Skyline Ranking à la IR

George Valkanas
Dept. of Informatics and Telecommunications
University of Athens
Athens, Greece
gvalk@di.uoa.gr

Apostolos N. Papadopoulos
Dept. of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
papadopo@csd.auth.gr

Dimitrios Gunopulos
Dept. of Informatics and Telecommunications
University of Athens
Athens, Greece
dg@di.uoa.gr

ABSTRACT

Skyline queries have emerged as an expressive and informative tool, with minimal user input and thus, they have gained widespread attention. However, previous research works tackle the problem from an efficiency standpoint, i.e., returning the skyline as fast as possible, leaving it to the user to manually inspect the entire skyline result. Clearly, this is impractical, even with a few dozen points. The techniques addressing this issue are computationally expensive, mapping to NP-Hard problems or having exponential complexity O(2^d) with respect to data dimensionality d. Moreover, the result is a *set*, lacking any quality-based ranking. In this paper, we propose a novel IR-style ranking mechanism for skyline points, based on the renowned *tf-idf* weighting scheme. We present efficient algorithms to compute the quality of a skyline point according to our technique, and induce a total ordering of the skyline set. Finally, we empirically evaluate the efficiency of our method with real-life and synthetic data sets.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process; H.2.8 [**Database Applications**]: Data Mining; H.2.4 [**Systems**]: Query Processing

Keywords

Skyline, Ranking, TF-IDF, Top-k

1. INTRODUCTION

Skyline queries were initially proposed in the context of databases in [2]. Their ability to empower multi-criteria decision making, with minimum user input, and their applicability in a series of domains has gained them considerable attention from the database community. Assuming $w.l.o.g.$ that smaller values are preferred, we say that point p = ($p.x_1$, $p.x_2$, ..., $p.x_d$) $\in \mathcal{D}$ *dominates* point q = ($q.x_1$, $q.x_2$, ..., $q.x_d$) $\in \mathcal{D}$ (and write $p \prec q$), when: $\forall i \in \{1, ..., d\}, p.x_i \leq q.x_i \land \exists j \in \{1, ..., d\} : p.x_j < q.x_j$. Simply put, p dominates q, if p is at least as good as q in every dimension, and it is strictly better than q in at least one. The skyline of $\mathcal{D}$, denoted as $\mathcal{S}$, is composed of all d-dimensional points that are not dominated by any other point.

Related Work and Motivation. Skyline queries are a well studied problem in the area of Computational Geometry [6], but have attracted considerable attention in the context of databases, when Börzsönyi et al introduced the *skyline operator* [2].

Several algorithms have been presented for skyline computation, with BBS [9] being the most preferred when using an index, due to its progressiveness and I/O optimality. Efficient algorithms have also been proposed in [5] and [10] for such cases where indexing cannot be applied. These works (as many more) focus on efficiency, i.e., retrieving the skyline as quickly as possible, and the result is a *set*, i.e. all points are equally important. In other words, there is no discrimination between the points, leaving it entirely to the user to select. Unfortunately, the skyline may contain far too many points: the skyline of randomly generated points is $\Theta(\log^{d-1}(n))$ [1]. In an era when "ten blue links" seem too many [7], returning approximately 10^3 skyline points from a dataset with 10^9 points, immediately negates the advantages of skyline queries.

To address the skyline cardinality explosion problem, the general direction is to return a subset of k skyline points, where k is a user- or application-defined parameter. The subset has some specific properties, e.g., collectively maximizes coverage [8], captures the contour of the skyline [11], diversifies the skyline [12, 11], etc. Nevertheless, these techniques fail to differentiate between the returned points based on some importance measure. Moreover, they are mapped to NP-Hard problems, so we can only efficiently approximate the solutions, unless P=NP.

Researchers have also investigated ranking of skyline points. We identify two categories, depending on the amount of information used to rank the points: In the first case, the entire dataset is used, and the importance of a skyline point is given by the number of points it dominates [9, 14]. The major shortcoming of the first category is that all dominated points are equally important. For example, a point p_1 dominated by 10 skyline points and another one p_2 dominated by 100 contribute the same weight to their dominators. Taking into account that skyline queries have an inherent relation to sorting [1], and that distance measures for sorted lists heavily rely on the relative positions of items, it feels counter-intuitive to use the same weight for all dominated points.

The second category relies on the skyline $\mathcal{S}$ alone, and typically uses dominance relations in subspaces [13, 4]. Consequently, such techniques ignore dataset characteristics. They are also generally inefficient, as they need to consider O(2^d) non-empty subspaces. Moreover, they favor skyline points with extreme values in a single dimension and have been shown to produce correlated results, whereas some of them [13] have not been sufficiently evaluated.

Contributions. To address these shortcomings, we present a novel ranking scheme for skyline points. We argue that a point's importance should be inversely proportional to the number of skyline points that dominate it. Additionally, our model distinguishes dominated points, based on their relative positions. For instance, if $sp_1 \prec a$, $sp_1 \prec b$, and a and b do not dominate each other, they contribute equally to sp_1. Otherwise, if $a \prec b$, then $score(a) > score(b)$. Therefore, our technique promotes skyline points that dominate *genuine* points, i.e., points which are not dominated by many others, and is a hybrid approach.

To capture both aspects in a single scoring function, we apply a modified version of the renowned tf-idf weighting scheme and present efficient algorithms that rank the skyline according to this scheme. Our contributions are briefly described as follows:

- We define a novel, generic and intuitive measure of importance for skyline points. Inspired by the renowned *tf-idf* weighting scheme from information retrieval, our method promotes skyline points that dominate *genuine* points.

- We present efficient algorithms that compute the top-k most important skyline points, given our measure.

- We provide an extensive experimental evaluation, in terms of efficiency using both real-life and synthetic datasets.

Roadmap. The rest of the paper is organized as follows. Section 2 gives the details of our scoring model and the algorithms proposed for the task at hand. In Section 4 we evaluate our techniques. Finally, Section 5 summarizes our findings and concludes the paper by discussing briefly future work in the area.

2. DP-IDP WEIGHTING SCHEME

Our proposed measure, *dp-idp*, which stands for *dominance power - inverse dominance power*, is inspired by the renowned *tf-idf* weighting scheme from Information Retrieval. The general rationale is that dominated points are not equally important, and that they impact skyline point differently. Therefore, their contribution depends on some local (per skyline point) and some global characteristics (the entire skyline), much like *tf-idf* uses local and global information to find important keywords in a document corpus. In the following paragraphs we present our ranking scheme.

2.1 Inverse Dominance Power

We will start with *inverse dominance power* (idp), which is easier to define, due to its more global view. The *inverse dominance power* of a point $p \in (\mathcal{D} \setminus \mathcal{S})$ is the number of skyline points which dominate p. This factor is similar to *idf* in the sense that the more frequently p appears in a skyline point's dominated set, the lower the importance of p. More formally:

$$idp(p) = \log \frac{|\mathcal{S}|}{|\{sp \in \mathcal{S} : sp \prec p\}|}$$

An interesting property of idp is the following: Assume a set of points $q_1, q_2, ..., q_m$ dominated by all skyline points, i.e., $\forall sp \in \mathcal{S}$, $sp \prec q_j$, $j = 1, .., m$. The contributing score of the q_i's will be 0, due to the log in the idp factor. Such points do not alter the ranking of $\mathcal{S}$, either with ours or simpler models (e.g., $|\Gamma(sp)|$), because they affect all skyline points the same.

2.2 Dominance Power

There are several ways we could define the *dominance power* of a dominated point. Given that we want to measure this factor with respect to a skyline point sp, we argue that its relative position to sp should matter. As a result, the same dominated point may contribute differently to different skyline points.

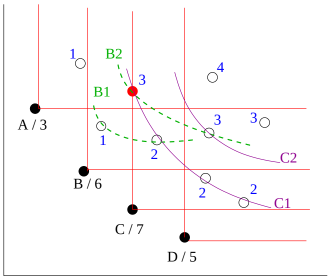


Figure 1: Example for the Dominance Power

To avoid introducing more artifacts in our model, we choose the dominance relation as our building block. More specifically, we find the *layer of minima* [1] $lm(p, sp)$ where the dominated point p falls in, with respect to sp. The dominance power of that point is then given by the inverse of the layer where it lies, i.e.,

$$dp(p, sp) = \frac{1}{lm(p, sp)}$$

Figure 1 portrays the skyline of a dataset, and the dominance region for each skyline point. Moreover, it shows the layers of minima for skyline points B and C, in dotted green and purple respectively. We observe that the red-filled point, dominated by both B and C, lies at different minima layers. Being easier to reach it from C, should render it more important for C. On the contrary, there is an additional layer for skyline point B prior to reaching that point, that decreases its importance. This is similar to *term frequency*, where the same term is weighted differently, depending on its occurrence in each document.

2.3 Putting it all together

Given our previous discussion, we can now formally introduce how we compute the importance of a skyline point *sp*. We use an additive model, because i) it is monotonous (dominating more points increases the overall importance) ii) it is comprehensive and iii) it leads to efficient computations, as we followingly discuss. Therefore, the importance of a skyline point *sp* is given by:

$$score(sp) = \sum_{p : sp \prec p} \frac{1}{lm(p, sp)} \times \log \frac{|\mathcal{S}|}{|\{sp' \in \mathcal{S} : sp' \prec p\}|}$$

The additive model also favors skyline points that dominate more *genuine* points, i.e., points dominated by few others in general (not just skyline points). This is important, because those skyline points are the reason why the dominated ones cannot be part of the skyline. For example, if we remove B from the skyline in Figure 1 (e.g., a hotel being fully booked), the point next to it will enter the skyline at once (similarly for the closest point dominated by A). Additionally, points dominated by the entire skyline still have no effect. Finally, note that the sum of $tf - idf$ values is also used in IR systems, to score the entire document against a query.

3. RANKING THE SKYLINE

Algorithm 1 gives a straightforward approach to rank the skyline $\mathcal{S}$ with our proposed scheme. For each skyline point *sp* (line 1),

[1] In the literature, the term *layer of maxima* is more common. Here, we use the term *layer of minima* because we assume that small values are preferable.

Algorithm 1 Baseline

```
   Input: Skyline $\mathcal{S}$, Dataset $\mathcal{D}$, Integer $k$
   Output: Ranked List
1: for every $sp \in \mathcal{S}$ do
2:     score( $sp$ ) $\leftarrow$ 0; layer $\leftarrow$ 1; $lm \leftarrow$ NextLayer( $sp, \emptyset$ );
3:     while ($lm \neq \emptyset$) do
4:         for every $p \in lm$ do
5:             score( $sp$ ) += $\frac{1}{layer} \times log\frac{|\mathcal{S}|}{|\{sp':sp' \prec p\}|}$;
6:         $lm \leftarrow$ NextLayer( $sp_i, lm$ ); layer++;
7: Order by descending score($sp$);
8: Return $k$ highest skyline points;
```

we extract one-by-one its minimal layers (lines 2–6). $NextLayer$ uses BBS [9] internally. For every point in each layer (line 4), we find how many in $\mathcal{S}$ dominate it, and together with the layer's index, we update the score of *sp* (line 5). After ordering the skyline in decreasing score order (line 7), we return the top-k ranked items.

Unfortunately, this approach is computationally expensive, due to repeated evaluations. It also computes the score of all skyline points, despite our interest in the top-k results. Finally, it lacks any notion of *progressiveness*, as we need to rank the entire skyline first. For all these reasons, we present an alternative approach, that relies on bounding the score of a skyline point.

3.1 Bounding the score

Bounding the score of a skyline point sp will help us reduce computations, by pruning away those that will not make it to the top-k positions. To achieve this, we use the number of points that *sp* dominates, $|\Gamma(sp)|$. We can then derive lower and upper bounds of the score of a skyline point, as shown in the next paragraphs.

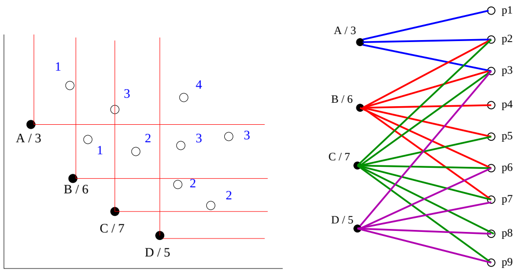
Loose Bounds. The simplest bounds consider each skyline point independently of the rest, and are derived as follows. A skyline point obtains its maximum score when *all* the points it dominates are in the same (first) layer, and they are *not* dominated by any other skyline point. In that case, the upper bound is:

$$\overline{score}(sp) = |\Gamma(sp)| \times \log |\mathcal{S}|$$

On the other hand, the lower bound is obtained when every point is dominated by the entire skyline $\mathcal{S}$. In that case, the score is 0, due to the $idp(sp)$ factor. However, this bound only holds for the skyline point sp_{min} with the minimum $|\Gamma(sp_{min})|$. The rest of the skyline dominates some points, which can not be domianted by sp_{min}. Consider, for instance, that $|\Gamma(sp_{min})| = 3$ and that $|\Gamma(sp')| = 5$. By definition, the 2 additional points dominated by sp' can not be dominated by sp_{min}, otherwise $|\Gamma(sp_{min})| = 5$. Therefore, the surplus will be dominated by $|\mathcal{S}| - 1$ skyline points, and a correlated distribution [2] will give the lowest score value. This gives a slightly better lower bound:

$$\underline{score}(sp) = \log \frac{|\mathcal{S}|}{|\mathcal{S}| - 1} \times \sum_{1}^{n-min_{\Gamma}} \frac{1}{i}$$

Collaborative Bounds. Despite their simplicity, the above bounds have limited pruning capability. Assume, for instance, a dataset $\mathcal{D}$, with $|\mathcal{D}| = 1M$ and $|\mathcal{S}| = 800$. If $|\Gamma(sp)| = 300K$, then $\overline{score}(sp) \simeq 871K$, and $\underline{score}(sp) \simeq 3 \times 10^{-3}$. Note that a skyline point sp' with $|\Gamma(sp')| = 1$, has an upper bound of ~2.9, making it eligible for consideration in the second round! Apparently, the computational gains of such bounds are easily swept away.



(a) Motivating Example (b) Bipartite Form

Figure 2: Example of skyline and bipartite domination graph

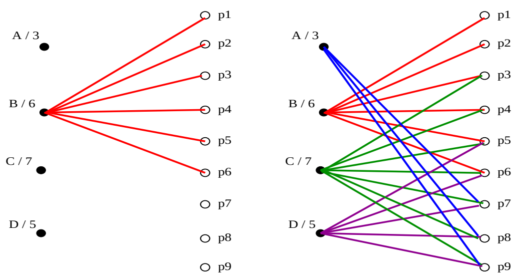
To address this issue, we derive stricter bounds, through additional information from other skyline points. To better understand this approach, we visualize the problem as a bipartite graph. Figure 2 shows a dataset, with its skyline and dominance regions on the left, and the resulting bipartite graph on the right. The left hand side of the graph contains the skyline, whereas the right hand side has the dominated points. We add an edge between a skyline point $sp \in \mathcal{S}$ and a dominated point $p \in \mathcal{D} \setminus \mathcal{S}$, iff $sp \prec p$.

We start with the upper bound. Due to the additive model, the score of a skyline point is maximized when the contribution of each dominated point is maximized. It is easy to see that dp is maximized when the dominated point is at the earliest possible layer. To maximize idp, we rely on the Pigeonhole Principle. For any two skyline points sp_1, sp_2, if $\Gamma(sp_1) + \Gamma(sp_2) > |\mathcal{D}|$, then at least $|\mathcal{D}| - (\Gamma(sp_1) + \Gamma(sp_2))$ dominated points are shared by sp_1 and sp_2. Having more common points reduces $idp()$, so we only consider the minimum overlap. The question now becomes "*How should we assign the common edges to maximize the score of a skyline point*"? Lemma 1 answers this question.

LEMMA 1. *Let sp be a skyline point, p_x and p_y two dominated points, where $sp \prec p_x$ and $sp \prec p_y$ and $lm(p_x) = lm(p_y) = l$. Assigning an edge to the point dominated by more skyline points gives a higher score(sp).*

PROOF. Let $\mathcal{S}_x$, $\mathcal{S}_y$ be the current sets of skyline points dominating p_x and p_y, respectively. Assigning an edge to either p_x or p_y gives two different bipartite graphs, with $\mathcal{S}'_x$ and $\mathcal{S}'_y$ being the new dominating sets of these points. It holds that $|\mathcal{S}'_x| = |\mathcal{S}_x| + 1$ (same for $\mathcal{S}'_y$), due to the new edge, i.e., one more dominating skyline point. The resulting bipartite graphs differ only in the assignment of this edge, which impacts the weights of p_x and p_y. The weights of all other dominated points remain unchanged. Assume that adding the edge to p_x yields a higher score. It so holds:

$$\frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}'_x|} + \frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}_y|} > \frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}_x|} + \frac{1}{l} \times \log \frac{|\mathcal{S}|}{|\mathcal{S}'_y|} \Rightarrow$$



(a) Dominance set (b) Upper Bound for B

Figure 3: Collaborative upper bound

[2] In a correlated distribution, each point is a minimal layer

$$\log(\frac{|\mathcal{S}|}{|\mathcal{S}'_x|}\times\frac{|\mathcal{S}|}{|\mathcal{S}_y|}) > \log(\frac{|\mathcal{S}|}{|\mathcal{S}_x|}\times\frac{|\mathcal{S}|}{|\mathcal{S}'_y|}) \Rightarrow |\mathcal{S}_x|\cdot|\mathcal{S}'_y| > |\mathcal{S}'_x|\cdot|\mathcal{S}_y| \Rightarrow$$

$$|\mathcal{S}_x|\cdot(|\mathcal{S}_y|+1) > (|\mathcal{S}_x|+1)\cdot|\mathcal{S}_y| \Rightarrow |\mathcal{S}_x| > |\mathcal{S}_y|$$

□

The above result tells us that a higher score is achieved by adding the extra edge to the dominated point with the higher indegree! Such a result can also be efficiently integrated in an algorithm to compute the upper bound of a skyline point's score. Figure 3(b) shows the edge assignment for the upper bound of skyline points B, using the above result.

A naive implementation of the upper bound can be very inefficient [3], because it requires too many counter updates for the commonly dominated points. Since we must compute the bound of each skyline point sp independently and repeatedly, as new layers are extracted, we need a more efficient approach. Algorithm 2 presents this improved technique.

Algorithm 2 `Score Upper Bounding`

```
    Input: $\mathcal{D}$, $\mathcal{S}$, Skyline Point $poi$, $minIDP$, $layer$
    Output: Upper bound of $poi$
 1: $v_{idp}$.push( $minIDP$ ); $v_{pnd}$.push( pending( $poi$ ) );
 2: Sort $\mathcal{S}$, in decreasing $|\Gamma(sp)|$;
 3: for every $sp \in \mathcal{S}$, $sp \neq poi$ do
 4:     surplus = $|\Gamma(poi)|$ - seen( $poi$ ) + $|\Gamma(sp)|$ > $|\mathcal{D}|$
 5:     if ( surplus > 0 ) then
 6:         $v_{pnd}$[last] = $v_{pnd}$[last] - surplus;
 7:         $v_{pnd}$.push( surplus );
 8:         $v_{idp}$.push( $v_{idp}$[last] + 1 );
 9: $ub \leftarrow 0$;
10: for ( $i$ = 0; $i < v_{idp}$.size(); $i$++ ) do
11:     $ub \leftarrow \frac{1}{layer+1} * v_{pnd}[i] * \log\frac{|\mathcal{S}|}{v_{idp}[i]}$
12: Return $ub$;
```

The improved algorithm takes as input the dataset $\mathcal{D}$, the skyline $\mathcal{S}$, the skyline point of interest poi, and two values $minIDP$ and $layer$. As we extract more layers for poi, we must compute the number of skyline points dominating each of the extracted points, which we need for the idp value. The minimum value that we have seen this far is stored in $minIDP$, and is different for each skyline point. This value practically tells us that any point in subsequent layers will be dominated by *at least* $minIDP$ skyline points, due to dominance being a transitive relation. The $layer$ tells us which was the index of the last layer of minima extracted for poi.

The algorithm uses two vectors, storing the $minIDP$ value and the number of *unseen* points, that have not yet been extracted for poi (line 1). For example, if $|\Gamma(poi)|$ = 100, and we have extracted 30 points, then $unseen(poi)$ = 70. The vectors practically store how many points (v_{PND}) can be dominated by that many skyline points (v_{IDP}). We sort the skyline points in decreasing order of their dominance power (line 2). We iterate over them (line 3), and select those points that will share common edges with poi, using the Pigeonhole Principle (lines 4–5). The surplus of points is removed from the last position (line 6) and is appended, incremented by 1 (lines 7–8). With these values, we can compute the upper bound according to the DP-IDP scheme (lines 10-11).

To better explain lines 5–7, assume $v_{pnd}[last]$ = 60, $v_{idp}[last]$ = 4, and $surplus$ = 25. This means that 60 points will be dominated by 4 skyline points and the current sp will share at least 25 dominated points with poi. As a result, we must add an edge (i.e., increment the idp) for an equal number of unseen points from poi. These must be selected from the points with maximum current idp, due to Lemma 1. Processing the skyline in decreasing order of dominance power ensures that we are properly assigning edges, and that the maximum idp is in the last positition. Given these values, 25 points will be computed with an idp of 5, which we append, whereas 60-25=35 will remain with an idp of 4, which we update.

For the lower bounds we could follow a similar reasoning. Unfortunately, the edge assignment problem in this case is not as easy. Although certain properties are self-evident, e.g., dp decreases with a correlated distribution, they do not necessarily result in the lowest possible score for a point. Consequently, we may have to reassign edges, and, possibly, reconsider the layer where some points are (i.e., break the correlated distribution). Therefore, in our current work, we will not pursue the collaborative lower bounds further, but plan to actively investigate it in our ongoing work.

3.2 Skyline Ranking with IR-style

Now that we have shown how we can efficiently bound the score of a skyline point, using easily extracted information, we turn our focus to finding the top-k most important skyline information, according to our DP-IDP weighting scheme. Algorithm 3 shows the general idea of execution of our technique, to efficiently compute the top-k skyline points. Our algorithm processes the points according to a prioritization scheme, and employs pruning of skyline points that will certainly not be in the final top-k result.

Algorithm 3 `SkyIR`

```
    Input: Skyline $\mathcal{S}$, Dataset $\mathcal{D}$, Integer $k$
    Output: Top-$k$ list

 1: for every $sp \in \mathcal{S}$ do
 2:     $sp_\Gamma \leftarrow |\Gamma(sp)|$
 3:     $sp_{score} \leftarrow 0$;
 4:     $priorityQueue$.enqueue( $sp_{prior}$, $sp$ );
 5: $kScore \leftarrow 0$;
 6: while (!$priorityQueue$.empty()) do
 7:     $poi \leftarrow priorityQueue$.dequeue();
 8:     if ( UpperBound( $poi$ ) ) < $kScore$ ) then
 9:         Discard $poi$;
10:         continue;
11:     if ( pending( $poi$ ) > 0 ) then
12:         $lm \leftarrow$ NextLayer( $poi$, $lm$ );
13:         $poi_{score} \leftarrow$ updateScore( $poi$, $lm$ );
14:         $added \leftarrow$ topk.insert( $poi$, $poi_{score}$ );
15:         if (!$added$ AND pending( $poi$ ) == 0) then
16:             Discard $poi$;
17:             continue;
18:         if ( topk[$k$] > $kScore$ ) then
19:             $kScore \leftarrow$ topk[$k$]
20:     if ( pending( $poi$ ) > 0 ) then
21:         $priorityQueue$.enqueue( $sp_{prior}$, $sp$ );
22: Return topk;
```

The algorithm starts by initializing appropriate information on the skyline points (lines 1-4), such as their dominance count, known score, and priority value, according to the prioritization scheme that we use (see below). We add each skyline point to a priority queue, using its priority value (line 4). We also initialize the k-th value, i.e., the value of the k-th ranked skyline point, to 0. We then enter a loop, each time extracting the top-most item from the queue poi

[3] Our experiments showed that this step alone can make up for up to 10 seconds of CPU processing time.

(line 7). If the upper bound of that point's score is below the k-th value, there is no need for further examination (lines 8 – 10). So we discard it and proceed with the next one from the priority queue. Otherwise, we extract the next layer of poi, provided there is one (lines 11–12). We update the point's score using this layer (line 13) and try to add poi in the top-k result. If the point was not added, and it can not be further updated, we discard it and proceed with the next point from the priority queue (lines 14–17). If the point was added, we keep track of the k-th value in the top-k result. If we can further update it, we compute its new priority and add it back in the priority queue (lines 20–21). The loop ends when the priority queue becomes empty, meaning no other points can update their score. The top-k list contains the final result.

Priority Schemes. Our SkyIR algorithm relies on a prioritization scheme to process the skyline points. In our current work we experiment with the following prioritization schemes.

- **Round Robin** (RRB): Items are processed in a round robin fashion. According to this scheme, we can not process the same skyline point twice, unless we have processed every skyline point first. This scheme also allows for an implementation that relies on arrays rather than the general priority queue, leading to faster (main memory) acceses.
- **Pending** (PND): The priority of an item is the number of points that it has not yet processed. For example, if a skyline point dominates 100 points, and it has already "seen" 30, its priority will be 70. Therefore, the more dominated points it has yet to see, the higher the priority of the skyline point.
- **Upper Bound** (UBS): The priority of a skyline point is given by the upper bound of its score. In other words, its priority is its potential to achieve a high final score. Similarly to the previous scheme, a higher upper bound results in a higher priority for the skyline point. Given that the upper bound can be used as a point's priority, it is even more important to have an efficient technique to compute it, like Algorithm 2.

4. PERFORMANCE EVALUATION

In this section, we report on the results of our experimental evaluation. The experiments were run on a Quad-Core @2.5GHz machine, with 8Gb RAM, running Linux. The code was written in C++ and compiled with g++ 4.7.2, with -O3 optimization. The datasets we consider were indexed by an aggregate R*-tree, with a 4Kb page size. An associated cache with 20% of the corresponding R*-tree's blocks was used with every experiment. Unless stated otherwise, the reported timings are in seconds, measured as CPU processing time and assuming a default value of 8ms per page fault.

Datasets and Algorithms. We generated datasets with *independent* (IND) and *anticorrelated* (ANT) distributions, as in [2], and also use Forest Cover [4]. Table 1 shows their basic properties. Although the datasets may seem rather small in size (up to 500K), one should keep in mind that our weighting scheme extracts *all* of the minimal layers for each skyline point. This problem is known to be difficult for high dimensionality even in the RAM model [3].

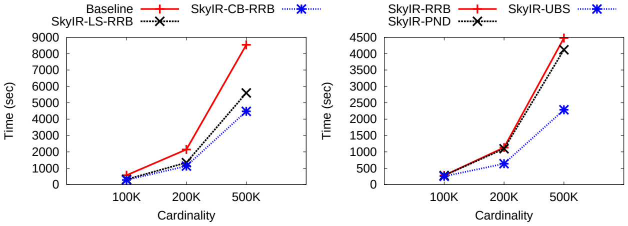
The algorithms that we evaluate are Baseline and SkyIR. For SkyIR we want to compare the performance of the **Loose** (LS) and **Collaborative** (CB) bounds, and how the three prioritization schemes affect the results. We use the abbreviations as suffixes to indicate what we compare each time.

Table 1: Dataset Statistics

Data set	Cardinality	Dimensionality
Independent (IND)	100K, 200K, 500K	2,3,4
Anticorrelated (ANT)	100K, 200K, 500K	2,3,4
Forest Cover (FC)	580K	2,3,4

Runtime. Figure 4(a) shows the total runtime for the independent distribution, when varying the dataset cardinality, with k=5. The naive approach is the worst, whereas SkyIR with collaborative bounds performs the best of the techniques, and we have obtained similar results when varying dimensionality and k.

As seen in Figure 4(b), the UBS prioritization scheme outperforms all others, resulting in up to 3× improvement compared to the baseline. Similar results are obtained for different priorities with the loose bounds, but the differences are less pronounced. An important observation from these plots is that the problem we are solving is not linear with the cardinality of points. The reason is that as the cardinality increases, there are more minimal layers to extract, and the computational costs are increased a lot, as a result of both more CPU processing and page faults.

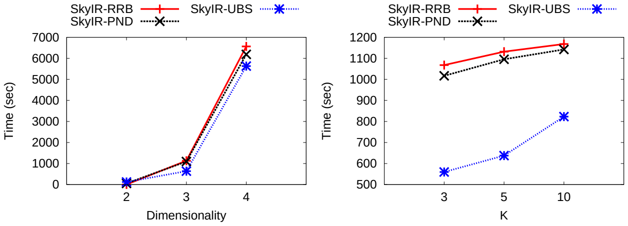


(a) Techniques (b) Prioritization for CB

Figure 4: Total runtime versus cardinality for IND, k=5

Figures 5(a)-(b) demonstrate how each prioritization scheme performs with the collaborative bounds. Figure 5(a) shows the performance when varying the data dimensionality. We observe that UBS performs the best for d = 3, 4, while being slightly worse for d=2. The reason for that is our array-based implementation, which is faster than the reordering of the priority queue maintained by PND and UBS. However, as seen in Figure 5(b) there is a huge improvement with UBS for d>2. The improvement increases with lower values of k, going up to 40%, because the collaborative bounds can prune away more points, reducing the computational costs.

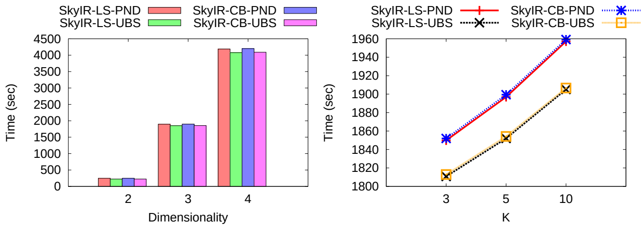
Figures 6(a) and (b) demonstrate how the prioritization schemes perform for the ANT, versus dimensionality and k, respectively. Once again, UBS is better than PND. The loose bounds appear to be slightly better than the collaborative, but not considerably. The difference comes from the fact that the loose bounds are less computationally intensive. The more interesting fact, however, is that ANT appears to be *easier* when compared to IND. In particular, for d=4, it takes ∼6000 seconds for CB to compute the top-5 for IND, whereas it takes ∼4000 seconds for ANT. The reason is again that IND has more layers to extract, and is more CPU hungry. Even though ANT has a lot of page faults, its CPU time is minimal.



(a) VS Dimensionality (k=5) (b) VS k (d=3)

Figure 5: Total runtime for various prioritization with IND, CB

[4] http://kdd.ics.uci.edu



(a) VS Dimensionality (b) VS k

Figure 6: Total runtime for ANT distribution

Finally, Figure 7 compares the loose and collaborative bounds, when applying the UBS technique on the real dataset FC. We observe that the CB technique performs better than LB for all tested dimensions.

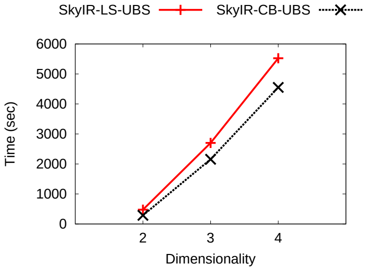


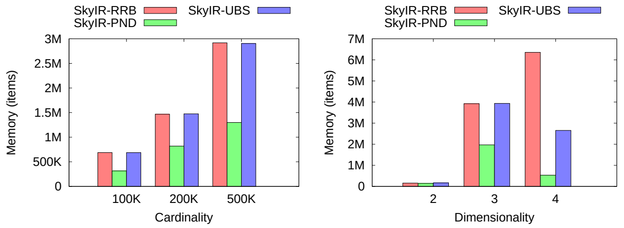
Figure 7: Forest cover

Memory Consumption. Finally, we compute the maximum number of items that we must maintain in memory while computing the top-k result. Figures 8(a) and (b) show this for IND and ANT, respectively, using the CB technique. We observe that the number of maintained items increases as the cardinality of IND also increases. On the other hand, increasing the dimensionality of ANT, does not have a similar effect: the number of memory points increases as we go from 2D to 3D, but drops again as we proceed to 4D. This may be explained again by the fact that ANT has less layers of minima to retrieve. For a fixed cardinality, more dimensions spread the points more, increasing the points retrieved with each layer. This decreases the information we must store to proceed to the next layer, giving as the plot of Figure 8(b).

Generally speaking, the schemes RRB and UBS behave almost the same (with the exception of ANT). We should stress the fact, however, that the pending scheme (PND) *always* results in less memory utilization. This is because the scheme will stick to a single point and try to reduce its number of pending points as much as possible, whereas the other schemes will rotate more over different points. This is an interesting outcome, because PND would be a good alternative in systems with limited resources.

5. CONCLUSIONS

In this paper, we proposed a novel model for ranking skyline points, based on the renowned *tf-idf* weighting scheme from Information Retrieval domain. We presented efficient techniques for finding the top-k result, by bounding the maximum score of a skyline point and employing pruning, combined with different visiting orders. We also experimentally evaluated the proposed bounding and prioritization schemes in terms of efficiency.



(a) IND (b) ANT

Figure 8: Maximum memory consumption for CB, k=5

Since the layers of minima problem is a difficult one (especially for the external memory model) as future work we plan to investigate the alternative to check only the first few layers of minima instead of computing the whole set. This is expected to improve performance at the cost of result accuracy. A second direction is to study the application of collaborative lower bounds in combination with the upper bounds studied in this work. Finally, we will work on the theoretical aspects of the problem in order to provide closed-form formulae for the cost of our algorithms. A potential starting point could be the application of a recurrence equation to estimate the number of layers combined with the cost of the BBS algorithm.

Acknowledgements: This work has been co-financed by EU and Greek National funds of the NSRF - Research Funding Programs: Heraclitus II fellowship, THALIS - GeomComp, THALIS - DISFER, ARISTEIA - MMD" and the EU funded project INSIGHT.

6. REFERENCES

[1] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, Oct. 1978.

[2] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[3] A. L. Buchsbaum and M. T. Goodrich. Three-dimensional layers of maxima. *Algorithmica*, 39:275–286, July 2004.

[4] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.

[5] A. Das Sarma, A. Lall, D. Nanongkai, and J. Xu. Randomized multi-pass streaming skyline algorithms. *Proc. VLDB Endow.*, 2(1):85–96, Aug. 2009.

[6] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22:469–476, 1975.

[7] C. Kwok, O. Etzioni, and D. S. Weld. Scaling question answering to the web. *ACM Trans. Inf. Syst.*, 19(3):242–262, 2001.

[8] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.

[9] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[10] C. Sheng and Y. Tao. On finding skylines in external memory. In *PODS*, pages 107–116, 2011.

[11] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009.

[12] G. Valkanas, A. N. Papadopoulos, and D. Gunopulos. Skydiver: A framework for efficient skyline diversification. In *EDBT*, pages 406–417, 2013.

[13] A. Vlachou and M. Vazirgiannis. Ranking the sky: Discovering the importance of skyline points through subspace dominance relationships. *Data Knowl. Eng.*, 69(9):943–964, 2010.

[14] M. L. Yiu and N. Mamoulis. Multi-dimensional top-*k* dominating queries. *VLDB J.*, 18(3):695–718, 2009.