Data & Knowledge Engineering 83 (2013) 1–19

Contents lists available at SciVerse ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

Subspace top-*k* query processing using the hybrid-layer index with a tight bound

Jun-Seok Heo [a], Junghoo Cho [b], Kyu-Young Whang [a,*]

[a] *Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea*
[b] *University of California, LA, USA*

ARTICLE INFO

Article history:
Received 28 April 2011
Received in revised form 10 July 2012
Accepted 14 July 2012
Available online 11 September 2012

Keywords:
Access methods
Query
Top-*k* queries
Subspaces
Linear scoring functions
Layering
Listing
Hybrid

ABSTRACT

In this paper, we propose the *Hybrid-Layer Index* (simply, the *HL-index*) that is designed to answer top-*k* queries efficiently when the queries are expressed on any *arbitrary subset* of attributes in the database. Compared to existing approaches, the HL-index significantly reduces the number of tuples accessed during query processing by pruning unnecessary tuples based on two criteria, i.e., it filters out tuples both (1) *globally* based on the combination of *all* attribute values of the tuples like in the layer-based approach (simply, *layer-level filtering*) and (2) based on *individual* attribute values specifically used for ranking the tuples like in the list-based approach (simply, *list-level filtering*). Specifically, the HL-index exploits the synergic effect of integrating the layer-level filtering method and the list-level filtering method. Through an in-depth analysis of the interaction of the two filtering methods, we derive a tight bound that reduces the number of tuples retrieved during query processing while guaranteeing the correct query results. We propose the HL-index construction and retrieval algorithms and formally prove their correctness. Finally, we present the experimental results on synthetic and real datasets. Our experiments demonstrate that the query performance of the HL Index significantly outperforms other state-of-the-art indexes in most scenarios.

1. Introduction

Computing top-*k* answers quickly is becoming ever more important as the size of databases grows and as more users access data through interactive interfaces [1]. When a database is large, it may take minutes (if not hours) to compute the complete answer to a query if the query matches millions of the tuples in the database. Most users, however, are interested in looking at just the top few results (ranked by a small set of attribute values that the users are interested in) and they want to see the results immediately after they issue the query.

As an example, consider a database of digital cameras, which has many attributes such as price, manufacturer, model number, weight, size, pixel count, sensor size, etc. Among these attributes, a particular user is likely to be interested in a small subset when they make a decision to purchase. For example, a user who wants to buy a cheap compact digital camera will be mainly interested in the price and the weight and may issue a query like

```
SELECT * FROM Cameras ORDER BY 0.5*price+0.5*weight ASC LIMIT k.
```

Another user who primarily cares about the quality of the pictures will be more interested in the pixel count and sensor size and issue a query like

```
SELECT * FROM Cameras ORDER BY 0.4*pixelCount+0.6*sensorSize DESC LIMIT k.
```

* Corresponding author.
E-mail addresses: jsheo@mozart.kaist.ac.kr (J.-S. Heo), cho@cs.ucla.edu (J. Cho), kywhang@mozart.kaist.ac.kr (K.-Y. Whang).

0169-023X/$ – see front matter © 2012 Published by Elsevier B.V.
http://dx.doi.org/10.1016/j.datak.2012.07.001

To handle scenarios like the above, we propose the *Hybrid-Layer Index* (simply, the *HL-index*) that is designed to answer top-*k* queries on an *arbitrary subset* of the attributes efficiently. There exist a number of approaches for efficient computation of top-*k* answers. For example, in their seminal work, Fagin et al. [2,3] designed a series of algorithms that consider a tuple as a potential top-*k* answer only if the tuple is ranked high in *at least* one of the attributes used for ranking. We refer to this approach as the *list-based approach* because the algorithms require maintaining one sorted list per each attribute. While this approach shows significant improvement compared to earlier work, it often considers an unnecessarily large number of tuples. For instance, when a tuple is ranked high in *one* attribute but low in all others, the tuple is likely to be ranked low in the final answer and can potentially be ignored, but the list-based approach has to consider it because of its high rank in that one attribute. As the size of the database grows, this becomes an acute problem because there are likely to be more tuples that are ranked high in one attribute but low overall.

To avoid this pitfall, Chang et al. [4] proposed an algorithm that constructs a global index based on the combination of *all* attribute values and uses this index for top-*k* answer computation. We refer to this approach as the *layer-based approach* because it builds an index that partitions the tuples into multiple layers. The layer-based approach avoids the pitfall of the list-based algorithms, but it also has the opposite problem. Because the index is constructed on *all* attributes, it does not perform well when the query ranks tuples by a small *subset* of the attributes. A tuple may be ranked high globally on many attributes, but it may be ranked low for a particular subset of attributes used for a query.

One simple way to address the drawback of the layer-based approach is to build one dedicated index per *subsets* of attributes and use the appropriate index for a query as in [5,6]. We refer to these approaches as the *view-based approach*. Clearly, view-based approaches lead to high query performance if the "closest" answers to the query issued by a user has been precomputed. Otherwise, they lead to low query performance. They can improve query performance by increasing the number of indexes, but the space overhead increases in proportion to the number of indexes [7].

Our proposed HL-index tries to avoid all pitfalls of the existing approaches in the following ways. By careful integration of the list-based and the layer-based approaches, it is able to filter out a tuple both by the *global* combination of *all* of its attribute values (like in the layer-based approach) and by the *individual* consideration of the particular attribute values used for ranking (like in the list-based approach). In addition, one HL-index can handle *any* queries on an *arbitrary subset* of the attributes avoiding the space overhead of the view-based approach. More precisely, we make the following contributions in this paper.

- We propose the HL-index that can be used for answering top-*k* queries on an arbitrary subset of attributes. The HL-index can be built for either (1) linear scoring functions (including monotone and non-monotone linear functions) or (2) monotone scoring functions (including linear and non-linear monotone functions). The HL-index has significantly more pruning power than existing approaches and does not require a separate index customized for each class of queries on different subsets of attributes.
- We present the algorithms for processing top-*k* queries using the HL-index. Through an in-depth analysis of the interaction of the list-based and layer-based approaches, we derive a tight bound to minimize the number of tuples that are retrieved during query processing and to guarantee the correctness of the computed results. We also provide formal proofs of correctness of those algorithms.
- We conduct extensive experiments comparing the performance of the HL-index with those of existing approaches on both synthetic and real data. The HL-index can exploit the synergic effect of the list-based approach and the layer-based approach by meticulous integration of the two approaches. As a result, the HL-index shows better performance over existing approaches for practically all settings in our experiments. In particular, our experiments show that the HL-index performs particularly well when the size of the database is large, leading to a factor of three or more improvement for a database of million tuples in our experiments.

The rest of the paper is organized as follows: We first go over related work in Section 2 and we formally define the top-*k* queries that we handle in Section 3. Then, in Section 4, we describe the HL-index construction algorithm and, in Section 5, explain the top-*k* query processing algorithm using the HL-index and prove its correctness. In Section 6 we present our experiments that compare the performance of the HL-index to existing approaches. We conclude the paper in Section 7.

2. Related work

There have been a number of methods proposed to answer top-*k* queries by accessing only a subset of the database. We categorize the existing methods into three classes: the *list-based approach*, the *layer-based approach*, and the *view-based approach*. We briefly review each of these approaches in this section.

2.1. Layer-based approach

The layer-based approach constructs a global index based on the combination of *all* attribute values of each tuple. Within the index, tuples are partitioned into multiple layers, where the *i*th layer contains the tuples that can potentially be the top-*i* answer. Therefore, the top-*k* answers can be computed by reading at most *k* layers. ONION [4], PL-index [8], and AppRI [9] are well-known methods of this approach.

ONION [4] builds the index by making layers with the vertices (or the *extreme points* [10]) of the *convex hulls* [11] over the set of tuples represented as point objects in the multi-dimensional space. That is, it makes the first layer with the convex hull vertices over the entire set of tuples, and then, makes the second layer with the convex hull vertices over the set of remaining tuples, and