bMap 数组下标表示,所以 SHAB 需要通过位运算获取各子区域与 q_r 的拓扑关系.

SHAB 首先根据 o_r 和 q_r 的拓扑关系在各维生成一组 bit 向量 qbit. qbit 的长度等于 bMap 中 $mbit$ 字段的长度(图 9 中$|mbit|=8$). qbit 的值表示 q_r 在基于网格$\mathcal{G}(|mbit|\times|mbit|)$划分下与那行(列)网格存在交集. 如图 10 所示,查询 q 基于网格$\mathcal{G}(8\times8)$的划分生成的 qbit 为{11111100,11111111}. 它表示 q_r 与第 2,3,4,5,6,7 列(0,1,2,3,4,5,6,7 行)网格相交.

在生成 qbit 后,SHAB 根据 qbit 和各子区域的 mId 判断它们之间的拓扑关系. 具体地,$\forall o_i\in\mathcal{O}$,本文用布尔表达式(8)判断 q_r 与 $o_i.r$ 的位置关系. 其中,$o_i.mid[u].mbit$ 表示 o_i 第 u 维对应的 $mbit$.

$$\bigwedge o_i.mid[u].mbit\wedge qBit[u]\neq 0 \tag{8}$$

直观地,如果$o_i.mid[u].mbit\wedge qBit[u]\neq0$,$o_i.r$ 至少在第 u 维与 q_r 相交. 例如:节点 C 在 X 维对应的 $mbit$ 为 00001100. 因为 00001100 ∧ 11111100 ≠ 0,q_r 与 C 在 X 维相交. 进一步讲,如果 $o_i.r$ 在任意一维都和 q_r 相交,$o_i.r$ 必定和 q_r 相交.

如果布尔表达式(8)成立,SHAB 近一步判断 $o_i.r$ 是否包含于q_r. 为此,SHAB 更新 qbit. 新的 qbit 表示 q_r 在基于网格$\mathcal{G}(|mbit|\times|mbit|)$划分下包含那些行(列)网格. 如图 10 所示,$q$ 基于网格$\mathcal{G}(8\times8)$划分下生成的新 qbit 为{11111000,11111111}. 它表示 q_r 与第 3,4,5,6,7 列(0,1,2,3,4,5,6,7 行)网格存在包含关系. 更新 qbit 后,SHAB 根据布尔表达式(8)近一步判断 q_r 与 $o_i.r$ 的位置关系.

与传统的节点间位置比较算法相比,基于位运算的比较算法有以下两个优点:① 由于 CPU 适合于位操作,因此基于位运算的位置比较算法更能充分 CPU 的计算优势;② 比较次数降低,当使用坐标判断节点的位置关系时,算法需要执行 4d 次坐标比较(判断相交和包含). 相反,本文只需 2d 次位运算就可以知道查询区域和节点间的位置关系.

4.5 基于 HGD 森林的多维数据管理

前文介绍了基于二维空间的 HGD-Tree. 因为 HGD-Tree 是以网格为基础构建的索引,所以当数据维度 d 大于 2 时,索引的空间代价和维护代价会急剧提高. 为了折衷查询性能和维护性能,本节提出了 HGD 森林解决这一问题.

给定参数 m,本文根据 m 将对象 o 的属性$\{a_1,a_2,\cdots,a_d\}$分为$\{a_1,a_2,\cdots,a_m\}$,$\{a_{m+1},a_{m+2},\cdots,a_{2m}\}$,…,$\{a_{\lfloor d/m\rfloor m+1},a_{\lfloor d/m\rfloor m+2},a_d\}$. 此后,本文为每 m 维数据独立的建立 HGD-Tree $\mathcal{I}_1,\mathcal{I}_2,\cdots,\mathcal{I}_{\lceil d/m\rceil}$和 BHG-Tree $\mathcal{BI}_1,\mathcal{BI}_2,\cdots,\mathcal{BI}_{\lceil d/m\rceil}$. 当处理更新时,算法同时维护这些索引. 由于维护算法已在上文描述,本节不再重复. 需要强调的是,虽然本文构建了多个索引维护多维数据,但是本文并不会为每个索引复制一个关于多维概率数据的副本. 相反,在每个索引中,我们只保留它们的 ID. 概率对象的内容在单独的空间内保存.

接下来,本节介绍查询算法. 给定查询 q,HGD-Tree 的 α 层索引$\mathcal{I}_{\alpha1},\mathcal{I}_{\alpha2},\cdots,\mathcal{I}_{\alpha\lceil d/m\rceil}$和 BHG-Tree 的 α 层索引$\mathcal{BI}_{\alpha1},\mathcal{BI}_{\alpha2},\cdots,\mathcal{BI}_{\alpha\lceil d/m\rceil}$,查询算法 SHAB 首先在$\mathcal{I}_{\alpha1}\cup\mathcal{BI}_{\alpha1},\mathcal{I}_{\alpha2}\cup\mathcal{BI}_{\alpha2},\cdots,\mathcal{I}_{\alpha\lceil d/m\rceil}\cup\mathcal{BI}_{\alpha\lceil d/m\rceil}$上寻找符合条件的对象. 其次,SHAB 对查询结果$\mathcal{R}_1,\mathcal{R}_2,\cdots,\mathcal{R}_{\lceil d/m\rceil}$做集合交操作. 最后,算法遍历候选集合$\mathcal{R}_1\cap\mathcal{R}_2\cap\cdots\cap\mathcal{R}_{\lceil d/m\rceil}$. 对于$\forall_o\in\mathcal{R}_1\cap\mathcal{R}_2\cap\cdots\cap\mathcal{R}_{\lceil d/m\rceil}$,SHAB 根据式(9)、(10)计算 o 出现在 q_r 的概率上下界.

$$ub(o,q)_1\times ub(o,q)_2\times\cdots\times ub(o,q)_{\lceil d/m\rceil} \tag{9}$$

$$lb(o,q)_1\times lb(o,q)_2\times\cdots\times ub(o,q)_{\lceil d/m\rceil} \tag{10}$$

我们需要强调的是:由于我们引入了参数 m,HGD-Tree 可以根据数据的特点和数据维度自适应的选择划分策略. 这样以来,HGD-Tree 可以避免维度过高而引发的问题. 本文将在实验部分讨论适合的参数 m.

5　实验分析

5.1　实验准备

本节首先讨论了数据集的获取. 接下来，本文讨论参数的设置和实验方法.

(1) 数据集. 实验的第 1 个真实数据集中国深沪两市约 2300 支股票近 10 年内的交易数据. 交易记录大约 5 GB 条，占用 150 GB 的磁盘空间. 为了降低数据规模，本文以周为单位将这些数据按照股票 ID 抽象成概率数据. 具体地，实验根据各支股票一周内的交易记录随机选取 $\sqrt{N}$(N 为该股票一周内的交易总量)条记录作为样本点. 然后，实验将这些条记录中的交易价格和交易数量当做概率数据的两个维度. 和文献[1,5-7]中一样，实验的第 2 个真实数据集是 Los Angeles 的位置数据. 本文根据这些数据生成边长为 500 的正方形，然后为每条数据根据正态分布生成 100 个样本点，并将这些数据作为概率数据. 第 2 个合成数据集模拟了正态分布的数据. 它分别包含了 10 TB 条 2d，3d，4d 数据. 与第 1 个合成数据集类似，本文按照 1∶10 000 的方式生成概率数据，且每条概率数据内仍然包含 100 个样本点.

此外，实验生成了两个合成数据集. 每个数据集包含 10 TB 条的多维数据. 第 1 个合成数据集是模拟环境监测得到的数据集. 该组实验生成 1 MB 个虚拟传感器，它们不间断的上报数据当数据收集到 10 TB，实验以小时为单位生成概率数据. 其中，每条概率数据内包含 100 个样本点.

(2) 参数设置. 在本实验中，两个重要参数需要测试. 它们是查询范围 q_r 和概率阈值 q_θ. 参照文献[1,5-7]中的参数设置方法，本文将查询范围从值域的 0.01^d 扩大到 0.1^d，默认值为 0.05^d. 概率阈值从 0.1 增加到 0.9，默认值为 0.5.

此外，本文重点讨论了数据更新频繁时的概率数据管理. 在这里，本节需要测试更新速度对索引性能的影响. 在默认条件下，实验将单次插入/删除的对象数目设置成 10；变化范围是[1,1 KB]. 表 2 描述了参数设置和默认值.

表 2　参数设置

测试参数	默认值	变化范围
查询范围	0.05^d	$[0.01^d, 0.1^d]$
概率阈值	0.5	[0.1,0.9]
索引规模	$0.5\|\mathcal{N}\|$	$[0.1\|\mathcal{N}\|, 0.9\|\mathcal{N}\|]$
更新规模	10	[1,1 KB]
数据集	股票数据	地理数据，合成数据 I 和 II
数据维度	2	2～6

(3) 实验方法. 本文首先从文件中读入 $0.5|\mathcal{N}|$ 条数据初始化 HGD-Tree. 接下来，实验分成以下两组：第 1 组实验是在索引不更新的条件下测试 HGD-Tree 的过滤能力、查询效率和空间代价. 查询的默认个数为 100；位置随机生成.

第 2 组实验是在数据频繁更新的情况下测试 HGD-Tree 的更新能力和综合性能. 在测试更新能力时，本文重复以下操作：① 从文件中读入 s 条概率数据；② 将它们插入到 HGD-Tree；③ 随机删除 s 条概率数据. 当数据集中所有数据被处理后，结束测试. 在测试综合性能时，实验方法每更新 s 条概率数据后提交 s 条查询. 最后，本节介绍实验的对比算法. 它们是 U-Tree 和 UD-Tree.

(4) 数据规模的变化. 本文首先测试概率模型对数据规模的影响. 表 3 展示了这 3 组数据集在引入概率模型后数据规模的变化. 如表 3 所示，在引入概率模型后，数据的规模大幅下降. 因此，索引的负担也会大幅减轻.

表 3　数据规模的变化

数据集名称	原始规模/条	概率对象/条
真实数据集 I	5 GB	0.954 MB
真实数据集 II	300 MB	0.0648 MB
合成数据集 I	10 TB	100 MB
合成数据集 II	10 TB	100 MB

5.2　实验分析

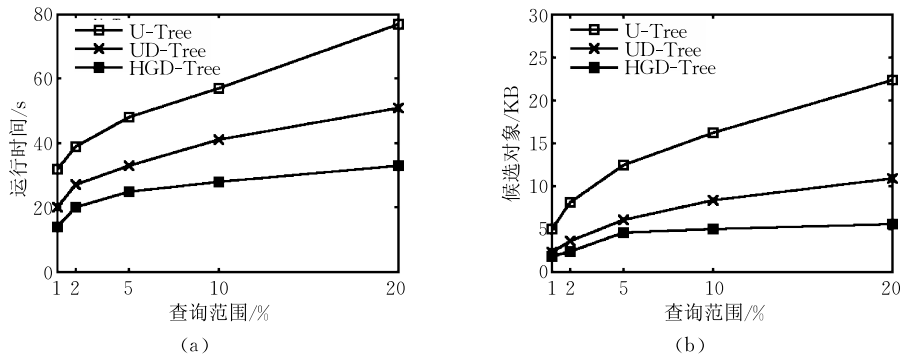
(1) 查询范围对索引的影响. 这组实验测试查询范围对索引性能的影响. 这组实验采用真实数据集 I 作为测试数据；其他参数均使用默认参数；

对比算法是 U-Tree 和 UD-Tree. 如图 11(a)所示，HGD-Tree 的过滤能力强于 U-Tree 和 UD-Tree. 特别地，随着搜索范围的扩大，基于这 3 种索引得到的候选集规模都有不同程度的扩大. 然而，相比较于 UD-Tree 和 U-Tree，HGD-Tree 候选对象集合的规模扩大的最小. 原因是：随着查询范围的扩大，与查询范围相交的概率对象也会随之增多. 在这种情况下，概率对象 SUM 的剪枝能力变得尤为重要. 图 11(b)对比了基于这 3 种索引的查询时间. 其中，HGD-Tree 的运行时间最短.

具体原因是：① 本文是以网格为基础构建的索引. 对于内存索引来说，它的访问速度最快；②SUM 的剪枝能力更强，这可以有效的减少验证次数；③ 基于位运算的 SUM 访问算法可以近一步降低计算代价从而降低查询时间.

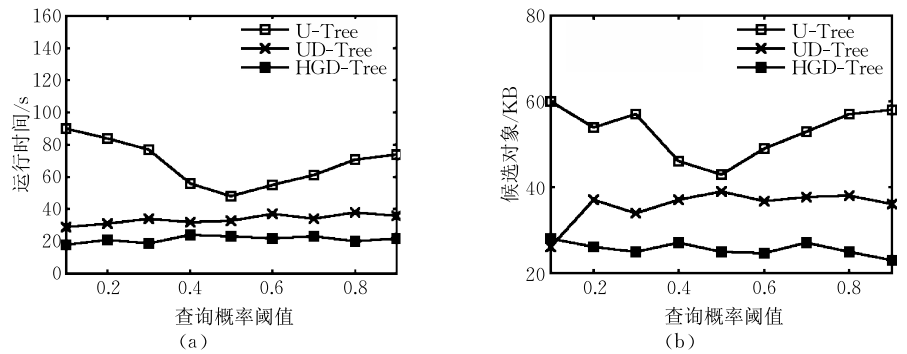
(2) 概率阈值对索引的影响. 这组实验测试的内

容是查询概率阈值对索引性能的影响.这组实验采用真实数据集Ⅰ作为测试数据;其他参数均使用默认参数;对比算法是 U-Tree 和 UD-Tree.如图 12(b)所示,概率阈值对于各种 SUM 的剪枝能力影响不大.与图 11(a)中的剪枝能力对比结果相似,HGD-Tree 的剪枝能力最强.UD-Tree 强于 U-Tree.图 13(b)对比了 3 种索引的总体运行时间.与图 11(b)中的结果相似,HGD-Tree 的查询时间最短.



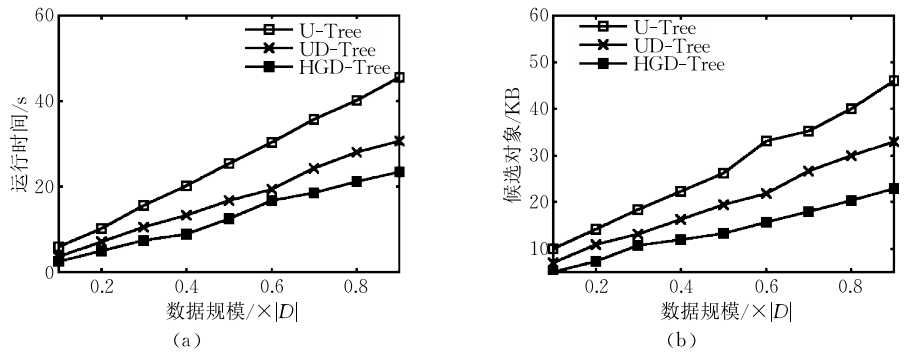
(a) (b)

图 11 查询范围对索引的影响



(a) (b)

图 12 概率阈值对索引的影响



(a) (b)

图 13 数据规模对索引的影响

(3)索引的查询效率.这组实验测试的内容是数据规模对索引性能的影响.这组实验采用真实数据集Ⅰ作为测试数据;其他参数均使用默认参数;对比算法是 U-Tree 和 UD-Tree.数据规模从 0.1 $\mathcal{N}$ 扩大到 0.9 $\mathcal{N}$.如图 13(a)所示,随着数据规模的扩大,基于这 3 种索引得到的候选集规模都有不同程度的扩大.然而,相比较于 UD-Tree 和 U-Tree,HGD-Tree 候选对象集合扩大的规模最小.原因是:随着数据规模的扩大,与查询范围相交的概率对象也会随之增多.在这种情况下,概率对象 SUM 对索引的性能产生至关重要的作用.因为基于 HGD-Tree 的 SUM 的过滤能力最强,数据规模对索引性能的影响最小.

(4)索引的空间效率.这组实验测试的内容是数据规模对索引空间效率的影响.这组实验采用真实数据集Ⅰ作为测试数据;数据规模从 0.1|$\mathcal{N}$|扩大到 0.9|$\mathcal{N}$|.对比索引是U-Tree和UD-Tree.如图 14 所示,随着数据规模的扩大,HGD-Tree 的空间代价上升幅度最小.其中,最主要的原因是 SUM^p 在大多数情况下只需较少的子区域表示一个划分.其次,

本文基于 bit 向量存储 SUM^p. 这种方法又近一步降低了空间代价. 如图 15 利用合成数据集再一次测试了索引的空间效率. 由实验结果可知，HGD-Tree 的空间代价最低.

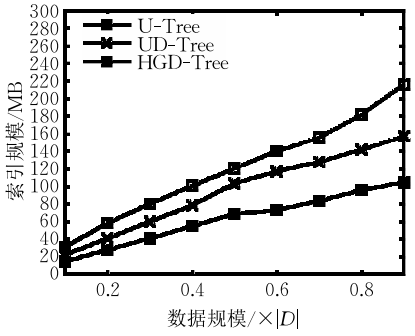


图 14　索引的空间效率

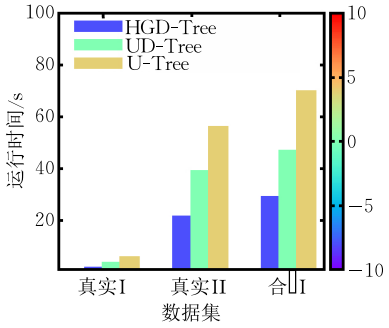


图 15　数据集对索引的影响

(5) 不同数据集对索引的影响. 这组实验测试的内容是不同数据对索引的性能的影响. 实验方法和参数设置同图 17 中的实验相同. 如图 15 所示，在不同测试数据集中，HGD-Tree 的性能最好. 这说明了 HGD-Tree 有着较为广泛的应用范围.

(6) 更新速度对索引的影响. 这组实验测试的内容是索引的更新能力. 本文在这组实验里并不考虑查询对索引的影响. 这组实验采用真实数据集 I 作为测试数据；其他参数均使用默认参数. 如图 16 所示，HGD-Tree 的动态更新能力最强. 这里有两方面原因：① HGD-Tree 使用以网格为基础管理概率数据. 该结构适用于在数据更新频繁的条件下使用；② 本文引入了一种缓冲机制批量的更新数据.

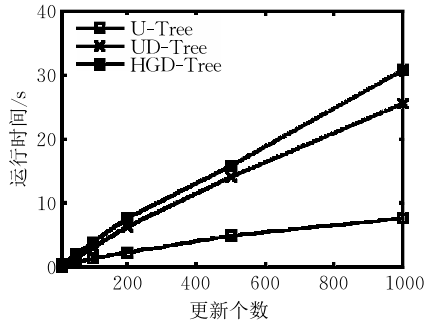


图 16　索引的更新效率

(7) 索引的综合性能. 这组实验采用真实数据集 I 作为测试数据；其他参数均使用默认参数；本文在这组实验里，查询和更新交替进行. 他们之间的比例为 1∶1. 如图 17 所示，HGD-Tree 的性能最好. 这里有两方面原因：① HGD-Tree 的更新能力远远强于其他索引；② HGD-Tree 的查询效率高于其他索引. 因此，HGD-Tree 的综合性能最好.

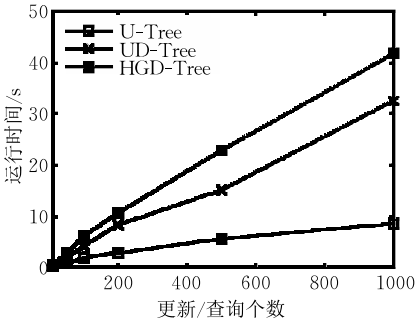


图 17　索引的综合效率

(8) 最适当的参数 m. 实验的最后，本文讨论了关于参数 m 的设置. 本节最后讨论关于参数 m 的设置. 本文采用的数据集为合成数据集，它包含了 6 个属性. 为了避免维度过高而引起高昂的空间代价和维护代价，本文将多维数据的属性按照其维度进行划分. 图 18、图 19 分别评估了不同划分下索引的过滤能力，计算代价和空间代价. 所有参数均为默认参数. 图 19 报告了参数 m 对算法过滤能力的影响. 其中，候选集个数为 $|\mathcal{R}_1 \cup \mathcal{R}_2 \cup \cdots \cup \mathcal{R}_{\lceil d/m \rceil}|$. 如图 20 可知，查询代价也是随着 m 的增大而降低. 但是，我们发现，当 $m \geq 2$ 时，下降速度变得非常缓慢. 其原因是：① 本文提出的索引具有较强的过滤能力. 因此，参与合并的候选对象个数也是有限的；② 本文采用位运算加速查询，它会额外降低算法的计算代价. 因此，当 m 减小是，查询性能的下降速度非常缓

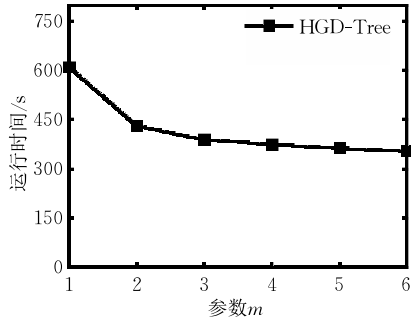


图 18　m 对计算代价的影响

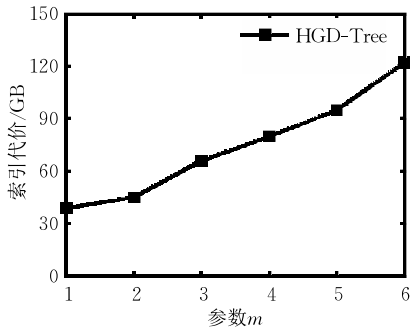


图 19 m 对索引空间代价的影响

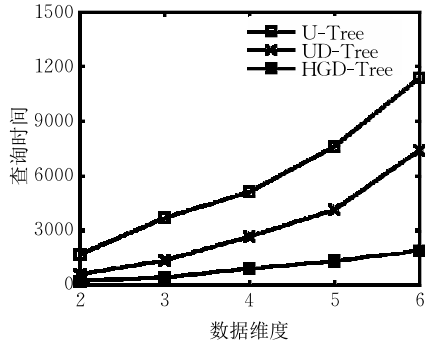


图 20 维度对查询性能的影响

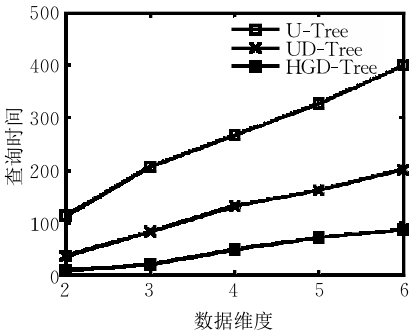


图 21 维度对更新性能的影响

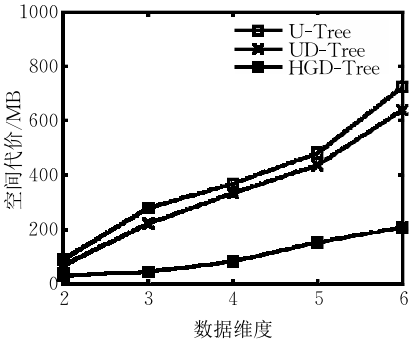


图 22 维度对空间代价的影响

慢.此外,随着 m 的增长,算法的空间代价快速增长.其根本原因是,当 m 增加是,对象 summary 包含的子区域个数迅速增加.因此索引的整体空间代价会增加.

(9)数据维度对索引的影响.这组实验测试的内容是数据维度对索引性能的影响.这组实验采用合成数据集 II 作为测试数据集;其他参数均使用默认参数.对比算法是 U-Tree 和 UD-Tree.数据维度从 2 维增加到 6 维.如图 20～图 22 所示,随着维度的增加 U-Tree 和 UD-Tree 的查询代价都快速上升.相反,HGD 森林的查询代价上升幅度相对缓慢.这有两方面原因:① SUM^p 的剪枝能力强于其他索引.当对多个索引的查找结果求交时,因为候选集中包含的对象个数较少,维度对 HGD-Tree 的影响是有限的;② HGD-Tree 是以网格为基础构建的.它在维度小于 2 时具有很大的优势.所以,HGD 森林的性能最好.如图 21 所示,随着维度的增加,3 种索引的更新代价都是呈线性增长的.其中,HGD-Tree 增长的最慢.其根本原因是:① HGD-Tree 在为对象构建 summary 速度较快;② 4.2.3 节中的批处理算法可以使得 HGD-Tree 快速的更新.如图 22 所示,随着维度的增加,所有索引的空间代价都是随着维度线性增长.因此,HGD-Tree 的空间代价依旧远远好于其他索引.

6 结 论

本文研究了支持频繁更新的概率数据范围查询的索引.本文首先提出了一种适用于管理数据更新频繁的索引 HGD-Tree.它不仅是一个平衡的结构,而且具有很强的动态更新能力.其次,本文提出了一种批处理算法处理索引的更新.再次,本文提出基于样本点距离的 SUM 构造算法.该算法可以用较少的空间代价构造出更能反应对象概率密度特征的 SUM.最后,通过大量实验验证了本文提出算法的有效性和实用性.

参 考 文 献

[1] Tao Y, Cheng R, Xiao X, et al. Indexing multi-dimensional uncertain data with arbitrary probability density functions// Proceedings of the Very Large Data Base (VLDB'05). Trondheim, Norway, 2005: 922-933

[2] Aggarwal C C, Yu P S. On high dimensional indexing of uncertain data//Proceedings of the International Conference on Data Engineering (ICDE'06). Cancun, Mexico, 2008: 1460-1461

[3] Agarwal P K, Cheng S-W, Tao Y, Yi K. Indexing uncertain data//Proceedings of the Symposium on Principles of Database Systems (PODS'09). Providence, Rhode Island, 2009: 137-146

[4] Kalashnikov D V, Ma Y, Mehrotra S, Hariharan R. Index

for fast retrieval of uncertain spatial point data//Proceedings of the Symposium on Geographic Information Systems (GIS'06). Virginia, USA, 2006: 195-202

[5] Zhang Y, Lin X, Zhang W, et al. Effectively indexing the uncertain space. IEEE Transactions on Knowledge and Data Engineering, 2010, 14(11): 1247-1261

[6] Zhang Y, Zhang W, Lin Q, Lin X. Effectively indexing the multi-dimensional uncertain objects for range searching// Proceedings of the Extending Database Technology (EDBT'12). Berlin, Germany, 2012: 504-515

[7] Zhang Ying, Zhang Wen-Jie, Lin Qian-Lu, et al. Effectively indexing the multidimensional uncertain objects. IEEE Transactions on Knowledge and Data Engineering, 2014, 22(9): 608-622

[8] Lian X, Chen L. Set similarity join on probabilistic data// Proceedings of the Very Large Data Bases (VLDB'10). Singapore, 2010: 650-659

[9] Zhou Ao-Ying, Jin Che-Qing, Wang Guo-Ren, Li Jian-Zhong. A survey on the management of uncertain data. Chinese Journal of Computers, 2009, 32(1): 1-16(in Chinese)
(周傲英, 金澈清, 王国仁, 李建中. 不确定性数据管理技术研究综述. 计算机学报, 2009, 32(1): 1-16)

[10] Lian X, Lin Y, Chen L. Cost-efficient repair in inconsistent probabilistic databases//Proceedings of the ACM Conference on Information and Knowledge Management (CIKM'11). Glasgow, Scotland, 2011: 1731-1736

[11] Jiang Bin, Pei Jian, Lin Xue-Min, Yuan Yi-Dong. Probabilistic skylines on uncertain data: Model and bounding-pruning-refining methods. Journal of Intelligent Information Systems, 2012, 38(1): 1-39

[12] Cheng R, Xia Y, Prabhakar S, et al. Efficient indexing methods for probabilistic threshold queries over uncertain data//Proceedings of the Very Large Data Base (VLDB'04). Toronto, Canada, 2004: 876-887

[13] Ohsawa Y, Sakauchi M. The BD-tree—A new n-dimensional data structure with highly efficient dynamic characteristics// Proceedings of the Computer Congress (IFIP'04). Paris, France, 1983: 539-544

[14] Shapiro J M. Embedded image coding using zerotrees of wavelet coefficients. IEEE Transactions on Signal Processing, 1993, 41(12): 3445-3462

[15] Freeston M. A general solution of the n-dimensional B-tree problem//Proceedings of the Special Interest Group on Management of Data (SIGMOD'95). San Jase, USA, 1995: 80-91

[16] Beckmann N, Kriegel H-P, Schneider R, Seeger B. The R*-tree: An efficient and robust access method for points and rectangles//Proceedings of the Special Interest Group on Management of Data (SIGMOD'90). New York, USA, 1990: 322-331

[17] Tao Yu-Fei, Papadias Dimitris, Sun Ji-Meng. The TPR*-tree: An optimized spatio-temporal access method for predictive queries//Proceedings of the Very Large Data Base (VLDB'03). Berlin, Germany, 2003: 790-801

[18] Berchtold S, Keim D, Kriegel H P. The X-tree: An index structure for high-dimensional data//Proceedings of the Very Large Data Base (VLDB'96). Bombay, India, 1996: 28-39

[19] Lazaridis I, Mehrotra S. Progressive approximate aggregate queries with a multi-resolution tree structure//Proceedings of the Special Interest Group on Management of Data (SIGMOD'01). Santa Barbara, USA, 2001: 401-412

[20] Ciaccia P, Patella M, Zezula P. M-tree: An efficient access method for similarity search in metric spaces//Proceedings of the Very Large Data Base (VLDB'97). Athens, Greece, 1997: 426-435

[21] White D A, Jain R. Similarity indexing with the SS-tree// Proceedings of the International Conference on Data Engineering (ICDE'96). New Orleans, Louisiana, 1996: 516-523

[22] Havran V, Bittner J. On improving KD-trees for ray shooting// Proceedings of the International Conference in Central Europe on Computer Graphics (WSCG'02). Pilsen, Czech, 2002: 209-216

[23] Chen S, Ooi B C, Tan K L, Nacismento N. ST^{2}B-tree: A self-tunable spatio-temporal B$^+$-tree index for moving objects //Proceedings of the Special Interest Group on Management of Data (SIGMOD'08). Vancouver, Canada, 2008: 209-216

[24] Nievergelt J, Hinterberger H. The grid file: An adaptable, symmetric multikey file structure//Proceedings of the Special Interest Group on Management of Data (SIGMOD'84). New York, USA, 1984: 38-71

[25] Zhu Rui, Wang Bin, Wang Guo-Ren. Indexing uncertain data for supporting range queries//Proceedings of the Web-Age Information Management (WIAM'84). Macau, China, 2014: 72-83

[26] Angiulli F, Fassetti F. Indexing uncertain data in general metric space. IEEE Transactions on Knowledge and Data Engineering, 2009, 24(9): 234-248

ZHU Rui, born in 1982, Ph. D. candidate. His research interests include sensor data management and uncertain data management.

WANG Bin, born in 1972, Ph. D., associate professor. His main research interests include big data management.

YANG Xiao-Chun, born in 1973, Ph. D., professor, Ph. D. supervisor. Her research interests include string data and parallel computing.

WANG Guo-Ren, born in 1966, Ph. D., professor, Ph. D. supervisor. His research interests include database, big data management.

Background

Along with the development of information technology, the amount of data generated by every walk of life becomes extremely huge, and the big-data management becomes a very hot topic. Because the scale of data is large, it is difficult to effectively store and real-time process them in an effectively way. As a novel solution, probabilistic model could effectively decrease the scale of data. It could use a probabilistic data for expressing a group of objects. Accordingly, the key problem caused by big data could be effectively solved. Thus, it is significant to manage probabilistic data.

As an important issue, range query over probabilistic data has been deeply studied. Their novel solution is to use a data structure to express the feature of object's probability density function and use this data structure for probabilistic pruning. Their problem the weakly probabilistic pruning ability, highly data structure constructing and updating cost.

In order to solve these problems, in this paper, we firstly study the problem of probabilistic range query over streaming probabilistic data. we propose a new index called HGD-Tree to manage probabilistic data in the stream. It could not only keep the balance of the tree but also has a strong updating power. And then, we propose a new way to construct the summary of each probabilistic data. Our result shows that both the constructing time and the pruning ability are all stronger than the previous works. Lastly, we evaluate the performance of HGD-tree.

This work is supported by the National Basic Research Program (973 Program) of China under Grant No. 2012CB316201, the National Natural Science Foundation of China under Grant Nos. 61272178, 61572122, 61173031, 61129002, 61532021, U1401256, and the National Natural Science Foundation of China for Outstanding Young Scholars (No. 61322208).