Efficient Dual-Resolution Layer Indexing for Top-k Queries

Jongwuk Lee, Hyunsouk Cho, Seung-won Hwang

Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
Pohang, Republic of Korea, 790-784
{julee, prory, swhwang}@postech.ac.kr

***Abstract*— Top-*k* queries have gained considerable attention as an effective means for narrowing down the overwhelming amount of data. This paper studies the problem of constructing an indexing structure that efficiently supports top-*k* queries for varying scoring functions and retrieval sizes. The existing work can be categorized into three classes: *list-*, *layer-*, and *view-based* approaches. This paper focuses on the layer-based approach, pre-materializing tuples into consecutive multiple layers. The layer-based index enables us to return top-*k* answers efficiently by restricting access to tuples in the k layers. However, we observe that the number of tuples accessed in each layer can be reduced further. For this purpose, we propose a *dual-resolution* layer structure. Specifically, we iteratively build *coarse-level* layers using *skylines*, and divide each coarse-level layer into *fine-level* sublayers using *convex skylines*. The dual-resolution layer is able to leverage not only the *dominance* relationship between coarse-level layers, named ∀*-dominance*, but also a relaxed dominance relationship between fine-level sublayers, named ∃*-dominance*. Our extensive evaluation results demonstrate that our proposed method significantly reduces the number of tuples accessed than the state-of-the-art methods.**

I. Introduction

With the exponential growth in database sizes, fighting the information flood is of critical importance in modern database systems. Toward this goal, *top-k queries* (or *ranked queries*) have gained considerable attention as an effective means for narrowing down the overwhelming amount of data. Top-k queries only return the best k tuples satisfying users' fuzzy information needs.

Specifically, a top-k query consists of a *retrieval size* k and a *scoring function* $\mathcal{F}$, which assigns a numerical score to each tuple in a target relation $\mathcal{R}$. A user specifies her/his *preference* by adjusting a *weight* w (or *importance*) of attributes, *e.g.*, the price is twice as important as the distance from the airport. (In this paper, we assume that the user preference is represented by a *linear combination function*, as widely used in the literature). The top-k query then returns the best k tuples as a focused and selected result set. To satisfy users with diverse preferences, efficient top-k query processing for varying weights is required.

To illustrate such a top-k query, we introduce a hotel-finding scenario, and formulate a top-k query based on an SQL-style expression, *i.e.*, ORDER BY and STOP AFTER k [1], [2].

Example 1 (Top-k queries) Consider a hotel retrieval system using a database *Hotel*(*hno*, *name*, *price*, *distance*, *city*).

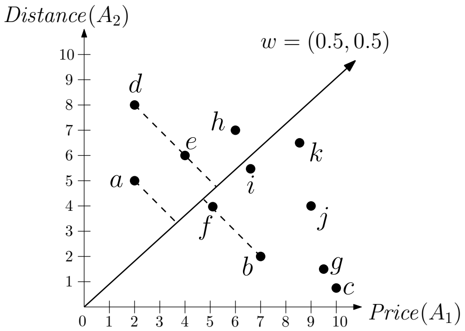


Fig. 1. Toy dataset in two-dimensional space

Suppose that a user "Alice" wants to find the best hotels in 'Washington DC' using two criteria, *i.e.*, low *price* and short *distance* to the airport. To search for the *top-5* hotels satisfying her preference with a weight (0.5, 0.5), she may formulate a top-k query as:

```
SELECT * FROM Hotel
WHERE city = 'Washington DC'
ORDER BY 0.5 * price + 0.5 * distance   ASC
STOP AFTER 5;                                (Q1)
```

Fig. 1 depicts a toy dataset stored in the Hotel database. In a geometric view, when representing a user-specific preference by a weight vector w (an arrow), we can identify the top-*5* tuples $\{a, b, f, d, e\}$, the scores of which can be computed by sweeping the perpendicular line (dashed), *e.g.*, $\mathcal{F}(a) = 3.5$.

Meanwhile, another user "Betty" may consider *price* to be more important, and issue a top-k query with a weight (0.75, 0.25) unlike Alice's query with (0.5, 0.5). According to user preferences, the top-k results may be different. Therefore, the database system should be able to support multiple top-k queries for varying weights efficiently. ■

For efficient top-k query processing, existing algorithms have exploited *pre-materialized* structures to avoid unnecessary data access in $\mathcal{R}$. Specifically, such structures can be categorized into the following three types:

- **Layer-based approach:** This approach [3], [4], [5], [6], [7] builds consecutive layers as a global index structure on all attributes $\mathcal{A}$. Each tuple in $\mathcal{R}$ is associated with some layer by one-to-one mapping. This approach then

1084-4627/12 $26.00 © 2012 IEEE
DOI 10.1109/ICDE.2012.73

traverses one layer at a time by leveraging the relationships between adjacent layers.

- **List-based approach:** This approach [8], [9], [10], [11], [12], [13], [14], [15], [16] exploits a set of *sorted* lists for each attribute. Given a scoring function $\mathcal{F}$, it accesses the sorted lists in a round-robin manner and aggregates the scores identified from each list until the best k tuples are found.
- **View-based approach:** This approach [17], [18], [19] makes use of pre-computed top-k queries as *views* in database systems. Given a new top-k query, it selects the most similar materialized top-k query, and reuses its computation.

This paper focuses on the layer-based approach, prematerializing tuples into multiple layers. Top-k answers can be found by accessing the first k layers regardless of the scoring functions. While existing approaches achieve efficiency by restricting access to the first few layers, they still give *complete access* to all tuples in a layer. This approach can be further optimized by giving *selective access*. Specifically, we consider the following two cases carefully:

1) **Complete access:** The first layer is a candidate set of *top-1* answers. This set can be identified as either a *convex hull* [20], [21], [22] or a *skyline* [23], [24], [25], [26], [27], [28].
2) **Selective access:** The *dominant graph* (DG) [5] defines the *dominance* relationship between tuples in adjacent layers. Using this relationship, DG selectively accesses some of the tuples in the k layers.

This paper aims to enable selective access to the first layer and to optimize the selective access further for the remaining layers. For this purpose, we propose a *dual-resolution layer*, which complements the advantages of the existing layer-based approaches. Specifically, the dual-resolution layer is comprised of *coarse-level* layers and *fine-level* layers. Tuples between two adjacent coarse-level layers have dominance relationships, named ∀-*dominance*. Each coarse-level layer is split into multiple *fine-level* sublayers. The tuples between adjacent fine-level layers have a new relaxed dominance relationship, named ∃-*dominance*, to restrict access to a few selected tuples in a coarse-level layer. In addition, to enable selective access to the first layer, we populate a virtual layer, *i.e.*, the *zero layer*, with a relationship to the tuples in the first layer.

We also develop efficient top-k query processing over the dual-resolution layer. Given a scoring function, we first access a *convex skyline* [4], [6], being a part of convex hull, as the fine-level layer in the first coarse-level layer. We then selectively access tuples by checking the ∀- and ∃-dominance relationships between the coarse- and fine-level layers, respectively.

To summarize, we believe that this paper makes the following contributions:

- We design a dual-resolution layer that consists of a coarse-level layer representing the skyline and a fine-level layer representing the convex skyline.
- We define a relaxed dominance relationship between the fine-level layers.
- We develop efficient top-k query processing by traversing the dual-resolution layer through the relationships between tuples.
- We devise an optimization technique for virtually populating the zero layer to achieve selective access in the first layer.
- We evaluate the efficiency of our proposed layering method by comparing it with the state-of-the-art methods for extensive synthetic datasets.

TABLE I

LIST OF NOTATIONS

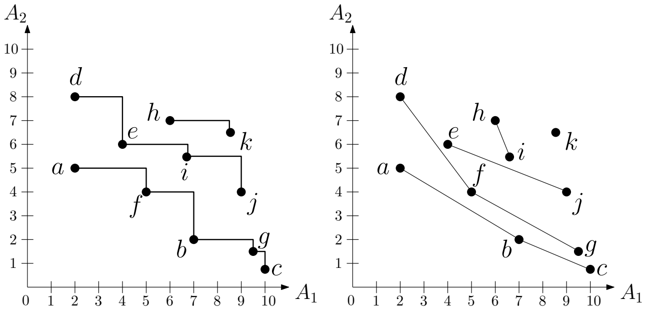
Notation	Definition
$\mathcal{R}$	A target relation for top-k queries, $\{t^1, \ldots, t^n\}$
n	Cardinality
$\mathcal{A}$	A attribute set, $\{A_1, \ldots, A_d\}$
d	The number of attributes
$dom(A_i)$	A domain of A_i
t, t'	A tuple in $\mathcal{R}$, $t = (t_1, \ldots, t_d)$
$\mathcal{F}$	A user-specific scoring function
w	A weight vector, $w = (w_1, \ldots, w_d)$
$\prec$	Dominance (or ∀-dominance)
SKY($\mathcal{R}$)	A skyline in $\mathcal{R}$
CSKY($\mathcal{R}$)	A convex skyline in $\mathcal{R}$
$\mathcal{L}$	A layer set, $\{L^1, \ldots, L^l\}$
l	The number of layers in $\mathcal{L}$
L^{ij}	j^{th} fine-level layer in L^i
$\mathcal{EDS}$	A ∃-dominance set, $\{t^1, \ldots, t^d\}$
$\precsim$	∃-dominance
$\mathcal{FA}$	A facet set $\mathcal{FA} = \{FA^1, \ldots, FA^r\}$

The remainder of this paper is organized as follows. Section II presents the preliminaries on top-k queries. Section III designs the dual-resolution layer for further optimizing existing layer-based structures, and Section IV develops efficient top-k query processing over the proposed dual-resolution layer. Section V optimizes top-k query processing by giving selective access to the first layer. Section VI reports our extensive evaluation results, and Section VII surveys related work. Finally, Section VIII concludes this paper.

II. PRELIMINARIES

This section first defines the basic notations to formalize the problem of top-k queries. Let $\mathcal{R} = (t^1, \ldots, t^n)$ be a target relation with d attributes $\mathcal{A} = (A_1, \ldots, A_d)$. The domain $dom(A_i)$ of each attribute is non-negative real values, *i.e.*, $dom(A_i) \rightarrow \mathbb{R}^+$. A tuple $t \in \mathcal{R}$ is represented by $(t_1, \ldots, t_d)$ such that $\forall i \in [1, d] : t_i \in dom(A_i)$. For simplicity, we assume that $dom(A_i)$ is normalized to [0,1]. Table I summarizes the notations used in this paper.

A top-k query consists of a scoring function $\mathcal{F}$ and a retrieval size k. We assume that the scoring function is a *linear combination function*, *i.e.*, $\mathcal{F}(t) = \sum_{i=1}^{d} w_i t_i$, where $w = (w_1, \ldots w_d)$ is a user-specific weight vector. Without loss of generality, the values of the weight vector are normalized to [0, 1] and $\sum_{i=1}^{d} w_i = 1$. In addition, we assume that the scoring function $\mathcal{F}$ preserves *monotonicity* [8], [11], *i.e.*, given two tuples t and t', if $t_i \leq t'_i$ for $A_i \in \mathcal{A}$, then



(a) Skyline layers

(b) Convex layers

Fig. 2. Skyline layers and convex layers using the toy dataset (Fig. 1)

$\mathcal{F}(t) \leq \mathcal{F}(t')$. Recall that these assumptions are commonly used in the literature [3], [4], [5], [6], [29].

Depending on user preferences, top-k queries may return k tuples with the highest or the lowest scores. Without loss of generality, in the rest of this paper, we assume that k tuples with the lowest scores are returned as top-k answers. The highest scoring tuples can be found by changing the sign of tuples.

We formally define the top-k query problem. We assume ties are broken arbitrarily by the unique identifiers to tuples.

DEFINITION 1 (TOP-k QUERY PROBLEM) Given a scoring function $\mathcal{F}$ and a retrieval size k, a *top-k query* returns an ordered set of k tuples $\{t^1, \ldots, t^k\}$, *i.e.*, $\mathcal{F}(t^1) \leq \ldots \leq \mathcal{F}(t^k) \leq \mathcal{F}(t^l)$ such that $t^l \in \mathcal{R} - \{t^i, \ldots t^k\}$.

A key challenge in computing top-k queries is to reduce the number of accessed tuples as much as possible. To address this problem, the existing work has exploited *pre-computed* indexing structures to avoid unnecessary data access. Specifically, these can be categorized into three classes: (1) *list-based*, (2) *layer-based*, and (3) *view-based* approaches. (Section VII will review these existing approaches in detail).

In this paper, we focus on a layer-based approach, where an indexing structure consists of consecutive multiple layers. Let $\mathcal{L} = \{L^1, \ldots L^l\}$ be a set of layers. Each tuple in $\mathcal{R}$ belongs to a layer with one-to-one mapping such that $L^i \cap L^j = \emptyset$ $(i \neq j)$ and $\bigcup_{i=1}^{l} L^i = \mathcal{R}$.

We next define two important notions used in the existing layer-based indexing structure.

First, a *skyline* [23], [25] is employed for the top-*1* candidates without scoring functions for constructing layers. The skyline is based on a *dominance* notion. It is said that a tuple t *dominates* another tuple t' if t is no worse than t' in all attributes. We formally define *dominance* and *skyline* below.

DEFINITION 2 (DOMINANCE) Given t and t', t *dominates* t' on $\mathcal{A}$, denoted as $t \prec t'$, if and only if $\forall\ i \in [1, d] : t_i \leq t'_i$ and $\exists\ j \in [1, d] : t_j < t'_j$.

DEFINITION 3 (SKYLINE) A tuple t is a *skyline tuple* if and only if any other tuple t' $(\neq t)$ does not dominate t on $\mathcal{A}$. The *skyline* is a set of skyline tuples such that $\text{SKY}(\mathcal{R}) = \{t \in \mathcal{R} | \nexists t' \in \mathcal{R} : t' \prec t\}$.

Skyline layers [5] can be built by identifying skylines in a sequential manner. Specifically, the first layer L^1 is the skyline of $\mathcal{R}$, and the i^{th} layer L^i $(i > 1)$ is the skyline of $\mathcal{R} - \bigcup_{l=1}^{i-1} L^l$. For example, Fig. 2(a) depicts the skyline layers using a toy dataset (Fig. 1). These consist of three layers such that $L^1 = \{a, b, c, f, g\}$, $L^2 = \{d, e, i, j\}$, and $L^3 = \{h, k\}$.

Alternatively, a *convex skyline* [30] can be used. A convex skyline is a subset of a *convex hull* [21], which is the smallest convex polyhedron including all the tuples in $\mathcal{R}$. Formally, we define the convex skyline as follows:

DEFINITION 4 (CONVEX SKYLINE) A tuple t is a *convex skyline tuple* if and only if t has the minimum score for any linear combination functions. The convex skyline is a set of convex skyline tuples such that $\text{CSKY}(\mathcal{R}) = \{t \in \mathcal{R} | t = \text{argmin}_{t^* \in R} \sum_{i=1}^{d} w_i t^*_i,\ w_i \in \mathbb{R}^+,\ \sum_{i=1}^{d} w_i = 1\}$.

As in the case of the skyline layers, we can build the *convex layers* [3], [4], [31] by identifying convex skylines sequentially. The first layer is the convex skyline of $\mathcal{R}$, and the i^{th} $(i > 1)$ layer is the convex skyline of $\mathcal{R} - \bigcup_{l=1}^{i-1} L^l$. Fig. 2(b) depicts the convex skyline layers using the toy dataset (Fig. 1). These consist of five layers such that $L^1 = \{a, b, c\}$, $L^2 = \{d, f, g\}$, $L^3 = \{e, j\}$, $L^4 = \{h, i\}$, and $L^5 = \{k\}$. The next section describes a new layer-based indexing structure that takes advantage of both skyline and convex layers.

III. PROPOSED LAYER-BASED INDEXING

In this section, we first observe existing layer-based indexing approaches and discuss their limitations (Section III-A). To address the problems, we then design a *dual-resolution layer*, which tightens the relationships between tuples in a fine granularity (Section III-B).

A. Limitations of Existing Layer-Based Approaches

Given a retrieval size k, layer-based approaches basically guarantee that tuples in the k layers contain top-k answers. We can thus identify top-k answers by reducing the search space by accessing all tuples in the k layers.

Existing approaches focus on optimizing the number of tuples accessed in the k layers. Depending on the method of layer construction, these are categorized into three classes.

Skyline-layer approach: The *dominant graph* (DG) [5] is constructed by computing skylines in a sequential manner. Specifically, the first layer L^1 is identified by a skyline in $\mathcal{R}$. The next layer L^i $(i > 1)$ is defined iteratively by finding the skyline in the remaining tuples $\mathcal{R} - \bigcup_{l=1}^{i-1} L^l$. This iteration continues until no tuples remain. In addition, the dominance relationship holds between tuples in adjacent layers, on the basis of which we can avoid the unnecessary access among dominated tuples, *i.e.*, *selective access*. Because tuples in L^1 are not dominated by any other tuples, DG requires *complete access* to identify a top-*1* tuple. In particular, because the number of skyline tuples is always greater than or equal to that of convex skyline tuples, the access cost for the first layer is greater than or equal to that of convex-layer approaches.

TABLE II

SELECTIVITY OF LAYER-BASED APPROACHES

(0: COMPLETE ACCESS, +: RELATIVE SELECTIVITY)

Approach	1st Layer	other layers
Convex-layer approach	0	0
Skyline-layer approach	0	++
Hybrid-layer approach	+	+
Our approach	+++	+++

Convex-layer approach: Onion [3] and AppRI [4] are built in the same manner as DG, except that they identify the convex skyline iteratively. To optimize Onion, AppRI reduces the number of tuples for each layer. Unlike skyline layers, however, both Onion and AppRI require *complete access* to all layers.

Hybrid-layer approach: The *hybrid-layer* index (HL) [6] is a hybrid of the layer- and list-based approaches. First, HL identifies layers by computing convex skylines iteratively (as in Onion), but the tuples in the layer are stored as sorted lists in increasing order of d attribute values. Leveraging the sorted lists, HL can reduce the number of tuples accessed from Onion by selectively accessing tuples for each layer on the basis of the list-based approach. Although HL gives *selective access* to all layers, the selectivity is limited.

Our goal is to complement the existing approaches by enabling selective access to all layers and providing a *systematic* optimization technique to improve the selectivity significantly in both cases. Table II illustrates the relative selectivity of the above three approaches and our approach.

B. Dual-Resolution Layer

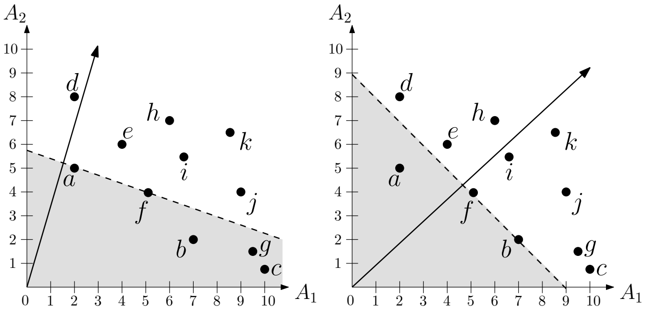
We extend the existing approaches by constructing layers of fine granularity. Specifically, each layer consists of a two-level structure, called a *dual-resolution* layer. We first build *coarse-level* layers using the skylines, and then divide each coarse-level layer into multiple *fine-level* sublayers using the convex skylines.

More specifically, the dual-resolution layer is constructed in two phases:

1) **Coarse-level layer construction:** We build skyline layers sequentially. The first layer L^1 is the skyline $\text{SKY}(\mathcal{R})$ and the other layers L^i $(i > 1)$ are the skylines, *i.e.*, $\text{SKY}(\mathcal{R} - \bigcup_{l=1}^{i-1} L^l)$.
2) **Fine-level layer construction:** We split the coarse-level layer into consecutive convex layers at a much finer granularity. Let L^{ij} be the j^{th} fine-level layer in the L^i coarse-level layer. For each coarse-level layer L^i, the first fine-level layer L^{i1} is the convex skyline $\text{CSKY}(L^i)$, and the other fine-level layers L^{ij} are the convex skylines, *i.e.*, $\text{CSKY}(L^i - \bigcup_{l=1}^{j-1} L^{il})$.

We next define the relationships between tuples in adjacent layers over the dual-resolution layer.

First, we can exploit the dominance relationship for coarse-level layers as in DG [5]. When sequentially accessing tuples from the ith to $(i+1)$th coarse-level layers, we can avoid the access of a tuple $t' \in L^{i+1}$ $(i > 1)$ unless every tuple $t \in L^i$ dominating t' is accessed. We formally state this dominance property as used in DG [5].



(a) $\mathcal{F}(a) < \mathcal{F}(f)$ (b) $\mathcal{F}(b) < \mathcal{F}(f)$

Fig. 3. Geometric view of tuple-wise relationships in the toy dataset (Fig. 1)

LEMMA 1 (DOMINANCE BETWEEN COARSE LAYERS [5]) For any scoring function $\mathcal{F}$, each tuple $t \in L^i$ always has a smaller score than a tuple $t' \in L^{i+1}$ such that $t \prec t'$, *i.e.*, $\forall t \in L^i, t \prec t' : \mathcal{F}(t) < \mathcal{F}(t')$.

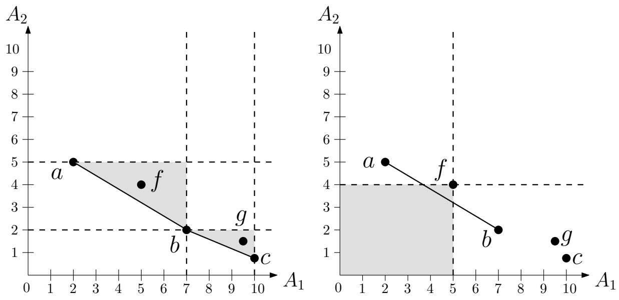
Proof: See reference [5]. ■

Second, we define a relaxed dominance relationship between adjacent fine-level layers to achieve selective access. Specifically, we split the coarse-level layer into multiple fine-level layers. Because no dominance relationship holds between two fine-level layers, we extend the dominance relationship for the coarse-level layer into a relaxed dominance relationship for the fine-level layer. To highlight this difference, we alternatively call the classical definition of dominance $\forall$-*dominance*, and define a new dominance notion as $\exists$-*dominance*. (We replace the $\forall$ quantifier with $\exists$ in the definition).

Our next question is how to define $\exists$-dominance. From a geometric viewpoint, if a tuple $t \in L^{ij}$ has a smaller score than another tuple $t' \in L^{i(j+1)}$, the half-space of the perpendicular hyperplane lying on t' contains t. For instance, tuple f has a greater score than tuples a and b in each case (Fig. 3). By shifting the weight vectors, we can identify the tuple-wise relationship by comparing the scores of tuples. In this case, no matter how the weight vector is adjusted, a or b are always contained to the half-space (gray colors in Fig. 3) of the perpendicular line lying on f. This implies that one of the tuple set $\{a, b\}$ always has a smaller score than f.

On the basis of this observation, we introduce an $\exists$-*dominance set* ($\mathcal{EDS}$). We consider a *hyperplane* $\mathcal{H}$ spanning a set of d tuples in d-dimensional space. Given a hyperplane $\mathcal{H}$ and a tuple t', it is said that the tuple set on hyperplane $\mathcal{H}$ is an $\mathcal{EDS}$ of a tuple t' if $\mathcal{H}$ is below tuple t' from the origin. Further, it is said that each tuple $t \in \mathcal{EDS}$ $\exists$-*dominates* t'. For example, a tuple set $\{a, b\}$ is represented by a line in two-dimensional space. As the line through $\{a, b\}$ is below tuple f, $\{a, b\}$ is an $\mathcal{EDS}$ of f, and a and b $\exists$-dominate f.

We formally define the $\exists$-*dominance set* and $\exists$-*dominance*. In addition, we state the $\exists$-dominance relationship between tuples in adjacent fine-level layers.



(a) Facets $\{a, b\}$ and $\{b, c\}$ (b) $\exists$-dominance for f

Fig. 4. A tuple set of L^1 in the toy dataset (Fig. 1)

DEFINITION 5 ($\exists$-DOMINANCE SET ($\mathcal{EDS}$)) A set of d tuples is an $\exists$-*dominance set* ($\mathcal{EDS}$), *i.e.*, $\{t^1, \ldots, t^d\}$, of a tuple t' on $\mathcal{A}$ if there exists a virtual tuple t^V that dominates t' such that t^V lies on the hyperplane $\mathcal{H}$ spanning the $\mathcal{EDS}$.

DEFINITION 6 ($\exists$-DOMINANCE) Given two tuples t and t', t $\exists$-*dominates* t' on $\mathcal{A}$, denoted as $t \precsim t'$, if and only if t is in the $\mathcal{EDS}$ of t'.

LEMMA 2 ($\exists$-DOMINANCE BETWEEN FINE LAYERS) For any scoring function $\mathcal{F}$, there exists a tuple $t \in L^{ij}$ that has a smaller score than a tuple $t' \in L^{i(j+1)}$ such that $t \precsim t'$, *i.e.*, $\exists t \in L^{ij}, t \precsim t' : \mathcal{F}(t) < \mathcal{F}(t')$.

Proof: According to Definition 4, a tuple $t \in L^{ij}$ always exists such that $\mathcal{F}(t) < \mathcal{F}(t')$ and $t \precsim t'$. ■

We then discuss how to construct $\exists$-dominance sets for the fine-level layer. Basically, the $\exists$-dominance sets in L^{ij} need to cover all tuples in $L^{i(j+1)}$, *i.e.*, every tuple in $L^{i(j+1)}$ forms the $\exists$-dominance relationship for any tuple in L^{ij}. Otherwise, we might fail to identify correct top-k answers, *i.e.*, some tuples in $L^{i(j+1)}$ cannot be accessed from L^{ij}.

One naive way of making $\exists$-dominance sets is to enumerate all possible tuple sets $\binom{|L^{ij}|}{d}$, where $|L^{ij}|$ denotes the cardinality of L^{ij}. Because the number of dominance sets increases exponentially with the number of attributes, this incurs prohibitive computation costs and thus defeats our purpose of performing selective access to tuples in $L^{i(j+1)}$.

To resolve this problem, we employ a set of *facets* representing a convex polyhedron [21]. The usual way to represent the convex polyhedron is to intersect the half-spaces of the hyperplanes. Let $\mathcal{FA} = \{FA^1, \ldots, FA^r\}$ be a set of *facets*, where each facet consists of a set of d tuples on $\mathcal{A}$. Because of this property, the facets can form *minimal* $\exists$-dominance sets to cover all tuples, *i.e.*, the facets encompass all tuples in the convex polyhedron. In particular, we consider the facet as the *hyperplane segment* to form $\exists$-dominance relationships. We thus check whether each facet can be an $\exists$-dominance set of another tuple in $L^{i(j+1)}$.

We now illustrate the relationships between adjacent fine-level layers using a running example.

EXAMPLE 2 (RELATIONSHIP FOR FINE-LEVEL LAYERS) Consider the first coarse-level layer in the toy dataset (Fig. 4). The coarse-level layer is split into two fine-level layers $L^{11} = \{a, b, c\}$ and $L^{12} = \{f, g\}$. Further, L^{11} consists of two facets $\{a, b\}$ and $\{b, c\}$ (solid lines in Fig. 4a). Because the facets represent the line segments, we only consider parts of the regions (gray triangles in Fig. 4a). For a tuple f, we thus check whether $\{a, b\}$ is an $\exists$-dominance set of f. The line segment constituting $\{a, b\}$ goes through the dominating region of f (gray rectangle in Fig. 4b), implying that there exists a virtual point t^V on the segment such that t^V dominates f. $\{a, b\}$ can thus be an $\exists$-dominance set of f, and a and b $\exists$-dominate f. ■

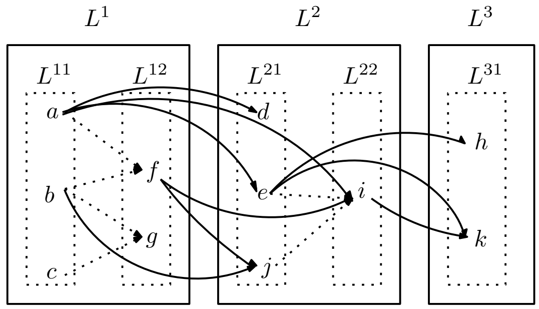


Fig. 5. Dual-resolution layer construction using the toy dataset (Fig. 1)

We lastly explain the overall procedure for constructing the dual-resolution layer (Algorithm 1). First, we compute the skylines for the coarse-level layers (line 6). For each coarse-level layer, we then compute convex skylines as the fine-level layers, and form the $\exists$-dominance relationships between fine-level layers (lines 7–16). After updating the fine-level layers, we also update the $\forall$-dominance relationships between coarse-level layers (lines 17–19). This iteration continues until each tuple in $\mathcal{R}$ is associated with an arbitrary layer.

We describe an example of constructing the dual-resolution layer.

EXAMPLE 3 (DUAL-RESOLUTION LAYER CONSTRUCTION) Fig. 5 depicts a dual-resolution layer using the toy dataset (Fig. 1). Conceptually, the entire dataset is partitioned into three coarse-level layers (solid boxes), *i.e.*, $L^1 = \{a, b, c, f, g\}$, $L^2 = \{d, e, i, j\}$, and $L^3 = \{h, k\}$. We then update tuples with $\forall$-dominance relationships (solid arrows), *e.g.*, a $\forall$-dominates $\{d, e, i\}$. Similarly, each coarse-level layer is divided into multiple fine-level layers (dotted boxes) such that $L^1 = \{\{a, b, c\}, \{f, g\}\}$, $L^2 = \{\{d, e, j\}, \{i\}\}$, and $L^3 = \{\{h, k\}\}$. In addition, the $\exists$-dominance relationships (dotted arrows) between adjacent fine-level layers are updated, *e.g.*, b and c $\exists$-dominate g. ■

IV. TOP-k QUERY PROCESSING

This section presents top-k query processing over the dual-resolution layer. Specifically, we first derive a *filtering* technique based on the $\forall$- and $\exists$-dominance relationships between adjacent coarse- and fine-level layers, respectively. We then describe the procedure of top-k query processing, building on the filtering technique over the dual-resolution layer. Finally,

Algorithm 1 BuildDualLayer($\mathcal{R}$)

```
Input: $\mathcal{R}$: a target relation on $\mathcal{A}$
Output: $\mathcal{L}$: a dual layer
$\mathcal{L} \leftarrow \{\}$. // Initialize a dual-resolution layer.
$i \leftarrow 1$. // Initialize the identifer of the coarse-level layer.
while $\mathcal{R} \neq \{\}$ do
    $L^i \leftarrow \{\}$. // Initialize the ith coarse-level layer.
    $j \leftarrow 1$. // Initialize the identifier of the fine-level layer.
    $\mathcal{S} \leftarrow \mathrm{SKY}(\mathcal{R})$. // Compute a skyline for $\mathcal{R}$.
    while $\mathcal{S} \neq \{\}$ do
        // Compute the jth fine-level layer in $L^i$.
        $L^{ij} \leftarrow \mathrm{CSKY}(\mathcal{S})$. // Compute a convex skyline for $\mathcal{S}$.
        if $j > 1$ then
            Update the $\exists$-dominance between $L^{i(j-1)}$ and $L^{ij}$.
        end if
        $L^i \leftarrow L^i \cup L^{ij}$. // Insert $L^{ij}$ into $L^i$.
        $\mathcal{S} \leftarrow \mathcal{S} - L^{ij}$. // Update current skyline $\mathcal{S}$.
        $j \leftarrow j + 1$. // Update the identifier of the fine-level layer.
    end while
    if $i > 1$ then
        Update the $\forall$-dominance between $L^{i-1}$ and $L^i$.
    end if
    $\mathcal{L} \leftarrow \mathcal{L} \cup L^i$. // Insert $L^i$ into $\mathcal{L}$.
    $\mathcal{R} \leftarrow \mathcal{R} - L^l$. // Update target relation $\mathcal{R}$.
    $i \leftarrow i + 1$. // Update the identifier of the coarse-level layer.
end while
return $\mathcal{L}$
```

we analyze the efficiency of the proposed top-k query processing.

Top-k query processing using layer-based structures can be viewed as a *graph traversal problem* as discussed for the existing algorithms [5], [6]. Specifically, we form tuples as *nodes*, and $\forall$- and $\exists$-dominance relationships as *edges*.

The key contribution here is to define an efficient filtering technique that avoids unnecessary access. For this purpose, we exploit the $\forall$- and $\exists$-dominance relationships for the coarse- and fine-level layers, respectively.

First, $\forall$-dominance can be represented as *solid edges* (Fig. 5) between adjacent coarse-level layers L^i and L^{i+1} in a *bipartite graph*. The $\forall$-dominance relationship preserves the *monotonicity* in the coarse-level layer (Lemma 1). Specifically, a directed solid edge e connects $t \in L^i$ and $t' \in L^{i+1}$ if t $\forall$-dominates t', *i.e.*, if $t \prec t'$, then $e : t \rightarrow t'$. This can be employed to identify top-k answer candidates (or prevent access to non-candidates). We formally state this property used in DG [5] as follows.

THEOREM 1 ($\forall$-DOMINANCE-ORDERED COARSE LAYERS) Given adjacent coarse-level layers L^i and L^{i+1}, a tuple $t' \in L^{i+1}$ can be in the top-k answers such that $t \prec t'$ if every tuple $t \in L^i$ is in the final top-$(k-1)$ answers.

Proof: See reference [5]. ■

Second, $\exists$-dominance can be represented as *dotted edges* (Fig. 5) between adjacent fine-level layers L^{ij} and $L^{i(j+1)}$, preserving the *monotonicity* in the fine-level layer (Lemma 2). Similarly, top-k answer candidates (and non-candidates) can be determined. We formally state this property for $\exists$-dominance.

THEOREM 2 ($\exists$-DOMINANCE-ORDERED FINE LAYERS) Given adjacent fine-level layers L^{ij} and $L^{i(j+1)}$, a tuple $t' \in L^{i(j+1)}$ can be in the top-k answers such that $t \precsim t'$ if any tuple $t \in L^{ij}$ is in the final top-$(k-1)$ answers.

Proof: According to Lemma 2, a tuple $t \in L^{ij}$ exists such that $\mathcal{F}(t) < \mathcal{F}(t')$ and $t \precsim t'$. Therefore, if t' is in the top-k answers, t' has to be in the top-$(k-1)$ answers. ■

Using these properties, we can maintain the *status* of a tuple in each layer. Depending on its status, we can easily identify whether or not a tuple needs to be accessed. Formally, we define the status of a tuple as follows.

DEFINITION 7 ($\forall$-DOMINANCE-FREE) A tuple $t' \in L^{i+1}$ is $\forall$-*dominance-free*, if and only if (1) t' has no connected edge from $t \in L^i$ or (2) every tuple $t \in L^i$ connected to t' is in the top-$(k-1)$ answers.

DEFINITION 8 ($\exists$-DOMINANCE-FREE) A tuple $t' \in L^{i(j+1)}$ is $\exists$-*dominance-free*, if and only if (1) t' has no connected edge from L^{ij} or (2) any tuple $t \in L^{ij}$ connected to t' is in the top-$(k-1)$ answers.

We now illustrate the statuses of tuples using our running example:

EXAMPLE 4 (STATUSES OF TUPLES) We continue the example for the dual-resolution layer (Fig. 5). We can identify that the tuples $\{a, b, c, f, g\}$ in the first coarse-level layer are $\forall$-dominance-free. Further, the tuples $\{a, b, c, d, e, j, h, k\}$ in the first fine-level layer of each coarse-level layer are $\exists$-dominance-free. In addition, when we access tuples sequentially in the dual-resolution layer, the statuses of other tuples can be changed. If a and f in L^1 are in top-$(k-1)$ answers, i is changed to be $\forall$-dominance-free. Similarly, if a or b in L^{11} are added to the top-$(k-1)$ answers, f is changed to be $\exists$-dominance-free. ■

On the basis of Theorems 1 and 2, when traversing tuples through the dual-resolution layer, we only compute tuples that are both $\forall$-dominance-free and $\exists$-dominance-free by $\mathcal{F}$, and avoid *unnecessary* computation of non-candidates. Formally, we state this filtering condition for tuples as follows.

THEOREM 3 (FILTERING CONDITION) A tuple t' only needs to be accessed if t' is both $\forall$-dominance-free and $\exists$-dominance-free.

Proof: We consider two cases for a tuple t'. First, when t' is not $\forall$-dominance-free, there exists a tuple t that $\forall$-dominates t'. According to Theorem 1, we can safely skip access to t' unless t is in the top-$(k-1)$ answers. Second, when t is not $\exists$-dominance-free, there exists a tuple t that $\exists$-dominates t'. According to Theorem 2, we can safely skip access to t' unless t is in the top-$(k-1)$ answers. ■

We explain the overall procedure of computing top-k processing over the dual-resolution layer (Algorithm 2). Let $\mathcal{Q}$ be a priority queue in which tuples are placed in order of the increasing scores by $\mathcal{F}$. First, we insert all tuples in L^{11} into a priority queue $\mathcal{Q}$, because they are already $\forall$-dominance-free and $\exists$-dominance-free (line 3). We then pop a tuple t with the smallest score from $\mathcal{Q}$ and insert t into a final top-k answer set

Algorithm 2 ComputeTopKProcessing($\mathcal{L}$, $\mathcal{F}$, k)

```
Input: $\mathcal{L}$: a dual layer, $\mathcal{F}$: a scoring function, $k$: a retrieval size
Output: $\mathcal{K}$: a set of top-$k$ tuples
 1: $\mathcal{K} \leftarrow \{\}$. // Initialize a top-k answer set.
 2: $\mathcal{Q} \leftarrow \{\}$. // Initialize a priority queue for top-k candidates.
 3: All tuples in $L^{11}$ are computed by $\mathcal{F}$, and are inserted into $\mathcal{Q}$.
 4: while $\mathcal{K}.size() \geq k$ do
 5:     $t \leftarrow \mathcal{Q}.pop()$. // Pop $t$ with the smallest score from $\mathcal{Q}$.
 6:     $\mathcal{K} \leftarrow \mathcal{K} \cup \{t\}$. // Insert $t$ into a top-k answer set $\mathcal{K}$.
 7:     for all $t'$ connected from $t$ in the coarse-level layer do
 8:         if $t'$ is $\forall$-dominance-free and $\exists$–dominance-free then
 9:             $t'$ is computed by $\mathcal{F}$ and is inserted into $\mathcal{Q}$.
10:         end if
11:     end for
12:     for all $t'$ connected from $t$ in the fine-level layer do
13:         if $t'$ is $\forall$-dominance-free and $\exists$-dominance-free then
14:             $t'$ is computed by $\mathcal{F}$ and is inserted into $\mathcal{Q}$.
15:         end if
16:     end for
17: end while
18: return $\mathcal{K}$
```

(lines 5–6). We also check the filtering condition (Theorem 3) for other tuples connected to t for coarse- and fine-level layers (lines 7–16). This process is iterated until the top-k answers are identified.

We describe an example of top-k query processing over the dual-resolution layer.

Example 5 Suppose that the retrieval size k is 3, and the weight vector w is (0.5, 0.5). Table III describes the actions of our proposed top-k query processing. After initializing a priority queue $\mathcal{Q}$ and an answer set $\mathcal{K}$, we access tuples $\{a, b, c\}$ in L^{11}, and enqueue them into $\mathcal{Q}$. Because tuple a is the top-*1* tuple, we pop a from $\mathcal{Q}$. We then update the status of tuples $\{d, e, f, i\}$ connected to a. Because tuples $\{d, e, f\}$ are changed to be $\forall$-dominance-free and $\exists$-dominance-free, we access those tuples, and insert them into $\mathcal{Q}$. As the top-*2* tuple, we pop b from $\mathcal{Q}$, and also update the status of tuples $\{g, j\}$ connected to b. Because g becomes $\forall$-dominance-free and $\exists$-dominance-free, g is inserted into $\mathcal{Q}$. Lastly, we pop f from $\mathcal{Q}$ as the top-*3* tuple. As a result, the top-3 answers are $\{a, b, f\}$. ■

We then prove the correctness of the proposed top-k processing.

Theorem 4 (Correctness) The proposed top-k processing returns correct top-k answers.

Proof: We prove this by induction.

1) Base case ($k = 1$): Our top-k query processing returns the tuple with the smallest score in L^{11}. Because one of the tuples in L^{11} always has a smaller score than tuples in other layers (Theorems 1 and 2), we can identify a correct top-1 answer.
2) Hypothesis ($k = i$): By inductive hypothesis, our top-k query processing returns correct top-k answers.
3) Induction ($k = i + 1$): When identifying the $(i + 1)$th answer, we consider (1) top-$(i + 1)$ candidates in $\mathcal{Q}$ or (2) tuples connected to *top-i* candidates whose statuses are both $\forall$-dominance-free and $\exists$-dominance-free. One of the tuples satisfying (1) or (2) always has a smaller score than other tuples (Theorems 1 and 2). Therefore, our top-k query processing returns correct top-$(i + 1)$ answers.

TABLE III
Top-k query processing over the dual-resolution layer (Fig. 5)

Step	Action	$\mathcal{Q}$	$\mathcal{K}$
1	Initialize $\mathcal{Q}$ and $\mathcal{K}$.	$\{\}$	$\{\}$
2	Access tuples in L^{11}.	$\{a, b, c\}$	$\{\}$
3	Pop a from $\mathcal{Q}$.	$\{b, c\}$	$\{a\}$
4	Update $\{d, e, f, i\}$ connected to a.	$\{b, f, d, e, c\}$	$\{a\}$
5	Pop b from $\mathcal{Q}$	$\{f, d, e, c\}$	$\{a, b\}$
6	Update $\{g, j\}$ connected to b.	$\{f, d, e, c, g\}$	$\{a, b\}$
7	Pop f from $\mathcal{Q}$	$\{d, e, c, g\}$	$\{a, b, f\}$

By induction, our top-k query processing always returns correct top-k answers. ■

Finally, we discuss the efficiency of the proposed top-k query processing by proving analytically that our algorithm incurs less *cost* than DG. (DG is known as the most efficient existing layer-based index). Specifically, we can view DG as a layer-based index that employs only coarse-level layers from dual-resolution layer indexing, and cannot take advantage of $\exists$-dominance relationships. We formally define the cost used to evaluate our top-k query processing, and analyze the cost of our top-k query processing. Recall that the cost model is also used in DG.

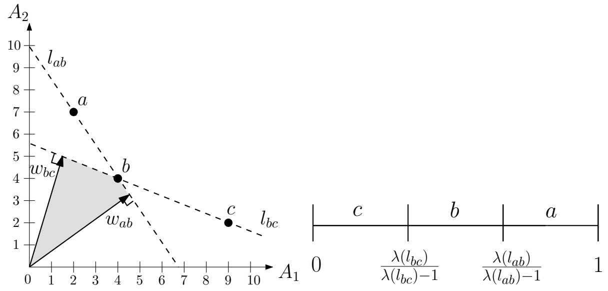
Definition 9 (Cost) The cost is the number of tuples that are both accessed and computed by $\mathcal{F}$ during top-k query processing.

Theorem 5 (Cost analysis) Our top-k query processing over the dual-resolution layer always incurs less or equal cost than DG.

Proof: We prove this by contradiction. Assume that DG can skip any tuples accessed by our top-k query processing. While DG evaluates tuples whose statuses are $\forall$-dominance-free, our top-k processing only evaluates tuples whose statuses are both $\forall$- and $\exists$-dominance-free. We thus compare the tuples evaluated by DG and ours: *tuples pruned by* $\forall$*-dominance-free* $\subseteq$ *tuples pruned by* $\forall$*-dominance-free* $\wedge$ $\exists$*-dominance-free*. If DG does not evaluate a tuple that is $\forall$- and $\exists$-dominance-free, then it does not guarantee to return correct top-k answers, which is a contradiction. Thus, the cost of DG has to include all the tuples evaluated by our top-k query processing. ■

V. Optimized Top-k Query Processing

This section presents optimization techniques for identifying a top-*1* tuple in L^{11}. Because all tuples in L^{11} are essentially $\forall$- and $\exists$-dominance-free, we need to give complete access to L^{11} (line 3 in Algorithm 2). To resolve this problem, we form a *virtual* zero layer L^0 to guide selective access to L^{11}. Recall that the tuples in L^0 (or *pseudo-tuples*) are only introduced for filtering purposes in L^{11} and do not exist in $\mathcal{R}$.


A_2
10
9
8
7
6
5
4
3
2
1
0
l_{ab}
a
b
c
l_{bc}
w_{bc}
w_{ab}
A_1
c
b
a
0
$\frac{\lambda(l_{bc})}{\lambda(l_{bc})-1}$
$\frac{\lambda(l_{ab})}{\lambda(l_{ab})-1}$
1

(a) Range bound for b (b) Weight ranges for w_1

Fig. 6. Geometric view of weight ranges in two-dimensional space

Specifically, we first discuss the structure of L^0 in two-dimensional space, which enables us to identify a top-*1* tuple by accessing only one tuple in L^{11} (Section V-A). In higher-dimensional space ($d \geq 3$), we develop a scalable alternative structure (Section V-B).

A. Top-1 Tuple Selection in Two-Dimensional Space

Ideally, given a weight vector $w = (w_1, w_2)$, one candidate tuple t in L^{11} can be identified for selective access by using the relationships between L^{11} and the virtual zero layer L^0. Let $\mathcal{W}$ denote a set of all possible weight vectors w. Considering that $\mathcal{W}$ is an infinite set, we cannot enumerate every weight vector $w \in \mathcal{W}$ to t. In contrast, we can represent $\mathcal{W}$ as a finite set of weight ranges and associate a *weight range* with t.

In the two-dimensional case, the following conditions hold: $0 < w_1, w_2 < 1$ and $w_1 + w_2 = 1$. Using this functional condition between w_1 and w_2, we can represent $\mathcal{W}$ as the one-dimensional range of w_1, since the corresponding ranges of w_2 can be determined by $1 - w_1$. Specifically, we represent $\mathcal{W}$ as a finite set of disjoint ranges of w_1, *i.e.*, $\{W^1, \ldots, W^{|L^{11}|}\}$ such that $\bigcup_{i=1}^{|L^{11}|} W^i = [0:1]$ and $W^i \cap W^j = \emptyset$ $(i \neq j)$.

For such partitioning, we leverage a set of facets $\mathcal{FA}$ representing L^{11}. In two-dimensional space, a facet is represented by a *line* across adjacent tuples. For example, suppose that L^{11} consists of three tuples $\{a, b, c\}$, where $\mathcal{FA} = \{\{a, b\}, \{b, c\}\}$ (Fig. 6a). For the sake of representation, let l_{ab} denote a line across tuples $\{a, b\}$, and w_{ab} denote a weight vector perpendicular to l_{ab}.

Given a weight vector w_{ab}, tuples a and b have the same scores, *i.e.*, $\mathcal{F}(a) = \mathcal{F}(b)$. By adjusting the *slopes* of the weight vectors, we can identify the weight ranges for a specific tuple. In particular, observe that the weight ranges are bounded to two lines represented by adjacent facets including the tuple because of the inherent convex skyline [20], [21], [32].

Fig. 6(a) illustrates the possible weight ranges (gray color) for tuple b using l_{ab} and l_{bc}. Because l_{ab} and l_{bc} are orthogonal to w_{bc} and w_{bc}, we can make the following inequality:

$$-\frac{1}{\lambda(l_{ab})} \leq \frac{1 - w_1}{w_1} \leq -\frac{1}{\lambda(l_{bc})},$$

where $\lambda(l_{ab})$ denotes a slope of l_{ab}, *i.e.*, $\lambda(l_{ab}) = \frac{b_2 - a_2}{a_1 - b_1}$.

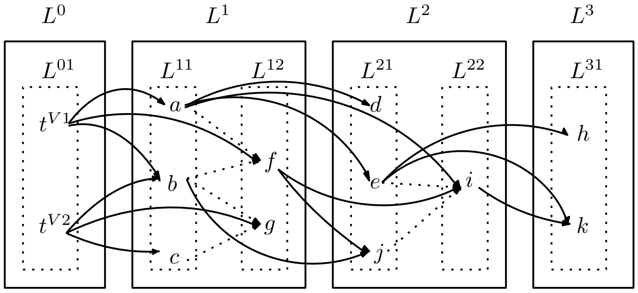


Fig. 7. Extended dual-resolution layer with L^0 using the toy dataset (Fig. 1)

We can replace the inequality as the range of w_1.

$$\frac{\lambda(l_{bc})}{\lambda(l_{bc}) - 1} \leq w_1 \leq \frac{\lambda(l_{ab})}{\lambda(l_{ab}) - 1}.$$

Fig. 6(b) describes the weight ranges for each tuple on the basis of this inequality. Once these ranges are identified, we retrieve the matching range for the given weight vector w using a binary search for w_1, and identify a top-*1* candidate. This search method incurs the logarithmic search cost $O(log|L^{11}|)$.

While this scheme is highly effective, its extension to higher dimensionality is non-trivial and expensive, as partitioning $\mathcal{W}$ in $(d-1)$-dimensional space becomes more complex. We thus exploit this scheme to inspire an alternative technique that can be scaled for the dimensionality.

B. Top-1 Tuple Selection in High-Dimensional Space

To achieve scalability over the dimensionality, we relax L^0 not to a single candidate t but to a *cluster* of tuples in L^1. While this intuition is similar to making the *pseudo-tuples* in L^0 as used in DG [5], our technique is more efficient by exploiting fine-level sublayers.

Specifically, we first group the tuples in L^1 using a clustering algorithm, *e.g.*, *k-means*. All tuples in a cluster C are dominated by a virtual tuple $t^V = (t_1^V, \ldots, t_d^V)$, where $t_i^V = \min_{t \in C} t_i$. We thus build a virtual zero layer, where virtual tuples can dominate tuples in L^1. Unlike in DG, we can form a dual-resolution layer for the pseudo-tuples. That is, the virtual tuples are divided into multiple fine-level layers, and the $\forall$- and $\exists$-dominance relationships hold between L^0 and L^1. Therefore, this construction can achieve higher selectivity for L^{11}.

To illustrate this, Fig. 7 shows an example of constructing L^0. For tuples in L^1, two clusters, $\{a, b, f\}$ and $\{c, g\}$, are identified. We make two pseudo-tuples, t^{V1} and t^{V2}, where the attribute values are made by the minimum values for the corresponding clusters. We then form the $\forall$-dominance relationship between L^0 and L^1. By generating L^0, we can thus provide selective access to L^1.

VI. Experiments

This section presents empirical evaluation results for our proposed dual-resolution layer. We first explain the experimental settings used. We then validate the indexing construction time and the number of tuples accessed for top-k query processing by comparing our technique with state-of-the-art layer-based methods for extensive synthetic datasets.

A. Experimental Settings

To evaluate our dual-resolution layer, we generated extensive synthetic datasets with four parameters: distribution, dimensionality, retrieval size, and cardinality. All the attribute values were positive real numbers, *i.e.*, $t_i \in (0, 1)$.

- Distribution: We generated two datasets, Independent (IND) and Anti-correlated (ANT), following the data generation instructions in [23].
- Dimensionality d: We varied the dimensionality d from 2 to 5. (Default: $d = 4$)
- Cardinality n: We varied the cardinality n from $100K$ to $500K$. (Default: $n = 200K$)
- Retrieval size k: We varied the retrieval size k from 10 to 50. (Default: $k = 10$)

For the scoring function $\mathcal{F}$, we randomly generated a user-specific weight vector $w = (w_1, \ldots, w_d)$ satisfying $\forall i \in [1, d]$: $0 < w_i < 1$ and $\sum w_i = 1$.

We then compared our dual-resolution layer with the state-of-the-art layer-based indices. Specifically, we implemented the following algorithms.

- DG: We implemented DG, which sequentially built skylines and tuples between adjacent layers were connected by $\forall$-dominance relationships. When computing skylines, we employed the state-of-the-art skyline algorithm BSkyTree [28].
- DG+: DG+ is the advanced algorithm of DG. We built a zero layer L^0 for the first layer by exploiting the pseudo-tuples using k-means clustering. For this optimization technique, we followed the instructions explained in [5].
- HL: We implemented HL, which built sorted lists for each convex skyline layer. For computing convex skylines, we modified the state-of-the-art convex hull algorithm QHull [22] available at *http://www.qhull.org*.
- HL+: HL+ is the optimized version of HL. When performing TA using sorted lists for each layer, we updated a tight threshold by accessing convex layers in a round-robin manner as described in [6]. (Because HL+ always showed better performance than HL, we do not report the results for HL in this paper).
- DL: We implemented DL that built coarse- and fine-level layers and tuples were connected by $\forall$- and $\exists$-dominance relationships. When computing skylines and convex skylines for the dual-resolution layer, we leveraged the same algorithms as used in DG and HL.
- DL+: We optimized DL by building a zero layer L^0 as discussed in Section V-B. Unlike DG+, L^0 was split into multiple fine-level layers to improve the selectivity for L^{11}.

As a measure to validate the performance, we employ *the number of tuples evaluated* (Definition 9). Other measures such as *query response time* and *I/O access cost* may affect the detailed implementation techniques of each indexing methods. In contrast, our measure is suitable for comparing the overall performance without this issue. In addition, this measure may be directly proportional to the query response time and I/O access cost.

TABLE IV
INDEX CONSTRUCTION TIME OF EACH ALGORITHM (SEC)
($k = 10$, $d = 4$, $n = 200K$)

Dist.	HL	HL+	DG	DG+	DL	DL+
IND	1.310	1.310	4.667	4.670	12.383	12.411
ANT	7.375	7.375	127.185	128.732	161.284	160.879

We assume that all implemented indices are in the main memory. However, these algorithms can be modified into disk-based algorithms, where tuples in the same layer are stored in the same disk block to reduce I/O cost, as discussed in [5].

All experiments were conducted using the Windows 7 with on an Intel Core i7 950 3.07 Ghz CPU and 24GB RAM, using C++ implementations of these algorithms.

B. Index Construction Time

We first compare the index construction time of algorithms (Table IV). Each parameter was set by default for each distribution (n = $200K$, d = 4, k = 10).

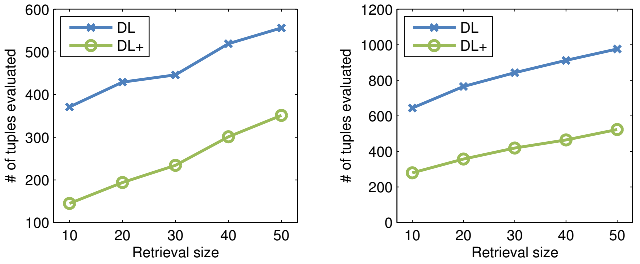
The dominant part of the index building time comes from making layers and forming the relationships between layers. Specifically, DG requires the skyline computation for each layer and dominance tests to connect the relationship between adjacent layers. In addition, the index building time of HL includes the convex skyline computation for each layer and the sorting of attribute values within each layer. In contrast, since DL requires us to compute both skylines and convex skylines for coarse- and fine-level layers, it encodes richer dominance relationships but incurs more computation costs than other indices.

In addition, to optimize the zero layer L^0 for DG and DL, the optimized versions, DG+ and DL+, require more construction time. However, because the number of tuples in the zero layer is small, this computation time is negligible (less than 1%). Meanwhile, HL+ shares the same index with HL (difference only in query processing part using a tighter threshold), and thus incurs the same index building time.

C. Performance Comparison between DL and DL+

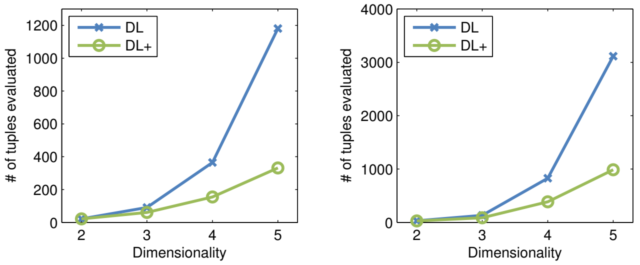
This section compares DL and DL+ for varying retrieval sizes and dimensionality. We used independent and anti-correlated datasets with k from 10 to 50, and d from 2 to 5 (n = $200K$). The main difference between DL and DL+ is the existence of the zero layer L^0, as discussed in Section V.

Fig. 8 shows the effect of L^0 with varying retrieval size. Regardless of the change in the retrieval size, the performance gap between DL and DL+ remains more or less constant, *i.e.*, DL+ is always two times better than DL in all settings. This implies that the access cost is proportional to the retrieval size k for both techniques.



(a) IND (b) ANT

Fig. 8. DL and DL+ with varying retrieval size k



(a) IND (b) ANT

Fig. 9. DL and DL+ with varying dimensionality d

Fig. 9 shows that supporting selective access to L^1 in DL+ results in a higher performance gain than in DL as the dimensionality increases. Observe that the gap between DL and DL+ tends to increase for high-dimensional datasets. For instance, when d = 5, DL+ accesses about three times fewer tuples than DL.

D. Effect of Retrieval Size

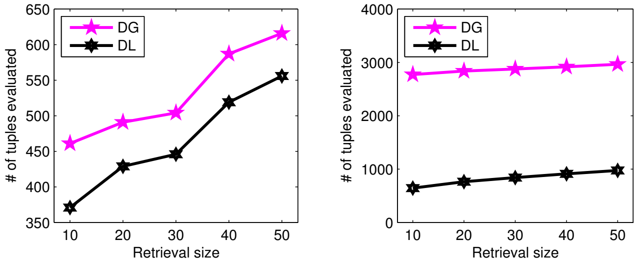
This section evaluates the effect of retrieval size by comparing five algorithms. We used independent and anti-correlated datasets with k from 10 to 50 (d = 4, n = 200K). In all settings, observe that our proposed algorithm DL (DL+) always outperforms DG (DG+) and HL+.

Regardless of the change in the retrieval size, DL (DL+) consistently outperforms DG (DG+) (Figs. 10 and 11). For instance, when $k = 50$, DL consistently accesses three times fewer tuples than DG for anti-correlated datasets, regardless of the retrieval size k. This observation is consistent with ours when comparing DL and DL+ (Fig. 8).

We next compare DL+ and HL+ with varying retrieval size. Observe that DL+ accesses far fewer tuples than HL+. In particular, as the retrieval size increases, the performance gap between DL+ and HL+ tends to increase (Fig. 12). For instance, when $k = 50$ for anti-correlated datasets, the access cost of DL is one order of magnitude smaller than that of HL. This is because the selective access in HL+, performing TA for sorted lists, is sensitive to the retrieval size.

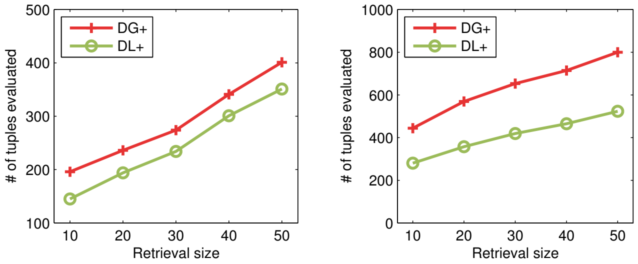
E. Effect of Dimensionality

This section evaluates the effect of dimensionality by comparing four algorithms. We used independent and anti-correlated datasets with d from 2 to 5 (k = 10, n = 200K). In all settings, DL (DL+) always accesses fewer tuples than DG (DG+) and HL+.



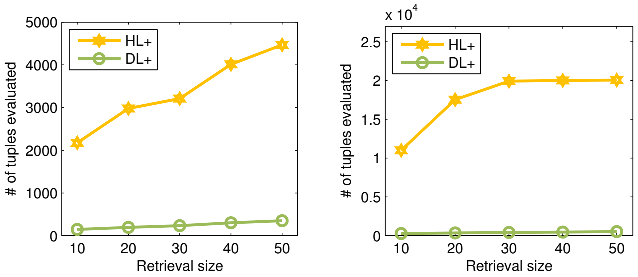
(a) IND (b) ANT

Fig. 10. DG and DL with varying retrieval size k



(a) IND (b) ANT

Fig. 11. DG+ and DL+ with varying retrieval size k

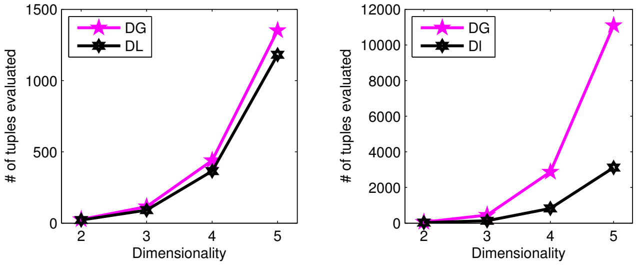


(a) IND (b) ANT

Fig. 12. HL+ and DL+ with varying retrieval size k

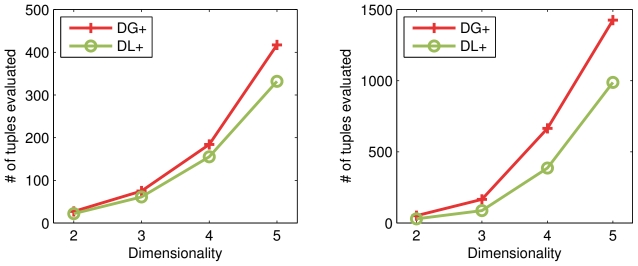
As the dimensionality increases, the performance gap between DL (DL+) and DG (DG+) increases linearly (Figs. 13 and 14). For instance, when $d = 5$, DL is about 2.5 times faster than DG for anti-correlated datasets. In particular, DL (DL+) is more efficient than DL (DL+) for anti-correlated and high-dimensional datasets.

This observation suggests that the optimization margin coming from the filtering technique using fine-level layers increases in high-dimensional data. Specifically, since our key contribution is the division of a coarse-level layer and the encoding of richer dominance relationships, *i.e.*, ∀- and ∃-dominance, DL and DL+ are relatively more effective, whereas other methods suffer from accessing the coarse-level layer with too many tuples. It is known from previous literature that such cardinality explodes when the dimensionality is high or



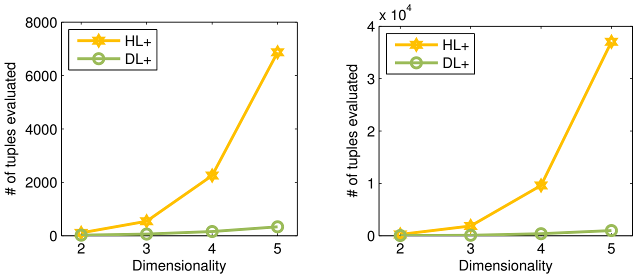
(a) IND (b) ANT

Fig. 13. DG and DL with varying dimensionality d



(a) IND (b) ANT

Fig. 14. DG+ and DL+ with varying dimensionality d



(a) IND (b) ANT

Fig. 15. HL+ and DL+ with varying dimensionality d

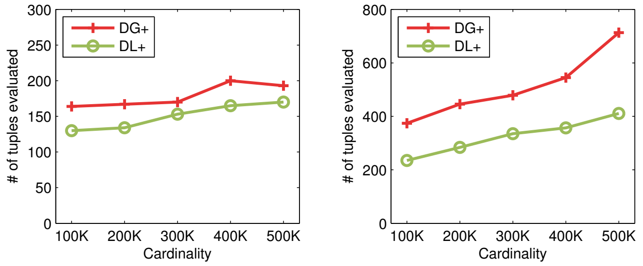
attributes are anti-correlated. This is also known as the curse of dimensionality problem. Using ∃-dominance relationship is thus effective for high dimensional and anti-correlated data.

We next compare DL+ with HL+ for varying dimensionality (Fig. 15). Observe that DL+ accesses far fewer tuples than HL+, as expected. For instance, when $d = 5$ for an anti-correlated distribution, the access cost of DL is two orders of magnitude smaller than that of HL.

F. Effect of Cardinality

This section evaluates the effect of cardinality by comparing two algorithms. We used independent and anti-correlated datasets with n from $100K$ to $500K$ (k = 10, d = 4). In all settings, our proposed algorithm DL always outperforms DG.

Fig. 16 shows that both DL+ and DG+ are less sensitive to cardinality comparing to the retrieval size and the dimensionality. This is because the layers enable proportional access to data, as observed from our evaluations with varying k. As a result, when $k \ll n$, the algorithms access highly selective tuples insensitive to n. (Because of space limitations, we do not show the graph comparing other algorithms. The findings are consistent with the results shown in Fig. 16.)



(a) IND (b) ANT

Fig. 16. DG+ and DL+ with varying cardinality n

VII. Related Work

Top-k queries are widely used in many real applications such as image [9], [10] and text [14] retrieval. The main concern for top-k query processing is to access a subset of tuples in the target relation as small as possible. The existing top-k processing algorithms are categorized into three classes: layer-, list-, and view-based approaches.

A. Layer-Based Approach

The layer-based approach materializes tuples into consecutive layers, and employs the relationships between layers to reduce data access. We categorize the existing layer-based approaches into three types: skyline-, convex-, and hybrid-layer approaches. We discuss how our proposed approach complements the state-of-the-art methods in each category.

- **Skyline-layer approach:** DG [5] finds skylines iteratively and materializes them as layers. The dominance relationships between layers enable selective access to layers. However, for high-dimensional or anti-correlated data with a large set of skylines, the cardinality of each layer is high and the performance starts to deteriorate. Our dual-resolution layer indexing addresses this limitation by further splitting each layer into fine-granularity sublayers.
- **Convex-layer approach:** Onion [3] materializes convex skylines into layers. Since dominance relationships do not exist between adjacent layers, Onion requires complete access to layers, and thus incurs higher access costs than DG. AppRI [4] improves Onion by designing a *robust* index, which reduces the sizes of tuples in each layer by counting the number of dominating points. However, AppRI cannot avoid complete access to layers either. In clear contrast, our dual-resolution layer defines the dominance relationships between the coarse- and fine-granularity layers, and makes use of those relationships to enable selective access to all layers.

- **Hybrid-layer approach:** HL [6] materializes convex skyline as layers, but builds d sorted lists within each layer for d attributes. This sorted ordering helps avoid unnecessary access within a layer, and thus enables selective access. However, compared to our systematic approach of defining and leveraging formal relationships, the selectivity is one order of magnitude higher, as we found empirically reported in Section VI.

B. List-Based Approach

The list-based approach exploits a set of sorted lists for efficient top-k processing. FA [8], TA [11], MPro [12], Upper [13], and Unified [15] are well-known algorithms using this approach. Given a scoring function $\mathcal{F}$, these algorithms access and aggregate the score of tuples by iteratively accessing the sorted lists in a round-robin manner until the best k tuples seen become the top-k answers. However, as sorted lists may have different importance (or weights) and may also be correlated, round-robin access may not be optimal. Some research work [16] has been carried out on optimal access scheduling of sorted lists. Meanwhile, the layer-based approaches are inherently robust to weights and correlations, and do not require such scheduling.

C. View-Based Approach

The view-based approach examines pre-computed top-k queries as *views* in database systems. The key idea of these approaches such as PREFER [17], [18] and LPTA [19] is to select the most similar materialized top-k query, and to reuse access and computation to compute a new top-k query. Specifically, PREFER determines a *watermark* tuple to obtain top-k answers from the existing query. LPTA optimizes PREFER by combining multiple top-k queries using a linear programming technique. A drawback is the overhead of storing and managing multiple top-k views.

VIII. Conclusion

This paper studied the problem of designing a layer-based index to support multiple top-k queries efficiently. In particular, we focused on the optimization goal of making selective access to each layer. To pursue this optimization goal, we designed a dual-resolution layer, in which each coarse-level layer was further divided into multiple fine-level sublayers at a fine granularity. We then defined $\exists$-dominance to improve the selectivity of access within the coarse-level layers, and designed an optimization technique to virtually build the zero layer to enable selective access to the tuples in the first layer. We analyzed that our dual-resolution layer outperformed the state-of-the-art methods.

Acknowledgement

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea and Microsoft Research, under IT/SW Creative research program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2011-C1810-1102-0008).

References

[1] R. Kimball and K. Strehlo, “Why decision support fails and how to fix it,” *SIGMOD Record*, vol. 24, no. 3, pp. 92–97, 1995.

[2] M. J. Carey and D. Kossmann, “On saying ”enough already!” in sql,” in *SIGMOD*, 1997, pp. 219–230.

[3] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith, “The onion technique: Indexing for linear optimization queries,” in *SIGMOD*, 2000, pp. 391–402.

[4] D. Xin, C. Chen, and J. Han, “Towards robust indexing for ranked queries,” in *VLDB*, 2006, pp. 235–246.

[5] L. Zou and L. Chen, “Dominant graph: An efficient indexing structure to answer top-k queries,” in *ICDE*, 2008, pp. 536–545.

[6] J. Heo, J. Cho, and K. Whang, “The hybrid-layer index: A synergic approach to answering top-k queries in arbitrary subspaces,” in *ICDE*, 2010, pp. 445–448.

[7] L. Zou and L. Chen, “Pareto-based dominant graph: An efficient indexing structure to answer top-k queries,” *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 5, pp. 727–741, 2011.

[8] R. Fagin, “Combining fuzzy information from multiple systems,” in *PODS*, 1996, pp. 216–226.

[9] S. Nepal and M. V. Ramakrishna, “Query processing issues in image (multimedia) databases,” in *ICDE*, 1999, pp. 22–29.

[10] U. Güntzer, W.-T. Balke, and W. Kießling, “Optimizing multi-feature queries for image databases,” in *VLDB*, 2000, pp. 419–428.

[11] R. Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for middleware,” *J. Comput. Syst. Sci*, vol. 66, no. 4, pp. 614–656, 2003.

[12] K. C. Chang and S. Hwang, “Minimal probing: supporting expensive predicates for top-k queries,” in *SIGMOD*, 2002, pp. 346–357.

[13] N. Bruno, L. Gravano, and A. Marian, “Evaluating top-k queries over web-accessible databases,” in *ICDE*, 2002, pp. 369–378.

[14] M. Theobald, G. Weikum, and R. Schenkel, “Top-k query evaluation with probabilistic guarantees,” in *VLDB*, 2004, pp. 648–659.

[15] S. Hwang and K. C. Chang, “Optimizing top-k queries for middleware access: A unified cost-based approach,” *ACM Trans. Database Syst.*, vol. 32, no. 1, pp. 1–41, 2007.

[16] H. Bast, D. Majumdar, R. Schenkel, M. Theobald, and G. Weikum, “Io-top-k: Index-access optimized top-k query processing,” in *VLDB*, 2006, pp. 475–486.

[17] V. Hristidis, N. Koudas, and Y. Papakonstantinou, “Prefer: A system for the efficient execution of multi-parametric ranked queries,” in *SIGMOD*, 2001, pp. 259–270.

[18] V. Hristidis and Y. Papakonstantinou, “Algorithms and applications for answering ranked queries using ranked views,” *VLDB J.*, vol. 13, no. 1, pp. 49–70, 2004.

[19] G. Das, D. Gunopulos, N. Koudas, and D. Tsirogiannis, “Answering top-k queries using views,” in *VLDB*, 2006, pp. 451–462.

[20] H. Edelsbrunner, *Algorithms in combinatorial geometry.* Springer, 1987.

[21] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry*. Springer, 2008.

[22] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Trans. Math. Softw.*, vol. 22, no. 4, pp. 469–483, 1996.

[23] S. Börzsönyi, D. Kossmann, and K. Stocker, “The skyline operator,” in *ICDE*, 2001, pp. 421–430.

[24] K. Tan, P. Eng, and B. C. Ooi, “Efficient progressive skyline computation,” in *VLDB*, 2001, pp. 301–310.

[25] D. Kossmann, F. Ramsak, and S. Rost, “Shooting stars in the sky: An online algorithm for skyline queries,” in *VLDB*, 2002, pp. 275–286.

[26] D. Papadias, Y. Tao, G. Fu, and B. Seeger, “An optimal and progressive algorithm for skyline queries,” in *SIGMOD*, 2003, pp. 467–478.

[27] J. Chomicki, P. Godfery, J. Gryz, and D. Liang, “Skyline with presorting,” in *ICDE*, 2003, pp. 717–719.

[28] J. Lee and S. Hwang, “BSkyTree: Scalable skyline computation using a balanced pivot selection,” in *EDBT*, 2010, pp. 195–206.

[29] J.-S. Heo, K.-Y. Whang, M.-S. Kim, Y.-R. Kim, and I.-Y. Song, “The partitioned-layer index: Answering monotone top-k queries using the convex skyline and partitioning-merging technique,” *Inf. Sci.*, vol. 179, no. 19, pp. 3286–3308, 2009.

[30] J. Matousek and O. Schwarzkopf, “Linear optimization queries,” in *Symposium on Computational Geometry*, 1992, pp. 16–25.

[31] B. Chazelle, “On the convex layers of a planar set,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 509–517, 1985.

[32] A. Vlachou, C. Doulkeridis, Y. Kotidis, and K. Nørvåg, “Reverse top-k queries,” in *ICDE*, 2010, pp. 365–376.