Discovering Influential Data Objects over Time

Orestis Gkorgkas[1], Akrivi Vlachou[1,2⋆], Christos Doulkeridis[3⋆⋆], and Kjetil Nørvåg[1]

[1] Norwegian University of Science and Technology (NTNU), Trondheim, Norway
[2] Institute for the Management of Information Systems, R.C. "Athena", Athens, Greece
[3] Department of Digital Systems, University of Piraeus, Greece
{orestis,vlachou,cdoulk,noervaag}@idi.ntnu.no

Abstract. In applications such as market analysis, it is of great interest to product manufacturers to have their products ranked as highly as possible for a significant number of customers. However, customer preferences change over time, and product manufacturers are interested in monitoring the evolution of the popularity of their products, in order to discover those products that are consistently highly ranked. To take into account the temporal dimension, we define the *continuous influential query* and present algorithms for efficient processing and retrieval of continuous influential data objects. Furthermore, our algorithms support incremental retrieval of the next continuous influential data object in a natural way. To evaluate the performance of our algorithms, we conduct a detailed experimental study for various setups.

1 Introduction

In online marketplaces, top-k queries are typically used to present a limited number of products ranked according to the user's preferences. This is extremely helpful for the user as it enables decision-making, without the need to inspect large amounts of possibly uninteresting results. In addition, the user is not overwhelmed by the available information and can retrieve results that satisfy her information need. As a result, an increasing amount of research has focused on efficient techniques for top-k query processing lately [6].

From the perspective of the product manufacturers top-k queries are of great interest as well, since the visibility of a product clearly depends on the number of different top-k queries for which it belongs to the result set. The reason for this is twofold: 1) users usually consider only a few highly ranked products and ignore the remaining ones, and 2) products that appear in the top-k result sets are far more likely to be chosen by a potential customer, because these products satisfy the customers' preferences. Recently, *reverse top-k queries* [14] were proposed to study the visibility of a given product. A reverse top-k query returns the set of user preferences (i.e., customers) for which a

⋆ The work of Akrivi Vlachou was supported by the Action "Supporting Postdoctoral Researchers" of the Operational Program "Education and Lifelong Learning" (Action's Beneficiary: General Secretariat for Research and Technology), and is co-financed by the European Social Fund (ESF) and the Greek State.

⋆⋆ The research of Christos Doulkeridis was supported under the Marie-Curie IEF grant number 274063 with partial support from the Norwegian Research Council.

given product is in the result set of the respective top-k queries. Intuitively, a product that appears in as many as possible top-k result sets, has a higher visibility and therefore also a higher impact on the market. This has naturally lead to the definition of the *most influential products* based on the cardinality of their reverse top-k result sets [17]. Identifying the most influential products from a given set of products is important for market analysis, since the product manufacturer can estimate the impact of her products in the market.

However, an important aspect of a product's influence that has not been taken into account yet is its variance over time as the user preferences change. The customers' criteria can differ significantly over time for various reasons. For example, in online marketplaces, new customers pose queries and new preferences are collected. In addition, customers that have already posed queries will disconnect after some time. As user preferences change over time, a product which appears consistently in the top-k results of as many customers as possible, thus satisfying many customers' criteria at any time, has a higher impact on the market than a product that is absent from those results. Therefore, these products are the best candidate products to advertise to potential customers, and it is important to identify such products efficiently.

In this paper, we study for the first time the problem of finding the product that belongs consistently to the most influential products over time, the *continuous influential products*. This is an important problem for many real-life applications. For example, the products advertised on the first page of an online marketplace should be the products that have the greatest impact on the market, i.e., the products that are the most popular among the customers. Since customers change all the time, the products that consistently belong to the most influential products over time are more probable to attract many potential customers at any time. It is therefore essential to identify the objects (products) that have high impact over a period of time and despite the fluctuation of preferences these objects remain among the most influential objects. From now on we will use the terms *product* and *object* interchangeably.

In the following, we first define formally the problem of continuous influential products and provide a baseline algorithm that sequentially scans all time intervals in order to retrieve the most continuous influential product. Then, we provide a bounding scheme in order to facilitate early termination of our algorithms and avoid processing time intervals that do not alter the result set. Summarizing, the main contributions of this paper are:

- We study, for the first time, the problem of identifying the data object that has the highest impact over time.
- An appropriate score of influence (called *continuity score*) based on the reverse top-k query is defined to capture the product impact over a period of time.
- We derive upper and lower bounds for the continuity score of a given object that lead to efficient algorithms for retrieving the most continuous influential product. Two different algorithms are presented that provide early termination based on the bounds, but follow different strategies in order to terminate as soon as possible.
- We conduct a detailed experimental study for various setups and demonstrate the efficiency of our algorithms.

The rest of this paper is organized as follows: In Section 2 we provide the necessary preliminaries, while in Section 3 we formulate the problem statement. Section 4 presents a baseline algorithm for finding the data object that belongs consistently to the most influential products. Section 5 provides the foundation for our bounding scheme and describes the two threshold-based algorithms. Our experimental results are presented in Section 6. Section 7 provides an overview of related work. Finally, in Section 8 we conclude the paper.

2 Preliminaries

Let $\mathcal{D}$ be a dataspace with n dimensions $\{d_1, \ldots, d_n\}$ and S be a set of data objects on $\mathcal{D}$. A data object is represented as a point $o = \{o[1], \ldots, o[n]\}$ where $o[i]$ is the value of the attribute d_i.

2.1 Time-invariant Case

Given a monotonic scoring function $f : S \rightarrow \mathbb{R}$, a top-$k$ query returns the k best objects $o \in S$ ranked based on their scores $f(o)$. The most important and commonly used case of scoring functions is the weighted sum function, also called linear. For a given data object o and a weighting vector $\mathbf{w}$, its score $f_{\mathbf{w}}(o)$ is equal to the weighted sum of the individual values of o: $f_{\mathbf{w}}(o) = \sum_{i=1}^{n} w[i]o[i]$, where $w[i] \geq 0$ ($1 \leq i \leq n$). The value of each dimension $w[i]$ of the vector $\mathbf{w}$ is a weighting (preference) score on dimension d_i. Without loss of generality we assume that (a) minimum values are preferable, and (b) for each vector $\mathbf{w}$ it holds that $\sum_{i=1}^{n} w[i] = 1$. We denote the result set of top-k query defined by a weighting vector $\mathbf{w}$ as $TOP_k(\mathbf{w})$.

Definition 1. (Top-k query)*: Given a positive integer k and a user-defined weighting vector $\mathbf{w}$, the result set $TOP_k(\mathbf{w})$ of the top-k query is a ranked set of objects such that $TOP_k(\mathbf{w}) \subseteq S$, $|TOP_k(\mathbf{w})| = k$ and $\forall o, o' : o \in TOP_k(\mathbf{w})$, $o' \in S - TOP_k(\mathbf{w})$ it holds that $f_{\mathbf{w}}(o) \leq f_{\mathbf{w}}(o')$.*

Given a data set S of objects, a set W of weighting vectors, an object q and an integer k, a reverse top-k query returns all weighting vectors $\{\mathbf{w}\} \in W$ for which $q \in TOP_k(\mathbf{w})$. We denote the result set of weighting vectors $\{\mathbf{w}\}$, as $RTOP_k(q) = \{\mathbf{w}\}$.

Definition 2. (Reverse top-k query [14])*: Given an object q, a positive number k and two data sets S and W, where S represents data objects and W is a data set of weighting vectors, a weighting vector $\mathbf{w} \in W$ belongs to the reverse top-k result set $RTOP_k(q)$ of q, if and only if $\exists o \in TOP_k(\mathbf{w})$ such that $f_{\mathbf{w}}(q) \leq f_{\mathbf{w}}(o)$.*

We can also define the *influence score* of a data object by simply setting a single value k that determines the scope of the reverse top-k queries that are taken into account for identifying influential data objects.

Definition 3. (Influence score [17])*: Given a positive integer k, a data set S, and a set of preferences (weighting vectors) W, the* influence score *of a data object o is defined as the cardinality $|RTOP_k(o)|$ of the reverse top-k query result set of object o.*

Based on the definition of influence score, we define the ranked set of m most influential data objects.

Definition 4. (Top-m most influential data objects [17])*: Given a positive integer k, a data set S, and a set of preferences (weighting vectors) W, the result set $ITOP_k^m$ of the top-m influential query is a ranked set of objects such that $ITOP_k^m \subseteq S$, $|ITOP_k^m| = m$ and $\forall o, o' : o \in ITOP_k^m$, $o' \in S - ITOP_k^m$ it holds that $|RTOP_k(o)| \geq |RTOP_k(o')|$.*

2.2 Temporal Model

We model the time domain $\mathcal{T}$ as an ordered set of V disjoint time intervals that cover the complete domain, i.e., $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$ and $\mathcal{T}_i \bigcap \mathcal{T}_j = \emptyset$ for $i \neq j$. We denote the start and end of time interval $\mathcal{T}_i$ with $t_s(\mathcal{T}_i)$ and $t_e(\mathcal{T}_i)$ respectively. Then, it also holds that $t_e(\mathcal{T}_i) = t_s(\mathcal{T}_{i+1})$, and that $t_s(\mathcal{T}_1)$ and $t_e(\mathcal{T}_V)$ denote the start and end of $\mathcal{T}$ respectively. Obviously, the number of time intervals V is user-specified and application-dependent, and its exact value depends on the desired level of detail for monitoring temporal changes.

In order to model the interval that a user is online, we associate the weighting vector representing the user preferences with a time interval. Thus, given a weighting vector $\mathbf{w}$ and by abusing notation slightly, we denote the start of this interval as $t_s(\mathbf{w})$ and its end as $t_e(\mathbf{w})$. We are now ready to define the validity of a weighting vector with respect to a time domain $\mathcal{T}$ that consists of time intervals.

Definition 5. (Validity of weighting vector)*: Given a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$ and a weighting vector $\mathbf{w}$, the validity of $\mathbf{w}$ with respect to $\mathcal{T}$ is the interval $[t_s(\mathcal{T}_i), t_e(\mathcal{T}_j))$, where $t_s(\mathbf{w}) \in \mathcal{T}_i$ and $t_e(\mathbf{w}) \in \mathcal{T}_j$.*

Based on Definition 5, we consider as the validity period of a weighting vector $\mathbf{w}$ the interval defined by the start and end of the time intervals ($\mathcal{T}_i$ and $\mathcal{T}_j$) that enclose $t_s(\mathbf{w})$ and $t_e(\mathbf{w})$ respectively. Henceforth, we will use $t_s(\mathbf{w})$ to refer to $t_s(\mathcal{T}_i)$ and $t_e(\mathbf{w})$ to refer to $t_e(\mathcal{T}_j)$.

3 Problem Formulation

Given a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, we define a total order $\prec$ such that $\mathcal{T}_i \prec \mathcal{T}_j$ if $t_e(\mathcal{T}_i) \leq t_s(\mathcal{T}_j)$ for any $\mathcal{T}_i, \mathcal{T}_j \in \mathcal{T}$. Furthermore, we use $ITOP_k^m(\mathcal{T}_i)$ to refer to the result set of the top-m most influential objects by taking into account only the weighting vectors that are valid in the interval $\mathcal{T}_i$.

In order to identify products that are consistently highly ranked for multiple users as time passes, we define the *continuity score* of an object $o \in S$.

Definition 6. (Continuity score)*: Given a data set S, a set of weighting vectors W, and a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, the* continuity score *$cis(o)$ of an object $o \in S$ is the maximum number of consequent intervals $\mathcal{T}_i$ for which o belongs to the top-m most influential data objects, i.e., $o \in ITOP_k^m(\mathcal{T}_i)$.*

The continuity score of an object is practically a measure of the object's aggregated influence over time. As we aim to discover the object with highest continuity score, we define the most continuous influential data object in a straightforward way.

Definition 7. (Most continuous influential data object)*: Given a data set S, a set of weighting vectors W, and a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, the* most continuous influential data object $o \in S$ *is the object for which it holds that* $\nexists o' \in S$ *such that* $cis(o') > cis(o)$.

We are now ready to formally define the problem of discovering the most influential object over time. Another closely related problem is the one of discovering a ranked set of the most influential object over time.

Problem 1. (Most continuous influential object): Given a data set S, a set of weighting vectors W, and a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, find the most continuous influential object $o \in S$.

Problem 2. (Top-N continuous influential objects): Given a data set S, a set of weighting vectors W, a time domain $\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_V\}$, and an integer N, find the ranked set of the N most continuous influential object $\{o_1, o_2, \ldots, o_N\} \in S$.

In this paper, we focus our attention to Problem 1 and present our algorithms for solving this problem. However, our algorithms can be extended in a straightforward way to solve also Problem 2. For the sake of simplicity we omit the details here.

4 Sequential Interval Scan

A baseline algorithm for solving Problem 1 is to compute the $ITOP_k^m(\mathcal{T}_i)$ sets for all time intervals $\mathcal{T}_i$ of $\mathcal{T}$ and simply follow a counting approach of the appearance of any data object o in consequent intervals. Then, the most continuous influential object is the one that appears in the $ITOP_k^m(\mathcal{T}_i)$ sets for the maximum number of consequent intervals. In the following, we refer to this algorithm as *Sequential Interval Scan* (*SIS*).

Intuitively, in each iteration (lines 2–10 of Algorithm 1), *SIS* examines the next consequent interval $\mathcal{T}_i \in \mathcal{T}$ and computes the set of most influential objects $ITOP_k^m(\mathcal{T}_i)$ within $\mathcal{T}_i$. For each retrieved object $o \in ITOP_k^m(\mathcal{T}_i)$, we maintain its current continuity score, which is derived based on the processed intervals so far. We use the concept of *alive object* to refer to any object retrieved in a previous interval $\mathcal{T}_j (j \leq i)$ that is influential in all intervals between $\mathcal{T}_j$ and $\mathcal{T}_i$ and also belongs to the most recently processed $ITOP_k^m(\mathcal{T}_i)$ set; we also refer to objects that stopped being influential at some intermediate interval between $\mathcal{T}_1$ and $\mathcal{T}_i$ as *dead objects*. To ensure correctness, *SIS* needs to maintain the alive objects in a list A and only a single dead object d, which is the one with the maximum continuity score among all other dead objects (lines 4–6). Then, the retrieved influential objects in $\mathcal{T}_i$ are examined, and if an object belongs to A (i.e., was and remains alive) then its score is increased by 1 (line 8), otherwise we add it to A (line 9). After having examined all intervals, the algorithm terminates and reports the object with maximum score among the alive objects and the dead object (line 10).

The main shortcoming of *SIS* is that it needs to evaluate the $ITOP_k^m$ query for all $|V|$ time intervals. In the following, we study how to derive appropriate score bounds, in order to find the most continuous influential object without processing all queries.

Algorithm 1: *Sequential Interval Scan* (*SIS*)

Input: S:data set; k, m: the parameters of the $ITOP_k^m$ queries; $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_V\}$.
Output: o: the most continuous influential object.

```
1  A← ∅, d←null            (A: alive objects, d: dead object)
2  for i = 1 ... V do
3      I ← ITOP_k^m(T_i)
4      forall the o ∈ A and o ∉ I do
5          A← A − {o}                  (remove dead objects)
6          d← objMaxScore({d} ⋃{o})
7      forall the o ∈ I do
8          if o ∈ A then o.incScore()              (increase score)
9          else A← A ⋃{o}                  (add new objects)
10 o ← objMaxScore(A⋃{d})
11 return o
```

5 Algorithms with Early Termination

SIS relies on processing multiple consequent intervals of $\mathcal{T}$ to produce the most continuous influential object. In fact, all our algorithms rely on the evaluation of multiple $ITOP_k^m$ queries in different intervals $\mathcal{T}_i$, in order to find the most continuous influential object, however these intervals are not necessarily consequent. In this sense, our algorithms treat the $ITOP_k^m$ computation as black-box, hence any existing techniques that solve efficiently the problem of indentifying influential objects can be directly exploited by our algorithms.

Let us assume that at some point during query processing, a subset of (not necessarily consequent) intervals of $\mathcal{T}$ have been processed. We define the following sets for any retrieved data object o.

Definition 8. *Given a data object o, a set of processed intervals* $\{\mathcal{T}_i\}$ *and a set of corresponding results sets* $\{ITOP_k^m(\mathcal{T}_i)\}$*, we define:*

- $\mathcal{T}^+(o)$ *is the set of intervals* $\{\mathcal{T}_i\}$*, such that* $\mathcal{T}_i \in \mathcal{T}^+(o)$ *if* $o \in ITOP_k^m(\mathcal{T}_i)$
- $\mathcal{T}^-(o)$ *is the set of intervals* $\{\mathcal{T}_i\}$*, such that* $\mathcal{T}_i \in \mathcal{T}^-(o)$ *if* $o \notin ITOP_k^m(\mathcal{T}_i)$
- $\mathcal{LB}(o)$ *is a maximal sequence of intervals* $\{\mathcal{T}_i, \mathcal{T}_{i+1}, ..., \mathcal{T}_j\}$*, such that* $\forall \mathcal{T}_z \in \mathcal{LB}(o)$: $\mathcal{T}_z \in \mathcal{T}^+(o)$
- $\mathcal{UB}(o)$ *is a maximal sequence of intervals* $\{\mathcal{T}_i, \mathcal{T}_{i+1}, ..., \mathcal{T}_j\}$*, such that* $\forall \mathcal{T}_z \in \mathcal{UB}(o)$: $\mathcal{T}_z \in \mathcal{T} - \mathcal{T}^-(o)$

We emphasize that according to Definition 8, $\mathcal{T}^+(o)$ and $\mathcal{T}^-(o)$ are sets of intervals, i.e., they may contain non-consequent intervals. Instead, the sequences $\mathcal{LB}(o)$ and $\mathcal{UB}(o)$ contain consequent intervals, and moreover they are of maximal size, i.e., there exists no other longer sequence of intervals with the same properties respectively.

By exploiting the above sets and sequences, we derive an upper and a lower bound on the score of any candidate most continuous influential object.

Lemma 1 (Score bounds)*: The continuity score of object o is bounded by the lower bound $L(o)$ and the upper bound $U(o)$, i.e., $L(o) \leq cis(o) \leq U(o)$, where $L(o) = |\mathcal{LB}(o)|$ and $U(o) = |\mathcal{UB}(o)|$ are the lengths of the sequences $\mathcal{LB}(o)$ and $\mathcal{UB}(o)$ respectively.*

Proof. By contradiction. Let us assume that $cis(o) < L(o)$. Then it holds that there exists a sequence of processed intervals of length $|\mathcal{LB}(o)|$ such that for each time interval $\mathcal{T}_i$ of $\mathcal{LB}(o)$ it holds that $\mathcal{T}_i \in \mathcal{T}$ and $o \in ITOP_k^m(\mathcal{T}_i)$, which leads to a contradiction since $cis(o)$ is defined by the sequence of maximum length (according to Definition 6). Similarly, the assumption $cis(o) > U(o)$ leads to a contradiction, because for each time interval $\mathcal{T}_i$ of the sequence that defines $cis(o)$, it holds that $\mathcal{T}_i \notin \mathcal{T}^-(o)$ for any set of processed intervals $\{\mathcal{T}_i\}$. In other words, the sequence of intervals whose length defines $cis(o)$ is always smaller or equal to the sequence $\mathcal{UB}(o)$ whose length defines $U(o)$, hence $cis(o) \leq U(o)$ which is a contradiction.

The lower bound $L(o)$ of o is equal to the continuity score of the object o, if we take into account only the time intervals that have been processed so far. The upper bound $U(o)$ of o is the continuity score of the object o, if we assume that for any time interval $\mathcal{T}_i$ that does not belong to $\mathcal{T}^-$ the object o belongs to $ITOP_k^m(\mathcal{T}_i)$ (because optimistically for all unprocessed time intervals, o may belong to the most influential objects).

Theorem 1 (Early termination condition) *The data object o is the most continuous influential object, if for any other data object o' it holds that $L(o) \geq U(o')$.*

Proof. By contradiction. Let us assume that o is not the most continuous influential object, even though it holds that $L(o) \geq U(o')$. Thus, there must exist another object o' which is the most continuous influential object (i.e., $cis(o) < cis(o')$). Then, it holds that $L(o) \leq cis(o) \leq U(o)$ and $L(o') \leq cis(o') \leq U(o')$. From these inequalities, we derive that $L(o) \leq cis(o) < cis(o') \leq U(o')$ and finally that $L(o) < U(o')$, which is a contradiction.

The intuition of the above condition for early termination is that if an object has a continuity score based on some processed time intervals that is definitely higher than the score of any other object, then it can be safely reported as the most continuous influential object, because the score of any other object cannot increase sufficiently in the remaining time intervals.

Algorithm *SIS* is oblivious of the derived bounds and examines all time intervals following a brute-force approach. Hence, we propose two algorithms, termed *Early Termination Interval Scan* (*TIS*) and *Early Termination Best-First Interval* (*TBI*), that exploit the bounds to provide early termination. However, despite using the same concept of bounding, *TIS* and *TBI* follow different strategies in order to terminate as soon as possible. *TIS* aims to maximize as quickly as possible the lower bound of the current most continuous influential object o and therefore examines time intervals sequentially. Instead, *TBI* aims to reduce the upper bound of any object o by breaking the longest unprocessed sequence of time intervals.

Algorithm 2: *Early Termination Interval Scan* (*TIS*)

Input: S:data set; k, m: the parameters of the $ITOP_k^m$ queries; $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_V\}$.
Output: o: the most continuous influential object.

A$\leftarrow \emptyset$, d$\leftarrow$null `(A: alive objects, d: dead object)`
$i = 1$, upperBound $= 0$, lowerBound $= -1$
while *lowerBound* $<$ *upperBound* **do**
  $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_i)$
  $i = i + 1$
  $o \leftarrow$objMaxScore(A$\bigcup${d})
  lowerBound $= o$.score()
  forall the $o \in A$ ***and*** $o \notin \mathcal{I}$ **do**
    A$\leftarrow$ A $-$ {o} `(remove dead objects)`
    d$\leftarrow$ objMaxScore({d} $\bigcup\{o\}$)
  forall the $o \in \mathcal{I}$ **do**
    if $o \in A$ **then** o.incScore() `(increase score)`
    else A$\leftarrow$ A $\bigcup$ {o} `(add new objects)`
  $o' \leftarrow$objMaxScore(A$-\{o\}$)
  upperBound$= max(o'$.score()+$(V - i), d$.score())
return o

5.1 Early Termination Interval Scan

In this section, we describe the Early Termination Interval Scan (*TIS*) algorithm. Similar to the *SIS* algorithm, *TIS* processes sequentially the time intervals of time domain $\mathcal{T}$. However, the significant advantage of *TIS* lies in the fact that it can terminate early and report the most continuous influential object o without processing the $ITOP_k^m$ query for all V time intervals $\mathcal{T}_i$.

Intuitively, the main objective of *TIS* is to increase the lower bound of any retrieved object, by scanning the time intervals sequentially. Notice that only consequent time intervals may lead to a higher lower bound. *TIS* takes advantage of the fact that the time intervals are processed sequentially and computes the $L(o)$ and $U(o)$ without maintaining the sets $\mathcal{T}^-(o)$ and $\mathcal{T}^+(o)$. The lower bound is defined as the continuity score of the current most continuous influential object, which can be computed by maintaining only the alive and dead objects similar to *SIS*. For *TIS*, the upper bound is defined as the maximum value of the score of the dead object or the second highest score of the alive objects plus the number of remaining time intervals.

Although these bound definitions of *TIS* are simpler than the ones of lower and upper bound in Lemma 1, it can be shown that they are equivalent. The reason for their simplicity is that *TIS* examines intervals sequentially, which is a special case of interval selection and the computation of the bounds can be simplified. Instead, the bound definitions of Lemma 1 and Definition 8 apply in the general case of selecting any interval for processing next (not necessarily in a sequential manner).

Algorithm 2 contains the pseudocode of *TIS*. In each iteration, the next interval of the time domain $\mathcal{T}$ is examined and the result set $ITOP_k^m(\mathcal{T}_i)$ is computed. For each retrieved object a score is maintained which is the maximum number of consequent

intervals for which this object belongs to the respective $ITOP_k^m$ sets. The retrieved data objects that belong to the most recent $ITOP_k^m$ set are considered to be alive, while we also keep track of the dead object with the highest score.

In more detail, as long as the termination condition does not hold (lines 3-15), the $ITOP_k^m$ set for the next time interval is computed and the alive and dead objects are updated (lines 8-10, 12, 13), similarly to the case of the *SIS* algorithm. Furthermore, in each iteration, the current most continuous influential object o is found (line 6). The current score of o defines the lower bound (line 7), as any other point must have a higher score to become the most continuous influential. Also, the alive object o' with the second highest score is found (line 14)[4]. The maximum possible score of any object (regardless of whether it has been retrieved or not) is equal to maximum value between the score of the dead object and the score of o' plus the number of remaining unprocessed intervals. This is because any object that is still alive in the best case scenario may be in the $ITOP_k^m$ set for all remaining time intervals. Also, the score of the dead object cannot be increased further. Notice that if the same object appears in the $ITOP_k^m$ set, it is considered to be a new alive object. Any new alive object can appear only in the $V - i$ remaining time intervals. Thus, if the termination condition holds, no object can exceed the score of the currently most continuous influential object and the algorithm safely reports this object as the result.

It should be noted that *TIS* reports the most continuous influential object over a time domain, however it does not report its score accurately. One can draw parallels with Fagin's NRA algorithm [4], which produces the top-k objects from ranked lists but without guaranteeing accuracy of scores. In order to calculate the exact continuity score of the most continuous influential object, we need to proceed until we find an interval where the object does not belong to the $ITOP_k^m$ set.

5.2 Early Termination Best-first Interval

In the following, we describe the Early Termination Best-first Interval (*TBI*) algorithm. The most important difference to *TIS* is that *TBI* follows a different strategy with respect to interval selection, namely *TBI* does not process intervals sequentially.

For each retrieved object o, *TBI* maintains the two sets $\mathcal{T}^+(o)$ and $\mathcal{T}^-(o)$ that correspond to the processed time intervals for which o belongs to or not to the most influential data objects respectively. This information is sufficient to derive the lower bound $L(o)$ and upper bound $U(o)$ of o. The algorithm first computes the influential objects $ITOP_k^m(\mathcal{T}_1)$ and $ITOP_k^m(\mathcal{T}_V)$. The following example demonstrates the information maintained by *TBI* at this point.

Example 1 *Let us assume that* $V = 6$, $m = 2$, *and that* $ITOP_k^m(\mathcal{T}_1) = \{o_1, o_2\}$ *and* $ITOP_k^m(\mathcal{T}_6) = \{o_2, o_3\}$. *Then,* TBI *maintains the following sets:* $\mathcal{T}^+(o_1) = \{\mathcal{T}_1\}$, $\mathcal{T}^-(o_1) = \{\mathcal{T}_6\}$, $\mathcal{T}^+(o_2) = \{\mathcal{T}_1, \mathcal{T}_6\}$, $\mathcal{T}^-(o_2) = \emptyset$, $\mathcal{T}^+(o_3) = \{\mathcal{T}_6\}$, $\mathcal{T}^-(o_3) = \{\mathcal{T}_1\}$. *In addition, the derived bounds are:* $L(o_1) = 1$, $U(o_1) = 5$, $L(o_2) = 1$, $U(o_2) = 6$, $L(o_3) = 1$, $U(o_3) = 5$.

[4] In the extreme case where $A - \{o\} = \emptyset$ we assume that $o'.score = 0$.

TBI iteratively selects a time interval that has not been processed yet and computes the influential objects in the selected time interval. Then, the bounds of retrieved objects can be updated as indicated in the following.

Example 2 *Continuing the previous example, assume that the next interval that is processed is* $\mathcal{T}_3$ *and* $ITOP_k^m(\mathcal{T}_3) = \{o_2, o_4\}$. *Then, the following sets are maintained:* $\mathcal{T}^+(o_1) = \{\mathcal{T}_1\}$, $\mathcal{T}^-(o_1) = \{\mathcal{T}_3, \mathcal{T}_6\}$, $\mathcal{T}^+(o_2) = \{\mathcal{T}_1, \mathcal{T}_3, \mathcal{T}_6\}$, $\mathcal{T}^-(o_2) = \emptyset$, $\mathcal{T}^+(o_3) = \{\mathcal{T}_6\}$, $\mathcal{T}^-(o_3) = \{\mathcal{T}_1, \mathcal{T}_3\}$, $\mathcal{T}^+(o_4) = \{\mathcal{T}_3\}$, $\mathcal{T}^-(o_4) = \{\mathcal{T}_1, \mathcal{T}_6\}$. *In addition, the bounds are updated as follows:* $L(o_1) = 1$, $U(o_1) = 2$, $L(o_2) = 1$, $U(o_2) = 6$, $L(o_3) = 1$, $U(o_3) = 3$, $L(o_4) = 1$, $U(o_4) = 4$.

The remaining challenge is how to select the most beneficial time interval for the next influential query to be processed, i.e., the time interval that will lead the algorithm to terminate as quickly as possible. *TBI* follows a best-first approach by selecting the time interval that will split the longest $\mathcal{UB}(o)$ sequence for any o in the queue. Intuitively, this "breaks" long sequences of unknown time intervals, in an attempt to reduce the upper bound of any data object.

In more detail, the next interval to be processed is selected in the following way. Given a candidate data object o and the corresponding $\mathcal{UB}(o) = \{\mathcal{T}_i, ..., \mathcal{T}_j\}$, the middle time interval $\mathcal{T}_z$ is computed such that $z = i + \lceil \frac{j-i}{2} \rceil$. If $\mathcal{T}_z \notin \mathcal{T}^+(o)$ then $\mathcal{T}_z$ is the next interval. Otherwise it means that $\mathcal{T}_z$ has been already processed and in this case the sequence $\{\mathcal{T}_i, ..., \mathcal{T}_z\}$ is tried to be split by finding the middle interval $\mathcal{T}_{z'}$ of it. If also $\mathcal{T}_{z'} \in \mathcal{T}^+(o)$, then the middle interval of $\{\mathcal{T}_z, ..., \mathcal{T}_j\}$ is examined if it qualifies for being the next interval. This is done recursively by examining always the longest sequence until an interval is found that does not belong to $\mathcal{T}^+(o)$. Note that it is guaranteed that such an interval exists, because otherwise $L(o) = U(o)$ and the algorithm terminates. Intuitively, computing $ITOP_k^m(\mathcal{T}_z)$ may break the longest sequence $\mathcal{UB}(o)$ in two smaller sequences if $o \notin ITOP_k^m(\mathcal{T}_z)$, thus reducing the upper bound, which will allow the algorithm to terminate faster.

During query processing, *TBI* keeps the retrieved data objects in a priority queue. The queue is sorted in descending order based on the upper bound $U(o)$ of each object o, so that immediate access to the object with the highest upper bound is provided. Algorithm 3 presents the pseudocode of *TBI*. First, the intervals $\mathcal{T}_1$ and $\mathcal{T}_V$ are processed and the retrieved objects are inserted in the queue (lines 1–4). The lower and upper bounds are initiated based on the object located at the head of the queue (lines 5, 6). In each iteration, we remove from the queue the object o (candidate object) with maximum upper bound $U(o)$ (line 8). Note that the candidate object is not necessary the object with the highest continuity score based on the processed partitions (which is the lower bound), and there may exist another object o' that has a higher score (lower bound) currently. But it is guaranteed that the algorithm cannot terminate at this iteration even if o' was processed next, because it holds that $L(o') \leq U(o')$ and $U(o') \leq U(o)$ so that the termination condition cannot hold. Thus, *TBI* does not process unnecessary time intervals.

After selecting the candidate o with the highest upper bound, *TBI* recursively selects the middle interval to be processed (line 9) and processes the query (line 10). Afterwards, the queue is updated (line 11), which means that every object in $ITOP_k^m(\mathcal{T}_i)$

Algorithm 3: *Early Termination Best-first Interval* (*TBI*)

Input: S:data set; k, m: the parameters of the $ITOP_k^m$ queries; $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_V\}$.
Output: o: the most continuous influential object.

1 $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_1)$
2 queue.update($\mathcal{I}$)
3 $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_V)$
4 queue.update($\mathcal{I}$)
5 upperBound $= U(queue.peek())$
6 lowerBound $= L(queue.peek())$
7 **while** *lowerBound<upperBound* **do**
8 $o \leftarrow queue.dequeue()$
9 $i =$ nextInterval($\mathcal{UB}(o)$) `(find next interval)`
10 $\mathcal{I} \leftarrow ITOP_k^m(\mathcal{T}_i)$
11 queue.update($\mathcal{I}$)
12 upperBound $= U(queue.peek())$
13 lowerBound $= L(o)$
14 queue.enqueue(o) `(add o back to queue)`
15 **return** o

is either added to the queue (if it is the first time that it was retrieved) or the existing object is updated by changing the corresponding $\mathcal{T}^+$ set. Moreover, for every object in the queue that does not belong in $ITOP_k^m(\mathcal{T}_i)$, the set $\mathcal{T}^-$ is updated.

The algorithm terminates when it holds that the candidate object o has $L(o) \geq U(o'), \forall o' \in$ queue. This is the *termination condition* (line 7), which means that o has a higher lower bound than the upper bound of the current head object o' in the queue.

In principle, we can also free part of the memory during the processing of the algorithm, by evicting candidate points that will never become the most continuous influential object. The condition for eviction is if a candidate object o has $U(o) \leq L(o')$, where o' is another candidate object.

6 Experimental Evaluation

In this section, we present the results of the experimental evaluation. All algorithms are disk-based and implemented in Java, and the experiments run on 2x Intel Xeon X5650 Processors (2.66GHz), 128GB. The index structure used was an R-tree with a buffer size of 100 blocks and the block size is 4KB.

6.1 Experimental Setup

Data sets. For the data set S we employ both real and synthetic data collections, namely uniform (UN), correlated (CO) and anticorrelated (AC). For the uniform data set, the data object values for all n dimensions are generated independently using a uniform distribution. The correlated and anticorrelated data sets are generated as described in [3].

In addition, we use two real data sets. NBA consists of 17265 5-dimensional tuples, representing a player's performance per year. The attributes are average values

of: number of points scored, rebounds, assists, steals and blocks. HOUSE (Household) consists of 127930 6-dimensional tuples, representing the percentage of an American family's annual income spent on 6 types of expenditure: gas, electricity, water, heating, insurance, and property tax.

For the data set W of the weighting vectors, two different data distributions are examined, namely uniform (UN) and clustered (CL). The clustered data set W is generated as described in [14] and models the case that many users share similar preferences. In more detail, first C_W cluster centroids that belong to the (n-1)-dimensional hyperplane defined by $\sum w[i] = 1$ are selected randomly. Then, each coordinate is generated on the (n-1)-dimensional hyperplane by following a normal distribution on each axis with variance σ_W^2, and a mean equal to the corresponding coordinate of the centroid. We consider $V = 100$ time intervals and assign a vector $\mathbf{w}$ to a time interval $\mathcal{T}_i$ ($i \in [1, 100]$) uniformly at random.

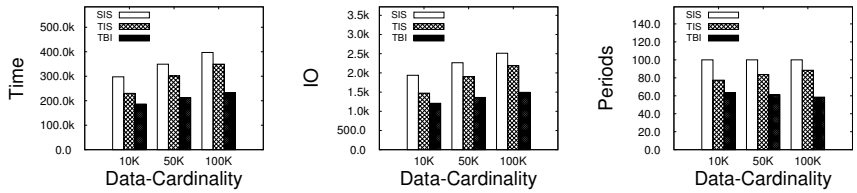
We conduct a thorough sensitivity analysis varying the dimensionality (2-5d), the cardinality (10K-100K) of S, the cardinality (100K-500K) of W the value of k (5-15), the value of m (5-15), and the number of intervals V (50-150). Unless explicitly mentioned, we use the default setup of: $|S| = 50K$, $|W| = 300K$, d=3, k=10, m=10, V=100, and uniform distribution for S and W. For the clustered data set W we use $C_W = 5$ and $\sigma_W = 0.1$, and try different values of σ_W.

Algorithms. We evaluate: a) sequential interval scan (*SIS*), b) early termination interval scan (*TIS*), and c) early termination best-first interval (*TBI*). All algorithms use the computation of the top-m most influential data objects as a black-box. In particular, the branch-and-bound algorithm proposed in [17] is employed for the underlying computation of influential objects.

Metrics. Our metrics include: a) the total execution time, b) the number of I/Os, and c) the number of processed time intervals by each algorithm. Notice that we do not measure the I/Os that occur by reading W, since this is the same for every algorithm and does not affect their comparative performance. For our experiments on synthetic data, we report the average of each metric over 10 different instances of the data set. We generate the different instances by keeping the parameters fixed and changing the seeds of the random number generator. We adopt this approach in order to factor out the effects of randomization.

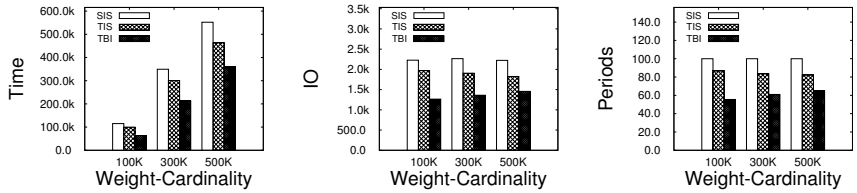
6.2 Performance of Query Processing

Effect of data set size $|S|$. Fig. 1 illustrates the performance of all algorithms when we vary the data set cardinality. For all metrics, *TBI* outperforms both *TIS* and *SIS*. In terms of time (Fig. 1(a)), *TBI* is significantly faster than the other algorithms, and more importantly its gain increases as the data set size increases. This is strong evidence that *TBI* scales gracefully with $|S|$. Similar observations can be made for the I/O metric depicted in Fig. 1(b). Fig. 1(c) depicts the number of processed intervals by each algorithm, which is a factor that affects all other metrics. *SIS* always processes the complete set of V intervals. *TIS* improves the performance of *SIS*, by exploiting the bounds and allowing for early termination. It should be clarified that *TIS* cannot process fewer than $V/2$ intervals to produce the correct result. Thus, in this setup ($V = 100$), *TIS* would in



(a) Time (b) I/O (c) Processed intervals

Fig. 1. Effect of varying data cardinality $|S|$.



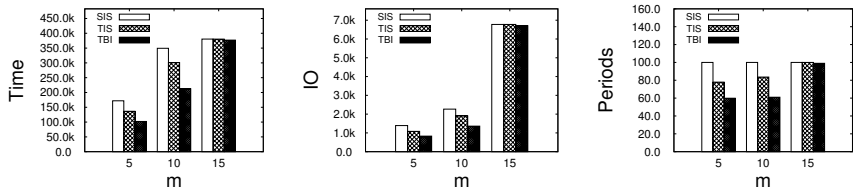
(a) Time (b) I/O (c) Processed intervals

Fig. 2. Effect of varying cardinality of weighting vectors $|W|$.

best case process 50 intervals. Still, *TBI* outperforms all other algorithms, which indicates that its best-first strategy for selecting the next interval performs more efficiently.

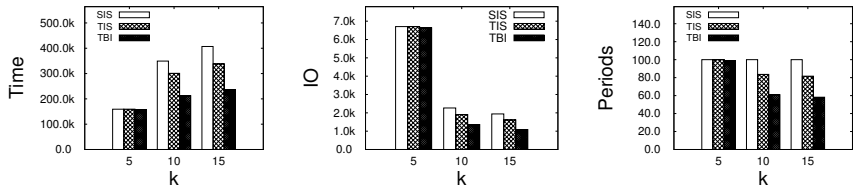
The advantage in the performance of *TIS* against *SIS* lies on the fact that *TIS* terminates when it is certain that the object with the highest continuity score cannot be surpassed. The advantage of *TBI* over *TIS* lies on the fact that TIS alters the upper and lower bounds each time by 1 interval while TBI splits the largest unseen interval in half. In the best case, every $2^{\lambda+1} - 1$ steps the upper bound will have been reduced to $|V|/(2^{\lambda} + 1)$, while for *TIS* the upper bound in the best case will have been reduced to $|V| - (2^{\lambda+1} - 1)$. Obviously in the early steps of *TBI* the upper bound and lower bound converge faster than in *TIS*.

Effect of varying cardinality of weighting vectors $|W|$. In Fig. 2, we study the effect of increasing the size of $|W|$. First, with respect to time (depicted in Fig. 2(a)), we observe that time increases linearly with $|W|$ for all algorithms. This is expected, since the size of W determines the number of user preferences, which is the number of potential top-k queries that may be evaluated. When the induced I/Os are considered, we see in Fig. 2(b) that all algorithms show a stable performance irrespective of $|W|$. Recall that we only measure the I/O induced on data set S, and this metric does not depend on W. Hence, this explains the stability of the measured I/O values. Fig. 2(c) shows the processed intervals by each algorithm. Also in this setup, *TBI* performs better than its competitors. It can be also observed that the size of W does not affect the number of processed intervals. The observations made for varying the data cardinality hold also here. The increased computation cost with respect to time is due to the fact that the complexity of the $ITOP_k^m$ queries increases when the weight cardinality rises.



(a) Time (b) I/O (c) Processed intervals

Fig. 3. Effect of varying m.



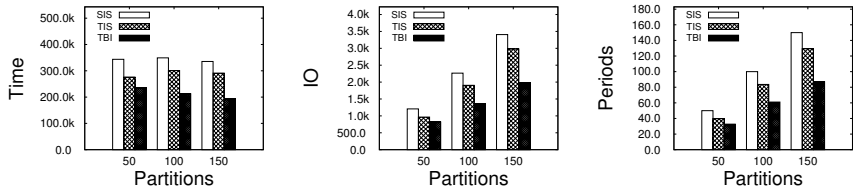
(a) Time (b) I/O (c) Processed intervals

Fig. 4. Effect of varying k.

Effect of varying m. Fig. 3 shows the effect of increasing the number of retrieved influential objects. *TBI* has a significant performance advantage over *SIS* and *TIS* when the value of m is relatively small. When m increases, we observe that all algorithms demonstrate similar performance. The reason for this behavior is that for larger values of m we observe that there exist data objects that have maximum continuity score equal to V. In other words, some data objects are influential in all V intervals. In this degenerate case, no algorithm can perform better than *SIS*, since all intervals must be processed in order to safely report the most continuous influential object.

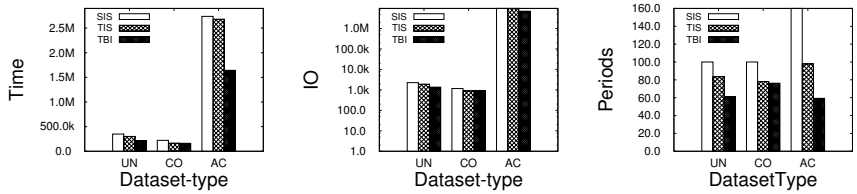
Effect of varying k. As k increases, all algorithms need more time to produce the results set as depicted in Fig. 4(a). For smaller values of k, all algorithms perform similarly because again there exist objects with maximum continuity score, which can only be reported when all intervals have been processed. For higher values of k, *TBI* performs better than all other algorithms.

Effect of varying V. Based on Fig. 5(a), we observe that *TIS* has a bigger advantage over *SIS* for small number of intervals, while *TBI* benefits more from large number of intervals. The reason is that the more the time intervals the smaller the possibility for an object to be influential in all of them. This fact is exploited by *TBI* which manages to reduce the upper bound fast in the first loops of its execution, and thus the lower bound and the upper bound converge fast and allow *TBI* to finish earlier that *SIS* and *TIS*. Contrary to the upper bound, the lower bound is expected to increase slowly when the time domain is partitioned with high granularity since many objects (including the one with the highest continuity score) are likely to disappear and re-appear from the $ITOP_k^m$ influential sets, and consequently the convergence between the bounds is delayed.



(a) Time (b) I/O (c) Processed intervals

Fig. 5. Effect of varying V.



(a) Time (b) I/O (c) Processed intervals

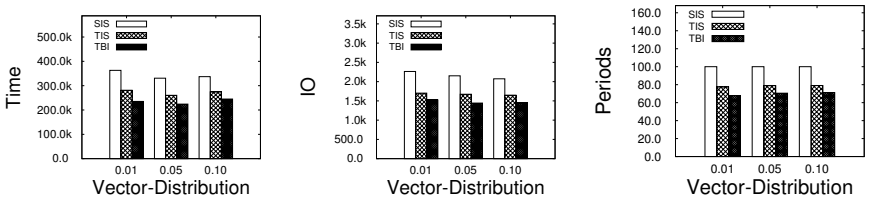
Fig. 6. Effect of varying the data distribution of S.

Effect of different data distributions of S. Fig. 6 compares the performance of the three algorithms when the set of data objects S follow different distributions, namely uniform (UN), correlated (CO) and anti-correlated (AC). Notice that we use log-scale in Fig. 6(b). Clearly, the cost of all algorithms (in terms of time and I/O) increases for AC. This is due to the more expensive processing of the underlying computation for influential data objects in the case of AC. However, as depicted in Fig. 6(c), the difference between the algorithms is significant in terms of processed intervals. Also, notice that *TBI* is not significantly affected by the challenging AC data distribution and processes comparable number of intervals, irrespective of the data distribution of S.

Effect of clustered data set W. Fig. 7 shows the results of using a clustered data set W for different values of σ_W. Smaller values of σ_W correspond to more clustered data sets, or in other words the weighting vectors are more compact with respect to the cluster centroids. For smaller values of σ_W, *TBI* performs better than the other algorithms. However, an interesting observation is that when σ_W increases, the performance of *TIS* tends to be similar to *TBI*.

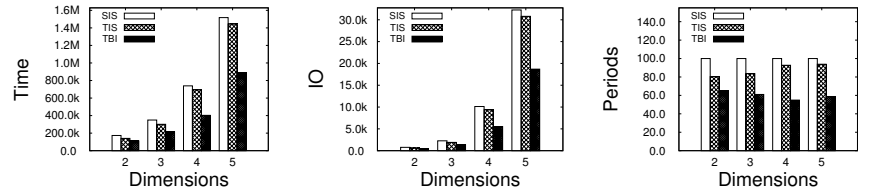
Table 1. Experimental results of real data sets NBA and HOUSE.

	NBA data set			**HOUSE data set**		
Algorithm	**Time(sec)**	**I/O**	**Proc. Intervals**	**Time(sec)**	**I/O**	**Proc. Intervals**
SIS	822.77	8119	100.0	903.00	9476	100.0
TIS	712.74	6988	85.9	867.33	9189	95.2
TBI	454.60	4508	55.6	865.58	9235	97.1



(a) Time (b) I/O (c) Processed intervals

Fig. 7. Effect of varying the standard deviation for clustered data set W.



(a) Time (b) I/O (c) Processed intervals

Fig. 8. Effect of varying dimensionality n.

Effect of increasing dimensionality. Fig. 8 illustrates the results for varying the number of dimensions. With respect to time (Fig. 8(a)) and I/O (Fig. 8(b)), the performance of all algorithms degrades with increased dimensionality. However, notice that *TBI* is less affected by the increased dimensionality, compared to the other algorithms. With respect to the number of processed intervals (Fig. 8(c)), we observe that this metric increases with dimensionality in the case of *TIS*. When *TBI* is considered, the metric drops for increased values of n. This means that *TBI* manages to process fewer intervals as n grows, however each top-k processing costs on average more for increased n, which explains why both time and I/O increase for *TBI* too.

Experiments with real data sets. Table 1 shows the results obtained for the two real data sets employed in our study (NBA and HOUSE). In both cases, the observed values follow the results and conclusions drawn from synthetic data.*TBI* outperforms the other two algorithms for the NBA data set. *TBI* needs almost half the time of *SIS* to identify the most influential object. In the case of HOUSE data set the difference between *TIS* and *TBI* is marginal but both outperform *SIS*. The higher the dimensionality of the problem the smaller is the probability that the most influential object will be influential for a long time interval. This fact reduces the advantage of *TBI* over *TIS* and the two algorithms have similar performance.

7 Related Work

Top-k queries have been well-studied in the last years to enable ranked retrieval of objects based on user preferences (for a thorough overview we refer to [6]). Recently, reverse top-k queries [14, 15] have been proposed to retrieve the set of users that have

a given object in their top-k list. An improved branch-and-bound algorithm for reverse top-k queries was proposed in [18], while [5] presents an approach that is beneficial when a large number of reverse top-k queries need to be processed. Another approach based on preprocessing all top-k queries for answering reverse top-k queries is presented in [21]. Moreover, in [16] the authors define the distance-based reverse top-k query and monitor its result set for mobile devices, when the values of one dimension (distance) change dynamically as devices move. Reverse queries are also studied in [2] following a unified approach. The authors examine the Inverse ϵ-Range, Inverse $k-$NN and Inverse Dynamic Skyline queries using a three-filter approach. The first two filters use only the query points whose number is usually small and the third query accesses the data points in ascending order of maximum distance from the query points.

Lately, several research initiatives have been proposed to study the influence of data objects. In this paper, we adopt the definition of influence that was first introduced by Vlachou *et al.* [17], where the influential objects are those that appear in the top-k lists of many users, i.e., have the larger reverse top-k results. A different definition of influence is used in [1], where the authors try to discover attractive products to users using the principle of skyline sets [3]. Other approaches try to identify the attributes of products that maximize its visibility [11] or the region in the space defined by the products' attributes where a product can be promoted [19, 20].

Jestes *et al.* [7] study the problem of performing top-k queries on a time window. They assume that the values of the objects change over time and instead of performing instant top-k queries they retrieve the top-k objects by ranking them after aggregating their scores in a query interval. Lee *et al.* [9] discuss the idea of objects that appear continuously in top-k queries over data streams. They focus on discovering objects that appear continuously on a moving window of time. In [13] the authors study techniques for *durable top-k search* in document archives, where the aim is to identify documents that are consistently in the top-k results of a given query. Kontaki *et al.* [8] study the problem of discovering the objects that remain the most dominant over a data stream. Our main difference towards these approaches is that they consider the ranking functions to be static while the values of the objects are changing while we consider the exact opposite. Other work related to top-k and time includes processing of top-k queries on temporal data where the aim is finding the top-k objects at a particular time [10], as well as monitoring top-k queries over sliding windows [12].

8 Conclusions

In this paper, we studied for the first time the problem of finding the *most continuous influential products* that belong consistently to the most influential products over time. To this end, we defined the continuous influential query, where the influence score is defined based on reverse top-k queries and it changes as user preferences change over a long time period. In order to be able to efficiently discover the continuous influential products, we studied the properties of the proposed continuity score and derived appropriate upper and lower bounds. In turn, this lead to the design of efficient algorithms with the salient property of early termination. To evaluate our approach, we conducted a thorough experimental study that demonstrates the efficiency of our algorithms.

References

1. A. Arvanitis, A. Deligiannakis, and Y. Vassiliou. Efficient influence-based processing of market research queries. In *Proc. of CIKM*, pages 1193–1202, 2012.
2. T. Bernecker, T. Emrich, H.-P. Kriegel, N. Mamoulis, M. Renz, S. Zhang, and A. Zfle. Inverse queries for multidimensional spaces. In D. Pfoser, Y. Tao, K. Mouratidis, M. Nascimento, M. Mokbel, S. Shekhar, and Y. Huang, editors, *Advances in Spatial and Temporal Databases*, volume 6849 of *Lecture Notes in Computer Science*, pages 330–347. Springer Berlin Heidelberg, 2011.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. of ICDE*, pages 421–430, 2001.
4. R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. In *Proc. of PODS*, pages 102–113, 2001.
5. S. Ge, L. H. U, N. Mamoulis, and D. W. Cheung. Efficient all top-k computation: A unified solution for all top-k, reverse top-k and top-m influential queries. *TKDE*, 25(5):1015–1027, 2013.
6. I. F. Ilyas, G. Beskales, and M. A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comp. Surv.*, 40(4), 2008.
7. J. Jestes, J. M. Phillips, F. Li, and M. Tang. Ranking large temporal data. *PVLDB*, 5(11):1412–1423, 2012.
8. M. Kontaki, A. N. Papadopoulos, and Y. Manolopoulos. Continuous top-k dominating queries. *TKDE*, 24(5):840–853, 2012.
9. M.-L. Lee, W. Hsu, L. Li, and W. H. Tok. Consistent top-k queries over time. In *Proc. of DASFAA*, pages 51–65, 2009.
10. F. Li, K. Yi, and W. Le. Top-k queries on temporal data. *VLDB Journal*, 19(5):715–733, Oct. 2010.
11. M. Miah, G. Das, V. Hristidis, and H. Mannila. Standing out in a crowd: Selecting attributes for maximum visibility. In *Proc. of ICDE*, pages 356–365, 2008.
12. K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *Proc. of SIGMOD*, pages 635–646, 2006.
13. L. H. U, N. Mamoulis, K. Berberich, and S. J. Bedathur. Durable top-k search in document archives. In *Proc. of SIGMOD*, pages 555–566, 2010.
14. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Reverse top-k queries. In *Proc. ICDE*, pages 365–376, 2010.
15. A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg. Monochromatic and bichromatic reverse top-k queries. *TKDE*, 23(8):1215–1229, 2011.
16. A. Vlachou, C. Doulkeridis, and K. Nørvåg. Monitoring reverse top-k queries over mobile devices. In *Proc. of MobiDE*, pages 17–24, 2011.
17. A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Identifying the most influential data objects with reverse top-k queries. *PVLDB*, 3(1-2):364–372, 2010.
18. A. Vlachou, C. Doulkeridis, K. Nørvåg, and Y. Kotidis. Branch-and-bound algorithm for reverse top-k queries. In *Proc. of SIGMOD*, 2013.
19. T. Wu, Y. Sun, C. Li, and J. Han. Region-based online promotion analysis. In *Proc. of EDBT*, pages 63–74, 2010.
20. T. Wu, D. Xin, Q. Mei, and J. Han. Promotion analysis in multi-dimensional space. *PVLDB*, 2(1):109–120, 2009.
21. A. Yu, P. K. Agarwal, and J. Yang. Processing a large number of continuous preference top-*k* queries. In *Proc. of SIGMOD*, pages 397–408, 2012.