# ANSWERING WHY-NOT QUESTIONS ON REVERSE SKYLINE QUERIES OVER INCOMPLETE DATA

**Bagus Jati Santoso[1)] and Tosca Yoel Connery[2)]**
[1, 2)] Department of Informatics, Institut Teknologi Sepuluh Nopember
Surabaya - Indonesia
e-mail: bagus@if.its.ac.id[1)], 5114100061@if.its.ac.id[2)]

**ABSTRAK**

*Dewasa ini, perkembangan query berbasis preferensi mendapat cukup banyak perhatian dari para peneliti dan pengguna data. Salah satu dari query berbasis preferensi yang paling popular adalah skyline query, yang mana akan memberikan subset himpunan superior yang tidak didominasi oleh titik manapun. Sebagai pengembangan dari skyline queries, lahirlah reverse skyline queries. Query ini bertujuan untuk mendapatkan informasi mengenai query points yang menjadikan sebuah data atau record sebagai hasil skyline query-nya.*

*Lebih lanjut, pengembangan IT yang berorientasi pada data menuntut ilmuwan agar dapat mengolah data dalam segala kondisi. Di dunia nyata, seringkali ditemui data multidimensi yang tidak lengkap, baik karena rusak, hilang, maupun perihal privasi. Untuk meningkatkan nilai guna dari sebuah kumpulan data, penelitian ini akan mendiskusikan mengenai salah satu masalah dalam pengolahan reverse skyline queries pada data tidak lengkap, yaitu masalah "why-not". Solusi yang diharapkan dari masalah "why-not" ini adalah saran dan langkah agar sebuah query point yang semula tidak menganggap sebuah record atau data tidak lengkap sebagai hasilnya, nantinya dapat menjadikan record atau data tersebut sebagai bagian dari hasil. Pada penelitian ini, akan terdapat diskusi dan pembahasan mengenai solusi dari problem tersebut beserta uji performansi untuk mengukur tingkat efisiensi dan efektifitasnya.*

***Kata Kunci**: incomplete data, reverse skyline, skyline query, why-not*

**ABSTRACT**

*Recently, the development of the query-based preferences has received considerable attention from researchers and data users. One of the most popular preference-based queries is the skyline query, which will give a subset of superior records that are not impeded by any records in the dataset. As the developed version of skyline queries, a reverse skyline query rise. This query aims to get information about the query points that make a data or record as the part of result of their skyline query.*

*Furthermore, data-oriented IT development requires scientists to be able to process data in all conditions. In the real world, there exist an incomplete multidimensional data, both because of damage, loss and privacy. In order to increase the usability over a data set, this study will discuss one of the problems in processing reverse skyline queries over incomplete data, namely the "why-not" problem. The considered solution to this "why-not" problem is the recommendation and steps so that a query point that does not initially consider an incomplete data, as a result, can later make the record or incomplete data as part of the results. In this study, there will be further discussion about the dominance relationship between incomplete data along with the solution of the problem. Moreover, some performance evaluations are conducted to measure the level of efficiency and effectiveness.*

***Keywords**: incomplete data, reverse skyline, skyline query, why-not*

## I. INTRODUCTION

IN a rapidly developing era recently, information technology is a vital aspect that covers almost all areas of human life. All activities and community works have been integrated recently with technology and databases, in order to achieve the optimum results. To be able to provide these benefits, an application often involves a considerable number of data in its computational process so that it requires the appropriate algorithm so that the application can deliver the correct results rapidly. Moreover, if we employ the standard database system, it may result in a lengthy execution time, consequently a new data structure that fit with the purpose of the application is required.

Based on the explained above conditions, a proper algorithm and purpose-specific data structure are critical to process the considerable number of data into some valuable information. With the continuous development of data structure and indexing, the result of specified data-related problems can be obtained without being constrained by the size and the velocity of data that needs to be evaluated.

Most research and discussions in the preference-based queries area are focused on numerous categories of problem queries and the search for state-of-the-art technology to tackle those problem. The most popular one is skyline queries. Firstly pioneered by Borzsony et al in [1], this paper proposes the query mechanism to obtain a set of data that are preponderant with respect to the dominance connection. Over literature, there exist extensive research results that related to the pioneer work. To progressively obtain the skyline points, Bitmap and Index algorithm were proposed in [2]. There also exist some works that integrate skyline query with non-centralized environment,

such as uncertain databases [3], mobile environment [4] and over sliding windows [5]. The author of [6] proposed an interesting problem, known as reverse skyline queries, in which provide the set of query points that consider a record or respected product as part of their skyline results.

However, there also exists an area where the refinement of query results is put forward. One example of the use of network and database technology is an evaluation system to perform the query refinement and find deficiencies in a product so that there still exist consumers who are not interested in the product. Moreover, we need to look for effective and efficient solutions to correct these deficiencies. This problem is well-known as the why-not problem. Why-not questions itself have gained considerable attention in the world of data engineering in recent years. A why-not problem has found application in many areas, such as the pioneer work of query result in [7]. He and Lo present a feature of explaining the missing tuples over the result of defined query. They developed algorithms to solve the why-not question over top-*k* query results efficiently. In [8], Herschel proposes a hybrid approach by forming the union of explanations and producing the larger set of missing answer. Further, an algorithm named Conseil is also created to produce the hybrid explanation. Moreover, in [9], Chen et.al discuss the why-not question especially over spatial keyword top-*k* queries. In providing solution, the author provides a query refinement-based algorithm using geometrical approach.

In real life, data does not always have a whole value over its multi-dimensions. There are circumstances where the attributes possessed by a data are empty or missing. This data is namely incomplete data. This incomplete data will significantly affect the algorithm to process. In the skyline queries, there exist pareto-based dominance relationships where if $a_1$ dominates $a_2$ and $a_2$ dominates $a_3$, then $a_1$ will automatically dominate $a_3$. However, this does not apply if it turns out that the data is incomplete. Incomplete data may lead to a cyclic dominance relationship where $a_1$ dominates $a_2$, $a_2$ dominates $a_3$ and $a_3$ dominates $a_1$. Hence, we need to design a specific algorithm to fit with the characteristics of incomplete data.

Encouraged by the above statements, we discuss the why-not question problem and propose the discussion on why-not question of reverse skyline queries over the incomplete data. According to our best, this is the first work that incorporate the incomplete data in the why-not questions over the incomplete data. Since the features of the data is completely different, a unique solution to tackle this problem needs to be proposed.

Overall, this work provides the contributions in the following:

1) we discuss the essential structure of the dominance connection over the incomplete data,
2) we define some theorems to facilitate the discussion and support the algorithm in tackling the why not question on reverse skyline queries over incomplete data,
3) we derive the useful properties and design a method to process the problem efficiently,
4) extensive experiments are carried out to evaluate the proposed solution.

The remainder of this paper is organized as follows. We review the works that related to this paper in Section 2. Moreover, we outline the problem definition of this work in Section 3. Section 4 provides the discussion over the proposed method to handle this problem. Extensive experiment outputs are discussed in Section 5. Finally, the concluding statements are provided in the last section.

## II. Related Work

There exist several works that have been published in accordance with why-not question in data engineering area. Chapman and Jagadish in [10] propose two algorithms that explain why a particular data item does not join the result of a given query in acceptable computation cost and running time. Moreover, Bidoit et al [11] propose algorithm named *NedExplain* that explain data missing from given query results by computing the provenance of why-not for monotone relational queries in aggregated scenario. In [12] and [13], by providing provenance-style answers, the authors discuss and propose some methods to tackle the problem of why-not on select-project-join queries. As an extension, there also exist some researches that focused on answering why-not in SPJUA (selection-projection-join-union-aggregation-grouping) queries [14][15].

In [16], the problem of why-not over reverse skyline area is discussed by Islam et al. Their work aims to obtain all records in which the query poins are contained in their dynamic skyline result. Their method leans to both the query and data point refinement approaches in producing purposeful answers.

Khalefa et al in [17] discuss the processing task of skyline query over the incomplete data. This works describes the comprehensive information about non-transitive dominance relationship between incomplete data. Two method are proposed to deal with the processing, named “Replacement and Bucket” and “ISkyline”. “Replacement and Bucket” is algorithm that adopted from the method of traditional skyline processing. On the other hand, “ISkyline” employs two optimization objects, called shadow skyline and virtual points, to provide the answer of why-not question on skyline queries while dealing with cyclic dominance relation over incomplete data.

## III. BACKGROUND AND SYSTEM MODEL

We define the why-not problem on reverse skyline queries over incomplete data (WNRSI) in this setction. Moreover, the system model used in this paper are also defined. The WNRSI problem trunks from some definitions of dominance concept and skyline queries that are subject to be discussed initially.

### A. *Skyline and Data Dominance*

Skyline contains the prominent records in which no other record over the dataset impedes them [1]. Compared to other data, skyline records are superior. Skyline queries do not need user to define a ranking function, and do not produce a bounded quantity of output. The dominance relationship between records depends on the semantics over each feature. A record *r* dominates another records if and only if *r* is better or at least equal than *s* in all dimension, and clearly better than *s* in at least one dimension. The size of skyline query output is bounded for the same dataset.

The advantage of skyline queries itself is enabling users to find a prominent, a potentially important object, due to its dominance in one or more dimensions. The unbounded size of the query results is the significant downside of a skyline query. In the extreme situations, skyline queries may return exactly one data, or the entire dataset, which both are not informative for the users.

### B. *Dynamic Anti-Dominance Region*

The dynamic anti-dominance region or $\overline{DDR}$ is the region that lies between the reference point and the skyline line. Any new point placed inside $\overline{DDR}$ will automatically become a new skyline. $\overline{DDR}$ will be used to determine the safe region for the moving-based refinement of query points with no possibility for being dominated by other points. Outside areas are called dynamic dominance regions (*DDR*) which contain all points dominated by skyline [18].

### C. *Dynamic Skyline*

Dynamic skyline (DSL) is a skyline that is based on a particular data as a reference point. In general, there is virtual line named *DSL(c)* which is a line that limits *DDR(c)* and $\overline{DDR}(c)$ [18]. The attribute value that is compared in the dynamic skyline is gained by using semantic around-by, which is the difference of value on each attribute between the reference data against the same attribute from the surrounding data. To facilitate the process of comparing any data that is part dynamic skyline, all data around the reference will be transformed into one same quadrant with the reference point as the center. In this study, the dynamic skyline is used to determine the skyline of each consumer or query point.

In Fig. 1, all records around the query-point *q* are transformed against *q* into the same quadrant. From the transformation result, records *p2', p3'* and *p4'* clearly dominate other points. Hence records *p2', p3'* and *p4'* become the dynamic skyline of *q*.

### D. *Reverse Skyline (RSL)*

Reverse skyline is a set of data that acts as a preference and concurrently picks the same query point *q* that

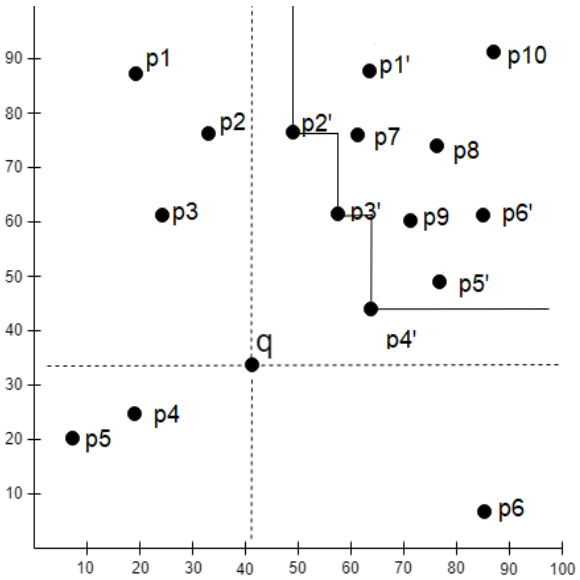



Fig. 1. The example of dynamic and reverse skylines

consider them as the dynamic skyline [18]. Reverse skyline is used to determine the consumers or query point that are interested in a product that has certain specifications.

As depicted in Fig. 1, $p_2$, $p_3$ and $p_4$ are dynamic skylines of $q$. Moreover, $q$ is the reverse skyline of $p_2$, $p_3$ and $p_4$. In this study, the reverse skyline is going to be utilized in order to determine all query point, or consumers, who are interested in the query point $q$.

*E. Skyline over Incomplete Data*

Incomplete data is a state where at least one missing value exists in the attribute of a data. It should be noticed that when two data are compared in the process of determining the skyline, only dimensions with complete data on both sides can be compared. While incomplete dimensions do not need to be compared.

This results to the inability of normal skyline query algorithm to be used since if *a* dominates *b* and *b* dominates *c*, *c* may just dominate a. This is called cyclic dominance relation [3] [4].

This incomplete data can be caused by several things such as damage to the storage device used, data loss, or just because of privacy where the data owner refuses to present it, hence the data cannot be involved in the process of determining the skyline.

As a result of the occurrence of cyclic and non-transitive relation dominance relations on incomplete data, the process of determining the skyline must use the ISkyline algorithm [17]. In this algorithm, the data that is processed must go through several stages so that it can be truly classified as a skyline in conditions of incomplete data. Data compared must be local skyline in the first stage of the process, then become an skyline expectant in the second stage, and finally become the final global skyline in the last stage.

In the first stage or local skyline, each data that is checked will be separated into certain bucket. A bucket is composed of all data that has the same bitmap representation and each bucket has a unique bitmap representation. Bitmap representation refers to the presence or absence of values on an attribute or dimension. The skyline results from each bucket are called the local skyline.

If a data successfully becomes a skyline in the local domain, it will be forwarded to the stage of the skyline candidate. At this stage the data being processed will be compared to the local skyline from another bucket. The number of candidate skylines is limited to *t* number of data so that there is no considerable number of data will be compared. When the number of candidate skylines has passed the *t* limit, a global skyline update will be carried out.

The final stage is the global skyline. In this process, all candidate skylines will be compared to the global skyline. All global skylines and non-dominated candidate skylines will survive as a global skyline and the candidate skyline variable will be re-initialized. All data that managed to survive to the end in this process can be called the actual skyline from incomplete data sets.

*F. Why-Not Questions over Incomplete Data*

Why not questions are problem that question why certain data or objects that are expected to be part of the query results do not appear as results. The answer to this question will explain what parts need to be adjusted so that the data can appear as a result of the query.

There are several ways that can be used to answer the problem of not questions, namely by correcting the why-not points (user preferences that do not become RSL), correcting the query points (the value of each product attribute or specification) or improving both. This must be done with minimum effort and without causing loss of customers from the list of customers that have been previously owned.

By using the first method, the why-not point *ct* will be moved to a safe location so that it is not dominated by another point. The first thing to do is to determine all the outermost points between the why-not points and the query point. The outermost points can be obtained by running a query window centered in *ct* according to the equation.

$$\Lambda \leftarrow window\_query(ct, q) \tag{1}$$

From the results of the query window, a point that meets the equation is selected.

$$\forall e_1 \in F, \nexists e_2 \in \Lambda : e_2 \prec_q e_1 \tag{2}$$

The location that can be used must be at least right in the middle between the outermost point and the query point in each dimension *ct.*

$$u_l^i = e_l^i + \frac{|e_l^i - q^i|}{2}, \forall_i \in \{1,\ldots,d\} \quad (3)$$

The other way is performing the refinement over the query points used. The moving task of the query point needs to consider the safe region *q.* This area is an intersection of all regions $\overline{DDR}$ of all RSL (q) so it does not lead to the loss of existing customers. This can only be done if the *SR (q)* intersects with $\overline{DDR}$ from *ct.*

The last method is done when the *dynamic anti-dominance region* $\overline{DDR}$ of the data that becomes the why-not point turns out not to have the same area with the intersections of all the dynamic anti-dominance regions of RSL, which is moving both the why-not points and the query points.

### *G. Virtual Points*

The utilization of the virtual points is motivated by the requirement to reduce the size of comparison tasks in obtaining the candidate of skyline result [18]. This can be done by performing initial screening of the prospective candidate skyline in the local skyline insertion process. Virtual point is data from another bucket that functions as a filter on a bucket so that the number of forwarding local skylines as a candidate skyline decrease.

Before a data is entered into the local skyline of the bucket, the data will be compared first with the virtual point specified on the bucket. If it is dominated by its virtual point, a data will not be regarded as a local skyline. Virtual points are obtained when a candidate skyline insertion process in which the local skyline and candidate skyline are dominated in this process will be used as virtual points on the bucket that dominates them. The use of virtual points in this study will help eliminate data on the local skyline, which should be dominated by a candidate skyline, but that skyline candidate has already been dominated by other data. The dominated skyline candidates will be removed from the list, this can cause the local skyline which must be dominated by the candidate skyline to remain a new candidate skyline. This is what causes the requirement for a virtual point to synchronize the local skyline that will go to the next stage.

## IV. PROPOSED SCHEME

Our solution is based on the safe region, in which there exist two main steps that must be performed to determine recommendations for change. The first stage is the preprocessing stage, then proceed with the main process. Both of these stages can be seen in Fig. 2.

The purpose of the proposed scheme is to get a recommendation in updating the value of the attribute in the product or data, and the user preferences if the product is not an attractive choice for the user. Please that if the modification is applied, we need to make sure that there should not be a single query point or costumer lost from the existing list of the customers. A product or data can be said to be interesting if it has features that are not dominated by other products to the preference of consumer.

### *A. Preprocessing Tasks*

This stage is the the initial task the proposed scheme. This stage includes the process of determining DSL for each query point. The process of determining the skyline for incomplete data is done by adopting the ISkyline algorithm [17]. Each data must be able to fulfill the three skyline stages so that it can be called the actual skyline on incomplete data. The first stage is the local skyline where a data cannot be dominated by a virtual point and another local skyline on the same bucket. Data that has successfully become a local skyline will be compared to all candidate skylines. After the number of candidate skylines reaches a certain number, the global skyline renewal process will be carried out. This global skyline is a combination of candidate skyline and the global skyline itself which is not dominated by other data and it is not dominated by shadow skyline.

Algorithm 1 describes the pseudo code of generating all dynamic skyline in pre-processing tasks. From all query points, algorithm will create each dynamic skyline. Each of their dynamic skyline are separated into bucket according to their bitmap.

### *B. The Main Method*

After the preprocessing task is complete, the system will utilize the results of the main process. In this process the system will provide recommendations for the most efficient changes that can be made. The things that will be done in this section are determining the reverse skyline of the product, determining the safe region of the product,

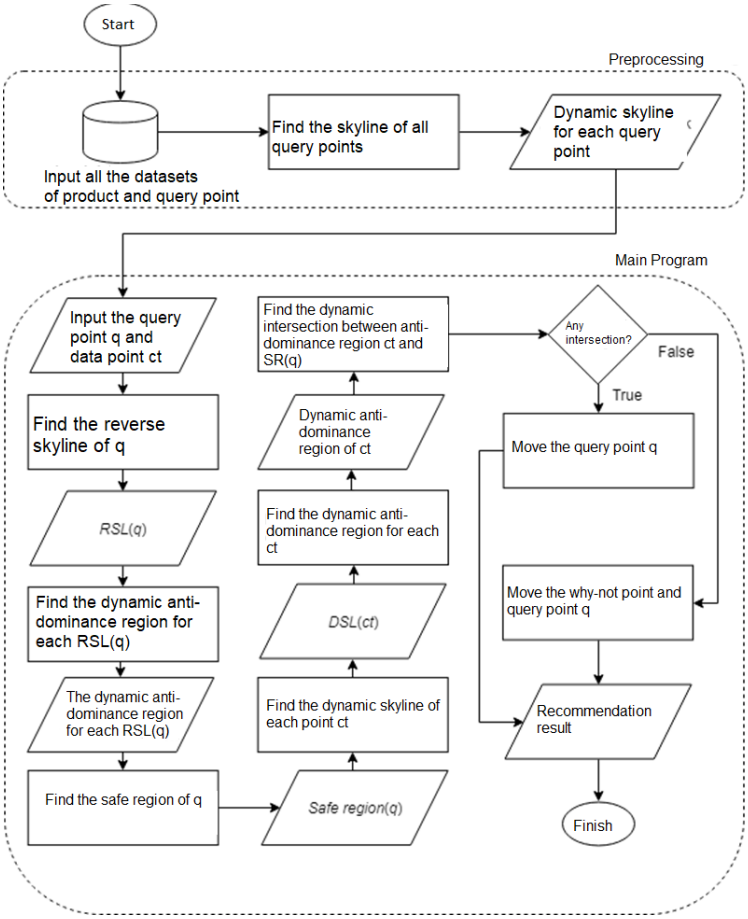



Fig. 2. The proposed scheme

**Algorithm 1** Generate All Dynamic Skyline

```
1.  customer_skyline ← {}
2.  for all pref ∈ list_customer do
3.      for all spec ∈ product_specification do
4.          is_skyline←
            Insert_Local_Skyline(transform(spec), bitmap)
5.          if is_skyline = TRUE then
6.              Insert_Candidate_Skyline(transform(spec),
                bitmap)
7.          end if
8.          update global_skyline when candidate_skyline
            reach a certain number
9.      customer_skyline = customer_skyline U global_skyline
```

determining the dynamic anti-dominance area of the query point who want to check, and determining the intersections of the two regions. All above tasks are performed by adopting the Algorithm 1 and Algorithm 2 of [18].

The why-not point *ct* will be compared to all skylines from consumer data. The tasks of insertion of skyline candidate is done by employing the Algorithm 3 of [18]. The process is performed identically with the skyline search process from the previous stage. There is additional task when there exist dominated data on performing skyline candidate insertion. These additional tasks are depicted in pseudocode over Algorithm 2, the virtual point insertion. Then we need to find any query point that make *ct* as the dynamic skyline. All consumers meeting these criteria will be made the RSL from *ct*. To determine the safe region of *ct*, the intersections of the entire reverse skyline of *ct* are needed. Moreovere, all product data *p* will be recounted by the skyline using the *ct* as the reference point. The global skyline insertion tasks are done by performing Algorithm 4 of [18]. Then from all those found skyline points, we can determine the $\overline{DDR}$ of *ct*.

To determine the refinement tasks, our proposed method first checks whether $\overline{DDR}$ (*ct*) intersects with safe region

of *q*, *SR(q)*. If there exist intersection between those two area, then the action that must be done is simply changing the value of the query point *q*. But if the two do not have intersections, it is necessary to change the values in the query points *q* and why not point *ct* simultaneously. In some cases, *q* query points may not be changed because the safe region, *SR (q)* region only includes the *q* point itself. This is caused by some consumers who are interested in *q* but have very different preferences so that the area of each consumer only intersects at the *q* point itself. The pseudo code to generate safe region over *q* are depicted in Algorithm 4.

Shortly after all safe region are formed, the query point *q* is able to be moved when there exists any intersection between safe region. In the other hand, when the intersection does not exist, the why-not point are transformed into the nearest part of safe region.

**Algorithm 2** Virtual Point Insertion

| | |
|---|---|
| **Global Variable** | *local_skyline, virtual_point, shadow_skyline* |
| **Input** | *Point P, Bitmap B* |
| **Output** | - |

```
1.  for each local_skyline L that has bitmap B do
2.      if L is dominated by P then
3.          move L to shadow_skyline[B]
4.  for each virtual_point V that has bitmap B do
5.      if V is dominated by P and have complete
        dimensions that are superset of, or same as P's
6.          remove V
```

**Algorithm 3** Generate DDR Prime(*Point Ct*)

| | |
|---|---|
| **Global Variable** | *product_list, node, local_skyline, candidate_skyline, global_skyline, shadow_skyline, virtual_point, bitmap, safe_region* |
| **Input** | *Ct* |
| **Output** | - |

```
1.  find all global_skyline of ct
2.  compare query_point q to all global_skyline of ct
3.  if q is not dominated then
4.      stop program //q is already in RSL(ct)
5.  else
6.      define top and bottom corner of each dimension
```

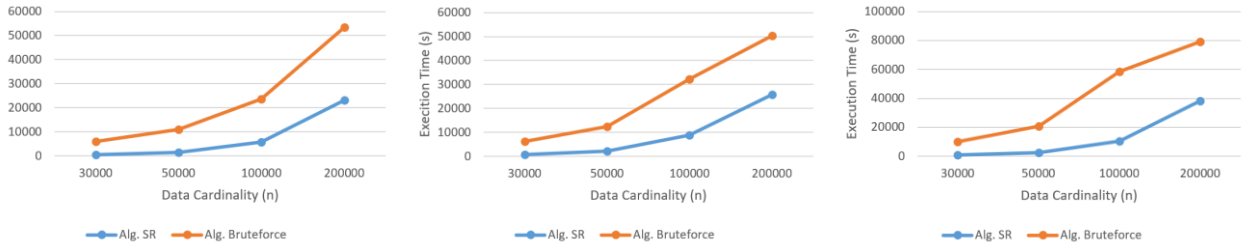
**Algorithm 4** Generate Safe Region

| | |
|---|---|
| Global Variable | *customer_skyline, query_point, safe_region* |
| Input | - |
| Output | - |

```
1.  for all customer_skyline
2.  SR(q) ← null
3.  for each cl ∈ RSL(q) do
4.      for each comparable data in DDR and transformed
        value of q do
5.      bottom = max(DDR, transformed query_point)
6.      top = min(DDR, transformed query_point)
7.      safe_region = safe_region U [bottom, top]
```

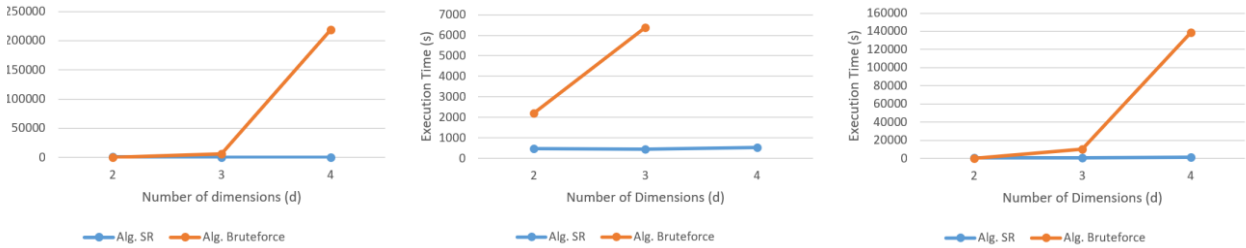
## V. PERFORMANCE EVALUATION

In performing the experiment, our works used the real-life dataset named Forest Cover (FC), and two synthetic datasets, which are anti-correlated (ANT) and independent (IND) datasets. The number of data *n* and dimension *d* are varied for each type of dataset.



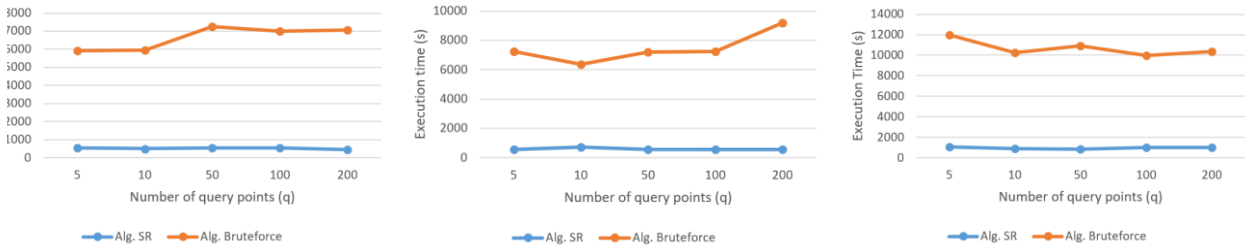

(a) IND (b) ANT (c) FC

Fig. 3. Data cardinality vs execution time




(a) IND (b) ANT (c) FC

Fig. 4. Number of dimensions vs execution time




(a) IND (b) ANT (c) FC

Fig. 5. Number of query points vs execution time

TABLE I
MEMORY USAGE OVER VARIANCE OF THE DATA CARDINALITY

| *n* | ***d* = 3, \|*q*\| = 10** | | | | | |
|---|---|---|---|---|---|---|
| | **IND** | | **ANT** | | **FC** | |
| | **SR** | **BF** | **SR** | **BF** | **SR** | **BF** |
| **30K** | 401M | 397M | 406M | 397M | 406M | 397M |
| **50K** | 404M | 397M | 410M | 397M | 410M | 397M |
| **100K** | 418M | 396M | 419M | 397M | 418M | 397M |
| **200K** | 436M | 396M | 437M | 397M | 436M | 396M |

TABLE II
MEMORY USAGE OVER VARIANCE OF THE NUMBER OF DIMENSIONS

| *d* | ***n* = 30K, \|*q*\| = 10** | | | | | |
|---|---|---|---|---|---|---|
| | **IND** | | **ANT** | | **FC** | |
| | **SR** | **BF** | **SR** | **BF** | **SR** | **BF** |
| **2** | 406M | 396M | 406M | 396M | 406M | 396M |
| **3** | 401M | 397M | 406M | 397M | 406M | 397M |
| **4** | 407M | 407M | 402M | 428M | 408M | 401M |

TABLE III
MEMORY USAGE OVER VARIANCE OF THE NUMBER OF QUERY POINTS

| \|*q*\| | ***n* = 30K, *d* = 3** | | | | | |
|---|---|---|---|---|---|---|
| | **IND** | | **ANT** | | **FC** | |
| | **SR** | **BF** | **SR** | **BF** | **SR** | **BF** |
| **5** | 406M | 396M | 406M | 397M | 406M | 397M |
| **10** | 401M | 397M | 406M | 397M | 406M | 397M |
| **50** | 406M | 396M | 401M | 397M | 406M | 397M |
| **100** | 406M | 396M | 407M | 397M | 406M | 397M |

### *A. Experimental Scenario*

The experiment is performed on each dataset with following variations for each of type of data:

a. Variation of data cardinality:**30,000**, 50,000, 100,000 and 200,000,
b. Variation of the number of dimensions: **3**, 5, 7 and 10,
c. Variation of query point: 5, **10**, 50, 100 and 200.

Please note that 30,000, 3, and 10 are the default value for data cardinality, number of dimensions, and query point, respectively.

### *B. Experimental Result: Execution Time*

In the experiment, we have measured the execution time by our proposed method (SR) compared to the brute force algorithm in virtually every case. The superiority of SR over the brute force algorithm in data cardinality case are depicted in Fig. 3. We get that our proposed method is the clear winner when the data cardinality varies over all datasets. This advantage factor is more evident when the data cardinality rises from 100.000.

Next, the outcome of our proposed method with respect to the dimension aspect is evaluated In Fig. 4, it is depicted that our SR completes slightly better than the brute force one from two to three dimensions. Over four dimensions, the performance of brute force algorithm becomes worse especially in ANT. We do not conduct performance over ANT in four dimensions, since it takes more than six hours to finish the computation.

Moreover, we also assess our proposed method with respect to the size of the query points. In all three datasets, as depicted in Fig. 5, we get that our proposed SR method overcomes the performance of the brute force one. The execution time of our proposed method is relatively stable in every running.

### *C. Experimental Result: Memory Usage*

In this paper, we have also conducted experiment over memory usage of SR and BF algorithms in all variance cases. Table I gives the memory usage over variance of the data cardinality. The result shows that there is no significant increment of memory usage when the data cardinality grows, both in SR and BF algorithms. Overall, the memory usage of SR algorithm is slightly exhaustive compared to the BF one.

Moreover, we also observe the memory usage over variance of the number of dimensions and query points. Table II and III shows the memory usage of the proposed SR compared to the brute force (BF). We can see that our proposed SR is slightly worse with respect to the memory usage. In accordance with the number of both dimensions and query points, our proposed SR consumes extra memory compared to the brute force one. We can clearly get that the memory usage of SR and BF increase when all variance factors increase.

## VI. CONCLUSION

This paper has discussed the problem of providing an answer to why-not question on reverse skyline queries over incomplete data. The incomplete multidimensional records exist in the real word because of damage, loss and privacy issues. The further discuss about a unique dominance relationship over the incomplete data is provided since it is dissimilar with the transitive nature of the dominance relationship in the complete one. We also have presented our proposed scheme to deal with the why-not question over the incomplete data in the reverse skyline area. By using real-world and the synthetic datasets, some experiments have been conducted to assess how our proposed scheme responses to the problem. The conducted experimental result show that our proposed method outdoes the brute force method in virtually all of the settings, with respect to the execution time.

## REFERENCES

[1] S. Borzsony, D. Kossmann, and K. Stocker, “The skyline operator,” in Proc. Int’l Conf. Data Engineering (ICDE), 2001, pp. 421–430.
[2] K.-L. Tan, P.-K. Eng, and B. C. Ooi, “Efficient progressive skyline computation,” in Proc. Int’l Conf. Very Large Data Bases (VLDB), 2001, pp. 301–310.
[3] X. Liu, D. Yang, M. Ye, and W. Lee, “U-skyline: A new skyline query for uncertain databases,” IEEE Trans. Knowledge and Data Eng., vol. PrePrints, 2012.
[4] X. Lin, J. Xu, and H. Hu. (2013). Range-based skyline queries in mobile environments. *IEEE Trans. Knowl. Data Eng. 25(4),* pp. 835-849.
[5] X. Lin, Y. Yuan, W. Wang, and H. Lu, “Stabbing the sky: efficient skyline computation over sliding windows,” in Proc. Int’l Conf. Data Engineering (ICDE), 2005, pp. 502–513.
[6] L. Chen, J. Xu, X. Lin, C. S. Jensen and H. Hu, "Answering why-not spatial keyword top-k queries via keyword adaption," in 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, 2016, pp. 697-708.
[7] Z. He and E. Lo. (2014). Answering why-not questions on top-k queries. *IEEE Trans. Knowl. Data Eng. 26(6),* pp. 1300–1315.
[8] Melanie Herschel. (2015). A Hybrid Approach to Answering Why-Not Questions on Relational Query Results. *J. Data and Information Quality 5(3).*
[9] L. Chen, J. Xu, X. Lin, C. S. Jensen and H. Hu, "Answering why-not spatial keyword top-k queries via keyword adaption," in 2016 IEEE 32nd International Conference on Data Engineering (ICDE), Helsinki, 2016, pp. 697-708.
[10] Adriane Chapman and H. V. Jagadish, "Why not?", In Proceedings of the 2009 ACM SIGMOD International Conference on Management of data (SIGMOD '09), 2009, pp. 523-534.
[11] N. Bidoit, M. Herschel, K. Tzompanaki. "Query-Based Why-Not Provenance with NedExplain", in Extending database technology (EDBT), 2014.
[12] J. Huang, T. Chen, A.H. Doan, J.F. Naughton, “On the provenance of non-answers to queries over extracted data”, in Proceedings of the VLDB Endowment 1.1, 2008.
[13] C. Zong, X. Yang, B. Wang, J. Zhang, “Minimizing explanations for missing answers to queries on databases”, in DASFAA, 2013, pp. 254–268.

[14] M. Herschel, M. Hernandez. (2010). Explaining missing answers to spjua queries. *VLDB Journal*, pp. 185–196.
[15] M. Herschel, M.A. Hernandez, W.C. Tan. (2009). Artemis: A system for analyzing missing answers, *VLDB Journal*, pp. 1550–1553.
[16] R. Zhou, C. Liu, Md. Saiful Islam, "On answering why-not questions in reverse skyline queries", in Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013), 2013, pp. 973-984.
[17] M.E. Khalefa, M.F. Mokbel, J.J. Levandoski, "Skyline Query Processing for Incomplete Data", in Proceedings of the 2008 IEEE 24th International Conference on Data Engineering (ICDE '08), 2008, pp. 556-565.
[18] M. S. Islam, R. Zhou, C. Liu, "On answering why-not questions in reverse skyline queries," in 2013 IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, 2013.
[19] R. Bharuka and P. S. Kumar, "Finding skylines for incomplete data" in Proceedings of the Twenty-Fourth Australasian Database Conference-Volume 137. Australian Computer Society, Inc., 2013.