Corrigendum to *Best position algorithms for efficient top-k query processing*, by R. Akbarinia, E. Pacitti and P. Valduriez, Information Systems 36, 6 (2011), 973-989

Reza Akbarinia, Esther Pacitti and Patrick Valduriez

In [APV11], we said that the proof of Theorem 6.1 in [FLN03] was incorrect, because we thought that random access could return the position of the seen data in addition to its score. But, as Dr. Ronald Fagin points out, the random access defined in [FLN03] does not return the positions. Therefore, we were wrong, and we apologize to the authors of [FLN03].

For context, we explain the assumptions of [FLN03], as given in their paper.

1 Definitions and Assumptions

There is a set of n *objects*, and a database that consists of m sorted lists $L_1, \ldots, L_m$, each of length n, with one entry in each list for each of the n objects. Each entry of L_i is of the form (R, x_i), where R is an object and $x_i \in [0, 1]$ is the *score* of R in list L_i, for $1 \leq i \leq n$. Each list L_i is sorted in descending order by the x_i value.

Each object has an overall score, that is obtained by combining its scores in each of the lists by a fixed *aggregation function*, such as min or sum. Thus, if t is the aggregation function, and if x_i is the score of object R in list L_i, for $1 \leq i \leq n$, then the overall score of object R is $t(x_1, \ldots, x_m)$. The goal is to find the top k objects, that is, the k objects with the highest overall scores, for some fixed k. In [FLN03] the focus is on *monotone* aggregation functions,, that is, aggregation functions t such that whenever $x_i \leq y_i$ for each i, then $t(x_1, , \ldots, x_m) \leq t(y_1, \ldots, y_m)$.

There are two modes of access to data. The first mode of access is *sorted access*. Here the system obtains the score of an object in one of the sorted lists by proceeding through the list sequentially from the top. Thus, if object R has the ℓth highest score in list L_i, then ℓ sorted accesses to list L_i are required to see this score under sorted access. The second mode of access is *random access*. Here, the system requests the score of object R in list L_i, and obtains it in one random access. If there are s sorted accesses and r random accesses, then the *sorted access cost* is sc_S, the *random access cost* is rc_R, and the *middleware cost* is $sc_S + rc_R$ (the sum of the sorted access cost and the random access cost), for some positive constants c_S and c_R.

We now define the notion of *instance optimality*. Let **A** be a class of algorithms, and let **D** be a class of legal inputs to the algorithms. We assume that we are considering a particular nonnegative performance cost measure $cost(\mathcal{A}, \mathcal{D})$, which represents the cost incurred by running algorithm $\mathcal{A} \in \mathbf{A}$ on input $\mathcal{D} \in \mathbf{D}$. In [FLN03], this cost is the middleware cost. We say that an algorithm $\mathcal{B}$ is *instance optimal over* **A** *and* **D** if $\mathcal{B} \in \mathbf{A}$ and if for every $\mathcal{A} \in \mathbf{A}$ and every $\mathcal{D} \in \mathbf{D}$ we have

$$cost(\mathcal{B}, \mathcal{D}) = O(cost(\mathcal{A}, \mathcal{D})). \quad (1)$$

Intuitively, instance optimality corresponds to optimality in every instance, as opposed to just the worst case or the average case.

An algorithm that does sorted and random accesses is said to "make wild guesses" if it performs some random access on an object not previously encountered by sorted access. In practice, no real-world algorithm in this setup would make wild guesses.

2 Discussion

In [FLN03] the *Threshold Algorithm (TA)* is introduced, and the following theorem is given:

Theorem 6.1 [FLN03]. *Assume that the aggregation function t is monotone. Let* **D** *be the class of all databases. Let* **A** *be the class of all algorithms that correctly find the top k answers for t for every database and that do not make wild guesses. Then TA is instance optimal over* **A** *and* **D**.

In our paper [APV11], we introduced a new algorithm called the *Best Position Algorithm (BPA)*. We observed that the proof of Theorem 6.1 does not go through to show that (1) holds when $\mathcal{A}$ is BPA and $\mathcal{B}$ is TA (in this case (1) may or may not hold, but at least the proof does not go through). We therefore concluded that there was an error in the proof of Theorem 6.1. However, BPA uses a stronger form of random access than that defined (or allowed) in [FLN03]: specifically, in our paper, in a random access of object R in list L_i, we learn not only the score of object R in list L_i (as in [FLN03]), but also the position of object R in list L_i.

But, as Dr. Ronald Fagin points out, this knowledge is not authorized in the model defined in [FLN03]. Therefore, BPA is not in the class **A** of algorithms covered by Theorem 6.1 in [FLN03], and so there is no error in the proof of Theorem 6.1.

References

[APV11] Reza Akbarinia, Esther Pacitti, and Patrick Valduriez. Best position algorithms for efficient top-k query processing. *Inf. Syst.*, 36(6):973–989, 2011.

[FLN03] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. System Sci.*, 66:614–656, 2003.