哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

数据库系统

论文概述

(2019年度春季学期)

姓　　名　朱明彦
学　　号　1160300314
学　　院　计算机学院
教　　师　高宏

计算机科学与技术学院

目录

第1章 解决的问题 **3**

第2章 采用的思想 **3**

2.1 数学模型方面 3

2.1.1 打车软件（BUA+QF Model） 4

2.1.2 Pokemon游戏（RUA+FCFS模型） 4

2.2 索引SCOB方面 5

2.2.1 符号定义 5

2.2.2 基本思想 6

2.3 算法TOAIN方面 6

第3章 基本算法描述 **6**

3.1 SCOB Index 6

3.2 TOAIN 6

第4章 算法分析 **7**

第5章 举例说明 **7**

论文概述

第1章 解决的问题

选择的论文 [1]为2018年PVLDB上面的一篇文章，**主要解决的问题是Road Networks上面动态的kNN查询**。

针对这个问题，作者主要有如下**三个方面**的工作：

1. 对于Road Networks上需要进行kNN查询的系统建立新的数学模型，并找到其中影响系统整体吞吐量的关键因素。
2. 建立了一个以Shortcut Graph为基础的索引，SCOB。
3. 设计了可以根据Update/Query的所用时间动态调整SCOB以最大化系统吞吐量的算法，TOAIN。

第2章 采用的思想

作者为了解决Road Networks上面动态的kNN查询，采用的思想是**先明确系统的目标（最大化系统的吞吐量），建立数学模型衡量寻找影响体统吞吐量的关键因素，对于关键的因素建立索引并利用设计的算法尽量提高关键因素影响下的部分，最终实现最大化吞吐量的目的**。

Notation	Description
(t_q, V_q)	expected/variance-of query time
(t_u, V_u)	expected/variance-of update time
λ_q, λ_u	query/update arrive rate
R_q	average query response time
R_q^*	average query response time bound, a QoS measure
λ_q^*	largest average throughput subject to an R_q^* constraint
T	update periodicity (under the QF model)
$m = \|\mathcal{M}\|$	number of objects
$G = (V, E)$	graph representing a road network
$d_G(u, v)$	shortest distance between u and v in G
$u \rightsquigarrow v$	a shortest path from u to v
$v \curvearrowright u$	a shortcut from u to v
$u_\downarrow$	the set of downhill objects of u

图 2.1: Notations

2.1 数学模型方面

作者主要考虑了两种不同的情况，分别对应于现实中两种不同的应用，即**如滴滴类似的打车软件系统和捕捉精灵球的Pokemon**。作者在考虑数学模型时，从**Query**和**Update**分别考虑

其影响，然后针对这两种不同的应用区分它们在Query和Update上面的差别，从而建立起来不同的数学模型。

以下两种应用都假设所有的查询到来是随机的，论文中为了方便假设Query符合泊松过程(Poisson Process)。

2.1.1 打车软件（BUA+QF Model）

BUA(Batch Update Arrival)模型是针对Update操作来说，其假设所有的元素都会进行Update操作，且都是在时间片的开始时刻进行，如果在某一个时间片里没有完成相应的Update操作，则直接舍弃未做操作，进行下一轮操作即可。**QF(Query First)是针对系统中队列模型来说的**，其假设Query在队列中的优先级高于Update。

基于上面的假设，在打车软件这样的系统中，对于每个Query是来自用户的，查询在其附近最近的K个车辆；每个Update是来自汽车的，其在指定间隔内向系统返回其最新的位置。这样，高优先级的Query可以尽量减少用户请求的查询的延迟，并且丢失少量Update使得某些车辆的信息具有很短距离的差距，所以这样的假设对于该应用是合理的。

利用在 [2]中的结论，可以得到式(1)，其中R_q为平均查询响应时间，t_q, V_q分别是查询时间的期望和方差，λ_q为Query到达的速率，即泊松过程中的参数。

$$R_q = \frac{\lambda_q \left(t_q^2 + V_q\right)}{2\left(1 - \lambda_{q^t q}\right)} + t_q \tag{1}$$

通过式(1)可以看到，**整体的查询响应时间随λ_q的增大而增大，这也与我们的直觉相符合，说明此模型具有一定的道理**。进而通过两个约束，**查询的响应时间不能超过用户能够忍受的最大值**和**在一个时间间隔内需要最少的用于Update的时间限制**，得到最终的λ_q的上界，如式(2)所示。

$$\lambda_q^* \le \min\left\{\frac{2\left(R_q^* - t_q\right)}{V_q + 2R_q^* t_q - t_q^2},\quad \left(T - mt_u\right)/\left(T \cdot t_q\right)\right\} \tag{2}$$

为了可解释性，将式(2)转化为式(3)。

$$\lambda_q^* \le \begin{cases} 1/t_q, & \text{if } \alpha\beta < 1/2 \quad (\text{ QoS-bound mode }) \\ (1-\beta)/t_q, & \text{if } \alpha\beta \ge 1/2 \quad (\text{Update-bound mode}) \end{cases} \tag{3}$$

其中$\alpha = R_q^*/t_q;\quad \beta = mt_u/T;\quad \gamma = V_q/t_q^2$；$\alpha$用来衡量平均查询相应时间的上界与平均查询时间的比值，β用来衡量一个时间片内处理Update所用时间的占比，γ是离散系数（coefficient of variation）的平方，相对于标准差其为无量纲量可以用在不同的索引算法中进行比较，一般$\gamma \in [0.1, 0.9]$。

2.1.2 Pokemon游戏（RUA+FCFS模型）

对于pokemon类似的以所在位置为中心，查询周围存在的Pokemon，并进行捕捉的这类游戏，其Update不是按照某一指定的时间间隔返回的，而是随机的，所以其Update操作不能在

使用BUA模型，而是转为RUA(Random Update Arrival)模型，同样的为了方便研究，**作者假设RUA的Update符合另一个泊松过程（与Query到来的泊松过程不同）**。

对于Query的优先级，Pokemon游戏很注重每个Pokemon的位置更新，所以该模型的队列模型使用了FCFS（First-come-first-served），即Query和Update具有相同的优先级，根据到来的顺序来进行操作。

其分析过程与2.1.1节类似，最终得到式(4)，衡量在该模型下的Query到来频率的上界。

$$\lambda_q^* \leq \min\left\{\frac{2\left(R_q^*-t_q\right)\left(1-\lambda_u t_u\right)-\lambda_u\left(V_u+t_u^2\right)}{V_q+2R_q^* t_q-t_q^2}, \frac{1-\lambda_{\mathrm{u}} t_u}{t_q}\right\} \tag{4}$$

2.2 索引SCOB方面

在索引和算法方面，作者做出了两个假设，这两个假设虽然在后面进行了放松，但本质上论文作者最终解决的问题也没有彻底摆脱这第一个假设的约束，这一部分在后面详细来说。两个假设分别是：

1. 抽象后的图G是无向图
2. 所有节点的Rank是不一样的

2.2.1 符号定义

SHORTCUT SET $SC_<$ 给定一个图$G=(V,E)$和一个定义在$V\to\mathbb{N}$上的Rank Function，则一条捷径(shortcut)$u\curvearrowright v\in SC_<$当且仅当1.$u,v\in V$，2.$r(u)<r(v)$，3.对于从$u$到$v$上最短路上的中间任意节点$z$均有$r(z)<r(u)$。

从上面的定义我们可以直接得到一个结论，如式(5)，其中$SC_<$是从G上导出的捷径图(shortcut graph)，因为不一定所有的G上的最短路都符合$r(z)<r(u)$即上述定义的第3条规则。

$$d_G(u,v)\leq d_{SC_<}(u,v)\quad \forall u,v\in V \tag{5}$$

SUMMIT NODE 给定一个最短路径(shortest path)$s\rightsquigarrow t$，则对于该最短路的Summit Node是$s\rightsquigarrow t$上具有最高权重的节点。

根据$SC_<$和summit node的定义以及第一个假设，我们可以直接得到式(6)，其中x是最短路径$s\rightsquigarrow t$上的summit node。

$$d_G(s,t)=d_{SC_<}(s,x)+d_{SC_<}(t,x) \tag{6}$$

DOWNHILL OBJECTS 论文中作者为了形象表示不同节点的rank不同，具有越高rank的节点在图中的位置越高，如图2.2所示。基于这种形象的表示，如果在$SC_<$中有一条从v指向u的shortcut，则称v是u的downhill object。

kDNNs 给定一个节点$u\in G$，离它最近的k个downhill objects被记作$kDNN(u)$。

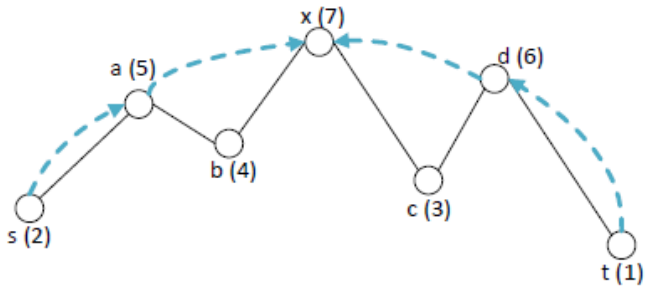

图 2.2: Query Algorithm

2.2.2 基本思想

根据式(5)和G是无向图的假设，可以有结论给定一个节点s和一个位于节点t的物体o，如果o是s的kNN之一，则有$o \in kDNN(x)$，x是最短路$s \rightsquigarrow t$上的$summit\ node$（其证明使用了证明其逆否命题的方法，主要利用三角不等式、式(5)进行放缩）。**通过以上结论，可以利用kDNN和一些特定的summit node可以得到一些源点的KNN结果。**

SCOB索引本质上也是利用了Dijkstra在图上进行搜索，而传统的方式对于一个有M个节点的图，需要进行M次s-climb(从源点到summit node的搜索)以及M次t-climb(从目标点到summit node的搜索)；而使用了SCOB索引**在每个summit node上记录在Shortcut Graph中的k个最近的downhill objects的id和其距离，以减少s-climb(t-climb)的次数，来提高效率，最终实现了仅需要从Query发出的节点进行一次s-climb就可以找到对应这次Query的kNN结果。**

具体如何维护SCOB索引，见3.1节。

2.3 算法TOAIN方面

第3章 基本算法描述

文中以伪代码形式讲述的算法共有4个，下面将分别进行描述。

3.1 SCOB Index

Query

Insert

Delete

3.2 TOAIN

Compute Rank

Algorithm 1: Query($SC_<$, q, k)

$\mathcal{R} \leftarrow \emptyset$; $d_0 \leftarrow$ longest distance of an object in $\mathcal{R}$ from q, initially ∞;
/* Conduct Dijkstra search from q on $SC_<$, with the following operations. */
for *each node p being visited in the Dijkstra search* **do**
  if $\|\mathcal{R}\| \geq k$ *and* $d_{SC_<}(q,p) > d_0$ **then**
    break ;
  for *object* $o \in kDNN(p)$ **do**
    add $[o, d_{SC_<}(q,p) + d_{SC_<}(o,p)]$ into $\mathcal{R}$;
    if $\|\mathcal{R}\| > k$ **then**
      remove from $\mathcal{R}$ the object with longest distance from q.
    update d_0;
return $\mathcal{R}$;

图 3.1: Query Algorithm

Algorithm 2: Insert($SC_<$, o, k)

/* Suppose o is located at node t; Conduct Dijkstra search from t on $SC_<$, with the following operations. */
for *each node p being visited in the Dijkstra search* **do**
  add $[o, d_{SC_<}(t,p)]$ into $kDNN(p)$;
  if $\|kDNN(p)\| > k$ **then**
    delete from $kDNN(p)$ the object with longest distance from p;

图 3.2: Insert Algorithm

第4章 算法分析

第5章 举例说明

参考文献

[1] Luo, S., Kao, B., Li, G., Hu, J., Cheng, R., & Zheng, Y. (2018). TOAIN: a throughput optimizing adaptive index for answering dynamic k NN queries on road networks. Proceedings of the VLDB Endowment, 11(5), 594-606.

[2] J. W. Cohen. The single server queue, volume 8. Elsevier, 2012.

Algorithm 3: `Delete(`$SC_<$`,` o`,` k`)`

```
   /* Suppose o is located at node t;                          */
1  if o ∈ kDNN(t) then
2      F ← ∅;
       /* Conduct Dijkstra search from t; lines 3∼6
          show the operations done in the search                */
3      for each node u being visited in the Dijkstra search do
4          if o ∈ kDNN(u) then
5              remove o from kDNN(u);
6              F ← F ∪ {u};
7      F* ←sort nodes of F in increasing ranks;
8      while F* is not empty do
9          p ← lowest ranked node in F*;
10         kDNN(p) ← at most k objects located at p;
11         for (v ⇝ p) ∈ SC_< do
12             for object o* ∈ kDNN(v) do
13                 add [o*, d_SC<(o*, v) + d_SC<(v, p)] to
                   kDNN(p);
14                 if ||kDNN(p)|| > k then
15                     delete from kDNN(p) the object with longest
                       distance from p;
16         remove p from F*;
```

图 3.3: Delete Algorithm

Algorithm 4: `ComputeRank(`G`,` h`)`

```
1  initialize r(u) to 0 for every node u ∈ G;
2  S ← V /* V is the node set of G.                             */
3  for grid level i from 1 to h do
4      impose K_i × K_i grid on G;
       /* Refer to text for the value of K_i                    */
5      Υ_i ← ∅;
6      for node v ∈ S do
7          conduct Dijkstra search from v within the 5 × 5 sub-grid,
           whose central cell contains v;
8          Υ_{i,v} ← cover nodes found during the Dijkstra search;
9          Υ_i ← Υ_i ∪ Υ_{i,v};
10     update r(z) to i for every node z ∈ Υ_i;
11     S ← Υ_i;
```

图 3.4: Compute Rank Algorithm