



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
2.2 索引SCOB方面	4
2.3 算法TOAIN方面	4
第3章 基本算法描述	4
3.0.1 SCOB Index	5
3.0.2 TOAIN	5
第4章 算法分析	6
第5章 举例说明	6

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系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)。

打车软件 (BUA+QF Model) BUA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Query操作来说，其假设Query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。

$$R_q = \frac{\lambda_q (t_q^2 + V_q)}{2(1 - \lambda_q t_q)} + t_q \quad (1)$$

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。

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

为了可解释性，将式(2)转化为式(3)。

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

其中 $\alpha = R_q^*/t_q$; $\beta = mt_u/T$; $\gamma = V_q/t_q^2$; α 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 $\gamma \in [0.1, 0.9]$ 。

Pokemon游戏 (RUA+FCFS模型)

2.2 索引SCOB方面

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。

3.0.1 SCOB Index

Algorithm 1: $\text{Query}(SC_{<}, q, k)$

```

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

```

图 3.1: Query Algorithm

Query

Algorithm 2: $\text{Insert}(SC_{<}, o, k)$

```

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

```

图 3.2: Insert Algorithm

Insert

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

```

/* Suppose  $o$  is located at node  $t$ ; */
1 if  $o \in kDNN(t)$  then
2    $\mathcal{F} \leftarrow \emptyset$ ;
   /* 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  $o \in kDNN(u)$  then
5       remove  $o$  from  $kDNN(u)$ ;
6        $\mathcal{F} \leftarrow \mathcal{F} \cup \{u\}$ ;
7    $\mathcal{F}^* \leftarrow$  sort nodes of  $\mathcal{F}$  in increasing ranks;
8   while  $\mathcal{F}^*$  is not empty do
9      $p \leftarrow$  lowest ranked node in  $\mathcal{F}^*$ ;
10     $kDNN(p) \leftarrow$  at most  $k$  objects located at  $p$ ;
11    for  $(v \curvearrowright p) \in SC_{<}$  do
12      for object  $o^* \in kDNN(v)$  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  $\mathcal{F}^*$ ;

```

图 3.3: Delete Algorithm

Delete

3.0.2 TOAIN

Compute Rank

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 4: ComputeRank (G, h)

```

1 initialize  $r(u)$  to 0 for every node  $u \in G$ ;
2  $S \leftarrow V$  /*  $V$  is the node set of  $G$ . */
3 for grid level  $i$  from 1 to  $h$  do
4     impose  $K_i \times K_i$  grid on  $G$ ;
5     /* Refer to text for the value of  $K_i$  */
6      $\Upsilon_i \leftarrow \emptyset$ ;
7     for node  $v \in S$  do
8         conduct Dijkstra search from  $v$  within the  $5 \times 5$  sub-grid,
9         whose central cell contains  $v$ ;
10         $\Upsilon_{i,v} \leftarrow$  cover nodes found during the Dijkstra search;
11         $\Upsilon_i \leftarrow \Upsilon_i \cup \Upsilon_{i,v}$ ;
12    update  $r(z)$  to  $i$  for every node  $z \in \Upsilon_i$ ;
13     $S \leftarrow \Upsilon_i$ ;

```

图 3.4: Compute Rank Algorithm