Integrating rankings: Problem statement

- Each object has m grades, one for each of m criteria.
- The grade of an object for field i is x_i.
- Normally assume $0 \leq x_i \leq 1$.
  - Typically evaluations based on different criteria
  - The higher the value of x_i, the better the object is according to the ith criterion
- The objects are given in m sorted lists
  - the ith list is sorted by x_i value
  - These lists correspond to different sources or to different criteria.
- Goal: find the top k objects.

Example

Grade 1
(17, 0.9936)
(1352,0.9916)
(702,0.9826)
...
(12, 0.3256)
...

Grade 2 2
(235, 0.9996)
(12, 0.9966)
(8201, 0.9926)
...
(17, 0.406)
...

Aggregation Functions

- Have an aggregation function F.
- Let $x_1, \ldots, x_m$ be the grades of object R under the m criteria.
- Then $F(x_1, \ldots, x_m)$ is the overall grade of object R.
- Common choices for F:
  - min
  - average or sum
- An aggregation function F is monotone if

$$F(x_1, \ldots, x_m) \leq F(x'_1, \ldots, x'_m)$$

whenever $x_i \leq x'_i$ for all i.

Other Applications

- Information retrieval
- Objects R are documents.
- The m criteria are search terms $s_1, \ldots, s_m$.
- The grade x_i: how relevant document R is for search term s_i.
- Common to take the aggregation function F to be the sum

$$F(x_1, \ldots, x_m) = x_1 + \cdots + x_m.$$

Modes of Access

- Sorted access
  - Can obtain the next object with its grade in list L_i
  - cost c_S.
- Random access
  - Can obtain the grade of object R in list L_i
  - cost c_R.
- Middleware cost:

$$c_S \cdot \ (\text{\# of sorted accesses}) + c_R \cdot \ (\text{\# of random accesses}).$$

Algorithms

- Want an algorithm for finding the top k objects.
- Naive algorithm:
  - compute the overall grade of every object;
  - return the top k answers.
- Too expensive.

Fagin's Algorithm (FA)

1. Do sorted access in parallel to each of the m sorted lists L_i.
   - Stop when there are at least k objects, each of which have been seen in all the lists.
2. For each object R that has been seen:
   - Retrieve all of its fields $x_1, \ldots, x_m$ by random access.
   - Compute $F(R) = F(x_1, \ldots, x_m)$.
3. Return the top k answers.

Fagin's algorithm is correct

- Assume object R was not seen
  - its grades are $x_1, \ldots, x_m$.
- Assume object S is one of the answers returned by FA
  - its grades are $y_1, \ldots, y_m$.
- Then $x_i \leq y_i$ for each i
- Hence

$$F(R) = F(x_1, \ldots, x_m) \leq F(y_1, \ldots, y_m) = F(S).$$

Fagin's algorithm: performance guarantees

- Typically probabilistic guarantees
- Orderings are independent
- Then with high probability the middleware cost is

$$O\left(N \cdot \sqrt[m]{\frac{k}{N}}\right)$$

- i.e., sublinear
- But may perform poorly
  - e.g., if F is constant:
  - still takes $O\left(N \cdot \sqrt[m]{k/N}\right)$ instead of a constant time algorithm

Optimal algorithm: The Threshold Algorithm

1. Do sorted access in parallel to each of the m sorted lists L_i. As each object R is seen under sorted access:
   - Retrieve all of its fields $x_1, \ldots, x_m$ by random access.
   - Compute $F(R) = F(x_1, \ldots, x_m)$.
   - If this is one of the top k answers so far, remember it.
   - Note: buffer of bounded size.
2. For each list L_i, let $\hat{x}_i$ be the grade of the last object seen under sorted access.
3. Define the *threshold value* t to be $F(\hat{x}_1, \ldots, \hat{x}_m)$.
4. When k objects have been seen whose grade is at least t, then stop.
5. Return the top k answers.

Threshold Algorithm: correctness and optimality

- The Threshold Algorithm is correct for every monotone aggregate function F.
- Optimal in a very strong sense:
  - it is as good as any other algorithm on <u>every</u> instance
  - any other algorithm means: except pathological algorithms
  - as good means: within a constant factor
  - pathological means: making wild guesses.

Wild guesses can help

- An algorithm "makes a wild guess" if it performs random access on an object not previously encountered by sorted access.
- Neither FA nor TA make wild guesses, nor does any "natural" algorithm.
- Example: The aggregation function is min; $k = 1$.

LIST L_1
(1, 1)
(2, 1)
(3, 1)
. . .
(n+1, 1)
(n+2, 0)
(n+3, 0)
. . .
(2n+1, 0)

LIST L_2
(2n+1, 1)
(2n, 1)
(2n-1, 1)
. . .
(n+1, 1)
(n, 0)
(n-1, 0)
. . .
(1, 0)

Threshold Algorithm as an approximation algorithm

- Approximately finding top k answers.
- For $\varepsilon > 0$, an ε-approximation of top k answers is a collection of k objects $R_1, \ldots, R_k$ so that
  - for each R not among them,

$$(1+\varepsilon) \cdot F(R_i) \;\geq\; F(R)$$

- Turning TA into an approximation algorithm:
- Simply change the stopping rule into:
  - When k objects have been seen whose grade is at least

$$\frac{t}{1+\varepsilon},$$

then stop.