第48卷 第1期
Vol.48 No.1
计 算 机 工 程
Computer Engineering
2022年1月
January 2022
·先进计算与数据处理·
文章编号：1000-3428(2022)01-0127-08 文献标志码：A 中图分类号：TP391

基于位置的路网Skyline查询处理研究

白 梅，苌仕涵，王习特
（大连海事大学 信息科学技术学院，辽宁 大连 116000）

摘 要：基于位置的路网Skyline查询可根据用户的需求及用户所处的位置，从大量数据中快速返回给用户期望的数据，但已有的道路网络技术需要计算大量的路网距离及数据点间支配关系的运算，导致查询效率较低。提出一种基于路网数据点的倒排索引查询算法DSR。通过计算少量数据点的路网距离求得最终结果，减小路网距离计算的代价，从而加快数据点间支配关系的判定，提升查询效率。在此基础上，在数据点更新情况下给出算法的动态维护，仅通过维护少量数据，DSR即可以快速地计算出Skyline集合。实验结果表明，与SSI、BSS等算法相比，该算法具有较高的查询效率，且时间性能明显提升。

关键词：Skyline查询；路网；数据点更新；倒排索引；查询处理

开放科学（资源服务）标志码（OSID）：

中文引用格式：白梅，苌仕涵，王习特.基于位置的路网Skyline查询处理研究[J].计算机工程，2022，48(1)：127-134.

英文引用格式：BAI M，CHANG S H，WANG X T.Research on location-based Skyline queries processing in road network[J].Computer Engineering，2022，48(1)：127-134.

Research on Location-based Skyline Queries Processing in Road Network

BAI Mei，CHANG Shihan，WANG Xite
（School of Information Science and Technology，Dalian Maritime University，Dalian，Liaoning 116000，China）

【Abstract】Location-based skyline queries can quickly return the expected information from massive data according to the user's needs and the user's location.However，the existing road network technologies require enormous calculations of road network distance and dominant relations between data points，which reduces the query efficiency.To solve the problem，an algorithm named DSR for inverted index query is proposed based on road network data points.The algorithm can get the final result by calculating the road network distance of a small number of data points，which greatly reduces the cost of calculating road network distance.The determination of the dominant relations between data points is also accelerated，and the query efficiency is improved.On this basis，the dynamic maintenance of the algorithm in the case of data point update is given.DSR can quickly calculate Skyline set only by maintaining a small amount of data.The experimental results show that compared with SSI，BSS and other algorithms，this algorithm displays higher query efficiency，and its performance grows with the data size.

【Key words】Skyline query；road network；data point update；inverted index；query processing
DOI：10.19678/j.issn.1000-3428.0060559

0 概述

Skyline是解决数据库中多目标决策问题的重要手段，其可以在多维数据查询中为用户推荐较好的选择，方便用户做出决策。具体地，给定一个多维数据点的集合P及数据点p，$p'\in P$，如果点p在每个维度中的值都比p'好，并且在至少一个维度上更好，则点p支配p'。所有不被集合中的其他点支配的点被称为Skyline点。

近年来，随着全球定位系统和无线通信技术的发展，基于位置的服务（LBS）逐渐地被越来越多的用户所使用，LBS系统的主要目的是获取用户所在的位置，并向用户提供即时信息以便用户做出决策。例如，在导航系统中，旅客在城市中是想要找到满足自己偏好的酒店。其中用户到酒店的距离是空间属性，酒店的价格、评分是非空间属性。但现有的基于位置的查询只是针对单一维度进行的查询，解决诸如“查找距离查询位置q最近的旅社”类似问题，显

基金项目：国家自然科学基金（61602076，61702072，61976032）；中国博士后科学基金面上项目（2017M611211，2017M621122，2019M661077）；辽宁省自然科学基金（20180540003）；赛尔网络下一代互联网技术创新项目（NGII20190902）。
作者简介：白 梅（1986—），女，副教授、博士，主研方向为数据管理、云计算、数据查询优化；苌仕涵，硕士研究生；王习特，副教授、博士。
收稿日期：2021-01-11 **修回日期：**2021-02-28 **E-mail：**changshihan09@163.com

然,这样单一的查询无法满足用户多样化的需求。

因此,在路网环境下基于位置的Skyline查询成为LBS研究的热点。近年来,一些学者针对路网数据中的Skyline问题进行了研究[1-3]。文献[1]提出一种SSI算法,主要是针对路网上每个数据点p计算一个Skyline区域,只要查询位置q位于该区域,p就是一个Skyline点。然而,该方法的问题在于提前建立的索引代价过大,并且在数据点更新时就不再适用。文献[2]提出一种最近邻算法,通过广度优先遍历的方式找到Skyline点,最近邻算法的问题在于用户必须提前给定阈值边界,否则会一直进行下去,直到遍历完整个路网。文献[3]提出一种Skyline查询方法,该方法基于空间数据点构建网络Voronoi图,并对查询点建立查询凸包,通过网络Voronoi图的性质与查询区域的位置关系对数据集约减,从而节省路网距离计算的开销。然而,该方法在构建Voronoi图索引时代价过大,当数据点更新时就不再适用。

本文设计一种对路网上的数据点进行管理和更新的倒排索引结构,并提出路网上的Skyline算法DSR。倒排索引结构可用于管理路网数据点的维度,并且根据数据点在各个维度的值由优到劣进行排序,使用该索引可以对不需要计算路网距离的数据点进行过滤,同时加快数据点间支配关系的判定。DSR算法采取有效的扫描策略,只计算小部分数据点的路网距离即可高效地进行支配判定,在数据点更新的情况下给出DSR算法的动态维护。最终采用真实的道路网数据,对本文算法的高效性以及有效性进行验证。

1 相关研究

近年来,在路网环境下的Skyline查询处理引起研究人员的关注。CHOMICKI等[4]介绍了Skyline的相关概念,并提出一些基础计算方法。DONG等[5]提出组合式Skyline的求解算法,并给出相关的更新算法。文献[6]提出空间Skyline的概念,即将路网距离换为欧式距离作为求解时考虑的一个维度。

DENG等[7]提出在路网上进行Skyline查询的方法,给出了相对应的算法,并围绕此问题,提出了相对应的协同扩张、下界约束等算法。然而,算法在大型路网上的距离计算成本过大。SAFAR等[2]提出在路网上运用NN算法,以查询点为起点进行广度优先遍历,创建候选表,每扫描到一个数据点就与候选表的数据点进行比较,直到达到阈值边界。

另外,文献[1]提出的SSI算法,提前是为每个路网上的数据点计算一个查询区域Skyline scope,只要发起查询的位置在该区域上,该数据点就是一个Skyline点。然而,该算法提前建立索引代价过大,且每当发生数据点的更新时,整个索引需要重新计算。文献[8-10]的算法都是在该索引基础上进行扩展而来。

文献[1]在针对路网环境下基于位置的Skyline查询提出了Cdε-SQ(Continuous d-ε-Skyline Query)和CKnn-SQ(Continuous K nearest neighbor-Skyline Query)两种算法。Cdε-SQ的主要目的是为了节省路网距离计算的开销,而CKnn-SQ的主要目的是返回距离查询位置最近的k个Skyline数据点。然而,文献[1]算法在路网距离计算方面存在一定缺陷,故而查询效率较低。随后,HUANG等[11]又提出了在动态路网上的Skyline查询。

文献[9-10,12]介绍了多重属性道路网(MCN)中传统Skyline查询问题(MCN Skyline)。MCN中每条边具有多维属性,例如该边的长度、经过该边所需要的时间、经过该边的费用等。

LIN[13]等与ZHOU[14]等研究了路网环境下针对移动对象的Skyline查询。该查询假设路网上所有的查询对象的位置都在不断变化,查询结果随时都在更新。XU[15]等研究了路网环境下针对用户发起查询位置的隐私保护问题。TAO[16]等研究了流环境中的Skyline计算。HUANG[17]等解决了在具有时变信息的动态道路网络中有效处理天际线内连续查询的问题。LIU[18]等提出一种ZINC模型,结合ZB树的优势,并支持高效的Skyline计算。

2 问题描述

本节给出路网下基于位置的Skyline问题的形式化定义。表1为本文所使用的符号。

表1 符号含义

Table 1 Meaning of symbols

符号	含义
D_{cont}	P的非空间属性上的维度空间
d_{spatial}	P的空间属性上的维度
$p_1[d_i]$	数据点p_1在维度d_i上的值
$p_1 \prec_q p_2$	p_1空间支配p_2
R	Skyline点结果集合
q	查询位置
p_1,p_2	路网上的数据点,P为数据点集合
$d_i$count	已经在维度d_i上扫描过的数据点个数
$p_i$num	数据点p_i被扫描过的次数

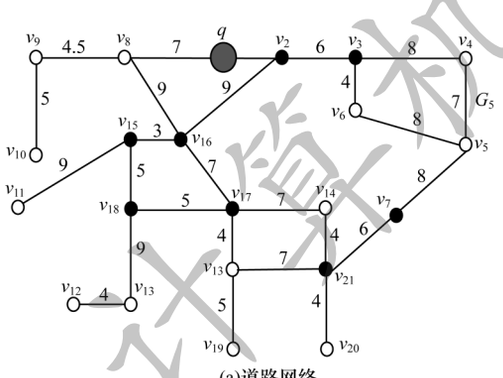
2.1 问题定义

道路网络建模为无向加权图$G=(V,E,W)$,其中:V是道路顶点集合;E是道路边集合;W是边长的集合。本文中讨论的距离都是路网上的最短路径

距离。

在本文中，要求数据点的位置不能重合，给定数据点$p\in P$，每个数据点由多个非空间维度和一个空间维度组成，其中P的非空间属性集合为$D_{\text{cont}}=\{d_1, d_2, \cdots, d_m\}$，数据点可以表示为$p=<p[d_1], p[d_2], \cdots, p[d_m]>$，其中$p[d_i]$表示$p$在维度$d_i$上的值。$P$的空间属性表示为$d_{\text{spatial}}$，任意数据点$p$的空间属性值是$p$的地理位置坐标，用一个三元组$<v_i, v_j, \text{distance}>$表示，其中，$v_i$和$v_j$表示$p$的路网中所在边的两端顶点，distance表示$p$到道路顶点$v_i$的距离，$\text{dis}(p, q)$表示数据点$p$和$q$的路网上的最短距离。

图1给出Skyline查询的示例，其中每个数据点对应一条旅社记录，具有评分、价格2个维度（注意，在该示例中用户希望评分越高越好和价格越低越好）。所以，根据图1中的旅社信息列表来评估，数据点p_1支配p_4，因为p_1评分维度和p_4相同，在价格维度优于p_4。



(a)道路网络

数据点	价格	评分	空间属性
p_1	130	6	v_9, v_{10}, 3
p_2	137	6	v_9, v_8, 0.5
p_3	139	4	v_8, v_2, 1
p_4	137	6	v_3, v_2, 1
p_5	230	8	v_3, v_4, 2
p_6	135	7	v_4, v_5, 2
p_7	134	7	v_5, v_7, 4
p_8	190	8	v_{15}, v_{18}, 2
p_9	136	8	v_{16}, v_{17}, 0.5
p_{10}	140	6	v_{18}, v_{17}, 0.5
p_{11}	230	6	v_{18}, v_{17}, 1
p_{12}	190	7	v_{17}, v_{18}, 0.5
p_{13}	140	8	v_{18}, v_{17}, 1
p_{14}	240	6	v_{17}, v_{18}, 0.5
p_{15}	280	8	v_{16}, v_{17}, 1
p_{16}	190	9	v_{14}, v_{13}, 3
p_{17}	80	3	v_{14}, v_{11}, 3
p_{18}	140	5	v_{17}, v_{19}, 5
p_{19}	230	6	v_7, v_{21}, 1
p_{20}	330	9	v_{21}, v_{20}, 2

(b)数据点信息

图1　Skyline查询示例

Fig.1　Example of Skyline query

例1　如图1(b)所示，p_1的地理坐标为(v_9, v_{10}, 3)，表示p_1在v_9、v_{10}边上，p_1到v_9的距离为3。查询位置q的地理坐标为(v_2, v_8, 4)。

与传统的Skyline查询不同，在道路网络中处理Skyline查询需要考虑数据点的空间属性。如图1所示，在z市中一共有20家酒店(p_1~p_{20})，用户在q位置召开会议，他想找到合适的酒店来给参会人员安排住宿，希望找到在价格和评分方面有优势并且距离开会地点近的酒店。在这种情况下，当用户从位置q发起查询时，需要首先计算每个酒店到用户的距离，然后结合价格、评分，来找到q的Skyline结果集。可以看出，p_4的价格和评分优于p_3(见图1)，并且因为p_4比p_3更接近q(即p_4在空间维度上更好)，则p_4支配p_3。最终，基于位置q返回满足用户条件的酒店集合是$\{p_1, p_4, p_5, p_6, p_7, p_{12}, p_{13}, p_{16}, p_{17}\}$。

定义1(空间支配)　给定路网G上的数据点集合P和查询位置q，$p_1, p_2\in P$，p_1关于q空间支配p_2(记作$p_1\prec_q p_2$)需要满足以下2个条件：

1)$\forall d_i\in D_{\text{cont}}\cup\{d_{\text{spatial}}\}$，$p_1[d_i]$不差于$p_2[d_i]$；

2)$\exists\, d_j\in D_{\text{cont}}\cup\{d_{\text{spatial}}\}$，$p_1[d_j]$好于$p_2[d_j]$。

定义2(路网Skyline集合)　给定路网G上的数据点集合P和查询位置q，道路网Skyline点集合就是不被其他数据点关于位置q空间支配的点的集合，记作$\text{SKY}(q, P)=\{p_2 \mid p_1, p_2\in P, \nexists p_1\prec_q p_2\}$。

例2　利用图1的数据集举例说明，若不考虑其空间属性，得到的Skyline集合为$\{p_1, p_6, p_7, p_9, p_{16}, p_{17}, p_{20}\}$，但在加入了空间属性后，得到的Skyline集合变为了$\{p_1, p_3, p_4, p_6, p_7, p_9, p_{16}, p_{17}\}$。

2.2　G-tree索引

本文采用G-tree索引[19]来快速计算路网上任意数据点到查询位置间的最短路网距离。

G-tree是将道路网递归地划分为子网络，并在子网络的顶部构造树结构索引，其中每个G-tree节点都对应一个子网络。针对图1对应的路网结构，进行的图划分结构如图2(a)所示，建立好的G-tree索引如图2(b)所示。其中，各非叶节点内都存储了该节点对应子图内边界点集合以及相应的距离矩阵，各叶节点内存储了该子图内边界点到该子图内其余路网节点的距离矩阵。

定义3(边界点)　给定图G的子图G_i，节点$b_1\in V_i$，节点b_2不属于V_i。若满足$\exists(b_1, b_2)\in E$，则称节点b_1、b_2为边界点，$B(G_i)$为G_i内的边界点集合。

给定数据点$p\in P$，以及查询位置q，下面介绍借助G-tree索引，求出$\text{distance}(p, q)$。

例3　图2展示了如何计算从q到p_{20}的距离，首先通过距离排序列表找到距离q和p_{20}最近的边界点v_2和v_{21}。随后通过哈希表(将边界点映射到叶子节

点)找到叶子节点G_3(对于v_2)和G_6(对于v_{21}),它们的最小公共祖先节点是G_0,通过节点G_3、G_1、G_2、G_6来计算最短路网距离,图2(c)中每个元素代表<v_i,distance(q,v_i)>。通过动态规划算法最终可以得到distance(q,p_{20})=35,如图2(c)所示,最短路径包括q、v_2、v_3、v_{21}、p_{20}。

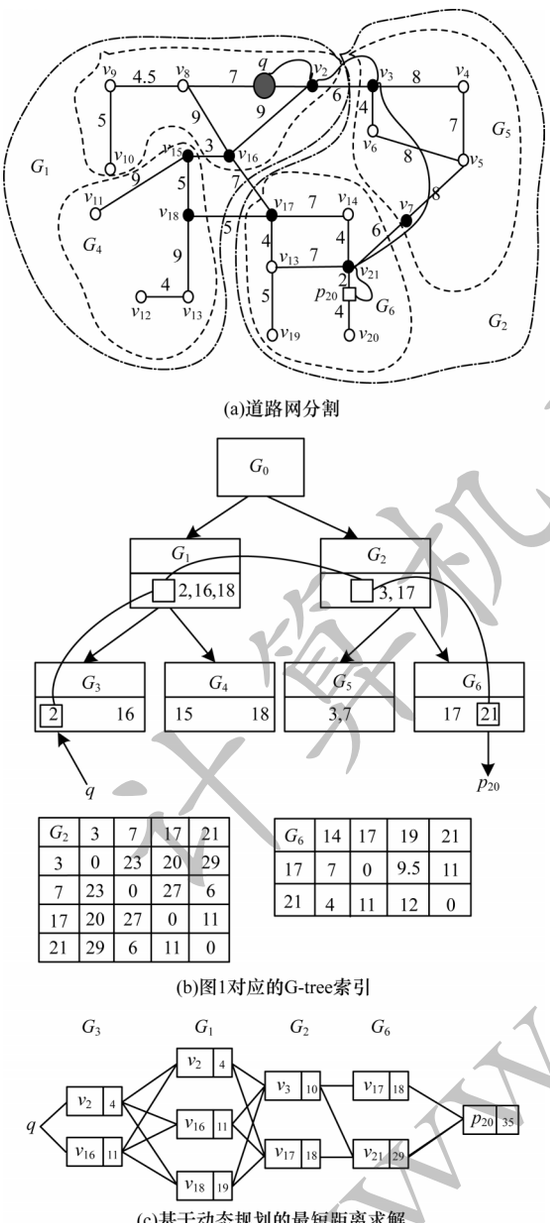


图2 基于G-tree索引的路网距离

Fig.2 Distance of road network based on G-tree index

3 查询处理算法DSR

本节介绍管理路网数据点的倒排索引,利用该索引提出DSR查询处理方法对数据提前进行处理,并过滤剪枝数据集,使得进行Skyline查询时不必扫描整个路网数据集,避免大量计算路网距离的开销。

3.1 管理路网数据点的倒排索引

对于路网上的数据点,采用特殊的倒排索引结构进行管理,在建立倒排索引时只针对将非空间维度的属性映射到倒排索引中,对于距离维度,在使用时才进行计算。具体地,对数据点在每一个维度上的值都按照从优到劣的顺序进行排序,距离维度初始时为空。

扫描策略:由于数据点可能在不同维度上表现各不相同,因此在扫描过程中,用count变量表示已经在该维度上扫描过的数据点个数。对每一个数据点p_i维护一个num变量,表示数据点被扫描过的次数。每次扫描时都选择count值最小的维度,按照数据点由好到坏的顺序进行扫描(若数据点值相同,则进行支配关系的判定)。

值得注意的是,针对距离维度,数据点的扫描顺序是在路网上从查询点q开始进行广度遍历,按照距离q由近及远的顺序进行处理,建立堆H用于存储距离维度已处理的信息。初始H={<q,0>},每次取堆首元素处理,若处理的元素是查询点或路网节点,则找到与该节点相连的数据点或路网节点重新加入堆中并按与q的路网距离进行排序。若堆首元素是数据点,直接进行Skyline结果判定,把该数据点加入距离维度,同时距离维度的count值加1。

例4 如图3所示,当第一次扫描到距离维度时,建立堆H,H={<q,0>}。初始处理q,找到与q直接相连的路网节点及数据点v_2、p_3和v_8,将其加入堆中,H={<v_2,2>,<p_3,4>,<v_8,5>}。然后处理v_2,此时出堆的元素为路网节点,找到与v_2相连的数据点以及路网节点加入H中,此时,H={<p_3,4>,<v_8,5>,<p_4,7>,<v_3,6>,<p_9,9>,<v_{16},11>}。

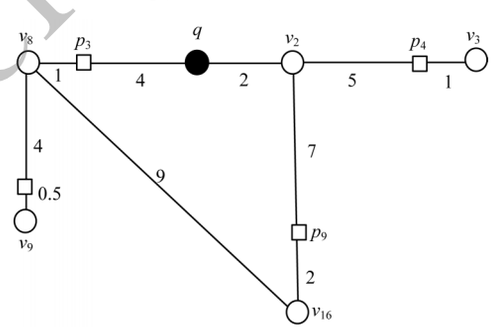


图3 距离维度扫描示例

Fig.3 Example of distance dimension scanning

随后处理p_3,此时出堆的元素为数据点,即距离维度第一次扫描到的数据点是p_3,进行Skyline结果判定,把该数据点加入距离维度,同时距离维度的count值加1。

扫描结束点:当数据点p_i在所有维度上都被扫描过一次时,称p_i为扫描结束点。

定理1 对于数据点p_i,若其被扫描过的次数与维度数相同,则对于未扫描过的数据点p_j(p_j的扫描次数为0)都一定被p_i支配。

证明　根据支配的定义，假设p_j不被p_i支配，则$\exists d_j \in D_{cont} \cup \{d_{spatial}\}$，$p_j[d_j]$好于$p_i[d_j]$。即在维度$d_j$上$p_j$的扫描次序在$p_i$之前，违背了给定的条件，在所有维度上$p_i$的扫描次序优于$p_j$。得证。

定理2　给定维度d_i，对于扫描到的在维度d_i上的数据点p_i，若p_i不被d_i上已扫描过的Skyline点支配，则p_i是Skyline点。

证明　若$\exists p_j$支配p_i，根据支配的定义，p_j支配p_i，需满足条件之一为$\forall d_j \in D_{cont} \cup \{d_{spatial}\}$，$p_j[d_j]$不差于$p_i[d_j]$；然而，定理2给定条件中$p_j$在维度$d_i$上的扫描次序在$p_i$之后，即$p_i$在维度$d_i$上优于$p_j$，与支配定义矛盾。得证。

如图4所示，当扫描到价格维度的数据点p_1时，只将其与p_{17}进行空间支配关系比较即可，而不用与p_{16}、p_{20}比较，大幅提高了计算效率。因此，对每一个维度d_i建立结果集R^i，存储该维度上已扫描过的所有Skyline点。

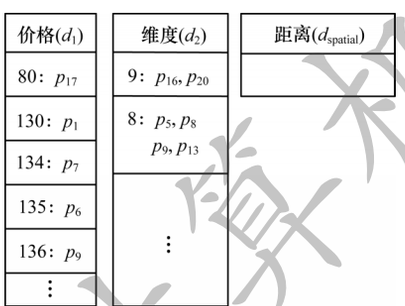


图4　初始倒排索引应用示例

Fig.4　Example of initial inverted index application

数据点处理策略：根据扫描策略可知，当前维度扫描到的数据点只有可能被当前维度结果集R^i中的数据点支配，同时，若数据点p_i在当前维度确定为Skyline点时，只处理1次，在其他维度再次扫描到p_i时，只对p_i的计数器加1，每个数据点第一次被扫描时，计算其距离维度上的值。

3.2　DSR算法

本节描述算法DSR查询的全过程。首先对整个数据集建立倒排索引，随后针对维度$d_i \in D_{cont} \cup \{d_{spatial}\}$建立结果集$R^i$，然后根据扫描策略开始扫描维度（若扫描维度为$d_{spatial}$时，借助堆$H$进行广度遍历），在得到待处理数据集$P_i$后，利用G-tree索引，计算其中数据点的距离维度，随后与R^i中的数据点进行支配关系的判定，将不被支配的数据点加入到结果集R中，最后输出结果集。

算法1　DSR算法描述

输入　路网$G=(V,E,W)$，索引r

输出　Skyline结果集R

1.初始化每个维度上的结果集$R^i=\varnothing$；
2.对所有维度$d_i \in (D_{count} \cup d_{spatial})$建立倒排索引，$d_{spatial}$初始为空；
3.建立数据维度初始堆H={<q,0>}；
4.While(计算未结束)
5.根据扫描策略得到count值最小的维度d_i；
6.If($d_i = d_{spatial}$)
7.处理H得到待处理数据集P_i；
8.Else
9.在非空间维度d_i上得到扫描数据集P_i；
10.计算出P_i中数据点的路网距离；
11.d_i.count= d_i.count +$|P_i|$；
12.Foreach($p_i \in P_i$)
13.If(p_i不被P_i中其他数据点空间支配)
14.If(p_i.num=0)
15.If(p_i不被R^i中的数据点空间支配)
16.将数据点p_i加入当前维度的结果集R^i；
17.p_i.num ++；
18.Else
19.p_i.num ++；
20.If(p_i.num = $|D_{count}|$)
21.扫描结束，退出循环；
22.对所有R^i取并集得到最终结果集R；
23.return R；

算法1第2行通过建立倒排索引来维护数据。第3行~第22行是算法主体，第5行根据count值确定扫描维度d_i，若d_i不是距离维度，则在得到待处理数据集后，算法第10行使用基于G-tree索引的动态规划算法来计算P_i中数据点的最短路径距离，其计算代价为$O(\mathrm{lb}\, f \times |V|)^2$，$V$是路网上节点的个数，$f$为G-tree上节点的分支数量。算法第12行~第23行是算法的主体，每扫描到一次数据点，若该数据点不被P_i中其他数据点空间支配且num值为0，则将数据点加入到R^i中且num值加1，当有一个数据点的num值与维度数相同时，此时，该数据点为扫描结束点，结束算法。DSR算法主要受$|R^i|$的影响，$|R^i|$为在维度d_i上的结果集，这部分算法的计算代价为$O(|R^i| \times |d|)$。算法第25行返回Skyline结果集R，算法整体时间复杂度为$O(\mathrm{lb}\, f \times |V| \times |R^i| \times |d|)^2$。

例5　在图5(a)中，在初始时各维度计数值分别为0、0、0。第一次扫描顺序为价格维度，扫描到的数据点为p_{17}，此时维度计数器被更新为1、0、0，p_{17}扫描次数更新为1。接着处理评分维度，扫描到的数据点为p_{16}、p_{20}，维度计数器被更新为1、2、0，p_{16}、p_{20}扫描次数更新为1、1。接着处理距离维度，堆H处理的第一个数据点是p_4，维度计数器被更新为1、2、1，p_4扫描次数更新为1。这样一直扫描下去，直到p_9的扫描次数更新为3，此时p_9成为扫描结束点，算法结束，最终结果集$R=\{p_1, p_3, p_4, p_6, p_7, p_9, p_{16}, p_{17}\}$。从图5(b)可以看到，DSR算法仅计算了少部分数据点的路网距离。

维度计数	扫描维度	扫描数据点及其计数	对应维度结果集
0, 0, 0	d_1	$p_{17.num}$=1	$\{p_{17}\}$
1, 0, 0	d_2	$p_{16.num}$=1, $p_{20.num}$=1	$\{p_{16}\}$
1, 2, 0	$d_{spaital}$	$p_{3.num}$=1	$\{p_3\}$
1, 2, 1	d_1	$p_{1.num}$=1	$\{p_1, p_{17}\}$
2, 2, 1	$d_{spaital}$	$p_{4.num}$=1	$\{p_3, p_4\}$
2, 2, 2	d_1	$p_{7.num}$=1	$\{p_1, p_{17}, p_7\}$
3, 2, 2	d_2	$p_{5.num}$=1, $p_{8.num}$=1 $p_{9.num}$=1, $p_{13.num}$=1	$\{p_{16}, p_9\}$
3, 6, 2	$d_{spaital}$	$p_{2.num}$=1, $p_{9.num}$=2	$\{p_3, p_4, p_9\}$
3, 6, 4	d_1	$p_{6.num}$=1	$\{p_1, p_{17}, p_7, p_6\}$
4, 6, 4	d_1	$p_{9.num}$=3	$\{p_1, p_{17}, p_7, p_6, p_9\}$

(a)扫描过程

价格(d_1)
80: p_{17}
130: p_1
134: p_7
135: p_6
136: p_9
⋮

维度(d_2)
9: p_{16}, p_{20}
8: p_5, p_8, p_9, p_{13}
⋮

距离($d_{spatial}$)
4: p_3
7: p_4
9: p_2, p_9

(b)倒排索引扫描过程

图5 扫描策略应用示例

Fig.5 Example of scanning strategy application

4 DSR算法动态维护

DSR算法的动态维护主要是在路网环境下，当数据点发生更新时DSR算法的动态维护。动态维护主要包括新增数据点的处理以及失效数据点的处理两个操作。

4.1 失效数据点的处理

定理3 假设数据点p_i在维度$d_i \in D_{total}$上已经被扫描过，若数据点p_i只被数据点p_{old}空间支配，且p_{old}不是扫描结束点，则p_i在维度d_i上的值一定不好于p_{old}，且优于当前扫描位置处的数据点。

证明 若只满足$p_{old} \prec_q p_i$，则$\forall d_i \in D_{cont} \cup \{d_{spatial}\}$，$p_{old}[d_i]$不差于$p_i[d_i]$，即在维度$d_i$上，$p_{old}$扫描位置一定不差于$p_i$，假设当前扫描位置处数据点为$p_j$，若$p_j \prec_q p_i$不成立，则需满足$\exists d_i \in D_{cont} \cup \{d_{spatial}\}$，$p_i[d_i]$好于$p_j[d_i]$，即在维度$d_i$上，$p_i$扫描位置好于$p_j$。得证。

若p_{old}不在现有的各个维度的结果集合并集中，则直接在倒排索引中删去。如果p_{old}属于结果集合，则首先需要找到所有只被p_{old}空间支配的数据点，具体做法是根据定理3，对所有已扫描过p_{old}的维度上表现不好于p_{old}但是优于当前扫描位置的数据点取交集，加入到候选集合R'中。再将R'中的数据点与R中的Skyline点进行空间支配关系的判定，R'中不被R的任意Skyline点空间支配的数据点即是只被p_{old}空间支配的数据点。随后，判断p_{old}是否为扫描结束点。若p_{old}是扫描结束点，则p_{old}失效后算法未结束，需要找到新的扫描结束点p_{end}。

例6 如图6所示，若被删除的数据点是p_9，则p_9是Skyline点，若此时的扫描位置为p_1，对在所有维度上表现不优于p_9但是优于p_1的数据点取交集，如图6中的黑框部分所示，对黑框部分中的数据点取交集，将不在R中的数据点加入到候选集R'中，得到$R'=\{p_5, p_8, p_{13}, p_{15}, p_{20}, p_{12}\}$，将$R'$中的数据点与$R$进行空间支配关系判定后，得到最终只被$p_9$支配的数据点为$p_5$、$p_8$、$p_{13}$、$p_{20}$。随后，由于$p_9$是扫描结束点，因此算法继续运行直到找到新的扫描结束点，输出最终结果集。

价格(d_1)
80: p_{17}
130: p_1
134: p_7
135: p_6
136: p_9
…

维度(d_2)
9: p_{16}, p_{20}
8: p_5, p_8, p_9, p_{13}
7.5: p_{15}, p_{11}
7: p_6, p_7, p_{12}

距离($d_{spatial}$)
4: p_3
7: p_4
9: p_2, p_9
10: p_5, p_{13}

图6 动态维护应用示例

Fig.6 Example of dynamic maintenance application

4.2 新插入数据点的处理

本文从以下2个方面考虑p_{new}本身是否会成为空间Skyline点以及p_{new}是否会支配掉结果集中的点：1)如果p_{new}不受任何维度上的结果集中的数据点支配，则p_{new}是空间Skyline点，将其加入到该维度对应的结果集中，反之，p_{new}被支配掉；2)根据定理1，首先需要将结果集中的数据点与p_{new}进行支配关系的判定(不在结果集中的数据点一定被空间支配)，因此需要将各个维度上结果集的并集中的数据点与p_{new}进行支配关系的判定(不在结果集中的数据点一定被空间支配)，随后在结果集中过滤掉所有被p_{new}空间支配的数据点。

算法2描述了DSR算法更新维护过程的细节。当数据点发生更新时，算法需要找到所有仅只被p_{old}空间支配的数据点，具体做法是：首先，找到候选集合R'中所有只被p_{old}空间支配的数据点，其次，每找到一个数据点就与R比较，判断其是否被R中的其他数据点空间支配。如果只能被p_{old}空间支配，则将其从候选集中取出添加到结果集中。算法第17行~第23行描述了在有新插入数据点p_{new}的情况下DSR的动态维护。

算法2 DSR算法动态维护

输入 路网$G=(V, E, W)$，查询点q，索引，查询点q，更新的数据点p_{new}，失效的数据点p_{old}

输出 Skyline结果集R

1.While(计算未结束)

2.删除失效数据点p_{old}//处理失效数据点p_{old}

3.If(p_{old}属于结果集)

4.找到当前扫描位置对应的数据点p_{now}

5.将各个维度上表现不好于p_{old}但是优于p_{now}的非Skyline数据点取交集，加入到候选集合R^t中

6.Foreach $p_i \in R^t$

7.If(p_{old}空间支配p_i)

8.If(p_i不被R^t中的数据点空间支配)

9.将 p_i 加入到R中
10.If（p_{old} 是扫描结束点）
11.继续执行算法1的第12行~第21行
12.//处理新数据点 p_{new}
13.If（p_{new} 不被R中的数据点空间支配）
14.将 p_{new} 添加到R中
15.Foreach $p_i \in R$
16.If（p_{new} 空间支配 p_i）
17.删除 p_i
18.return R；

算法2第3行~第11行描述了当倒排索引中的数据点失效时的情形，若 p_{old} 属于结果集合，根据算法第3行~第11行，算法找到只被 p_{old} 支配的数据点的计算代价为 $O(|R^i|)$。若 p_{old} 不属于结果集合，算法只需要 $O(1)$ 次操作，第13行~第17行描述了在有新插入数据点 p_{new} 的情况下DSR的动态维护。在处理新插入数据点 p_{new} 时，需要判定 p_{new} 是否被 R 中的数据点支配，在最坏情况下，要将 p_{new} 与 R 中所有数据点进行比较，计算代价为 $O(|R^i|\times|d|)$，随后，删除掉所有被 p_{new} 支配的数据点，DSR算法动态维护的计算代价为 $O(|R^i|\times|d|)$。

5　实验结果与分析

本节主要验证所提DSR算法的性能。该实验环境配置为Inter Core i5 7300HQ 2.50 GHz CPU，8 GB内存，1T硬盘，Windows 10操作系统的PC。算法用C++语言编写。为验证本文所提算法的有效性，将SSI算法和BSS算法作为对比实验。

设定　对于G-tree，设定 f 为4，τ 为64。本文使用真实路网数据验证算法性能。真实数据采用的是两个真实的路网数据集，出处来自GitHub开源项目TransportationNetworks（网址为 https://github.com/bstabler/TransportationNetworks），第1个路线图是奥尔登堡罗德（德国的一个城市）网络，由约6 000个节点和7 000条边组成，第2个路线图是加利福尼亚公路网，其中包含21 048个节点和21 693条边。这些数据集的统计信息如表1所示。实验默认在道路网络上模拟生成1 000个对象。每个对象都有3个非空间属性，其值均匀分布在[0，100]范围内。实验参数如表2所示。

表2　实验参数

Table 2　Experimental parameters

参数	默认值	变化范围
数据点更新速度	1	1，10，50，100，150
数据点集规模	10^3	500，1 000，15 000，2 000

5.1　数据点数量变化对算法性能的影响

在第1组实验中，本文研究了非空间维度的变化对SSI、BSS算法以及本文DSR算法的性能影响。图7所示为数据点集规模变化的实验结果。可以看出，SSI算法要优于本文DSR算法以及BSS算法，这是因为SSI算法提前计算了所有数据点两两之间的空间支配关系以及互相之间的距离，只用确定查询位置 q 的位置，即可直接返回结果集。而DSR算法的效率远远优于BSS算法，这是因为DSR算法在映射后采用倒排索引进行扫描，在找到扫描结束点后，可以直接结束算法，节省了计算时间与计算成本。

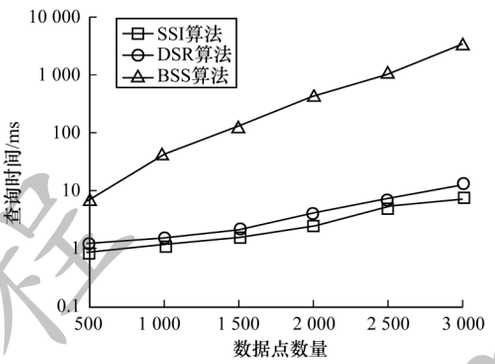


图7　数据点数量对查询时间的影响

Fig.7　Impact of data points number on query time

5.2　数据点更新速度对查询性能的影响

在第2组实验中，本文研究了数据点更新变化对BSS、SSI算法以及本文DSR算法的性能的影响。如图8所示，数据点更新速度从1到150。可以看出，随着数据点更新速度的增加，3种算法的性能都有所下降，但DSR算法远远优于BSS算法以及SSI算法，主要是由于后者需要进行大量的路网距离计算，而基于倒排索引的DSR算法只需要计算部分数据点的路网距离，在找到扫描结束点之后，就能直接返回整个Skyline结果集。

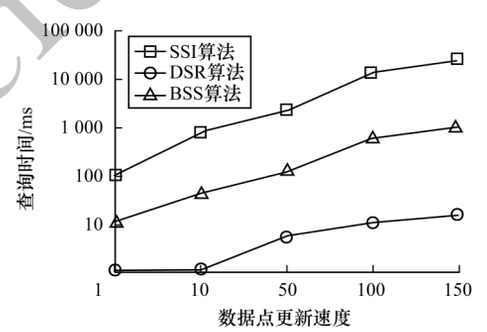


图8　数据点更新速度对查询时间的影响

Fig.8　Impact of data point update speed on query time

5.3　数据点数量对索引建立时间的影响

在第3组实验中，本文研究了数据点数量的变化对SSI、BSS算法及本文DSR算法性能的影响。如图9所示，数据点数量从范围为500到2 000。可以看到，随着数据点规模的增加，3种算法索引的建立时间性能都有所增加，但是DSR算法远远优于另外2种算法，主要原因是由于BBS算法需要进行大量的数据点路网距离计算，而SSI索引的建立需要计算好所有数据点之间的两两支配关系，以及遍历整个路网集，可以看到当数据集增加到一定规模后，计算

代价过大。而DSR算法不需要直接计算出数据点之间的距离,所需的倒排索引以及G-Tree索引的建立时间明显优于前两种算法。

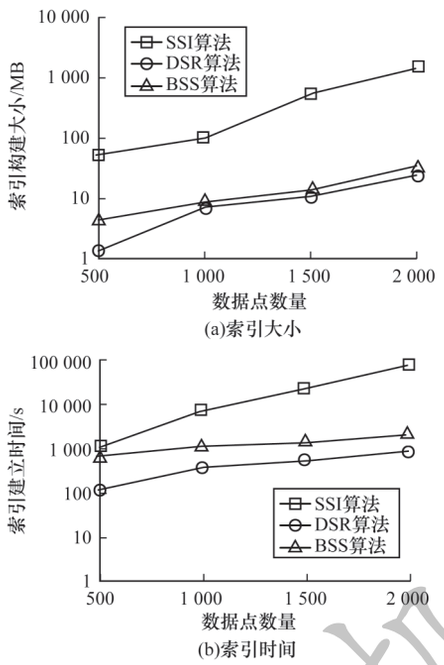


图9 数据点数量对索引建立时间及大小的影响

Fig.9 Impact of number of data points on index creation time and size

6 结束语

本文提出一种在路网环境下基于倒排索引的Skyline查询优化方法。将管理路网数据点的倒排索引引入Skyline查询,在查询前进行大量的过滤剪枝,从而提高实验的查询效率。在引入倒排索引的基础上,采用提前分组的形式对算法进行优化,提高对数据点过滤的有效性,加快算法的计算速度。实验结果验证了本文算法的高效性。下一步将运用DSR算法从数据集中快速返回给用户可控数量的Skyline结果,以达到提高算法查询效率、帮助用户快速决策的目的。

参考文献

[1] HUANG Y K,CHANG C,LEE C. Continuous distance-based Skyline queries in road networks[J]. Pergamon,2012,37(7):611-613.

[2] SAFAR M,EL-AMIN D,TANIAR D. Optimized Skyline queries on road networks using nearest neighbors[J]. Personal and Ubiquitous Computing,2011,15(8):845-856.

[3] 李松,窦雅男,郝晓红,等. 道路网环境下K-支配空间Skyline查询方法[J]. 计算机研究与发展,2020,57(1):227-239.
LI S,DOU Y N,ZHANG X H,et al. K-domination space Skyline query method in road network environment[J]. Computer Research and Development,2020,57(1):227-239. (in Chinese)

[4] CHOMICKI J,GODFREY P,GRYZ J,et al. Skyline with presorting[EB/OL]. [2020-12-10]. https://www.researchgate.net/publication/4053418.

[5] 董雷刚,刘国华. 组合Skyline的求解与更新算法[J]. 计算机工程,2017,43(6):195-201,206.
DONG L G,LIU G H. Solving and updating algorithm of combined Skyline[J]. Computer Engineering,2017,43(6):195-201,206. (in Chinese)

[6] LI Q Y,ZHU Y Y. Skyline cohesive group queries in large road-social networks[C]//Proceedings of the 36th IEEE International Conference on Data Engineering. Washington D. C. ,USA:IEEE Press,2020:397-408.

[7] DENG K,ZHOU X F,SHEN H T. Multi-source Skyline query processing in road networks[C]//Proceedings of the 23rd IEEE International Conference on Data Engineering. Washington D. C. ,USA:IEEE Press,2011:796-805.

[8] MIAO X Y,GAO Y J,GAO S,et al. On efficiently answering why-not range-based Skyline queries in road networks[J]. IEEE Transactions on Knowledge and Data Engineering,2018,30(9):1697-1711.

[9] 李佳佳,臧寅旭,刘向宇,等. 面向时间依赖路网的空间索引方法[J]. 计算机工程,2019,45(5):127-134.
LI J J,ZANG Y X,LIU X Y,et al. Spatial index method for time-dependent road network[J]. Computer Engineering,2019,45(5):127-134. (in Chinese)

[10] HUANG Z Y,LU H. Continuous Skyline queries for moving objects[J]. IEEE Transactions on Knowledge and Data Engineering,2006,18(12):1645-1658.

[11] LI R H,QIN L,YE F H,et al. Finding Skyline communities in multi-valued networks[J]. The VLDB Journal,2020,29(6):1407-1432.

[12] LI R,YU J X,MAO R. Efficient core maintenance in large dynamic graphs[J]. IEEE Transactions on Knowledge and Data Engineering,2014,26(10):2453-2465.

[13] LIN X,XU J L,HU H B. Range-based Skyline queries in mobile environments[J]. IEEE Transactions on Knowledge and Data Engineering,2013,25(4):835-849.

[14] 周剑刚,秦小麟,张珂珩,等. 基于道路网多移动用户动态Skyline查询[J]. 计算机科学,2019,46(9):73-78.
ZHOU J G,QIN X L,ZHANG K Y,et al. Dynamic Skyline query of multiple mobile users based on road network[J]. Computer Science,2019,46(9):73-78. (in Chinese)

[15] XU J L,TANG X Y,HU H B,et al. Privacy-conscious location-based queries in mobile environments[J]. IEEE Transactions on Parallel and Distributed Systems,2010,21(3):313-326.

[16] TAO Y,PAPADIAS D. Maintaining sliding window Skylines on data streams[J]. IEEE Transactions on Knowledge and Data Engineering,2006,18(3):377-391.

[17] HUANG Y K,KAINZ W. Within Skyline query processing in dynamic road networks[J]. ISPRS International Journal of Geo-Information,2017,6(5):137-150.

[18] LIU B,CHAN C Y. ZINC:efficient indexing for Skyline computation[J]. Proceedings of the VLDB Endowment,2010,4(3):197-207.

[19] ZHONG R,LI G,TAN K L,et al. G-Tree:an efficient and scalable index for spatial search on road networks[J]. IEEE Transactions on Knowledge and Data Engineering,2015,27(8):2175-2189.

编辑 索书志