

Telescope: Zooming to Interesting Skylines

Jongwuk Lee, Gae-won You, and Seung-won Hwang

POSTECH, Pohang, Korea
{julee, gwyou, swhwang}@postech.ac.kr

Abstract. As data of an unprecedented scale are becoming accessible, skyline queries have been actively studied lately, to retrieve “interesting” data objects that are not dominated by any other objects, *i.e.*, *skyline objects*. When the dataset is high-dimensional, however, such skyline objects are often too numerous to identify truly interesting objects. This paper studies the “curse of dimensionality” problem in skyline queries. That is, our work complements existing research efforts to address this “curse of dimensionality”, by ranking skyline objects based on *user-specific* qualitative preference. In particular, Algorithm *Telescope* abstracts skyline ranking as a dynamic search over skyline subspaces guided by user-specific preference with correctness and optimality guarantees. Our extensive evaluation results validate the effectiveness and efficiency of Algorithm *Telescope* on both real-life and synthetic data.

1 Introduction

As data of an unprecedented scale are becoming accessible, skyline queries have been actively studied lately, to retrieve “interesting” data objects that are not “dominated” by any other objects, *i.e.*, *skyline objects*. To illustrate, Example 1 shows how skyline queries can be used to identify interesting objects.

Example 1 (Skylines). Consider a basketball coach trying to recruit ideal players excelling in six dimensions d_1, \dots, d_6 , such as the number of games played, rebounds, assists, blocks, steals, and points as illustrated in a toy dataset Table 1. Intuitively, player A is a better choice than B , if A is superior to B in all dimensions, *i.e.*, A *dominates* B . This coach will thus be recruiting players that are not dominated by any other player, or, skyline objects. In Table 1, all data objects *e.g.*, $\{t_1, t_2, t_3, t_4, t_5\}$ tie as skyline objects, while such objects have different trade-offs. For instance, t_3 is superior to t_1 in 5 dimensions except one, *i.e.*, d_6 . In other words, skyline queries generate numerous ties as they do not judge trade-offs among skyline objects.

As Example 1 illustrates, skyline queries do not require users to specify how to judge trade-offs across dimensions. While such property makes it simpler for users to formulate a query, at the same time, by not differentiating skyline objects with varying trade-offs, *i.e.*, treating all as *ties*, the number of skylines is often too numerous to identify truly promising objects with respect to user preference, especially, when the number of dimensions is large, *i.e.*, “curse of

Table 1. Toy dataset for Example 1

	d_1	d_2	d_3	d_4	d_5	d_6
t_1	5	2	1	1	2	5
t_2	3	1	2	2	5	4
t_3	7	3	5	3	4	1
t_4	1	4	4	4	1	3
t_5	2	5	3	6	3	2

dimensionality”. It is thus non-trivial to identify truly interesting objects from many skylines.

Recently, there have been research efforts to address this “curse of dimensionality” problem, focusing on a specific subset of the skylines, in the following two directions: First, references [1,2] propose to find more interesting skylines by identifying skylines for a subspace of total dimensions. For instance, in Table 1, if we compute the skylines of some subspace, say, $\{d_1, d_2, d_3\}$, some tuples are no longer skylines, *e.g.*, t_2 is dominated by t_3 . Based on this observation to identify interesting subsets of skylines, references [1,2] study computation of skylines in varying subspaces— More specifically, reference [1] studies how to amortize the cost of computing skylines over all possible subspaces, while reference [2] studies how to identify *decisive* subspaces for each skyline object. However, these works do not study how to use subspace for generating a result with desirable size. References [3,4] thus study how to rank skyline objects using various ranking criteria based on subspace skylines. However, the proposed ranking criteria cannot adapt to user-specific information needs, *i.e.*, *user-oblivious*.

This paper combines the advantages of the two approaches. First, our framework alleviates users from identifying an appropriate subspace, which we automatically identify for user-specified preference and retrieval size. Second, unlike user-oblivious ranking approach, we adapt to user-specific information needs. Further, the information we obtain from users is highly intuitive, *i.e.*, a qualitative partial ranking over data dimensions. For instance, in Example 1, a coach trying to recruit defense players can simply specify he takes rebounds more seriously than game points, *i.e.*, $rebound > game\ points$. Summing up, our work complements existing skyline works by ranking skylines based on user-specific criteria and retrieval size. In a clear contrast, our work distinguishes itself from rank query processing [5,6,7,8,9] as well, which requires users to specify a complete and quantitative ranking function for all tuples, which is often much harder to formulate than qualitative partial preference used in our framework, as pointed out in [10]. In particular, our proposed Algorithm *Telescope* abstracts skyline ranking problem as a dynamic search problem over skyline subspaces, guided by user-specified qualitative preference. In summary, we believe this paper has the following contributions:

- We propose the framework to rank skyline objects based on user-specific qualitative preference over dimensions to effectively identify truly interesting points.
- We abstract skyline ranking as a dynamic search over skyline subspaces and develop a dynamic search algorithm guided by user-specified preference and retrieval size. Algorithm *Telescope* is provably correct and optimal.
- We implement Algorithm *Telescope* and extensively evaluate the effectiveness and efficiency over both a real-life dataset and synthetic datasets.

This paper is organized as follows. Section 2 briefly reviews existing efforts related to our work. Section 3 discusses preliminaries on ranking skyline objects based on qualitative user preference over dimensions. In particular, Section 4 proposes Algorithm *Telescope* for ranking skyline. Finally, Section 5 validates the effectiveness and efficiency of Algorithm *Telescope*.

2 Related Work

In many applications, skyline queries have been actively studied to efficiently identify promising objects over the dataset having multiple dimensions. Reference [11] was a pioneering work, which devised block nested loop (*BNL*), divide-and-conquer (*D&C*) and B-tree-based algorithms for identifying skyline objects.

Later, reference [12] followed to improve *D&C* algorithm by pruning dominated objects by partitioning the data space using the nearest neighbor object. Similarly, reference [13] developed sort-filter-skyline (*SFS*) algorithm using pre-sorting lists, which improves existing *BNL* algorithm. However, these algorithms focused on efficient computation of skyline objects and did not study the curse of dimensionality problem.

Recently, there have been research efforts to identify interesting subsets of skylines to address this curse of dimensionality problem, in the following directions: First, reference [1] proposes *skycube* structure, adopting the cube structure of OLAP environments, which amortizes the cost of computing skylines over all possible subspaces. Meanwhile, reference [2] studies how to identify *decisive* subspace for each tuple. However, these works do not study how to identify a result with desirable size. Second, references [3,4] study ranking skyline objects by adopting varying interesting metrics. More specifically, reference [3] uses *skyline frequency* metrics that counts the number of subspaces where each tuple belongs to skylines. Further, reference [4] uses *k-dominant skyline* metrics which considers skylines in *k*-dimensional subspace as well. While these rankings help users to identify more interesting skylines by focusing on the top results in the ranking, the ranking criteria are user-oblivious, which can fail to adapt to user-specific needs.

In summary, this paper complements existing works by identifying more interesting subset of skyline objects with user-specified preference and retrieval size. In particular, Algorithm *Telescope* abstracts the problem as a dynamic search

guided by user-specified preference over numerous skylines, which finds truly interesting skylines with correctness and optimality guarantees.

3 Preliminaries

As illustrated in Example 1, a downside of skyline queries is that the size of skyline objects can be large when dataset is high dimensional. While this “curse of dimensionality” problem has been actively studied lately, existing works do not address the challenge of adapting to user preference in identifying truly interesting objects among the skyline objects. To address this challenge, we define the notion of user preference, and its properties.

Table 2. Notations used

Notation	Definition
\mathcal{S}	The dataset
\mathcal{D}	The dimension set
t_i	A tuple in \mathcal{S}
n	The number of dimensions in \mathcal{S}
d_i	A data dimension ($1 \leq i \leq n$)
\mathcal{W}	User’s preference dimensions ($\mathcal{W} \subseteq \mathcal{D}$)
\mathcal{V}	a subset of preference dimensions \mathcal{W} ($\mathcal{V} \subseteq \mathcal{W}$)
m	The number of user preference dimensions in \mathcal{S}
w_i	A user preference dimension ($1 \leq i \leq m$)
$t_i(d_j)$	The value of a tuple t_i on d_j
$SKY(\mathcal{D})$	Skyline on the dimension space \mathcal{D}

We first formally define *dominate* and *skyline* in Definition 1 and Definition 2, respectively, by using notations introduced in Table 2. These definitions are consistent with the definition of skyline queries used in existing works [1,2,3,4,11,12,13].

Definition 1 (Dominate). A tuple t_i dominates another tuple t_j on \mathcal{D} if and only if $\forall d_k \in \mathcal{D}, t_i(d_k) \geq t_j(d_k)$ and $\exists d_s \in \mathcal{D}, t_i(d_s) > t_j(d_s)$ ¹.

Definition 2 (Skyline). A tuple t_i is a skyline object on \mathcal{D} if and only if any other tuples $\forall t_j (\neq t_i) \in \mathcal{S}$ do not dominate t_i on \mathcal{D} . $SKY(\mathcal{D})$ denotes the set of skyline objects on \mathcal{D} in \mathcal{S} .

¹ Note that this definition is based on *max* skyline operator [11], while it can be straightforwardly extended to another operator as well, *i.e.*, *min*, where t_i dominates t_j on \mathcal{D} and only if $\forall d_k \in \mathcal{D}, t_i(d_k) \leq t_j(d_k)$ and $\exists d_s \in \mathcal{D}, t_i(d_s) < t_j(d_s)$.

We then define user preference to be specified by each user. We view user preference as a qualitative ranking of some data dimensions, *i.e.*, *strict partial order* on \mathcal{D} . More formally, we define user preference \mathcal{W} as the *ordered set* of some data dimensions, *i.e.*, $\mathcal{W} = \{w_1, w_2, \dots, w_m\}$ such that $w_1 > w_2 > \dots > w_m$ when w_i is some $d_j \in \mathcal{D}$. We define its semantics as follow: (1) For dimension d_i included in \mathcal{W} , d_i is preferred over any other dimensions not included in \mathcal{W} . (2) Among the dimensions included in \mathcal{W} , preference follows the order of \mathcal{W} . That is, over subspaces having dimensions of same size, the preference is determined *lexicographically*, *e.g.*, $\{w_1, w_2\} > \{w_1, w_3\}$.

Example 2 illustrates this notion intuitively, followed by formal definition of *subset precedence* in Definition 3. Further, based on this subset precedence definition, we rank skyline objects by the precedence of subspace as Theorem 1.

Example 2 (User Preference). Continuing from Example 1, to recruit good guard candidates, the coach considers assists, steals, and game points dimensions are more importantly than the rest of dimensions. Especially, when the coach has preference of assists $>$ steals $>$ points, players with strength in assists and steals will be preferred over those with strength in steals and game points.

Definition 3 (Subset Precedence). *For any subset $\mathcal{V} = \{v_1, v_2, \dots, v_{m'}\}$ of \mathcal{W} , *i.e.*, $\mathcal{V} \subseteq \mathcal{W}$, we define a subspace $\mathcal{V}^i = \mathcal{V} - \{v_{m'-i}\}$ of size $m' - 1$ has higher precedence over any other subspaces $\mathcal{V}^j = \mathcal{V} - \{v_{m'-j}\}$ of the same size, if and only if $i < j$, *i.e.*, $v_{m'-i} < v_{m'-j}$.*

Theorem 1 (Skyline Preference). *For user preference $\mathcal{V} \subseteq \mathcal{W}$, skyline preference follows the order of subset precedence, *i.e.*, $SKY(\mathcal{V}^0) > SKY(\mathcal{V}^1) > \dots > SKY(\mathcal{V}^{m'-1})$ where $\mathcal{V}^i = \mathcal{V} - \{v_{m'-i}\}$. Skyline preference implies (a) already seen skylines outrank the rest to be accessed among unseen skylines, (b) when already seen skylines show again, the rank of skylines follows the subset precedence initially accessed.*

With the notion of preference defined, we make observations on key properties of skyline subspaces. These properties observed play essential roles in showing the correctness and optimality of our framework which essentially implies a dynamic search over the lattice structure of subspaces of \mathcal{W} as illustrated in Fig. 1(left). For simplicity, we first assume that every tuple has a different value for each dimension just for now, which are formally defined as *distinct value assumption*. (We will later relax this assumption.)

Definition 4 (Distinct Value Assumption). *Any two data t_i, t_j in \mathcal{S} are $\forall d_k \in \mathcal{D}, t_i(d_k) \neq t_j(d_k)$.*

With distinct value assumption, previous works have observed that skylines on \mathcal{D} subsume the skylines of its subspace, *i.e.*, *skyline monotonicity* holds, as formally described below.

Theorem 2 (Skyline Monotonicity with Distinct Value Assumption). *Given dataset with distinct value assumption, a tuple $t_i \in SKY(\mathcal{V}')$ is included in $SKY(\mathcal{V})$ where $\mathcal{V}' \subseteq \mathcal{V} \subseteq \mathcal{D}$.*

Proof. See [1].

This monotonicity will be later used to ensure that our framework of exploring skylines subspace lattice (as illustrated in Fig. 1) is correct, *i.e.*, does not return a non-skyline object. However, observe that this monotonicity is conditional to the distinct value assumption, as the example below illustrates.

Example 3 (Example without distinct value assumption). Consider a dataset of 5 objects over 2-dimensions (X, Y) , *i.e.*, $\mathcal{S} = \{a(1, 6), b(3, 6), c(4, 5), d(6, 4), e(6, 2)\}$, which does not satisfy the distinct value assumption. Observe that, while overall skylines on two dimensions $\{X, Y\}$ are $\{b, c, d\}$, skyline on its subspace $\{X\}$ and $\{Y\}$ is $\{d, e\}$ and $\{a, b\}$, respectively. The monotonicity no longer holds as the skylines for $\{X\}$ and $\{Y\}$ are not subsumed by those of $\{X, Y\}$.

While this may seem to compromise the correctness of our framework, we can easily extend the correctness guarantee for any dataset with and without distinct value assumption by replacing the skyline of any subspace \mathcal{V}^i in lattice of Fig. 1 as the intersection with its parent in the lattice, *i.e.*, $SKY(\mathcal{V}^i) \cap SKY(\mathcal{V})$ for its parent \mathcal{V} , as we further discuss in the next section. With this extension, the monotonicity is preserved, as $(SKY(\mathcal{V}^i) \cap SKY(\mathcal{V})) \subseteq SKY(\mathcal{V})$ trivially holds.

4 Algorithm Telescope

In this section, we propose our algorithm that identifies truly interesting skyline objects by adapting to user-specific preference and retrieval size. We name our algorithm *Telescope*, for a telescope helping each user to effectively and efficiently focus on interesting skylines depending on user-specific preference. Toward this goal, Algorithm *Telescope* leverages the preliminaries as discussed in Section 3, to rank skyline objects, which is essentially a dynamic search of $2^m - 1$ subspaces considering m preference dimensions among n dimensions.

To illustrate Algorithm *Telescope*, we use the scenario in Example 2. Suppose that a user specifies his/her preference as $\mathcal{W} = \{w_1 = d_3, w_2 = d_5, w_3 = d_6\}$ and retrieval size $k = 3$ for our toy dataset Table 1. As discussed in Section 3, all subspaces of \mathcal{W} can be represented as the lattice graph in Fig. 1(left). We illustrate how Algorithm *Telescope* works in Fig. 1(left): First, we compare the number of skylines of \mathcal{W} , *i.e.*, $|SKY(\mathcal{W})|$, with retrieval size k . Second, we consider the subspace having the highest precedence, *i.e.*, $\mathcal{V}^0 = \{w_1, w_2\}$ and insert its skylines, *i.e.*, $SKY(\mathcal{V}^0) = \{t_2, t_3\}$, into desirable results, and move on to its right sibling, *i.e.*, subspace $\mathcal{V}^1 = \{w_1, w_3\}$, by the order of subset precedence. Third, the number of skylines in \mathcal{V}^0 and \mathcal{V}^1 exceeds the retrieval size k *i.e.*, $|SKY(\mathcal{V}^0) \cup SKY(\mathcal{V}^1)| > k$. Fourth, we zoom into the subspaces of \mathcal{V}^1 , or $\{w_1\}$ and $\{w_3\}$, to identify more desirable k results among them, *e.g.*, $\{t1, t2, t3\}$ by adding $\{t1\}$ in $\{w3\}$.

Further, we can transform this lattice graph into a *left-skewed graph* by pruning multiple links to common descendants, as we will formally state in Definition 5. To illustrate, consider adjacent subspaces, *e.g.*, $\{w_1, w_2\}$ and $\{w_1, w_3\}$, sharing

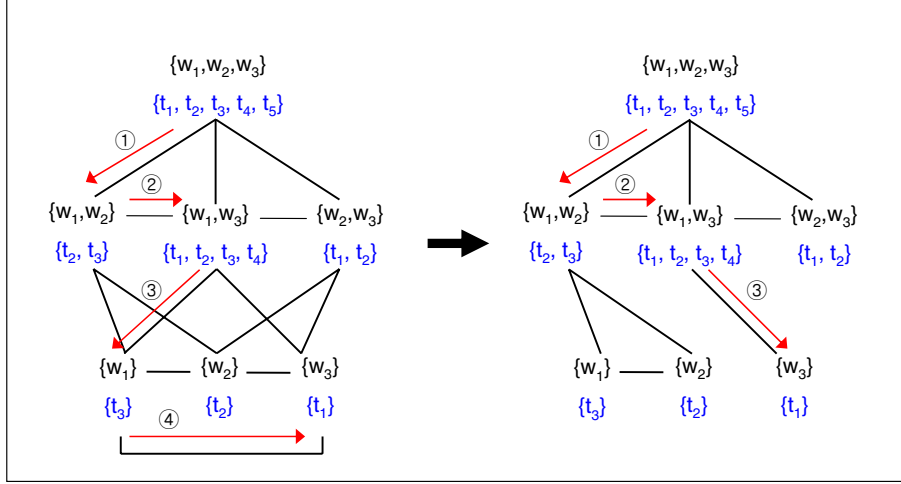


Fig. 1. The lattice graph and left-skewed graph

a common descendant, *e.g.*, $\{w_1\} \subseteq \{w_1, w_2\} \cap \{w_1, w_3\}$. As there are multiple links to $\{w_1\}$ from both $\{w_1, w_2\}$ and $\{w_1, w_3\}$, we keep only the link from the ancestor of higher precedence, *i.e.*, $\{w_1, w_2\}$. With such transformations for every adjacent nodes, we can eliminate multiple links to common descendants by keeping only the link from the highest precedence parent, as Fig. 1(right) illustrates. With this graph transformation, Algorithm *Telescope* guarantees to visit nodes in the descending order of precedence (which will be later used for correctness proof), and at the same time, guarantees not to visit any node twice in traversal (which will be later used for optimality proof).

Definition 5 (Graph transformation). *The lattice graph can be transformed into a left-skewed graph, by eliminating a link from adjacent nodes \mathcal{V}^i and \mathcal{V}^j to its common descendant \mathcal{V}^k by keeping only a single link to the common descendant, in particular, the one from the node with the highest precedence.*

By leveraging the left-skewed graph, Fig. 2 shows the pseudo-code of Algorithm *Telescope*. We describe how Algorithm *Telescope* works for the user-specified preference \mathcal{W} and retrieval size k as followed:

1. Compare the number of skylines for the root node, *i.e.*, $|SKY(\mathcal{W})|$, with the retrieval size k .
2. When $|SKY(\mathcal{W})| > k$, push all subspaces \mathcal{V} of \mathcal{W} with $m - 1$ dimensions in \mathcal{W} into the stack (except for already seen common descendants). The precedence of subspaces will be preserved if pushed from right to left in Fig. 1(right), *i.e.*, low precedence to high, as pushing and popping from the stack will reverse the order.
3. Pop a subspace \mathcal{V} from the stack, and decide whether to insert its skylines $SKY(\mathcal{V})$ into the results \mathcal{Z} as follow:

- If $|\mathcal{Z} \cup SKY(\mathcal{V})| > k$, go to step 2 to insert its subspaces to the stack, *i.e.*, move on the leftmost child of current node.
- If $|\mathcal{Z} \cup SKY(\mathcal{V})| \leq k$, insert new skyline points $SKY(\mathcal{V})$ into \mathcal{Z} . If $|\mathcal{Z}| < k$ still holds, go to step 3, *i.e.*, move on the right sibling of current node. Otherwise terminate.

Note that, for all child nodes, the monotonicity of skyline (Theorem 2) assures that every data object in the lattice is a part of skylines of \mathcal{W} . Further, when accessing siblings in the order of subset precedence, skyline preference theorem (Theorem 1) ensures the already seen objects outrank the rest to be accessed among unseen objects, *i.e.*, the rank of skylines is decided by the subset precedence initially accessed. Putting together, any search over lattice with two modes of access— (a) to child node with the highest precedence, *i.e.*, leftmost child node, and (b) to sibling node in the decreasing order of precedence, is correct, as we formally state below.

Algorithm Telescope($\mathcal{S}, \mathcal{W}, k$)

Input

- \mathcal{S} : dataset
- $\mathcal{W} : \{w_1, w_2, \dots, w_m\}$
- k : retrieval size

Output

- \mathcal{Z} : skylines with respect to k

Procedure

- $\mathcal{T}, \mathcal{U}, \mathcal{V}$ // Stack, superset of current set, and current set
- $\mathcal{T} \leftarrow \{\}, \mathcal{Z} \leftarrow \{\}$ // Initialize the stack and results.
- **if** $|SKY(\mathcal{W})| > k$ **then**
 - $\mathcal{T}.push(\mathcal{W})$ // Push \mathcal{W} into the stack.
 - **while** (\mathcal{T} is not empty **and** $|\mathcal{Z}| < k$)
 - $\mathcal{V} \leftarrow \mathcal{T}.pop()$
 - // Traverse subsets of \mathcal{V} if exceed the retrieval size k .
 - **if** $|\mathcal{Z} \cup SKY(\mathcal{V})| > k$ **then**
 - $\mathcal{U} \leftarrow \mathcal{V}$ // Keep track of the superset of the current set.
 - // Insert subsets into the stack except for shared subsets.
 - **for** $i := 0$ **to** $m' - 1$:
 - $\mathcal{T}.push(\mathcal{V}^i)$ // Push subsets \mathcal{V}^i except the common descendants.
 - **else** // Insert skylines into \mathcal{Z} , and move on next precedence subset.
 - $\mathcal{Z}.insert(SKY(\mathcal{V}))$
 - // Move on superset \mathcal{U} of the last subset \mathcal{V} having no child.
 - **if** \mathcal{V} is the last subset having no child of \mathcal{U} **then**
 - Call some deterministic **tie breaker** *e.g.*, object ID.
 - **else if** $|SKY(\mathcal{W})| < k$ **then**
 - $\mathcal{Z}.insert(SKY(\mathcal{W}))$
 - $Telescope(\mathcal{S} - \mathcal{Z}, \mathcal{W}, k - |\mathcal{Z}|)$
 - **else**
 - Terminate.

Fig. 2. Algorithm *Telescope*

Theorem 3 (Correctness of Algorithm Telescope). *For user preference \mathcal{W} and retrieval size k , Algorithm Telescope returns correct k results such that, each object in the output queue is (a) a part of skylines and (b) outranks the objects yet to be retrieved, at any point during execution.*

Proof. Immediate from Theorems 1 and 2, every node in the lattice contains only the skyline objects. Further, after the graph transformation in Definition 5, Algorithm Telescope ensures no unseen object, yet to be accessed, outranks the skyline objects found in prior by design (as discussed before).

Theorem 4 (Optimality of Algorithm Telescope). *After the transformation in Definition 5, Algorithm Telescope only visits the node that is absolutely necessary to identify the top- k skyline objects, for user-specified preference \mathcal{W} and retrieval size k . The worst number visited of Algorithm Telescope is $O(m)$ which is minimal traversal to obtain the top- k skylines.*

Due to the space limitation, we leave the proof to our extended report [14] and only report the proof sketch for this theorem. As discussed for the correctness proof, Algorithm Telescope ensures that no unseen skyline outranks the ones accessed in prior. Further, using proof by contradiction, we can claim Algorithm Telescope terminates after visiting only the absolutely necessary nodes for correctness, and nothing else. Summing up, Algorithm Telescope is provably correct and optimal for the user-specified preference \mathcal{W} and retrieval size k .

5 Experiments

This section reports our experimental results to validate the effectiveness and efficiency over our Algorithm Telescope. First, to validate effectiveness using real-life data, Section 5.1 reports our evaluations over real-life NBA player statistics. Second, to validate efficiency in extensive problem settings, Section 5.2 reports our evaluations over synthetic data of varying problem settings. Our experiments was carried out on a Intel(R) Xeon(TM) machine with 3.20 GHz dual processors and 1GB RAM running LINUX. Algorithm Telescope was implemented in C++ language.

5.1 Real-Life Data Set

In this section, we validate the effectiveness of Algorithm Telescope by the quality of the skyline objects retrieved from the real-life data. In particular, we use NBA dataset (available from www.nba.com), resulting in 19112 players with 16 numeric attributes including game points, number of rebounds, assists, steals, and blocks. We evaluate with a scenario of identifying $k = 10$ skyline objects over user-specified preferences on three dimensions, which is a useful query for a basketball coach in our illustrative example in Example 2– For instance, when the coach needs to recruit a guard, he may need to focus more on the number of assists or steals, than the number of rebounds or blocks. Similarly, when the coach recruits a center, he may focus on the skylines over the number of rebounds or blocks.

Table 3. Skylines for guard and center positions

Preference: Assists > Steals > Points		Preference: Rebounds > Blocks > Points	
Position	Player	Position	Player
C	Wilt Chamberlain 1961	C	Wilt Chamberlain 1960
C	Wilt Chamberlain 1962	C	Wilt Chamberlain 1961
C	Wilt Chamberlain 1963	C	Artis Gilmore 1971
G	Nate Archibald 1972	C	Artis Gilmore 1973
G	Don Buse 1975	C	Bob Mcadoo 1974
G	Michealray Richardson 1979	C	Kareem Abdul-jabbar 1975
G	John Stockton 1987	C	Mark Eaton 1984
G	John Stockton 1988	G	Michael Jordan 1986
G	John Stockton 1990	G	Michael Jordan 1987
G	John Stockton 1991	C	Patrick Ewing 1989

We implement the above two scenarios of recruiting good guard candidates (with qualitative preference of assists > steals > points) and good center candidates (with qualitative preference of rebounds > blocks > points) and report the results in Table 3. In contrast to existing skyline ranking, which is user-oblivious, our skyline results effectively adapt to the user-specific needs and identify ideal candidates. Observe from Table 3 that our top-10 results correctly identify legendary NBA guards and centers respectively, except all-round players such as Wilt Chamberlain or Michael Jordan who happen to play the roles of both a guard and a center.

5.2 Synthetic Data Set

With the effectiveness of our Algorithm *Telescope* illustrated in Section 5.1, this section discusses the efficiency of our Algorithm *Telescope* over synthetic data of varying problem settings, such as the user preference dimensionality m , data size $|\mathcal{S}|$, and retrieval size k .

More specifically, we randomly generate synthetic data with and without correlations, which is a deciding factor for the number of skylines. Intuitively, if data dimensions are anti-correlated, the number of skylines explodes, as even a tuple dominated in some dimension A is likely to be superior in another dimension B , anti-correlated to A . We thus generate synthetic data using uniform random generator for all dimensions (for **Independent** dataset). In addition, for correlated data, we synthetically introduce correlation by randomly generating $t_i(d_1)$ and generating the rest, *i.e.*, $t_i(d_j)$ for $j > 1$ within the range of $t_i(d_{j-1}) \pm \alpha$ (for **Correlated** dataset) and $-t_i(d_{j-1}) \pm \alpha$ (for **Anti-correlated** dataset). In Fig. 3(a), (b), and (c) report the response time of Algorithm *Telescope* for the above three datasets, over varying dimensionality m , data size $|\mathcal{S}|$, and retrieval size k respectively. Besides, Fig. 3(d) reports the number of the visited nodes in the lattice graph and the left-skewed graph with the datasets.

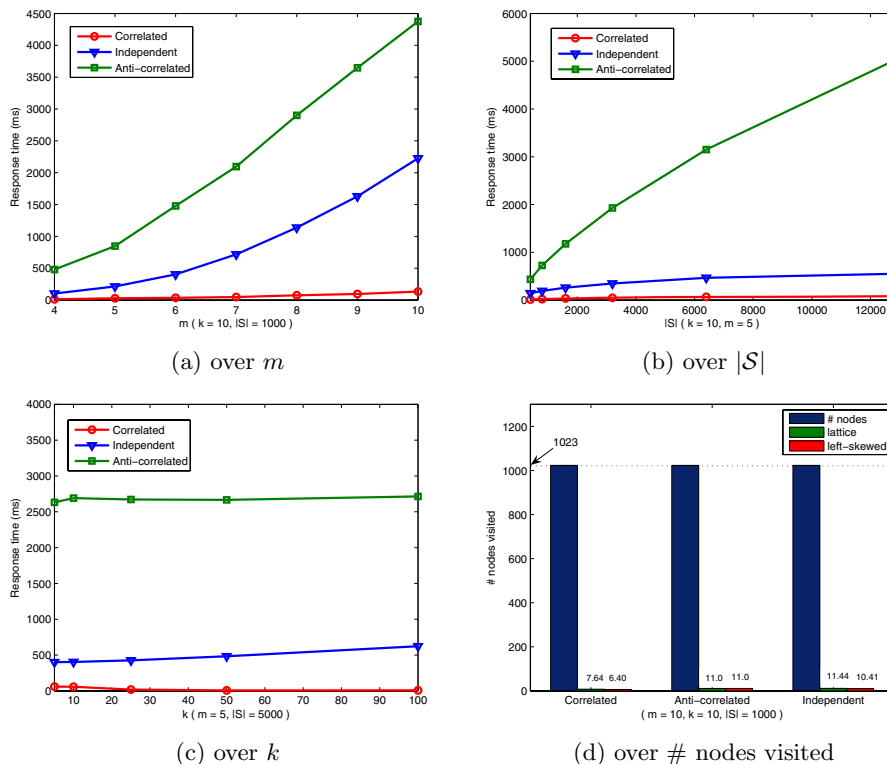


Fig. 3. Response time / The number of nodes visited

Fig. 3(a) reports performance results over varying dimensionality m . Observe that, Algorithm *Telescope* by only visiting the provably minimal number of nodes in the left-skewed graph, scales gracefully over m in all datasets, while the number of nodes to visit explodes exponentially over m , *i.e.*, $O(2^m)$. The performance degradation of anti-correlated dataset can be best explained by the explosion of the number of skylines.

We similarly report our scalability results over cardinality and retrieval size in Fig. 3(b) and (c) respectively. Observe that, in all settings, Algorithm *Telescope* ensures high scalability and low response time *e.g.*, less than 0.6×10^{-2} second in all evaluations. In addition, Fig. 3(d) reports the efficiency of pruning in Algorithm *Telescope*. Observe that Algorithm *Telescope*, by using the lattice graph in Fig. 1(left), only visits approximately 1 % of all nodes. Further, by transforming the lattice graph into the left-skewed graph in Fig. 1(right), we can further reduce the number of nodes visited by 17 %.

6 Conclusion

This paper studies how to alleviate the curse of dimensionality problem in skyline queries. More specifically, Algorithm *Telescope* zooms into truly interesting

skyline objects, guided by user-specified qualitative preference and retrieval size, which complements existing works that either require users to formulate a rather cumbersome query or do not adapt to user-specific information needs. In particular, Algorithm *Telescope* abstracts skyline ranking as a dynamic search over skyline subspaces to efficiently and effectively identify truly interesting objects for a specific user. Our extensive performance study validates both the effectiveness and efficiency of Algorithm *Telescope* on real-life and synthetic data.

References

1. Yidong Yuan, Xuemin Lin, Qing Liu, Wei Wang, Jeffery Xu Yu, and Qing Zhang. Efficient computation of the skyline cube. In *VLDB 2005*, 2005.
2. Jian Pei, Wen Jin, Martin Ester, and Yufei Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB 2005*, 2005.
3. Chee-Yong Chan, H.V. Jagadish, Anthony K.H. Tung Kian-Lee Tan, and Zhenjie Zhang. On high dimensional skylines. In *EDBT 2006*, 2006.
4. Chee-Yong Chan, H.V. Jagadish, Kian-Lee Tan, Anthony K.H. Tung, and Zhenjie Zhang. Finding k-dominant skyline in high dimensional space. In *SIGMOD 2006*, 2006.
5. Ronald Fagin. Combining fuzzy information from multiple systems. In *PODS 1996*, pages 216–226, 1996.
6. Ronald Fagin, Amnon Lote, and Moni Naor. Optimal aggregation algorithms for middleware. In *PODS 2001*, 2001.
7. Nicolas Bruno, Luis Gravano, and Amelie Marian. Evaluating top-k queries over web-accessible databases. In *ICDE 2002*, 2002.
8. Kevin C. Chang and Seung-won Hwang. Minimal probing: Supporting expensive predicates for top-k queries. In *SIGMOD 2002*, pages 346–357, 2002.
9. Seung-won Hwang and Kevin C. Chang. Optimizing access cost for top-k queries over web sources. In *ICDE 2005*, 2005.
10. Hwanjo Yu, Seung won Hwang, and Kevin Chen-Chuan Chang. RankFP: A framework for supporting rank formulation and processing. In *ICDE 2005*, 2005.
11. Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE 2001*, 2001.
12. Donald Kossmann. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB 2002*, 2002.
13. Jan Chomicki, Parke Godfery, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *ICDE 2003*, 2003.
14. Jongwuk Lee, Gae-won You, and Seung-won Hwang. Telescope: Zooming to interesting skylines. In *POSTECH Technical Report*, 2006.