On Continuously Monitoring the Top-k Moving Objects with Relational Group and Score Functions

[PhD Showcase]

Jian Wen*
University of California, Riverside
900 University Ave.
Riverside, CA 92521
jarodwen@gmail.com

ABSTRACT

With the wide usage of location tracking system, continuously mining relationships among moving objects over their location changes is possible and also important to many real applications. This paper shows a novel continuous location-based query, called continuous relational top-k query or *CRTQ*, which continuously monitors the k moving objects with the most significant relations with the other objects by user-defined relational group and score functions. Although this kind of query can be implemented as a special case of the top-k join query using SQL, this straight-forward way is too expensive to be applicable widely. This paper also discusses the properties of this novel query, which leads to the flexibility of the query and also the difficulty for a generalized solution. An efficient algorithm is proposed for a special type of CRTQ over spatial data set with closeness grouping function and monotone increasing score function. Finally the main contributions of this showcase and also our future research plans are discussed.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: DATABASE MANAGEMENT—*Database applications*

General Terms

Relational Top-k Query

Keywords

Relational, Top-k, Query, Algorithm

1. INTRODUCTION

Top-k query [3] has drawn much attention in database community due to its wide usage in both database and data mining applications. Given a data set R and a preference function F, a top-k query Q returns the k tuples in R with the highest scores according to F. The score function F for a specific object r gets all its input from the attributes of r, that is, the score of an object is a comparable value according to its characteristics which then be used in the sorting process.

However, many recent applications require monitoring the relations between the observed object and all the other objects in the data set, while getting the most interesting relations continuously, which intuitively follows the top-k pattern. The following two examples show this new trend.

EXAMPLE 1. *SDSS Q18: The Q18 in SDSS data set, one of the benchmark database queries over the images of objects on the sky observed in the SDSS project, tries to find all objects within 30 arcseconds of one another that have very similar colors: that is where the color ratios u-g, g-r, r-i are less than $0.05m$. Two relations are involved into the query,* photoPrimary*, which contains the information of physical objects on the sky observed, and* Neighbors*, a logical view which contains the information about neighbor objects for each primary object in* photoPrimary*.*

To illustrate our problem we derive a top-k query from it: we want to know the k objects having the most similar colors with their neighbors within 30 arcseconds. Here a score function is defined as the sum of the absolute difference on the color ratios u-g, g-r, r-i. So the derived SQL query is like:

```
SELECT distinct P.ObjID
FROM photoPrimary P, Neighbors N, photoPrimary L
WHERE P.ObjID = N.ObjID
   and L.ObjID = N.NeigoborObjID
   and P.ObjID < L.ObjID
   and abs((P.u - P.g) - (L.u - L.g)) < 0.05
   and abs((P.g - P.r) - (L.g - L.r)) < 0.05
   and abs((P.r - P.i) - (L.r - L.i)) < 0.05
   and abs((P.i - P.z) - (L.i - L.z)) < 0.05
ORDER BY abs((P.u - P.g) - (L.u - L.g))
    + abs((P.g - P.r) - (L.g - L.r))
    + abs((P.r - P.i) - (L.r - L.i))
    + abs((P.i - P.z) - (L.i - L.z))
LIMIT k;
```

*Supervised by Professor Donghui Zhang and Professor Vassilis Tsotras

Q18 in SDSS is known as the longest-running query among the total 20 queries, since it compares every object in *photoPrimary* with all its neighbors in a user-defined area. Our new top-k query above derived from Q18 intuitively requires a further sorting step so its running time must be even longer than the original. When frequent updates on the image data set are considered, the naive way of updating neighbors of all objects for their scores will cause significant performance overhead.

Example 2. *Continuously Monitor Top-k Unsafe Moving Objects: For maintaining public safety, police force needs to monitor many moving objects at the same time, such as crowds walking over the city in the Independence Day, or cash transport vans towards different bank branches. We want to make sure that each of such objects is safe, which is, protected by the proper amount of police force. If any object has less protecting force around than it requires, it is unsafe. An very useful query for managing the police force is to monitor the top-k unsafe objects while both the protected objects and police force keep updating their locations.*

The common characteristic of these two tasks is that it queries on relations: Example 1 is on the similarity, while Example 2 is on the coverage. In a straight-forward way both of them require a full scan over the entire data set to obtain the score for each single object. When an object is updated, in order to make sure that the top-k results are still correct, naively a total re-run is necessary since (1) all the objects whose scores are effected by the update should be re-calculated, and (2) once the scores of some objects are updated, a top-k query should be re-issued to make sure that the results include the tuples with the highest scores.

In this paper these queries are generalized into a novel top-k query, called *continuous relational top-k query* or *CRTQ*, which intuitively extracts the most significant relationship among tuples from the given data set instead of the characteristic of a single object in the traditional top-k query. This paper shows that although this kind of query can be processed in a traditional database system, it costs too much in both the I/O and CPU aspects since naively each update would trigger a total recalculation for updated top-k information.

An efficient solution for this query in the continuous environment is difficult, however, due to its flexibility on grouping and scoring. For this reason, in this paper we show an efficient solution for a special case of the continuous relational top-k query, continuous monitoring top-k unsafe moving objects, whose score and group functions satisfy certain conditions for utilizing the algorithm. Although this solution cannot be applied directly to the generalized relational top-k query model, basic optimal strategies like pruning and indexing are considered in order to provide helpful inspection on the possible generalized solutions in the future research plan.

Existing efforts related to this new query include top-k aggregation query [4] and top-k join query [2]. Both these two top-k queries extend the traditional top-k query and also provide efficient algorithms for them. However, top-k aggregation query only focuses on the attributes in a single relation, while top-k join query algorithms cannot handle dynamic updates efficiently. In [5] a continuous top-k query algorithm is proposed, however in our problem no sliding window is considered (so no data object is expired within a fixed life time).

The rest of this paper is organized as follows. Section 2 describes the definition of continuous relational top-k query and its properties. Then in Section 3 a special case of CRTQ, continuous monitoring top-k unsafe moving objects from Example 2 is proposed and analyzed, with an efficient query answering algorithm called BasicCTUO. Finally, in Section 5 this showcase is summarized with our contributions and future research plans.

2. CONTINUOUS RELATIONAL TOP-K QUERY

In this section the continuous relational top-k query is introduced. The data model assumption will be defined at first, then the formal definition of our new query.

2.1 Data Model

Consider a relation R describing objects with a list of m properties $A_1, A_2, ..., A_m$. We assume that there exist attributes which can be quantified for numeric comparison and scoring.

Data objects are updated continuously. An update may (1) update the values of attributes of an existing object, or (2) insert a new data object into the relation, or (3) delete an existing data object.

Here we use Example 2 to illustrate this model. For monitoring top-k unsafe moving objects over a 2-D space, a relation R containing all the moving objects (both police force and objects to be protected) is maintained on a centralized server. These objects can update their location through the GPS system. The server also responds to queries from users, specially here the continuous relational top-k queries, e.g., a query monitoring the k most unsafe cash transport vans.

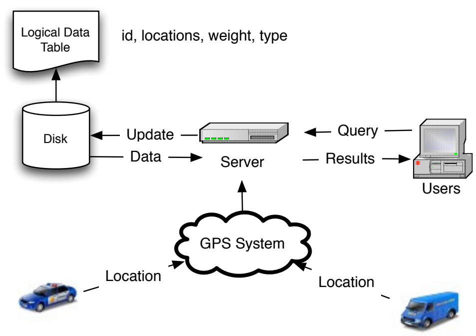

Figure 1: The model for monitoring top-k unsafe moving objects.

2.2 Problem Definition

From the two examples showed previously, in order to get the scores for the final top-k query, for each object in the relation, it is necessary to (1) find objects which will be considered into the score function (so for each object there will be a group of such objects), and also (2) define the score function over the object itself and also the group containing other objects we have found. These two conditions can be formalized using two functions defined as follows.

DEFINITION 1. *Group Function: A group function, F_G, accepts the input of two data tuples r_0 and r_g, and returns a boolean value illustrating whether r_g will be involved into the score calculation of r_0. That is, $F_G : r_0 \times r \rightarrow BOOL$.*

DEFINITION 2. *Score Function: A score function, F_S, accepts the input of a data tuple r_0 and a group of tuples L, and returns an integer value as the score for r_0. Here L satisfies $L \subseteq R$ and $\forall r \in L, F_G(r_0, r) = true$, and also $\forall r' \in \{R-L\}$, $F_G(r_0, r') = false$.*

A group function defines the membership requirement for a given relational top-k query. Intuitively, this function describes the relationship we want to monitor. For instance, in Example 1 the group function can be defined as $F_G(r_0, r) = \{r$ is within 30 arcseconds of $r_0.\}$, while in Example 2 the group function can be defined as $F_G(r_0, r) = \{$The distance between r and r_0 is no more than 1 mile.$\}$.

A score function computes the score from the input tuples approved by the group function. Since both the data object r_0 and tuples in L are involved into the score function, it defines the measurement on the relationship between r_0 and its group L. This measurement can be simply some aggregation like either of the two examples we have showed, or some complex algorithm quantifying the relationship using attributes related.

Now the CRTQ problem can be defined as:

DEFINITION 3. *Relational Top-k Query: Given a relation R, and the size of the output k, a* **Relational Top-k Query** *$Q\{R, k, F_G, F_S\}$ returns the k tuples with the highest scores according the given group function F_G and score function F_S.*

2.3 Properties

As a novel top-k query, relational top-k query has many interesting properties, which leads to its wide applicability and also difficulty in solution probing.

2.3.1 Group Function

The definition of a group in CRTQ is very flexible. Although a group can be generated for a data object from the group function, it is not the same as clustering in data mining area. In clustering objects are grouped based on their similarity, which is, the items in a group will have higher similarity, while items between different groups will have lower similarity. In CRTQ, this can be simply achieved by group functions making constraints on the measurement of similarity, for example, $F_G(r_0, r) = \{$Similarity $(r_0, r) \geq s\}$. Furthermore, group functions can be used to describe more other relations, such as difference, variation, etc., which cannot be considered as similarity. Some good examples are like "find k objects farthest from r_0", and "the object whose value of A_i is different from $r_0.A_i$ by at least l". When considering the continuous situation, group functions can illustrate some continuous relations, such as "in the past t time period, the difference of A_i between r_0 and r has never been larger than l", etc. However, this flexibility also brings the difficulty to find a generalized solution for this query. We will discuss this more at the end of Section 3.

2.3.2 From top-k to CRTQ

CRTQ has a strong connection with the traditional top-k query. A relational top-k query is in fact a special case of top-k join query [2], where all joins are self-joins (which is the reason for its name using "relational"). Formally, given a relational top-k query $Q\{R, k, F_G, F_S\}$, a corresponding top-k join query can be defined as follows:

SELECT $R_1.key$ *as key*, $F_S(R_1, R_2^*)$ *as score*
FROM R R_1, R R_2
WHERE $F_G(R_1.r, R_2.r) =$ true
ORDER BY *score*
GROUP BY $R_1.key$
LIMIT k;

Notice that here we use R_2^* to present the tuples grouped for the score computation of a single object in R_1. These values are collected by the *GROUP BY* clause with the score function F_S as a customized aggregation function.

Although answering such a query in the static environment is no worse than running a top-k join query, it is really expensive to issue it in a continuous environment. Existing algorithms for the static environment mentioned in [2] cause overhead due to the expensive cost on maintaining the natural joins when updating. The flexibility of the group function makes this exploration even harder. As far as we know, there has been no efficient algorithm yet working for the continuous top-k join queries.

CRTQ is not always so difficult to be solved efficiently, however. Although the flexibility of the grouping and scoring strategy increases the difficulty, efficient solutions are still possible when group and score functions are restricted. Later in Section 3, an efficient algorithm is showed for the CRTQ when (1) the group function is closeness-grouping, and (2) the score function is monotone-increasing.

3. CONTINUOUSLY MONITOR TOP-K UNSAFE MOVING OBJECTS

This section shows that for a special case of the relational top-k query from Example 2, called continuous monitoring top-k unsafe moving objects in a spatial database, an efficient solution is possible. We extend the original optimal algorithm in [6] for monitoring top-k unsafe places whose locations will not be changed, and show the efficient strategy for relation top-k query in such a special category.

3.1 Definition

Consider R as a data set of moving objects such as cash transport vans and police cars. Each data tuple has the schema as $\{$*id* INT, *type* BOOLEAN, pos_x DECIMAL, pos_y DECIMAL, *safety* INT, PRIMARY KEY *id*$\}$. Assume that an object with *type* as *true* will be a protect unit and otherwise be an object to be protected. *Safety* indicates the weight of the safety required (for objects to be protected) or of the safety provided (for protect units). We also assume that all the protect units have the same protect region of u (the furthest distance it can reach for protection is d_u). The safety of a protected object r is defined as the difference between its safety weight and the sum of safety weights of protect units whose protect regions contain this object, which is

$$\sum_{r \in r_i.u} r_i.\text{safety} - r.\text{safety}$$

formally. So our problem is defined as

Definition 4. *The Continuous Top-k Unsafe Moving Object query continuously monitors the k moving objects in R with the lowest safeties.*

So from this definition we have

$$F_G(r_0, r) = \{\sqrt{(r_0.x - r.x)^2 + (r_0.y - r.y)^2} \leq d_u\}$$

and

$$F_S(r_0, L) = \sum_{r \in L} r.\text{safety} - r_0.\text{safety}$$

And the corresponding SQL query in the top-k join query pattern will be

SELECT $R_0.id$ as id,
(SUM(R_1.safety) - R_0.safety) as score
FROM R R_0, R R_1
WHERE $R_0.id \neq R_1.id$
AND $Sqrt((R_0.pos_x - R_1.pos_x)^2$
$+ (R_0.pos_y - R_1.pos_y)^2) \leq d_u$
GROUP BY $R_0.id$
ORDER BY score DESC
LIMIT k;

3.2 BasicCTUO

We extend our efficient algorithm for the continuous top-k unsafe place query (BasicCTUP[6]) to fit into our new query, which we call it as Basic Continuous top-k Unsafe moving Objects algorithm (BasicCTUO). The most important change is that static places are replaced by dynamic moving objects. Here a stream-based two-level storage structure is used, where data are stored on disks, and all updates are firstly written into the memory then synced onto the disk. The computations of CRTQ queries will also be processed inside of the memory. So in order to improve the performance, the loading process from the disks should be minimized.

The whole space is partitioned into a set of non-intersecting cells, and a grid for this partition is maintained in memory. Each cell in the memory has a pointer to the data of objects in it on the disk. A lower bound flag as an integer is also associated with each cell, indicating the lowest safety weight gained by the moving objects inside from the protect units reachable. An example is showed in Fig. 2, where the space is partitioned into 9 cells, and the lower bound of cell 5 is -2. When an update is coming, only the cells being influenced are updated to maintain the updated lower bound. In this way, the cost of maintaining the natural join result in CRTQ is decreased by limiting the monitoring granularity by cells instead of single objects in the whole Cartesian product space.

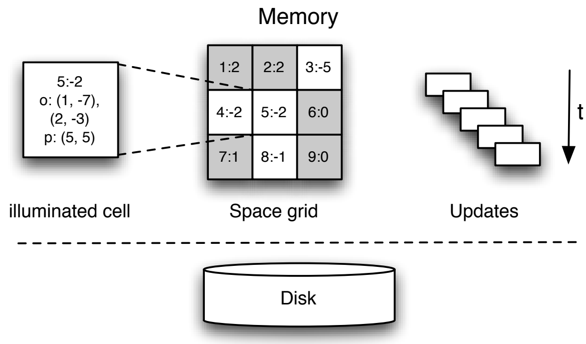

Figure 2: An example for BasicCTUO.

Here the same "illuminating" strategy from [6] is applied to keep as few as possible active cells (which are illuminated) into the memory while other darken ones onto the disk. The status of a cell will be changed based on the updates of objects inside. Our algorithm tries to minimize the number of illuminated cells during the updates, which further decreases the number of cells to be monitored and improves the performance.

The whole algorithm is consisted of two main parts, the initialization part and the maintenance part. Since the initialization part is working on a static data set, it is exactly the same as the BasicCTUP. Changes are made in the maintenance part, where different strategies are used for updating protected moving objects and protect units. Details of the algorithms can be found below.

Algorithm 1 BasicCTUO: Initialize

Require: A relational top-k query $Q = \{R, F_G, F_S, k\}$
Ensure: Get the initial top-k unsafe objects, and initialize the lower bound of all cells. Cells containing top-k objects are illuminated and maintained in memory.

```
Initialize the grid based on the space partition.
for each cell in the grid do
    for each protected object in this cell do
        Calculate the safety score by probing the protect
        units in this cell and cells around.
    end for
    Update the lower bound safety weight of the cell lb.
end for
Set SK_u = +∞ as the upper bound safety weight of the
top-k list
Invoke Alg. 2 to update the top-k list.
```

In details for updating the status of cells, all the cells con-

Algorithm 2 BasicCTUO: Update the top-k list.

Require: A grid with updated lower bound on each cell.
Ensure: The top-k list is updated.
Illuminate all cells whose lower bound is less than SK_u
for each cell in the grid in the increasing order of the lower bound **do**
  if The lower bound of this cell lb is larger than SK_u **then**
    Break this loop and return the top-k objects.
  else
    Find the partial top-k unsafe objects in this cell, then update the top-k list using these objects.
    Update SK_u to be the current upper bound of the top-k list (if this top-k list is not full, set $SK_u = +\infty$).
  end if
end for

taining (or possibly containing) a top-k object will *be illuminated* for further checking. If none of the protected objects in one cell is in the top-k list, this cell will be *darken.* The content of an illuminated cell will be maintained in memory for fast monitoring in the future, while for dark cells only their lower-bounds will be maintained in memory. In [6] more details have been showed to update the lower bound of the dark cell without loading its content into the memory, and also an optimized algorithm is described to overcome the "flashing" problem on updating the cells. Since all of these strategies can be applied directly to our algorithms, we omit these details here.

Algorithm 3 BasicCTUO: Maintenance for updating protected objects

Require: An update of a moving object.
Ensure: The top-k list is updated.
if The object is a protected object. **then**
  if The object is in an illuminated cell **then**
    Calculate its safety by checking only the police force in illuminated cells;.
  else
    Set the safety of the object as its safety requirement (the safety lower bound of this cell should be updated too).
  end if
  Invoke Alg. 2 to update the top-k list when the safety of this object is smaller than SK, or a dark cell is just illuminated.
else
  Update the lower bound of the cells intersecting with the protect region before and after updating.
  if All these updated cells have lower bounds higher than SK_u **then**
    Nothing need to be updated for the top-k list.
  else
    Invoke Alg. 2 to update the top-k list.
  end if
end if

4. PERFORMANCE ANALYSIS

In this section mainly two algorithms are compared: the naive algorithm and BasicCTUO algorithm. Instead of using any pre-processing or pruning technologies, the naive algorithm uses a brute-force strategy processing all the data points influenced by the update. Here the data points "influenced" refer to the ones whose safety weights may be changed due to the update. To minimize possible systematic error on the running time, the number of cells loaded when running is considered as the main measurement on the performance of algorithms, since the main goal of BasicCTUO is to reduce the unnecessary cost of loading data between memory and disk every time when updating.

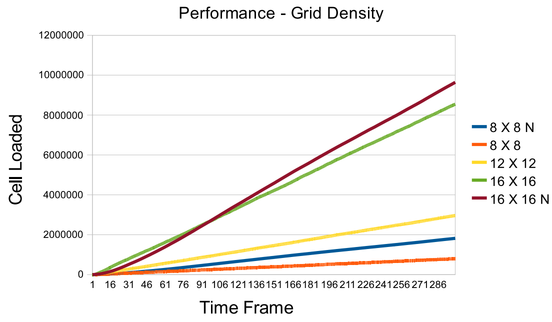

Figure 3: Naive and BasicCTUO performance on different grid density.

The algorithms are implemented in Java on a Mac OS X 10.5.8 system with a 2.2 GHz Intel Core 2 Duo CPU and 4 GB 667 MHz DDR2 SDRAM. All the data are generated by the Network-based Generator of Moving Objects [1] along the Oldenburg road network. Different configurations on parameters, including *the monitoring time, the grid density and the protection radius*, are used in our experiments to illustrate different properties of BasicCTUO. Fig. 3 shows the performance when the grid density of the map is changed. It is clear that BasicCTUO always beats the naive algorithm with the same grid density (N means the naive algorithm), however the increasing grid density causes the performance downgrade of BasicCTUO due to the overhead of monitoring more small cells within the same protect radius. Fig. 4 shows the performance of the BasicCTUO changes when the radius of the protection region varies, where a larger protection region also indicates the overhead on more cells to be monitored.

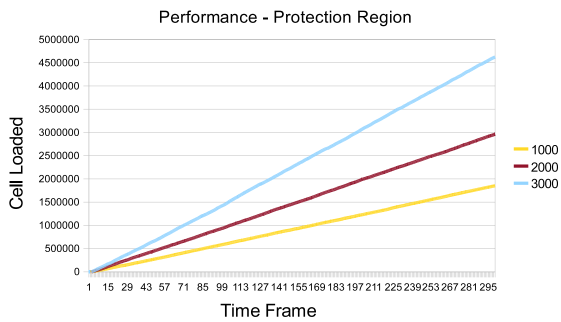

Figure 4: BasicCTUO on different protection radius.

4.1 BasicCTUO for a Category of CRTQ

Our BasicCTUO algorithm can be extended onto other spatial continuous relational top-k queries, if these queries have the similar characteristics on their group and score functions. In details, they should satisfy the following conditions.

- Objects are grouped based on the closeness to the reference point. This condition utilizes the grid index to minimize the overhead of join operations. It can be formally defined as the *closeness-grouping property*:

  PROPERTY 1. *Closeness Grouping: For data objects* r_0, r_1 *and* r_2*, if* $dist(r_0, r_1) \leq dist(r_0, r_2)$ *and* $F_G(r_0, r_2)$ *= true, then* $F_G(r_0, r_1)$ *= true. Here* dist *is the Euclidean distance of the two objects.*

  For example, this property holds on a k-NN group function like $F_G(r_0, r) = \{r$ is one of the k-NN of $r_0.\}$

- The score function is monotone increasing on each object in the group. This condition makes sure that updates can be pre-processed over the grid in memory without inspecting the content of the cell. Formally it can be defined as the *monotone scoring*:

  PROPERTY 2. *Monotone Scoring: Given a data object* r_0 *and its group* L*, consider that* $r \in L$ *and* $r.A$ *participates the score function* F_S*, then we say* F_S *is a monotone increasing score function if after updating* r *into* r' *(*L *is then into* L'*),* $r.A \leq r'.A$*, we have* $F_S(r_0, L) \leq F_S(r_0, L')$*.*

  So a more complex score function for the safety measurement is still feasible for BasicCTUO, if this score function is monotone on the safety weight of each protect unit in the group.

Here we can show that the BasicCTUO algorithm cannot be applied to Example 1. The group function of Example 1 can be defined as

$$F_G(r_0, r) = \{r \text{ is within 30 arcseconds of } r_0.\}$$

And its score function is

$$F_S(r_0, L) = \sum_{l \in L} \sum_{c} \text{abs}((r_l)_c - (r_0)_c)$$

where $c \in \{u-g, g-r, r-i, i-z\}$. Although F_G is a closeness grouping function, F_S is not monotone due to the existence of abs function. So BasicCTUO algorithm is not working for Example 1. For the same reason, BasicCTUO cannot be generalized to any CRTQ on which the two conditions do not hold.

5. CONCLUSIONS AND RESEARCH PLANS

This showcase proposed a new continuous top-k query, called continuous relational top-k query. As a special case of the top-k join query where only natural joins are used, this kind of query retrieves the k objects with the most significant "relationship" with others in the data set. A formal definition of this query is proposed, with the relationship modeled as a group function F_G and a score function F_S. It also showed that due to the flexibility of grouping and scoring, it is difficult to find a generalized efficient solution for this query. However, one of its special applications in the spatial database, continuously monitoring top-k unsafe moving objects, was showed with an efficient solution avoiding the overhead to re-calculate every time after updating.

Our main contributions in this work are:

- The formal definition of a CRTQ has been proposed, including the relationship using group and score functions, which is helpful to formalize the real applications of this query and translate such a query into a SQL query.
- The flexibility of a relational top-k query is theoretically discussed, and also the connection between a relational top-k query with a top-k join query.
- An efficient algorithm, BasicCTUO for the special category of CRTQ, is provided. This category requires that for a query its group function should be closeness-grouping and score function be monotone increasing, in order to use BasicCTUO.

Our on-going research plan includes (1) classifying possible categories of the group functions to divide the solution space into proper sub problems, and (2) extending CTUO for continuous relationships.

6. REFERENCES

[1] Thomas Brinkhoff. A framework for generating network-based moving objects. *Geoinformatica*, 6(2):153–180, 2002.

[2] Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, pages 754–765. VLDB Endowment, 2003.

[3] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):1–58, 2008.

[4] Chengkai Li, Kevin Chen-Chuan Chang, and Ihab F. Ilyas. Supporting ad-hoc ranking aggregates. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 61–72, New York, NY, USA, 2006. ACM.

[5] Kyriakos Mouratidis, Spiridon Bakiras, and Dimitris Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD '06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 635–646, New York, NY, USA, 2006. ACM.

[6] Donghui Zhang, Yang Du, and Ling Hu. On monitoring the top-k unsafe places. In *ICDE '08: Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 337–345, Washington, DC, USA, 2008. IEEE Computer Society.