Top-*k* Query Algorithms

Outline

- Definitions – Objects, Attributes and Scores
- Querying Fuzzy Data
- Top-*k* query algorithms
  - Naïve Algorithm
  - Fagin's Algorithm (FA)
  - Threshold Algorithm (TA)
  - No Random Access Algorithm (NRA)
- Comparing top-*k* query algorithms
- References

Objects, Attributes and Scores

- Each object X_i (i.e house) has m scores ($r_{i1}, r_{i2}, \ldots, r_{im}$), one for each of m attributes (i.e. size, price, location).
- Objects are listed, for each attribute sorted by score.
- Each object is assigned an overall score by combining the attribute score using aggregate function or combining rule.
- Aim: Determine k objects (i.e. top k houses) with the highest overall score.

	R_1	R_2	R_3
X_1	1	0.3	0.2
X_2	0.8	0.8	0
X_3	0.5	0.7	0.6
X_4	0.3	0.2	0.8
X_5	0.1	0.1	0.1

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

Querying Fuzzy Data - Example

- Given the following relational structure

	R_1	R_2	R_3
X_1	1	0.3	0.2
X_2	0.8	0.8	0
X_3	0.5	0.7	0.6
X_4	0.3	0.2	0.8
X_5	0.1	0.1	0.1

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

Query: Select top-2 for the **sum** aggregate function (total scores).

i.e. Finding the top 2 houses which have the highest total scores.

Monotonicity property: An aggregation function t is monotone

if $t(x_1,\ldots,x_m) <= t(x'_1,\ldots,x'_m)$ whenever $x_i <= x'_i$ for every i.

Naïve Algorithm

1. Compute overall score for every object by looking into each sorted list.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

Naïve Algorithm

1. Compute overall score for every object by looking into each sorted list.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

X_1	1.5
X_2	1.6
X_3	1.8
X_4	1.3
X_5	0.3

Naïve Algorithm

2. Return *k* objects with the highest overall score.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

X_3	1.8
X_2	1.6
X_1	1.5
X_4	1.3
X_5	0.3

Return top-2 objects

Fagin’s Algorithm

1. Sequentially access all the sorted lists in parallel until there are k objects that have been seen in all lists.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

Fagin's Algorithm

1. Sequentially access all the sorted lists in parallel until there are k objects that have been seen in all lists.

	R_1		R_2		R_3
X_1	1	X_2	0.8	X_4	0.8
X_2	0.8	X_3	0.7	X_3	0.6
X_3	0.5	X_1	0.3	X_1	0.2
X_4	0.3	X_4	0.2	X_5	0.1
X_5	0.1	X_5	0.1	X_2	0

Since k = 2, and X_1 and X_3 have been seen in all the 3 lists

Fagin’s Algorithm

2. Perform random accesses to obtain the scores of all seen objects

	R_1		R_2		R_3
X_1	1	X_2	0.8	X_4	0.8
X_2	0.8	X_3	0.7	X_3	0.6
X_3	0.5	X_1	0.3	X_1	0.2
X_4	0.3	X_4	0.2	X_5	0.1
X_5	0.1	X_5	0.1	X_2	0

Fagin's Algorithm

3. Compute score for all objects and return the top-k

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

X_3	1.8
X_2	1.6
X_1	1.5
X_4	1.3

Return top-2 objects

Threshold Algorithm

1. Access the elements sequentially

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

Threshold Algorithm

1. At each sequential access

   (a) Set the threshold *t* to be the aggregate of the scores seen in this access.

	R_1		R_2		R_3
X_1	1	X_2	0.8	X_4	0.8
X_2	0.8	X_3	0.7	X_3	0.6
X_3	0.5	X_1	0.3	X_1	0.2
X_4	0.3	X_4	0.2	X_5	0.1
X_5	0.1	X_5	0.1	X_2	0

t = 2.6

Threshold Algorithm

1. At each sequential access

    (b) Do random accesses and compute the scores of the seen objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

$t = 2.6$

X_1	1.5
X_2	1.6
X_4	1.3

Threshold Algorithm

1. At each sequential access

   (c) Maintain a list of top-k objects seen so far

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

$t = 2.6$

X_2	1.6
X_1	1.5

Threshold Algorithm

1. At each sequential access

   (d) Stop, when the lowest score of the top-k are greater or equal to the threshold.

	R_1		R_2		R_3
X_1	1	X_2	0.8	X_4	0.8
X_2	0.8	X_3	0.7	X_3	0.6
X_3	0.5	X_1	0.3	X_1	0.2
X_4	0.3	X_4	0.2	X_5	0.1
X_5	0.1	X_5	0.1	X_2	0

t = 2.1

X_3	1.8
X_2	1.6

Threshold Algorithm

1. At each sequential access

(d) Stop, when the lowest score of the top-k are greater or equal to the threshold.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

$t = 1$

X_3	1.8
X_2	1.6

Threshold Algorithm

2. Return the top-*k* seen so far

Since, objects are sorted based on scores in descending order, the overall score of any unseen object cannot be greater than the threshold **t**. On the other hand, the overall score of any object in the top-k result must be greater or equal to **t.** Hence, the current top-k is the final result.

	R_1		R_2		R_3
X_1	1	X_2	0.8	X_4	0.8
X_2	0.8	X_3	0.7	X_3	0.6
X_3	0.5	X_1	0.3	X_1	0.2
X_4	0.3	X_4	0.2	X_5	0.1
X_5	0.1	X_5	0.1	X_2	0

t = 1

Return the objects

X_3	1.8
X_2	1.6

Algorithm 1: THRESHOLD ALGORITHM(T, k)

```
Input  : A table T which contains ratings based on different attributes, and k.
Output : The top k results which have the largest scores.
Initialize TopK ;
foreach row_i in the table T do
    foreach cell_c in the row_i do
        Obj_1 ← the object in cell_c;
        if Topk does not contain Obj_1 then
            Score_1 ← overall score of Obj_1;
            if the size of TopK ≥ k then
                Score_2 ← the lowest score in TopK;
                if Score_2 < Score_1 then
                    Remove an object Obj_2 whose score is Score_2 from TopK;
                    Add Obj_1 into TopK;
                Th ← compute the threshold of row_i;
                if Th < the lowest score in TopK then
                    break;
            else
                Add Obj_1 into TopK;
return TopK;
```

No Random Access Algorithm

1. Access sequentially all lists in parallel until there are k objects for which the lower bound is higher than the upper bound of all other objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

	LB	UB
X_1	1	2.6
X_2	.8	2.6
X_4	.8	2.6

No Random Access Algorithm

1. Access sequentially all lists in parallel until there are k objects for which the lower bound is higher than the upper bound of all other objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

	LB	UB
X_2	1.6	2.2
X_3	1.3	2.1
X_1	1	2.3
X_4	0.8	2.3

No Random Access Algorithm

1. Access sequentially all lists in parallel until there are k objects for which the lower bound is higher than the upper bound of all other objects.

	R_1
X_1	1
X_2	0.8
X_3	0.5
X_4	0.3
X_5	0.1

	R_2
X_2	0.8
X_3	0.7
X_1	0.3
X_4	0.2
X_5	0.1

	R_3
X_4	0.8
X_3	0.6
X_1	0.2
X_5	0.1
X_2	0

	LB	UB
X_3	1.8	1.8
X_2	1.6	1.8
X_1	1.5	1.5
X_4	0.8	1.6

No Random Access Algorithm

2. Return top-k objects for which the lower bound is higher than the upper bound of all other objects.

	R_1		R_2		R_3
X_1	1	X_2	0.8	X_4	0.8
X_2	0.8	X_3	0.7	X_3	0.6
X_3	0.5	X_1	0.3	X_1	0.2
X_4	0.3	X_4	0.2	X_5	0.1
X_5	0.1	X_5	0.1	X_2	0

	LB	UB
X_3	1.8	1.8
X_2	1.6	1.8
X_1	1.5	1.5
X_4	0.8	1.6

Return top-2 objects

Comparing top-*k* query algorithms

- Naïve algorithm
  - Buffer space required is equal to the number of database objects.
  - Each entry is looked in the *m* sorted lists. The cost is linear in database size.
  - Not efficient for a large database.
- Fagin’s algorithm (FA)
  - Large buffer space is required.
  - Random access is done at the end to get the missing scores.
  - Cost optimal under certain aggregate functions.
- Threshold algorithm (TA)
  - Buffer space required is bounded by *k*.
  - Score of an object not seen during algorithm execution is less than the threshold due to monotonicity property of aggregate function.
  - Less object access is required compared to FA because when *k* common objects have been seen in FA, their scores are higher or equal to threshold in TA.
  - May perform more random access than FA because in FA random access is done at the end only for the missing scores.
- No Random Access (NRA) algorithm
  - Only sorted access is performed.
  - May not report the exact object scores, since it uses bounds to determine top *k*.

References

- Combining Fuzzy Information: An Overview. *Ronald Fagin.*
- A Survey of Top-*k* Query Processing Techniques in Relational Database Systems. *Ilyas, Beskales and Soliman.*
- 'Web Information Search' lecture notes. *Prof. Leonardi (http://www.dis.uniroma1.it/~leon/didattica/webir/)*