An Efficient Pruning Method
to Process Reverse Skyline Queries*

AH HAN, YOUNGBAE PARK AND DONGSEOP KWON+
Department of Computer Engineering
Myongji University
Yongin-si, Gyeonggi-do, 449-728 Korea

Several algorithms for processing reverse skyline queries have been proposed in recent literature. However, these algorithms are based on pre-processing approaches, and hence involve complex procedures and waste storage space due to inefficient use of storage. In addition, they are not robust to frequently changing data as, they have to re-compute and update the pre-computed results. To overcome these issues, this paper proposes a novel algorithm to efficiently process reverse skyline queries using an approach based on two pruning methods: the search-area pruning method and the candidate-objects pruning method. Utilizing these pruning methods, the algorithm is able to process reverse skyline queries efficiently even in situations where data is changing frequently. The proposed algorithm also effectively reduces the inefficient use of storage under existing approaches for storing pre-computed results. We conducted extensive experiments to show that the proposed algorithm shows better performance compared to existing approaches regardless of the dimension, distribution, or size of the data.

Keywords: skyline, reverse skyline, preference queries, query processing, database

1. INTRODUCTION

Given a set of d-dimensional points, the skyline operator returns all the points from the set that are not dominated by any other point [1]. A point dominates another one if it is as good or better in all dimensions and better in at least one dimension. For example, suppose that you are going to Santa Monica Beach in Los Angeles and you are asking a travel agent to pick up a hotel that is cheap and close to the beach. Points in Fig. 1 (a) depict hotels in this example. Unfortunately, the travel agent cannot decide which hotel is best for you since the hotels nearby the beach are more expensive. However, they can at least recommend you hotels that you may be interested in. In Fig. 1 (a), the hotels represented by point 6 and 7 are better than the other hotels in both dimensions. The two hotels dominate the others. In this example, these two points is the result of skyline operator.

On the contrary to skyline, reverse skyline queries retrieve all the points whose skyline contains the given query point [2-4]. For example, points from \( p_1 \) to \( p_9 \) in Fig. 1 (b) represent favorite hotels of different users. One may prefer a hotel close to the beach, but the other may not prefer the hotel because it is too close to the beach so that it might be
noisy and not compatible. Similarly, one may prefer a cheapest hotel, but the other may prefer more expensive hotels. Suppose that you have a new hotel for promotion, represented by point \( q \). Unless contacting all the customers, you may save the cost by contacting the targeted customers who may interested in the new hotel. A reverse skyline query with point \( q \) can find the customers whose skyline hotels may contain the new hotel. For example, one whose favorite hotel is \( p_2 \) must not be interested in the hotel \( q \) because hotel \( p_2 \) is closer to \( p_8 \) than the hotel \( q \) in both dimensions, which means \( p_2 \) dominates \( q \). On the contrary, one whose favorite hotel is \( p_1 \) may be interested in the hotel \( q \) because no other hotel dominates the hotel \( q \). As the same way, you can get \( p_1, p_2, p_3, p_5 \), and \( p_7 \) as a result of the reverse skyline query. The customers who prefer those hotels are possible to have some interests on the new hotel.

![Fig. 1. Example of the hotel database.](image)

To process reverse skyline queries efficiently, Dellis et al. have proposed Branch and Bound Reverse Skyline (BBRS) and Reverse Skyline using Skyline Approximations (RSSA) algorithms [2]. The BBRS algorithm has two steps. First, it computes candidates for a reverse skyline query by using a branch-and-bound skyline algorithm. Then it examines the candidates whether it can be a final result or not by issuing a window-range query over the candidates. Although the processing procedure of the BBRS is simple and intuitive, the filtering step is time-consuming. To improve the performance of the BBRS, the RSSA pre-computes all the skylines of each object and stores a selected number of points as an approximation of the resulting points. Using the points stored in the filter step, the RSSA can identify reverse skylines more efficiently. Although the RSSA performs better than the BBRS, it also suffers from serious problems. Since based on a pre-computation, it requires a large amount of storage. Although the RSSA includes an approximation technique to reduce the storage requirements, the technique also involves a trade-off between processing time and required storage space. Using a smaller number of approximation cells may reduce the storage required for the pre-computed result, but it may also increase the total processing time for a reverse skyline query because more window queries would be needed in the refinement step to find the final result using the pre-computed result. In addition, the RSSA should re-compute all the pre-processed results when data change. Therefore, it is not applicable for the RSSA to use real applications where data change often. Moreover, it is hard to use the approximation technique
PRUNING METHOD FOR REVERSE SKYLINE QUERIES

for a high-dimensional dataset because higher dimensional data requires more space. Therefore, this approach is generally used for two-dimensional datasets only [2].

To overcome these limitations, this paper proposes a new algorithm for efficiently computing reverse skyline queries, which we call Pruning-Based Reverse Skyline (PBRS). The proposed algorithm is based on two pruning methods, namely the search-area pruning method and the candidate-object pruning method. These two pruning methods filter out unnecessary objects as earlier as possible and choose much less objects as candidates. Whenever finding a new candidate, unlike previous algorithms such as the BBRS and the RSSA, the PBRS prunes candidates continuously with these two pruning methods. With these pruning methods, the proposed algorithm can reduce the number of candidates for refinement step noticeably, compared with the BBRS and the RSSA. In addition, the PBRS refine candidates without any further disk accesses while the BBRS and the RSSA refine candidates with expensive window queries over all data objects. Although the RSSA stores the approximation of pre-calculated information for the refinement step, it still needs to execute expensive window queries to refine candidates. To overcome this, the PBRS keeps a set of necessary objects in main memory, and refines candidates with the set of the objects. As a result, the PBRS outperforms the BBRS and the RSSA even though the RSSA uses a pre-processing approach.

The rest of this paper is organized as follows: Section 2 provides brief reviews of prior works for computing skylines and reverse skylines. Section 3 defines the problem and basic concepts. Section 4 proposes the PBRS algorithm with the detailed procedure. Section 5 presents experimental results to prove the efficiency and the effectiveness of the proposed algorithm. Finally, Section 6 concludes the paper.

2. RELATED WORKS

The skyline query, first proposed in [1], has been widely studied and has proven very useful in several applications, such as multi-criteria decision making, market analysis, environmental surveillance and quantitative economics research [1, 5-16]. Several algorithms have been proposed for skyline query computation: the block-nested loop algorithm [1]; the divide-and-conquer algorithm [1]; the bitmap algorithm [5]; the index-based algorithm [5]; the nearest-neighbor-based algorithm [6]; and the branch-and-bound skyline algorithm [7]. Among these, the Branch and Bound Skyline (BBS) algorithm through the R-tree index is widely used for its effectiveness and simplicity. We also extend the BBS procedure in this paper to support the reverse skyline query.

Dellis and Seeger proposed the definition of a reverse skyline query [2]. They also introduced two algorithms for processing the reverse skyline query: the Branch-and-Bound Reverse Skyline (BBRS) algorithm and the Reverse Skyline using Skyline Approximations (RSSA) algorithm. The BBRS algorithm first finds candidates for a reverse skyline query, which they called as a global skyline. Then it re-examines all the candidates whether the query point can be a skyline of them or not. This step is simple but time-consuming task. The RSSA algorithm is proposed to reduce the computation cost for the post-processing of the BBRS. RSSA pre-computes and maintains all the skylines of each object and identifies the reverse skyline points by checking the location of the given query. Note that the RSSA does not store the final answer. The RSSA stores additional pre-computed information, and uses them for refining candidates. Although the
RSSA performs better than the BBRS, it has a problem for modifying data. When data change, it should re-compute all the approximations and store them. Therefore it is not applicable for general applications with changing data.

There have been also research efforts for applying reverse skylines to a special application or environments such as data streams, uncertain databases, and spatial databases. Ling et al. defined a novel algorithm to handle a data stream that provides continuous, high-speed data elements called Divide-and-Conquer Reverse Skyline (DCRS) [4]. The DCRS uses its own index structures for computing skyline over data stream, called as Divide-and-Conquer tree (DC-tree). Xiang et al. proposed a Probabilistic Reverse Skyline (PRS) algorithm to support uncertain objects [3]. Lim et al. proposed a reverse skyline queries over metric spaces with spatial locations [17]. Han et al. proposed reverse skyline queries over regions not points [18]. These algorithms differ from this work because they need a special assumption or constraints on data such as spatial locations and region objects. As the same as [2], our work deals with general reverse skyline queries over multi-dimensional points. However, [17] defines different kinds of reverse skyline queries such as “Monochromatic Reverse Skyline” and “Bichromatic reverse skyline”, which are not compatible with general reverse skyline queries.

We also presented the basic idea of the pruning-based reverse skyline query processing in [19, 20]. However, the previous works include the result of the early stage of research only. This paper is an extended version of them, with more concrete and clear definitions, concepts, and analysis of the detailed algorithm. This paper also contains the result of an extensive experiment to prove the correctness and the effectiveness of the proposed algorithm.

3. PROBLEM DEFINITION

Before presenting a new algorithm for processing reverse skyline queries, we first define the problem formally in this section. The following is a definition of the skyline query from [2].

Definition 1 (Skyline Query) Let $D = (D^1, ..., D^d)$ be a $d$-dimensional data space and $P \subseteq D$ be a data set. A point $p \in P$ can be represented as $p = (p^1, p^2, ..., p^d)$ with $p^i \in D^i$, $i \in \{1, ..., d\}$. A point $p \in P$ is said to dominate another point $q \in P$, denoted as $p \prec q$, if (1) for every $1 \in \{1, ..., d\}$: $|q_i - p_i| \leq |q_i - p_j|$, and (2) for at least one $j \in \{1, ..., d\}$: $p_j \leq q_j$. The skyline of $P$ is a set of points $SL \subseteq P$ which are not dominated by any other point. That is, $SL = \{p \in P \mid \nexists p \in P: p \prec q\}$.

There is no query point in the basic skyline query, also called as the static skyline query, as in Definition 1. However, to define the reverse skyline query, we should have a dynamic version of the skyline query with a query point. The dynamic skyline query which has a dynamic query point can be defined as follows [2]:

Definition 2 (Dynamic Skyline Query) Given a query point $q$ and a data set $P$, a Dynamic Skyline Query (DSQ) according to $q$ retrieves all data points in $P$ that are not dynamically dominated. A point $p_1 \in P$ dynamically dominates $p_2 \in P$ with regard to the query point $q$ if (1) for all $i \in \{1, ..., d\}$: $|q_i - p_1| \leq |q_i - p_2|$, and (2) at least one $j \in \{1, ..., d\}$: $q_j \leq p_2$.
Based on the definition of dynamic skyline, the reverse skyline of a point can be defined in [2] as follows:

**Definition 3 (Reverse Skyline Query)** Let \( P \) be a \( d \)-dimensional data set. A Reverse Skyline Query (RSQ) according to the query point \( q \) retrieves all points \( p_1 \in P \) where \( q \) is in the dynamic skyline of \( p_1 \). Formally, a point \( p_1 \in P \) is a reverse skyline point of \( q \in P \) iff \( \exists p_2 \in P \) such that (a) for all \( i \in \{1, ..., d\} \) : \( |p_2^i - p_1^i| \leq |q^i - p_1^i| \) and (b) for at least one \( j \in \{1, ..., d\} \) : \( |p_2^j - p_1^j| < |q^j - p_1^j| \).

To compute the reverse skyline query efficiently, Dellis et al. defines another kind of skyline called as the global skyline, and proves that the reverse skyline set is always a subset of the global skyline set [2]. We also use the concept of the global skyline in this paper. The following is the definition of the global skyline from [2].

**Definition 4 (Global Skyline)** A point \( p_1 \in P \) globally dominates \( p_2 \in P \) with regard to the query point \( q \) if (1) for all \( i \in \{1, ..., d\} \) : \( (p_1^i - q_i)(p_2^i - q_i) > 0 \), (2) for all \( i \in \{1, ..., d\} \) : \( |p_1^i - q_i| \leq |p_2^i - q_i| \) and (3) for at least one \( j \in \{1, ..., d\} \) : \( |p_1^j - q_j| < |p_2^j - q_j| \). The global skyline of a point \( q \), \( GS(q) \), contains those points which are not globally dominated by another point according to \( q \).
also presented; (2) The RSSA should re-compute the stored dynamic skylines whenever an object is inserted, deleted or modified; (3) The approximation mechanism to reduce storage is hard to scale up for high-dimensional dataset.

To overcome these limitations, we propose a new efficient processing algorithm for reverse skyline queries. The proposed algorithm does not store or use any pre-computed data. Therefore, it requires less space and saves the processing time for pre-computation. In addition, there is no additional work for updating data. It is more suitable for real applications with data changes. The proposed algorithm is basically similar with the BBRS. It traverses the R-tree nodes with a heap data structure as branch-and-bound manner. However, while the BBRS refines a leaf object with expensive window queries over whole dataset, the proposed algorithm has continuously pruning and refining a candidate list with two pruning methods, which we called as a search area pruning method and a candidate object pruning method. We call this algorithm the Pruning-Based Reverse Skyline algorithm, or PBRS. Since the PBRS replaces expensive window queries over whole data points with an in-memory pruning steps, it outperforms the prior algorithms for reverse skyline queries including the BBRS and the RSSA.

In Sections 4.1 and 4.2, the two pruning methods are described in detail. A refinement step is presented in Section 4.3. These three sections explain the steps involved in the algorithm. To aid the reader’s understanding, we describe the complete procedure of PBRS using pseudo code in Section 5.

4.1 Search Area Pruning Method

The BBRS and the RSSA select objects which are not globally dominated by others as candidates for reverse skylines. Consequently, if an object is selected for a candidate, the region which is globally dominated by the object is excluded from a search area for further search for other candidates. For example, if we find that point \( p_8 \) in Fig. 3 (a) is a candidate, we do not have to check any object in the upper right area of \( p_8 \) because any points in the area should be dominated by \( p_8 \). Then, after finding \( p_8 \) as a candidate, the search area can be shrunk as a shadowed area in Fig. 3 (a).

However, we can shrink the search area more. While points \( p_7 \) and \( p_9 \) should be still checked after choosing the point \( p_8 \) in Fig. 3 (a) for the BBRS and the RSSA, the point \( p_9 \) should not be checked after choosing the point \( p_8 \) in Fig. 3 (a).
cannot be a candidate because the candidate point \( p_8 \) is closer to \( p_9 \) than the query point \( q \), which means \( p_8 \) dominates \( q \) dynamically for \( p_9 \). Therefore the query point \( q \) cannot be a dynamic skyline of \( p_9 \). For the result, we can define a search area of a candidate object as follows:

**Definition 5 (Search Area for a Point)** Let \( D = (D_1, \ldots, D_d) \) be a \( d \)-dimensional data space and \( p \in D \) be a data set. A point \( p \) can be represented as \( p = (p_1, p_2, \ldots, p_d) \) with \( p_i \in D_i \), \( i \in \{1, \ldots, d\} \). Given a point \( p \in P \) and a query point \( q \), \( SA(p, q) = \{ t \mid \forall t \in D \text{ and } |t_i - q_i| \leq \frac{1}{2}|p_i - q_i| \text{ for any dimension } i \in \{1, \ldots, d\} \}. Fig. 3 (b) depicts \( SA(p_8, q) \) as the shadowed area.

**Definition 6 (Search Area for Set of Points)** Let \( L = \{ p_1, p_2, \ldots, p_n \} \) be a set of points in data space \( D \). \( \text{SA}(L, q) = \text{SA}(p_1, q) \cap \text{SA}(p_2, q) \cap \cdots \cap \text{SA}(p_n, q) \). If we also select \( p_7 \) as a candidate of \( \text{RSL}(q) \) from Fig. 3 (b), \( \text{SA}(\{p_7, p_8\}, q) \) can be defined as the shadowed area in Fig. 3 (c). Therefore, we can prune any object not in the \( \text{SA} \) of the candidate list. We call this pruning method as **Search Area Pruning Method**, shortly **SAPM**. To guarantee the correctness of the SAPM, we present the Lemma 1 as follows:

**Lemma 1** Given \( L = \{ p_1, p_2, \ldots, p_n \} \) as a list of candidates of \( \text{RSL}(q) \), iff \( s \) is a point not in \( \text{SA}(L, q) \), \( s \) is not a reverse skyline point of \( q \).

**Proof:** If \( r \notin \text{SA}(L, q) \), then \( r \notin \{ \text{SA}(p_1, q) \cap \text{SA}(p_2, q) \cap \cdots \cap \text{SA}(p_n, q) \} \) from Definition 6. This means that \( r \notin \text{SA}(p_1, q) \), or \( r \notin \text{SA}(p_2, q) \) or \( \cdots \) and \( r \notin \text{SA}(p_n, q) \). Therefore, there exist \( \exists t \in L \) which satisfies \( r \notin \text{SA}(t, q) \). From Definition 5, this means that \( t \) dominates \( q \) relative to its distance to \( r \). Therefore \( r \) is not in \( \text{RSL}(q) \).

Since the search area used in the PBRS is about a half of that used in the BBRS or RSSA, we can reduce the number of candidates more efficiently. This leads to reduce expensive overheads in the refinement step. Consequently, the overall cost for computing reverse skylines decreases greatly in spite of not using any pre-processing techniques.

### 4.2 Candidate Object Pruning Method

Although the number of candidates in the PBRS is much smaller than that in the BBRS, we can reduce the number of objects in the candidate list by applying another pruning method. Whenever examining a new entry including MBRs of internal nodes or points of leaf nodes, we filter out objects from the candidate list with the entry. Since this step needs no additional disk access, it is an effective approach to reduce the number of candidates, compared to the expensive window query in the refinement step in the BBRS or the RSSA.

We define an area, named Check Area (CA), for checking an object in the candidate list whether the object can be pruned or not by an entry as follows:

**Definition 7 (Check Area)** Given a point \( p \in P \) and a query point \( q \in D \), \( CA(p, q) \) is
defined as \{ t | t \in D \text{ and } |t_i - q_i| \leq |p_i - q_i| \text{ for all } i \in \{1, \ldots, d\} \}. CA(p, q) is a rectangular region whose center point is \( p \), and the length of each side the length in each dimension is twice of the distance between \( p \) and \( q \) in the corresponding dimension.

Fig. 4 shows the CA for point \( p_5 \), \( CA(p_5, q) \) as the shadowed area.

\[
\text{Lemma 2: If } CA(p, q) \text{ includes any point in dataset } P \text{ except } p \text{ or any MBR of internal node for dataset } P \text{ except the node including } p, \text{ } p \text{ is not a reverse skyline point of } q.
\]

**Proof:** First, Let \( s \in P \) and \( s \neq p \). If \( CA(p, q) \) includes \( s \), \( s \) dominates \( q \) relative to its distance to \( p \). Therefore, \( p \) is not a reverse skyline point of \( q \). Next, Let \( m \) be a MBR of an index node for dataset \( P \). If \( CA(p, q) \) includes \( m \), there exists at least one point in \( CA(p, q) \). Therefore, for the same reason of the first case, there is a point which dominates \( q \), and \( p \) is not a reverse skyline point of \( q \).

Therefore, we can remove an object in the candidate list by using the CA. If any object or any MBR of nodes is included in CA of a candidate object, then we can remove the candidate object from the candidate list. We call this pruning method as Candidate Object Pruning Method (COPM). Lemma 2 guarantees the correctness of the COPM. As the same as in the SAPM, the COPM does not need any additional disk accesses. All the process in the COPM executed an already-fetched entry or object with in-memory list of candidates. Compared to the BBRS and the RSSA, the two proposed pruning methods, the SAPM and COPM, reduce the number of candidates effectively without any additional disk access cost.

**4.3 Refinement Step**

After traversing all the nodes in the R-tree for the above pruning phase, we should refine the candidate list because there can be false positives in the candidate list. Although the BRRS and the RSSA use a window query for this refining step, it is time consuming because it requires disk accesses over all data. Instead of window queries, we
maintain a list of objects in the main-memory for the final refinement step to avoid any window query. Pruned objects during the pruning phase can be divided into two groups. The first group consists with objects which are dominated by any object in the candidate list. Objects in this group have no need to consider for checking again in the refinement step, because the dominants of the objects can be used for the refinement step. However, the second group in which objects are not dominated by any object in the candidate, we should refine the candidate list again with this group of object. To avoid any disk accesses, we keep the second group of objects in the main-memory. Fortunately, most of objects are dominated by other object, and can be eliminated. Only a small number of objects will remain in the second group. Considered modern systems having large amount of main-memory, keeping the non-dominated list is an affordable solution for reducing the cost in the refining step.

In Fig. 5, for example, the system traverse objects with an order from \( p_1 \) to \( p_{10} \). Suppose that we have \( \{p_1, p_2, p_3, p_5, p_8\} \) as a candidate list, and now the turn for checking point \( p_9 \). Point \( p_9 \) will be discarded because it is not in \( SA(p_8, q) \), and Point \( p_8 \) will also be removed because \( CA(p_8, q) \) includes \( p_8 \). Then, we have \( \{p_1, p_2, p_3, p_5, p_7\} \) as the candidate list. Therefore point \( p_{10} \) cannot be pruned anymore, and we get \( \{p_1, p_2, p_3, p_5, p_7, p_8, p_{10}\} \) as a final candidate list. However, \( p_{10} \) cannot be a reverse skyline point of \( q \), because \( CA(p_{10}, q) \) includes \( p_9 \). Since we discard \( p_9 \) earlier than checking \( p_{10} \), this false positive has occurred. However, if we keep all non-dominated objects in the non-dominated list instead of discarding it, \( p_9 \) can be remained, and \( p_{10} \) can be successfully eliminated in the refinement step with the non-dominated list.

With two pruning methods and the modified refinement step, the PBRS needs only one traverse of an R-tree while the BBRS and RSSA need repetitive disk traverses of the R-tree because of window queries for the refinement step. This is a main reason why the PBRS outperforms previous algorithms without any pre-processing.

### 4.4 Algorithm of the PBRS

Algorithm 1 shows the main body of the algorithm. We assume that all data are in-
dexed by an R*-tree [21]. There are three types of list used in the algorithm. The RSL is a candidate list of the final result, and the C keeps any object which is or was a candidate. That is, the C includes the RSL. The ND is a list for non-dominated objects kept for the final refinement step. First, all entries of a root node are pushed into a heap sorted by distance from a query point, and then the top of the heap is sequentially popped until it is emptied. If the popped entry is globally dominated by any candidate, it will be discarded regardless of the node type. Before deleting the node, we have to discard candidates which cannot be reverse skyline points because of the popped node, using the COPM.

If the popped node is not discarded, the next step depends on the type of node. If it is an intermediate entry, we check whether each child entry of the node is globally dominated by any candidate or not. If the child is dominated, we discard it after running the COPM with the child entry. Otherwise, the child entry is pushed into the heap. In the case of leaf entry, the SAPM and the COPM are executed for pruning objects. When the heap becomes empty, the remaining objects in the reverse skyline list are checked once more through a refinement step. Finally, remaining objects in the RSL list are the result of the reverse skyline of the given query point.

Algorithm 1: Pruning-based reverse skyline

```
1: procedure PBRS(R-tree R, Query point q)
2:     RSL ← ø, C ← ø, ND ← ø
3:     // RSL = reverse skyline list, C = candidate list, ND = non-dominated list
4:     insert all entries of a root R in a heap H sorted by distance from q
5:     while H is not empty do
6:         remove top entry e from H
7:         if e is globally dominated by some point in C then
8:             CandidateObjectPruning(RSL, e, q);
9:             discard e
10:        else[e is not globally dominated]
11:            if e is an intermediate entry then // MBR
12:                for all child c of e do
13:                    if c is globally dominated by C then
14:                        CandidateObjectPruning(RSL, c, q);
15:                        discard c
16:                else
17:                    insert c into H
18:                end if
19:            end for
20:        else[e is a leaf entry] // Objects
21:            SearchAreaPruning(RSL, C, ND, e, q);
22:            CandidateObjectPruning(RSL, e, q);
23:        end if
24:     end if
25:     end while
26:     Refinement(RSL, ND, q);
27:     RETURN RSL
28: end procedure
```
Algorithm 2: Search area pruning method

```
1: procedure SearchAreaPruning(List RSL, List C, List ND, point p, Query point q)
2:   if p∈SA(C, q) then
3:     insert p into C and RSL
4:   else
5:     insert p into ND
6:     discard p
7:   end if
8: end procedure
```

Algorithm 3: Candidate objects pruning method

```
1: procedure CandidateObjectPruning(List RSL, Entry e, Query point q)
2:   for all object p in RSL do
3:     if CA(p, q) includes e then
4:       delete p from RSL
5:     end if
6:   end for
7: end procedure
```

Algorithm 4: Refinement step

```
1: procedure Refinement(List RSL, List ND, Query point q)
2:   for all object p in RSL do
3:     for all object o in ND do
4:       if CA(p, q) includes o then
5:         discard p from RSL
6:     end if
7:   end for
8: end for
9: end procedure
```

Algorithm 2 shows the SAPM, and Algorithm 3 explains the COPM. Finally Algorithm 4 is the refinement step.

5. EXPERIMENTAL RESULTS

In this section, we present several experimental results to evaluate the performance of the proposed algorithm. First, we examine the pruning capabilities of each pruning method and compare them with the RSSA algorithm which is one of the existing algorithms for computing the reverse skyline. Moreover, we survey and analyze the efficiency of the PBRS algorithm with respect to pure query response time and disk I/O accesses. The performance data were evaluated according to the following parameters: (1) number of datasets; (2) dimensions; (3) distribution of the data points.

5.1 Experimental Evaluation

For the experiments, we generated synthetic datasets as follows: The number of objects in a dataset varies from 10K to 100K. The dimension of a dataset varies from 2 to 5. Three different distributions of data, which are Uniform, Gaussian, and Anti-correlate, are used in the experiments. The range of values in each dimension is [0:100,000]. All experiments were performed with an Intel Quad-core Q6600 2.4 GHz CPU, 4GB of RAM, and a SATA ST340062 400GB hard disk. The BBRS, RSSA and PBRS algorithms were implemented in Java using JDK 1.6. We ran 100 random reverse skyline
queries and an average of the results is used for experiments. We assume that there is no disk cache in operating systems and simply count the number of read requests and write requests from the simulator for estimating the number of disk accesses.

Fig. 6 shows the average number of reverse skyline points in a dataset with 100,000 points and three different types of data distributions. While the number of reverse skyline points in the uniform dataset is generally more than that in the others, the number of result in the anti-correlated dataset increases more sharply as the number of dimension increases.

5.2 Pruning Capabilities

To study the pruning capability of the PBRS, we count the number of candidate objects remaining after each pruning method. In this experiment, we use a dataset with 100K uniformly distributed points.

In Fig. 7, GSL represents the number of candidate objects remaining after global skyline queries. The global skyline query is used for the BBRS and the RSSA as a pruning method. SAPM is the number of candidate objects when only the SAPM is applied in the PBRS. COPM is the number of candidate objects when the SAPM and the COPM are applied. Finally, RSL represents the number of reverse skyline points in the dataset. The number of candidate objects is one of the most important factors in the performance of reverse skyline queries because the refinement for the candidates is generally an expensive step. As you can see in Fig. 7, in case of the dataset with 5 dimensions, GSL makes about 5 times more candidates than SAPM does. This means that the previous approaches, the BBRS and the RSSA, should examine about 5 times more objects. The SAPM can prune more than 98% of objects. With SAPM and the COPM together, only about 1% of data objects remain as candidates. Compared with the number of the final answer, which are depicted as RSL in Fig. 7, there are much smaller false positives in the result of the SAPM and the COPM. Moreover, as the number of dimensions increases, the number of candidates in the BBRS and the RSSA increases more rapidly. That is the reason why the PBRS is less affected by the increase of data dimension.
5.3 Query Response Time

In this experiment, we measured the response time of each algorithm with varying the number of dimensions and the number of objects in dataset. The response time does not include any pre-processing time such as building indexes.

Fig. 8 depicts the response time of each algorithm with 100K uniformly–distributed points. For the two-dimensional dataset, all three algorithms showed similar performance. However, the dimension of objects increases, the performance of the BBRS and the RSSA are rapidly degraded. On the other hand, the PBRS shows much stable performance although the response time increases slowly. For the 5-dimensional dataset, the PBRS is about 24.43 times faster than the RSSA and about 125.55 times faster than the BBRS. As the number of objects increases, the response time of the BBRS and the RSSA also increases rapidly. However, the response time of the PBRS is less affected by the increase of the number of data. This is because the number of candidates in the PBRS is much smaller than that in the others and the PBRS process all the pruning and refining step with in-memory lists. As the dimension or the number of data increases, the number of candidates in the BBRS and the RSSA increases sharply as you can see in Fig. 7. Moreover they should execute an expensive window query over each candidate to refine the result. Therefore, the performance of the BBRS and the RSSA deteriorates rapidly as the dimension or the number of data increases.

![Fig. 8. Query response time vs. Dimensions and sizes (uniform).](image)

![Fig. 9. Query response time vs. Distribution (N = 50,000).](image)
Fig. 9 shows the result of the experiments with different distributions of data. The PBRS outperforms the BBRS and the RSSA for all of the cases. For the anti-correlate data in Fig. 9 (b), the performance of the RSSA is worse than the BRRS. This is because the approximation in the RSSA is generally inefficient for the anti-correlate data.

5.4 Number of Disk Accesses

One of the reasons why the PBRS outperforms the others in large data with higher dimension is that the PBRS needs less number of disk accesses because it does not execute window queries for refinement step. Instead of executing window queries over all data point, the PBRS refines the result by comparing candidates with an in-memory list of objects as in Algorithm 4. Although maintaining an in-memory list requires more spaces, it is effective to reduce number of disk accesses, which is one of the main causes of the performance deterioration.

![Fig. 10. The number of node access vs. Dimensions.](image)

Fig. 10 (a) shows the number of disk accesses for the RSSA and the PBRS with uniformly distributed datasets of 50K objects. Although both of the number of disk accesses increase as the dimension increases, the PBRS needs less disk accesses than the RSSA and the gap between two algorithms becomes wider as the dimension increases. For the 5-dimensional data, the PBRS requires about 33% less disk accesses than the RSSA. Fig. 10 (b) shows the effect of data size. The number of disk accesses is directly proportion to the number of dimensions and the number of data points.

5.5 Summary of Experimentations

The experiments show that the PBRS outperforms the BBRS and the RSSA. The PBRS is more effective for the large data set with higher dimension than other competitors. The main reason for performance improvements in the PBRS is using more effective pruning methods and avoiding unnecessary disk accesses. First, the PBRS filter out unnecessary data very effectively by using two pruning methods, the SAPM and the
PRUNING METHOD FOR REVERSE SKYLINE QUERIES

COPM. Compared to the BBRS and the RSSA, the candidate list of the PBRS contains only few false positives. Consequently the PBRS can reduce the cost of the expensive refinement step as much as possible. Section 5.2 shows the comparison of the pruning performances. Second, the PBRS uses an in-memory refinement step. It keeps a set of necessary-but-discarded objects in memory, and refines candidates with the set. Therefore the PBRS does not need additional disk accesses for the refinement step while the RSSA and the PBRS needs a lot of disk accesses for the refinement step. Section 5.4 shows the comparison of the number of disk accesses. With these two factors, the PBRS shows a better performance even in case of the higher dimensional data or large dataset, as seen in Section 5.3.

Although the RSSA uses an approximation and the pre-processing, it cannot precompute the final answer. The RSSA only store pre-computed dynamic skylines with an approximation technique, and use the dynamic skyline as another filter for candidates. Therefore, the RSSA still need to compute global skylines and refine the candidate as in the BBRS. However, by using more efficient pruning methods and an in-memory refinement step, the PBRS shows a superior performance than the RSSA without any approximation or preprocessing.

6. CONCLUSION

Reverse skyline queries retrieve a set of points that have a query point in their dynamic skyline. Several algorithms including the BBRS and RSSA algorithms are proposed for computing the reverse skylines, but they have some limitations. In this paper, we have proposed an efficient method for computing the reverse skyline based on pruning methods, which we call pruning-based reverse skyline algorithm, shortly PBRS. The PBRS uses two efficient pruning methods, which is the SAPM and the COPM. These pruning methods effectively filter out unnecessary objects and minimize the number of candidates. The PBRS also uses an in-memory refinement step. Since the PBRS keeps necessary objects in memory during the pruning step, there is no need to access disk for refinement step. The experimental result shows that the PBRS has a better performance than the BBRS and the RSSA. The PBRS is less affected by the increase of the dimension and the number of data.

In future work, we plan to study reverse skyline queries for datasets with the dynamic properties, and efficient clustering and parallelizing methods for reverse skyline queries for big data analysis.

REFERENCES

3. X. Lian and L. Chen, “Monochromatic and bichromatic reverse skyline search over uncertain databases,” in Proceedings of the International Conference on ACM Spe-


Ah Han received her B.S., M.S., and Ph.D. degrees in Computer Science from Myongji University, Korea, in 2007, 2009, and 2012, respectively. Her current research interests include preference queries, data mining and spatio-temporal databases.

Youngbae Park was a Professor in the Department of Computer Engineering at Myongji University, Korea, until 2012. He received the Ph.D. in Computer Science from Seoul National University, Korea, in 1993. His main research interests include data mining, data warehouses, and spatial databases.

Dongseop Kwon received his B.S., M.S., and Ph.D. degrees in Computer Science from Seoul National University, Korea, in 1998, 2000, and 2005, respectively. After graduating, he worked at Samsung Electronics Co., Ltd., for a year. He joined the Department of Computer Engineering, Myongji University in 2006, where he is currently an Associate Professor. His current research interests include database systems, data mining, and bioinformatics.