Finding Skylines for Incomplete Data

Rahul Bharuka **P Sreenivasa Kumar**

Department of Computer Science & Engineering
Indian Institute of Technology Madras
Chennai, India
{rahulb, psk}@cse.iitm.ac.in

Abstract

In the last decade, skyline queries have been extensively studied for different domains because of their wide applications in multi-criteria decision making and search space pruning. A skyline query returns all the *interesting* points in a multi-dimensional data set that are not dominated by any other point with respect to all dimensions. However, real world data sets are seldom complete, i.e. data points often have missing values in one or more dimensions. Traditional skyline query processing algorithms developed for complete data can not be easily adapted for such situations because of the non-transitive and potentially cyclic nature of dominance relation that arises in the case of incomplete data. Unfortunately, skyline query processing for such incomplete data has not received enough attention. We propose an efficient *Sort-based Incomplete Data Skyline* (SIDS) algorithm to compute the skyline points over incomplete data. Extensive experiments on both real world and synthetic data sets demonstrate the efficiency and scalability of our approach over current state of the art approach.

Keywords: Skyline query processing, incomplete data, missing data.

1 Introduction

Preference queries return answers that are relevant to user requirements. A subclass of preference query, called *skyline* query, that retrieves pareto-optimal points from a data set, was first proposed by Börzsönyi et al. (2001). Given a set of points in a d-dimensional space S, a point p is said to dominate another point q, if p is better than or equal to q in all dimensions and p is strictly better than q in some dimension s_i of S. Those points that are not dominated by any other point in the data set are called *skyline points* and the set of skyline points is known as the *skyline*. Skyline is useful in several applications, such as those requiring personalization, multi-criteria decision making, or search space pruning.

For example, consider a customer who wants to rent a flat in a city and is browsing a real estate website for the same. Each flat has several features such as area, rent, number of bedrooms, location, distance from the office. It would be difficult and time consuming for the customer to make a choice, if all the flats in the database which are up for rent, are presented to her. Flats which are not worse than any other flat with respect to all features, are of *interest* to the customer. Firing the skyline query for such scenario will eliminate unwanted flats and would present a manageable set of interesting flats to the user.

Movie rating site (e.g. MovieLens[1]) is another example where computing skylines could be useful. MovieLens captures, among other things, movie ratings that are given by the users. Now, consider a person interested in highly-rated movies. In such a case, if each user is treated as a dimension then a movie will be a point in the d-dimensional space, where d is the number of users. Computing the skyline will prune all low-rated movies in the data set and return the top-rated movies as the answer.

Owing to its wide range of applications, skyline query has been extensively studied in the past. With the exception of Khalefa et al. (2008) and Zhang et al. (2010), all skyline query processing approaches assume completeness of data, i.e. all dimension values are present for all the data points. But this assumption may not hold in many real life scenarios. For example, in the MovieLens data set, it is obvious that every movie may not be rated by all users, resulting in incomplete dimensions for a movie. In such a case, comparing a pair of movies becomes difficult when they are rated by different sets of users. *Transitivity* of dominance relation, which is the basis of all traditional skyline query processing algorithms does not hold for such incomplete data sets. For example, we may have a situation where data point p dominates q, q dominates r, but p does not dominate r. Instead, point r may dominate p. Under this *cyclic* dominance relation, none of the three points belong to the skyline, as each point is dominated by at least one other point. Consequently, a dominated point can not be discarded immediately, as it could potentially dominate certain points occurring later in the data set. Hence, due to the presence of non-transitive and cyclic nature of dominance relation, the traditional skyline query processing algorithms in their current form can not be applied for incomplete data.

The naïve approach, for computing the skyline over incomplete data, would be to perform exhaustive pairwise dominance checks for all input points and discard the dominated ones. However, this method becomes inefficient with increase in the size or dimension of a data set. Khalefa et al. (2008) were the first to formally address the problem of skyline query processing over incomplete data, by proposing an algorithm called *ISkyline*. The algorithm reduces the number of comparisons required for the skyline computation by using virtual points and shadow skylines, implemented using the *Bucket* data structure. How-

[1] http://movielens.umn.edu

ever, the use of Bucket data structure and bitmap representation for determining comparable Buckets slows down the algorithm. Moreover, memory utilization of ISkyline is quite high when each Bucket has a large number of comparable Buckets, which is very common in real world data sets.

Zhang et al. (2010) addressed the problem of incomplete data by converting an incomplete point p to a complete point p'. Their approach plugs-in the value of missing dimensions i using a probability distribution function formed by non-missing values on dimension i. They have also proposed extensions to existing skyline query processing algorithms for computing skyline over the modified data set. However, such substitution for missing values may not be advisable in certain situations like recommender systems. For example, in a cell phone database, all handsets may not have features like *touch-screen*.

We adopt the definition of dominance for incomplete data from Khalefa et al. (2008) and consider the case where missing values are not guessed. Our approach assumes presorted data, based on dimension values. The contributions of this paper are as follows:

- We propose a new skyline processing algorithm, *Sort-based Incomplete Data Skyline* (SIDS), for incomplete data.
- We provide a proof of correctness for the SIDS algorithm.
- We demonstrate efficiency of our approach over ISkyline algorithm using detailed experiments.

The rest of the paper is organized as follows: preliminary discussion and problem definition are presented in Section 2. Section 3 surveys the related work. In Section 4, we give a details discussion of the SIDS algorithm. The experimental results are discussed in Section 5 and Section 6 concludes this paper.

2 Preliminaries

Throughout this paper, without loss of generality, we assume that greater values are preferred over the smaller ones. We also use '-' to represent missing value for a dimension. Dimensions with known values for a data point are referred to as *complete dimensions*, whereas dimensions with missing values for a data point are referred to as *incomplete dimensions*.

2.1 Problem Definition

Given a d-dimensional data set D, where each point has at least one known dimension value and other dimension values may or may not be known, find the set of points in D which are not dominated by any other point in D.

2.2 Dominance

In a d-dimensional space, S, with dimensions $M = \{s_1, s_2, ..., s_d\}$, a point p is said to dominate another point q, if $\forall s_i \in M$, $p.s_i \geq q.s_i$ and $\exists s_j \in M$, $p.s_j > q.s_j$, where $p.s_i$ denotes the i^{th} dimension value of point p.

In case of incomplete data, the dominance relation may become non-transitive and cyclic. To handle these inconsistencies, Khalefa et al. (2008) proposed a new definition of dominance, which is discussed in the following section.

Point	u_1	u_2	u_3	u_4	u_5
m_1	2	-	4	2	-
m_2	-	3	4	4	-
m_3	-	-	1	-	4
m_4	3	-	-	3	3
m_5	4	-	2	5	4
m_6	-	2	-	-	-
m_7	2	4	2	4	3

Figure 1: Sample movie-rating data set

2.3 Dominance for Incomplete Data

Consider two points p and q over a d-dimensional space. Let M', where $|M'| \leq d$, be the set of complete dimensions, common for points p and q. Then, point p is said to dominate q if, $\forall s_i \in M'$, $p.s_i \geq q.s_i$ and $\exists s_j \in M'$, $p.s_j > q.s_j$.

In other words, we apply traditional dominance relation across common dimensions for points where both values are known. For example, Figure 1 shows a sample movie-rating data set, storing ratings for seven movies given by five users. Each entry r_{ij} in the movie-user matrix corresponds to the rating given to movie i by user j. The sample data is 20% incomplete. Consider points m_1 and m_2, where u_3 and u_4 are the only dimensions across which values are known for both points. Thus, we only consider u_3 and u_4 dimensions for determining the dominant point. Here, m_2 dominates m_1, because m_2 and m_1 have same value for dimension u_3 but m_2 has larger value than m_1 for dimension u_4. As mentioned earlier, non-transitivity and cyclic-dominance may arise in case of incomplete data. For example, consider points m_1, m_3 and m_4. Here, point m_1 dominates m_3 in third dimension, while point m_3 dominates m_4 in fifth dimension. However, point m_1 does not dominate m_4. Thus, dominance relation is non-transitive. Moreover, point m_4 dominates m_1 which makes the dominance relation cyclic. In such a case, none of these three points is considered as a skyline point, because each point is dominated by at least one other point.

3 Related Work

Börzsönyi et al. (2001) first introduced the Skyline operator and proposed BNL, D&C and an algorithm using B-tree for skyline evaluation. Among these, BNL and D&C algorithms do not assume any preprocessing on input data. Since then, various skyline query processing algorithms, using different approaches, have been proposed by the research community. For example, Kossmann et al. (2002), Papadias et al. (2005) and Tan et al. (2001) made use of index structures to progressively report skylines. SFS and SaLSa algorithms proposed by Chomicki et al. (2003) and Bartolini et al. (2006) respectively, sort input data using monotone sorting function. While SFS uses presorted data to put most dominant points first in the window and there by reducing the number of passes required to compute the skyline, SaLSa uses presorting to terminate the search for additional skyline points early. In contrast, our approach sorts data in non-ascending order for each dimension and stores such d sorted lists, to determine processing order of points.

Skyline queries have been proposed for domains such as partially-ordered [Chan et al. (2005)], spatial [Sharifzadeh & Shahabi (2006)] and distributed databases [Balke et al. (2004)]. Researchers have

also proposed reverse skyline [Dellis & Seeger (2007)] and probabilistic skyline [Pei et al. (2007)] queries. Hose & Vlachou (2012) have presented a comprehensive survey of skyline query processing in highly distributed environments. Unfortunately, all of the above mentioned skyline evaluation approaches were proposed for complete data and can not be easily adapted for incomplete data.

Different variants of traditional skyline query have also been proposed over the years like constrained, subspace and dynamic skyline queries. The skyline for the input data set depends on the definition of dominance used by the algorithm. The traditional dominance relation is rather strict, resulting in large size of the skyline. Hence, to restrict the skyline size, a variant of traditional skyline, called *k-dominant* skylines was proposed by Chan et al. (2006). In a k-dominance relation, a point p is said to k-dominate another point q if, p is better than or equal to q in any subset of dimensions with size k and p is strictly better than q in one of these k dimensions. Similar to the case of incomplete data, non-transitive and cyclic dominance may arise in a k-dominance relation. Hence, algorithms proposed for finding k-dominant skylines can be adapted for computing skylines over incomplete data.

Our approach for evaluating skyline over incomplete data is based on *Sorted Retrieval Algorithm* (SRA) proposed by Chan et al. (2006) for k-dominant skylines. SRA algorithm is inspired by rank aggregation algorithms proposed by Fagin et al. (2001) to find *top-k* objects under some monotone aggregation function. In SRA algorithm, k is a constant and is specified by the user to find the k-dominant skyline over a complete data set. Whereas in our approach, k is the count of complete dimensions for a point in an incomplete data set and it varies for different points. Unlike SRA algorithm which iteratively processes the batch of data points in a sorted array that have same next best value, our approach iteratively choses the next data point from a sorted array for processing. When comparing a pair of points a and b, SRA algorithm only checks whether a dominate b, while SIDS algorithm simultaneously also checks whether b dominates a.

The two approaches, specifically proposed for the skyline computation over incomplete data till date, are summarized in the following sections:

3.1 ISkyline Algorithm

Khalefa et al. (2008) proposed a new definition of dominance for incomplete data and introduced *ISkyline* algorithm for computing the skyline. ISkyline uses *Bucket* data structure and bitmap representation for determining pairs of comparable points. The bitmap vector for a point represents each missing dimension by 0's and complete dimension by 1's. All points belonging to the same Bucket have same bitmap representation and thereby the same set of complete dimensions. Thus, transitivity of dominance relation holds for points belonging to the same Buckets. ISkyline also utilizes two concepts viz. *shadow skylines* and *virtual points*, to reduce the number of comparisons required for computing the skyline.

A virtual point is formed by considering only the common dimensions in the bitmaps of a point and a Bucket having comparable but not equal bitmap. Virtual points reduce the number of points advancing to *candidate_skyline* list, by exploiting information about points in other Buckets to prune points in *local_skyline* list of a Bucket. ISkyline maintains *local_skyline* list and *shadow_skyline* list for each Bucket. The skyline for points within each Bucket is called *local_skyline*. *global_skyline* is the list of current skyline points for the data set. *shadow_skyline* is the list of *local_skyline* points for each Bucket which are not in *global_skyline*. The algorithm also uses $updated\_flag$ to keep track of new insertions to the *shadow_skyline* list of a Bucket.

ISkyline employs a three phase approach for determining the skyline set. Each phase filters out some of the non-skyline points before passing it to next. Phase I determines whether a new point should be stored in *local_skyline*, *shadow_skyline* or can be safely pruned. Only real *local_skyline* points are propagated to the next phase.

In phase II, a new *local_skyline* point p is compared with all comparable points q in *candidate_skyline* list. If p dominates q, then q is deleted from the *candidate_skyline* list and a virtual point is inserted into point q's *local_skyline* list. Similarly, if q dominates p, point p is simply discarded and a virtual point is inserted into point p's *local_skyline* list. If p is not dominated by any of the *candidate_skyline* points then it is added to *candidate_skyline* list. ISkyline algorithm uses a tuning parameter t that controls frequency of updating *global_skyline*. When the number of points in *candidate_skyline* list reaches a minimum threshold t, phase III starts.

In phase III, each point in *global_skyline* list is validated against incoming *candidate_skyline* points and dominated points from both lists are moved to *shadow_skyline* of corresponding Buckets. Remaining points in *global_skyline* list are then compared against *shadow_skyline* list of Buckets with comparable but not equal bitmap and $True$ $updated\_flag$. If any one of the *shadow_skyline* point dominates *global_skyline* point p, then p is immediately deleted from *global_skyline* list and added to *shadow_skyline* list of its Bucket. Similarly, all remaining points in *candidate_skyline* list are compared against comparable but not equal bitmap Buckets irrespective of the $updated\_flag$. If any one of the *shadow_skyline* point dominates *candidate_skyline* point q, then q is immediately deleted from the *candidate_skyline* list and added to *shadow_skyline* list of its Bucket. Then, *candidate_skyline* and *global_skyline* lists are merged to create an updated *global_skyline* list. Finally, *candidate_skyline* list is cleared and all $updated\_flag$ are set to $False$. After the end of input, *global_skyline* list is once again updated to produce the final skyline set.

Though ISkyline algorithm reduces the number of comparisons required for skyline computation, it's performance is highly susceptible to the relative ordering of input data. This is because it does not assume any preprocessing. The comparison count also changes with tuning parameter t, used for refreshing the results. On the other hand, our proposed approach necessarily sorts data on individual dimensions and hence the performance of our algorithm is stable. Also, memory utilization of ISkyline algorithm is quite high, as for each Bucket, pointers to comparable Buckets are stored. Use of bitmaps and virtual points further adds to the memory usage of the algorithm. Moreover, the overhead of bitmap storage and computation slows down the algorithm. One more shortcoming of ISkyline algorithm is that, it can not progressively report the skyline points, i.e. final skyline points can only be confirmed after processing the last data point. In contrast, our algorithm can immediately returns a skyline point if it has been *processed* sufficient number of times.

3.2 Using Generalized Framework Approach

Zhang et al. (2010) proposed a generalized framework to guide extension of conventional skylines to different variants and extended their analysis to support data sets with incomplete data. Their approach defines a probability distribution function (pdf) of the values across each dimension i, by observing all the non-missing values for that dimension. A generic mapping is then defined, to convert an incomplete point p to a complete point p', by retaining all non-missing values and estimating the concrete value for each missing values, based on the known pdf on that dimension. Consequently, a mapping dominance (MD) relationship is defined to facilitate comparison between a pair of points.

In essence, this approach plugs-in the estimated values for missing values in a data set. The authors then proposed extensions over existing algorithms to handle incomplete data. However, we note that such an approach of plugging-in values for missing values is only viable when the incompleteness ratio of data set in each dimension is small. Using this approach for sparse data sets may produce undesirable results. Such approach is also not advisable in many practical situations where trust and transparency of results matter, like recommender systems and in case of sensitive domains where user may tolerate false negatives but can not afford false positives. Thus, in our approach, we assume that missing values for dimensions are not to be guessed and hence we do not compare this approach with ours'.

4 Proposed Approach

This section describes our approach and introduces an efficient *Sort-based Incomplete Data Skyline* (SIDS) algorithm for evaluating the skyline over incomplete data, based on the definition given in Section 2.3. The input data set is presorted in non-increasing order for each dimension, to determine processing order of points. The proposed approach chooses one of the dimensions in round-robin fashion and then, the point having the next best value in that dimension is chosen for processing. The intuition behind this is to process relatively dominant points early, so that non-skyline points can be pruned as early as possible. Consequently, less number of comparisons would be required for determining the skyline and this would in turn reduce the execution time of the algorithm. The algorithm initially considers all points in the data set as candidate skyline points and then iteratively removes dominated points from the candidate set. If a point has not been pruned yet and has been processed k times, where k is the count of complete dimensions for the point, then it is determined to be a skyline point and can be returned immediately. The reason is, any point with k complete dimensions can be dominated in at most k dimensions. The detailed proof of correctness for this, is given in Subsection 4.2. Thus, our algorithm can progressively return skyline points whenever the above condition is satisfied. The pseudo-code of our approach is given in Algorithm 1 and is explained below.

Algorithm 1: Sort-based Incomplete Data Skyline

```
   Input  : d-dimensional incomplete data set D
   Output: ResultSet, the skyline for data set D
 1 foreach dimension s_i ∈ M do
 2 |   sort D in non-ascending order of each point's s_i
   |   value and store pointers for the sorted points in
   |   array D_i[1...|D|]
 3 |   initialize the array pointer for D_i, ptr_i=0
 4 end
 5 foreach p ∈ D do
 6 |   initialize processedCount(p) = 0
 7 |   initialize dimCount(p) =number of complete
   |   dimensions in p
 8 end
 9 initialize CandidateSet = D, ResultSet = φ and
   iter = 0
10 while CandidateSet ≠ φ and atleast one point is yet to
   be processed do
11 |   nextDim = iter % |M|
12 |   p = D_nextDim[ptr_nextDim]
13 |   isDominated = False
14 |   if processedCount(p)=0 then
15 |   |   foreach c ∈ CandidateSet do
16 |   |   |   if p and c have not been compared before
   |   |   |   then
17 |   |   |   |   if p dominates c then
18 |   |   |   |   |   remove c from CandidateSet
19 |   |   |   |   else if c dominates p then
20 |   |   |   |   |   isDominated = True
21 |   |   |   |   end
22 |   |   |   end
23 |   |   end
24 |   |   if p ∈ CandidateSet and isDominated then
25 |   |   |   remove p from CandidateSet
26 |   |   end
27 |   end
28 |   processedCount(p) = processedCount(p) + 1
29 |   if p ∈ CandidateSet and
   |   processedCount(p)=dimCount(p) then
30 |   |   move p from CandidateSet to ResultSet
31 |   end
32 |   ptr_nextDim = ptr_nextDim + 1
33 |   iter = iter + 1
34 end
35 return ResultSet
```

4.1 SIDS Algorithm

The algorithm takes as input a d-dimensional incomplete data set D and outputs *ResultSet* - the skyline set for D. In steps 1-2, data points in D are sorted in non-ascending order for each dimension $s_i \in M$ and such d sorted sets are stored in array $D_i[1...|D|]$. Thus, for each sorted array $D_i, \forall i \in [1, |M|]$, we have monotonously descending array such that $D_i[j] \geq D_i[j+1], \forall j \in [1, |D|)$. Step 3 initializes an array pointer ptr_i, which points to the data point to be processed next from sorted array D_i. It is important to note that, data points with missing value in dimension s_i are not stored in the sorted array D_i. Also note that, to save space, only ids of the actual data points are stored in each array D_i. Steps 5-8 initialize two variables for each point p in D: (1) *processedCount*, which keeps track of number of times a point has been processed from the sorted arrays and (2) *dimCount*, which stores the number of complete dimensions the point p has. We maintain two sets viz. *CandidateSet* and *ResultSet*. Step 9 initializes *CandidateSet*, which stores candidate skyline points, to data set D. *ResultSet* stores points which are determined to be the final skyline points and is initialized to ϕ. In step 10, variable *iter* is initialized and it determines the sorted array to be considered next.

Sorted arrays D_i are processed in round-robin fashion and the next dimension is determined in step 11. If any array pointer ptr_i reaches end of array, we simply continue with the next dimension until *CandidateSet* becomes empty or all points have been processed at least once, whichever is the earlier. In step 12, the point p to be processed next is retrieved.

Step 13 initializes *isDominated* flag for point p to $False$. If $processedCount(p) = 0$, i.e. point p has not been processed before, then only, it is compared with points c in $CandidateSet$ (steps 15-23). Points p and c are compared only if they have not been compared before (step 16). This condition avoids unnecessary comparisons by appropriate bookkeeping. If any point in $CandidateSet$ is dominated by p, it is immediately removed from the $CandidateSet$ (steps 17-18). However, if p is in $CandidateSet$ and is dominated by any point in $CandidateSet$, it is also removed from $CandidateSet$ at the end of iteration (steps 24-26) as more than one point in $CandidateSet$ may dominate p. For each occurrence of p, $processedCount(p)$ is incremented by one (step 28). If $processedCount(p)$ is equal to number of complete dimensions p has and p is not pruned yet, it is removed from $CandidateSkyline$ and progressively added to the $ResultSet$ (steps 29-31). Step 32 increments $ptr_{nextDim}$ to point to the subsequent data point in array $D_{nextDim}$ and step 33 increments counter *iter* to point to next sorted array. Finally, the algorithm terminates when $CandidateSet$ becomes empty or all points have been processed at least once and $ResultSet$ is returned as the answer.

If all points in data set have been processed at least once before $CandidateSkyline$ becomes empty, then the algorithm can be safely terminated early. Because no point (which is yet to be processed) is left to be compared against $CandidateSkyline$ points. Thus all points in $CandidateSkyline$ belong to the skyline and are moved from $CandidateSkyline$ to $ResultSet$.

The condition (step 16), for avoiding unnecessary comparisons is implemented with the help of two data structures as follows. Each point p may have, (1) $processedTimestamp(p)$ which stores the time when p was processed for the first time, only if it is not pruned and (2) $deleteTimestamp(p)$ which stores the time when p was deleted, only if p is yet to be processed. Then, the next point p' to be processed, should be only compared with a point c in $CandidateSet$, if c has not been processed yet or $processedTimestamp(c) > deleteTimestamp(p')$ i.e. if c was processed after point p' has been pruned. This is because, points in $CandidateSet$ which were processed before p' has been deleted, have already been compared with p'.

4.2 Proof of Correctness

Each point in data set D is stored in at most d arrays and may be processed in different orders. We say that p is a skyline point when following conditions hold:

I. Point p has not been pruned yet, and

II. p has been processed k times, where k is the number of complete dimensions in point p.

The first condition implies that, point p is still in $CandidateSet$. While the second condition implies that, points that have been processed before p was processed for the k^{th} time, could not dominate p.

Suppose there exists a point p' that dominates p, where p' may or may not be a skyline point. Any point p having k complete dimensions can be dominated in at most k dimensions. Let us assume that p' has larger value than p in the last dimension s_d and p' has same value as p on all other *common* dimensions that are known in both points. (Note that, these are the minimal requirements for point p' to dominate p.) This implies that p' should occur earlier than p in at least one array D_d. For dimensions $M\backslash\{s_d\}$, p and p' may occur in any order. Thus, we have

u_1	u_2	u_3	u_4	u_5
m_5	m_7	m_1	m_5	m_3
m_4	m_2	m_2	m_2	m_5
m_1	m_6	m_5	m_7	m_4
m_7		m_7	m_4	m_7
		m_3	m_1	

Figure 2: Sorted arrays for sample data set

movieId	m_1	m_2	m_3	m_4	m_5	m_6	m_7
dimCount	3	3	2	3	4	1	5

Figure 3: Number of complete dimensions for points

two cases here: (1) **Case 1:** *p' occurs before p in at least one of the $M\backslash\{s_d\}$ dimensions.* So, p' would be processed first and thus p' should have pruned p, irrespective of whether p' is in $CandidateSet$ or not. However, by condition (I), point p is still in $CandidateSet$. Hence this case leads to a contradiction. (2) **Case 2:** *p' occurs after p for all dimensions $M\backslash\{s_d\}$.* Thus, p would be processed before p'. Here, we have two cases: (a) **Case a:** *p' is in $CandidateSet$.* Thus, p and p' would be compared and p' should have pruned p. But, according to condition (I), p is still in $CandidateSet$, and hence this case also leads to a contradiction. (b) **Case b:** *p' is not in $CandidateSet$.* Since p' occurs before p in dimension s_d, processing of p' leads to p being pruned. As p is still in $CandidateSet$, this case leads to contradiction. Hence, all the cases that arise, lead to a contradiction and thus, there can not exist a point p' that dominates p.

Note that every point occurring after point p in a sorted array, has value less than or equal to the value of p across the corresponding dimension. Hence, any point that is *processed*, after p has been processed k times, can not dominate p. Thus, a point p is a skyline point whenever condition (I) and (II) holds.

4.3 Example

We explain our algorithm using sample movie-rating data set D, given in Figure 1. First, data points in D are sorted in non-ascending order for each dimension $u_1...u_5$. Figure 2 shows five sorted arrays, one for each dimension u_i. Now, ptr_i is initialized with first entry in the respective arrays. Figure 3 shows the number of complete dimensions for each movieId. $processedCount(m_i)$ for each movieId is also initialized to 0. $CandidateSet$ is initialized to movieIds m_1 to m_7. As we are processing arrays in round-robin fashion, in the first iteration, movie m_5 is selected from array D_1. The points m_3, m_4 and m_7 get dominated by m_5 and thus, are removed from the $CandidateSet$. And we set $processedCount(m_5) = 1$. In the second iteration, m_7 is used to prune m_6 and $processedCount(m_7) = 1$. Note that, here we do not compare m_7 with m_5 as they have already been compared before. Also, the point being processed is compared only with points that are still in $CandidateSet$. In the third iteration, m_1 is selected which does not prune any movie, but gets dominated by m_2 and hence is removed from $CandidateSet$. Fourth iteration sets $processedCount(m_5)$ to 2. In the next three iterations, points m_3, m_4 and m_2 are selected respectively. None of these points dominate any point in $CandidateSet$. In iteration 9, we have $processedCount(m_2) = 3$, which is equal to $dimCount(m_2)$. Hence, m_2 is a *skyline* point and

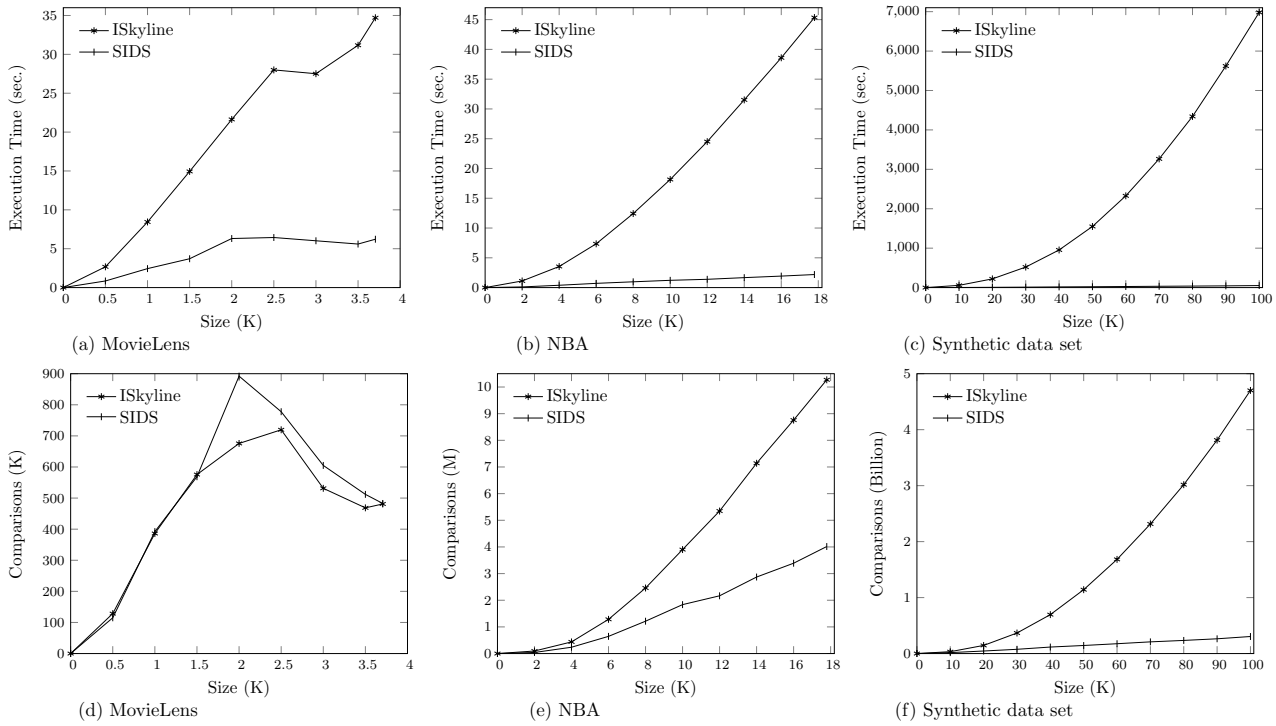

Figure 4: Scalability - effect of varying data size

is moved from *CandidateSet* to *ResultSet*. By iteration 12, all points in the data set have been processed at least once. Hence, the algorithm terminates and we move all remaining points in *CandidateSet* i.e. $\{m_5\}$ to *ResultSet*. Finally, $\{m_2, m_5\}$ is returned as the skyline set.

4.4 Analysis

The order in which the dimensions are chosen for SIDS algorithm affect the actual number of comparisons on a given data set. Currently next dimension is chosen in a round-robin manner as suggested by Chan et al. (2006). It might be an interesting direction to look at various strategies for choosing the optimal order of dimensions. If first few entries in each of the sorted array D_i could prune large number of non-skyline points, then the algorithm saves unnecessary comparisons between incomparable points and thus finishes early. On the other hand, when non-skyline points are not pruned early, the performance of the algorithm is relatively poor. As the algorithm uses presorted data, its performance remains same irrespective of the input order. The memory utilization of our algorithm is stable, as it uses minimal data structures and does not store any redundant data. The detailed performance analysis is presented in the Experimental Section below.

5 Experiments

We have implemented ISkyline algorithm of Khalefa et al. (2008) and our SIDS algorithms in C++. This section discusses the experimental settings used and compares the performance of the aforementioned algorithms.

5.1 Experimental Settings

For evaluating the performance of the proposed approach, we have used both real world and synthetic data sets. MovieLens and NBA[2] are real world data sets while the third data set is synthetically generated using the benchmark data generator provided by Börzsönyi et al. (2001).

MovieLens is a movie rating data set where 3706 movies have been rated by 6040 users on the scale of 1-5. The data set is 95% incomplete and contains only 1 million ratings. For evaluation purpose, each user is treated as a dimension for a movie.

NBA data set contains, among other things, statistics about 17 skills of basketball players during regular season in the period 1946-2005. The data set consists of 19,112 tuples, with multiple tuples for some players who have switched teams during the season. As suggested by Bartolini et al. (2006), we have manually deleted tuples with team name "total" for such players and thus obtained 17,791 records. NBA is a correlated data set, where all attribute values are positively correlated. That is, point good in one dimension tends to be good in all other dimensions as well. Thus, few "good" points dominate large number of points, resulting in small size of the skyline. As the NBA data set is complete, we have explicitly removed values from it to generate 20% incomplete data set.

A 15-dimensional synthetic data set of 100K points is generated using an uniform distribution where attribute values are independent of each other. The synthetic data generated by benchmark data set generator is complete and thus values are explicitly removed from the data set to make it 20% incomplete. We could not perform experiments on 100-dimensional synthetic data of size of 100K because for such a high dimensions, approximately all Buckets in ISkyline algorithm become comparable with each other. Storing pointers to such a large number of comparable Buckets alone, as required by ISkyline algorithm, needs huge amount of memory which is beyond our system's capacity.

Note that, all the results reported here are evalu-

[2] http://basketballreference.com

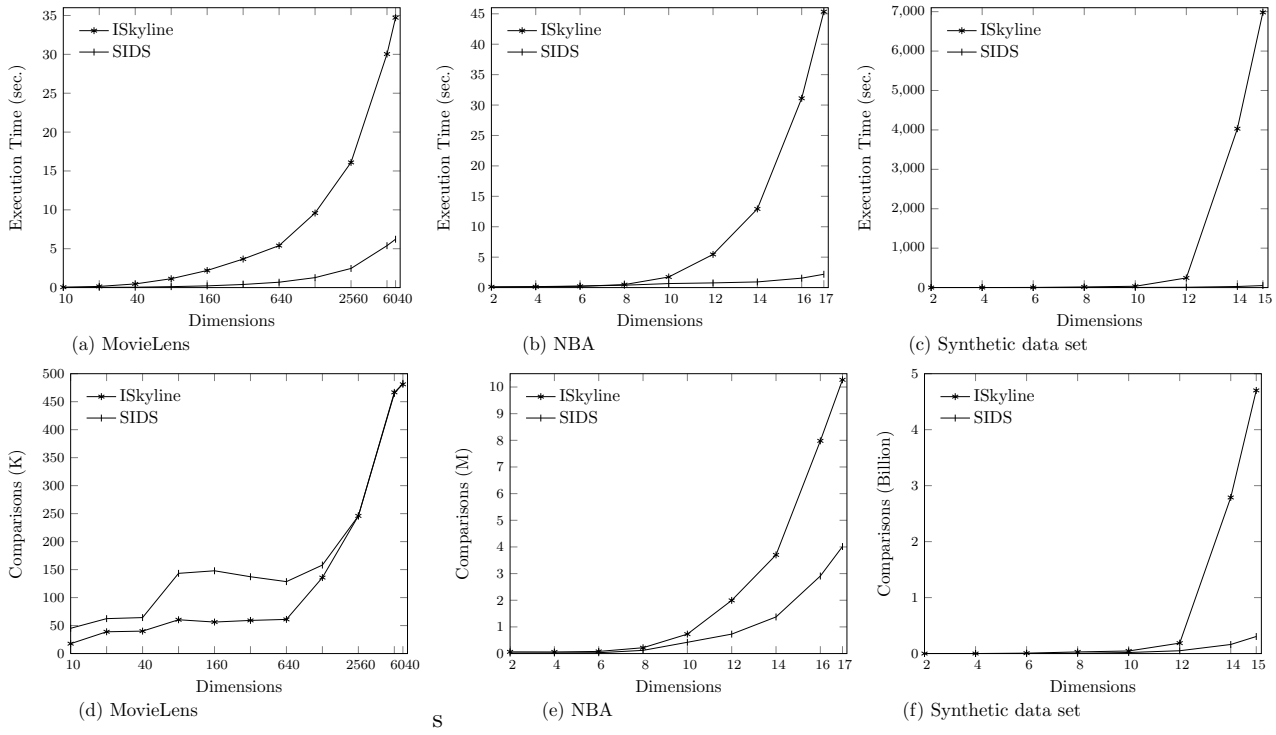

Figure 5: Dimensionality - effect of varying dimension

ated on same random ordering of data for both algorithms. The sorting time for SIDS is included in the execution time of the algorithm. The *tuning* parameter t, in ISkyline algorithm is set to 200, 20, 100 for MovieLens, NBA and Synthetic data set respectively, as suggested in the paper. All the experiments are performed on Intel Quad-core 2.83GHz processor machine, having 4GB RAM and running Ubuntu 10.04 LTS with kernel 2.6.32-41-generic. We report performance based on execution time taken and number of comparisons incurred by each algorithm.

5.2 Scalability

Figure 4 shows the effect of varying data set size on SIDS. As shows in Fig. 4(a), 4(b) and 4(c), for all three data sets, SIDS steadily outperforms ISkyline algorithm in terms of execution time. The reason is that, our approach does not have the overhead of maintaining bitmaps, virtual points and comparable Buckets.

In case of MovieLens data set, total number of Buckets are very close to the data set size, resulting in each Bucket approximately having only one or two elements. Thus, ISkyline does not benefit from the pruning power of virtual points and shadow skyline. For MovieLens data set (Fig. 4(d)), SIDS incurs more comparisons than ISkyline algorithm when data set size is 2K or higher because MovieLens is an anti-correlated data set to some extent. Thus, each point in the sorted array prunes relatively less number of points and hence subsequently more comparisons are required to prune remaining non-skyline points.

For NBA and Synthetic data sets, our approach outperforms ISkyline in terms of the number of comparisons also, as depicted in Fig. 4(e) and 4(f). The performance improvement comes from the fact that, in both data sets, first few points in sorted arrays dominate large number of points, thus reducing the execution-time and comparisons incurred. For example, in NBA data set with 2000 points, only 9 points collectively prune 1885 points in the first scan itself and for Synthetic data set with 20000 points, only 30 points prune 3903 points. In case of Synthetic data set, number of comparisons incurred by SIDS are orders of magnitude less than that of ISkyline. In contrast, ISkyline performs poorly because it processes data sequentially in the input order.

5.3 Dimensionality

The results depicted in Figure 5 show the effect of dimensionality on SIDS for MovieLens, NBA and Synthetic data sets. The performance of SIDS when varying dimensions is very similar to that of size. SIDS algorithm continues to have superior performance in terms of execution time for all three data sets (Fig. 5(a), 5(b) and 5(c)).

For MovieLens data set, our approach performs worse than ISkyline in terms of comparisons incurred (Fig. 5(d)). The reason is, even though SIDS prunes non-skyline points efficiently in first few scans of sorted arrays, it incurs extra comparisons in validating the skyline. ISkyline, on the other hand, makes use of local skyline and shadow skyline concepts to reduce redundant comparisons. However, note that, the reduction in number of comparisons incurred by ISkyline does not lead to any benefit in terms of overall execution time.

On NBA and Synthetic data sets, our approach outperforms ISkyline in terms of number of comparisons and execution time. For 17-dimensional NBA data set, SIDS incurs only 4 million comparisons in contrast to 10 million incurred by ISkyline.

5.4 Incomplete Ratio

We have used incomplete ratio of 10%, 50% and 90% for NBA data set, to analyze the effect of incomplete ratio on both algorithms. The y-axis in Figure 6(a) denotes the ratio of execution time taken by SIDS to that of ISkyline. SIDS algorithm runs in less than

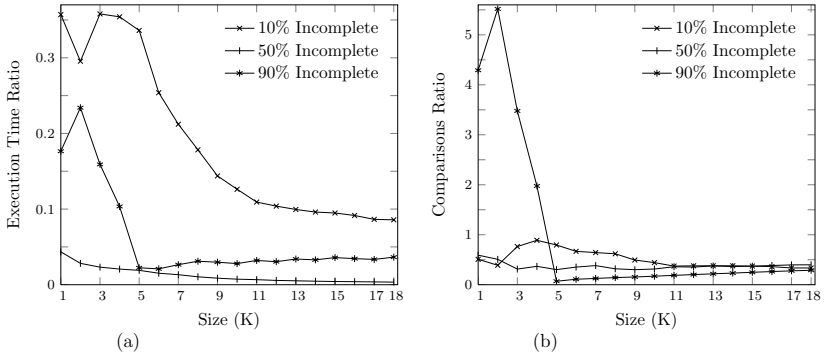

Figure 6: Effect of varying incomplete ratio (NBA data set)

40% of execution time taken by ISkyline algorithm for all the three incomplete ratios. Observe that, for NBA data set with 50% incomplete ratio, SIDS runs in less than 5% execution time taken by ISkyline algorithm.

Similarly, the y-axis in Figure 6(b) denotes the ratio of the number of comparisons incurred by SIDS to that of ISkyline. On NBA data set, with 90% incomplete ratio and data set size less than 5K, SIDS incurs significantly large number of comparisons. This is because, SIDS does not terminate immediately after pruning all non-skyline points and majority of the comparisons incurred by it are for validating the skyline points. With the exception of this, SIDS incurs much less comparisons compared to ISkyline. For example, on NBA data set with 50% incomplete ratio, SIDS incurred less than 60% of the comparisons required by ISkyline. Thus, the performance of SIDS algorithm is superior to ISkyline and scales well with respect to incomplete ratio of the data set.

6 Conclusion

In this paper, we introduced the *Sort-based Incomplete Data Skyline* (SIDS) algorithm for computing the skyline over data sets with missing values. We explained the need for a new algorithm, optimized specifically for such incomplete data sets. The proposed approach efficiently uses sorting on columns to prune non-skyline points early. We hypothesized and proved that, if a data point has been processed as many times as the number of complete dimensions in it, then it is a skyline point. Experimental results show superiority of our approach over ISkyline algorithm. Our algorithm is efficient, scalable and has stable performance, besides utilizing far less memory across both real world and synthetic data sets.

References

Balke, W.-T., Güntzer, U. & Zheng, J. X. (2004), Efficient distributed skylining for web information systems, *in* 'Proceedings of the international conference on Extending Database Technology', pp. 256–273.

Bartolini, I., Ciaccia, P. & Patella, M. (2006), Salsa: computing the skyline without scanning the whole sky, *in* 'Proceedings of the ACM international Conference on Information and Knowledge Management', pp. 405–414.

Börzsönyi, S., Kossmann, D. & Stocker, K. (2001), The skyline operator, *in* 'Proceedings of the International Conference on Data Engineering', pp. 421–430.

Chan, C. Y., Eng, P.-K. & Tan, K.-L. (2005), Stratified computation of skylines with partially-ordered domains, *in* 'Proceedings of the ACM SIGMOD international conference on Management of Data', pp. 203–214.

Chan, C. Y., Jagadish, H. V., Tan, K.-L., Tung, A. K. H. & Zhang, Z. (2006), Finding k-dominant skylines in high dimensional space, *in* 'Proceedings of the ACM SIGMOD International conference on Management of Data', pp. 503–514.

Chomicki, J., Godfrey, P., Gryz, J. & Liang, D. (2003), Skyline with presorting, *in* 'Proceedings of the International Conference on Data Engineering', pp. 717–719.

Dellis, E. & Seeger, B. (2007), Efficient computation of reverse skyline queries, *in* 'Proceedings of the international conference on Very Large Data Bases', pp. 291–302.

Fagin, R., Lotem, A. & Naor, M. (2001), Optimal aggregation algorithms for middleware, *in* 'Proceedings of the ACM SIGMOD-SIGACT-SIGART symposium on Principles of Database Systems', pp. 102–113.

Hose, K. & Vlachou, A. (2012), 'A survey of skyline processing in highly distributed environments', *The VLDB Journal* **21**(3), 359–384.

Khalefa, M. E., Mokbel, M. F. & Levandoski, J. J. (2008), Skyline query processing for incomplete data, *in* 'Proceedings of the International Conference on Data Engineering', pp. 556–565.

Kossmann, D., Ramsak, F. & Rost, S. (2002), Shooting stars in the sky: An online algorithm for skyline queries, *in* 'Proceedings of the international conference on Very Large Data Bases', pp. 275–286.

Papadias, D., Tao, Y., Fu, G. & Seeger, B. (2005), 'Progressive skyline computation in database systems', *ACM Transactions on Database Systems* **30**(1), 41–82.

Pei, J., Jiang, B., Lin, X. & Yuan, Y. (2007), Probabilistic skylines on uncertain data, *in* 'Proceedings of the international conference on Very Large Data Bases', pp. 15–26.

Sharifzadeh, M. & Shahabi, C. (2006), The spatial skyline queries, *in* 'Proceedings of the international conference on Very Large Data Bases', pp. 751–762.

Tan, K.-L., Eng, P.-K. & Ooi, B. C. (2001), Efficient progressive skyline computation, *in* 'Proceedings of the international conference on Very Large Data Bases', pp. 301–310.

Zhang, Z., Lu, H., Ooi, B. C. & Tung, A. K. H. (2010), 'Understanding the meaning of a shifted sky: a general framework on extending skyline query', *The VLDB Journal* **19**(2), 181–201.