Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 649271, 19 pages
http://dx.doi.org/10.1155/2015/649271

Research Article

Reverse Skyline Computation over Sliding Windows

Junchang Xin,[1] Zhiqiong Wang,[2] Mei Bai,[1] and Guoren Wang[1]

[1]*College of Information Science and Engineering, Northeastern University, Shenyang 110819, China*
[2]*Sino-Dutch Biomedical and Information Engineering School, Northeastern University, Shenyang 110169, China*

Correspondence should be addressed to Junchang Xin; xinjunchang@ise.neu.edu.cn

Received 9 May 2015; Accepted 25 November 2015

Academic Editor: Tadeusz Kaczorek

Reverse skyline queries have been used in many real-world applications such as business planning, market analysis, and environmental monitoring. In this paper, we investigated how to efficiently evaluate continuous reverse skyline queries over sliding windows. We first theoretically analyzed the inherent properties of reverse skyline on data streams and proposed a novel pruning technique to reduce the number of data points preserved for processing continuous reverse skyline queries. Then, an efficient approach, called Semidominance Based Reverse Skyline (SDRS), was proposed to process continuous reverse skyline queries. Moreover, an extension was also proposed to handle n-of-N and (n_1, n_2)-of-N reverse skyline queries. Our extensive experimental studies have demonstrated the efficiency as well as effectiveness of the proposed approach with various experimental settings.

1. Introduction

The skyline operator [1] and its variations [2–15] have been widely applied in many applications involving multicriteria decision-making. Specifically, given a set P of points, the *skyline* of P comprises all the points which are not *dominated* by any other point in P. Generally, a small value is assumed to be better in all dimensions. Here, we say a point x *dominates* another one y, if x is not worse than y for all dimensions ($\forall i,\ x[i] \leq y[i]$), and x is better than y for at least one dimension ($\exists j,\ x[j] < y[j]$). Figure 1 illustrates an example of skyline in a 2-D space, where points a and g on the bottom-left line are the skyline points.

As a variation of skyline, given a query point, *dynamic skyline* [4–7] contains all the points which are near to the query point in all dimensions. As illustrated in Figure 1, the query point is $b = (b[1], b[2])$, and the dynamic skyline with respect to b is $\{h, q, c\}$. The other points are farther to b than h (or q, c) in all dimensions.

Compared with dynamic skyline, *reverse skyline* is proposed from an opposite perspective. If q is one of the dynamic skyline points of x, then x is called a *reverse skyline point* of q [7]. As shown in Figure 1, b is a reverse skyline point of q, since q is contained in the dynamic skyline of b. The reverse skyline query is very useful for many applications. For example, in an online used car trading system, each used car is evaluated in various aspects such as brand, price, and mileage. If a dealer wants to take a new used car to sell, it is desirable for the car to attract as many customers as possible. Intuitively, if a customer is interested in an existing car in the trading system, s/he might be also interested in the dynamic skyline cars of this car. Therefore, we can conduct a reverse skyline query with respect to the new one to retrieve all the cars in the system and find the cars whose dynamic skyline contains the new one. The bigger the number of reverse skyline is, the better it is.

In such applications, the dealer may want to continuously monitor the trading system for selecting the customers who will recommend the new used car. As the price of used cars is always fluctuant, the information too long before may not be quite relevant to the current used car recommendation. Therefore, we tend to only focus on the most recent (e.g., within a week) used car information, that is, the reverse skyline query over sliding windows.

Although reverse skyline query processing has been well studied in recent years [7, 10–13], there is only one previous

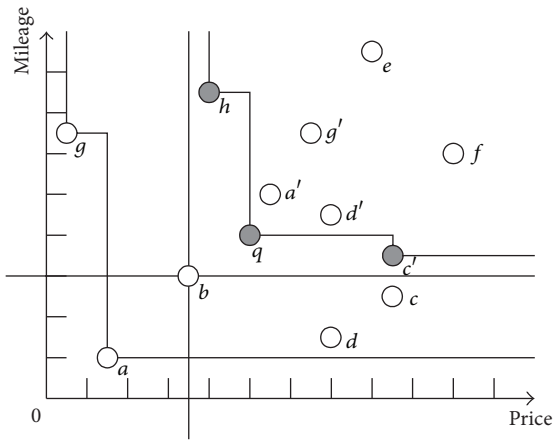


Figure 1: Example of skyline and its variations.

work, *Divide-and-Conquer Reverse Skyline* (DCRS) [16], that has studied the reverse skyline over sliding windows. Once the sliding window moves, a new point is inserted into the window and the oldest point is expired from the window. Reverse skyline query over the sliding window will return the reverse skyline result in the current sliding window once the sliding window moves. The DCRS algorithm needs to preserve all points in the sliding window, which will consume a lot of memory space, and the maintenance of such a large number of points usually consumes a large amount of CPU time. In this paper, we propose an efficient algorithm, called *Semidominance Based Reverse Skyline* (SDRS), to continuously answer the reverse skyline on data streams. Our contributions can be summarized as follows:

(i) We propose an efficient algorithm SDRS to process the reverse skyline queries over the sliding window. By using the semidominance relationships and first-in-first-out property of the sliding window, SDRS only maintains a small number of points in the sliding window. Also, by maintaining a reasonable structure of each reserved point, SDRS can quickly calculate the new reverse skyline once the sliding window moves.

(ii) By building and maintaining a 2D R-tree, SDRS is easily extended to deal with n-of-N and (n_1, n_2)-of-N reverse skyline queries over the sliding window.

(iii) Last but not least, extensive experiments show that our proposed SDRS approach can efficiently support continuous reverse skyline queries, including n-of-N reverse skyline queries and (n_1, n_2)-of-N reverse skyline queries.

2. Related Work

Dynamic skyline was first introduced by Papadias et al. [4, 5]. Given a query point, dynamic skyline can return all the points which are near to the query point in all dimensions. Sharifzadeh and Shahabi [17] proposed the concept of spatial skyline queries, and the derived spatial attributes of each point were defined by Euclidean distances from the point to some query points. Deng et al. [6] presented the multisource skyline query in the application of road networks, in which the dynamic attributes of each mapped point were defined as the relative network distances to multiple query points. Chen and Lian [18, 19] proposed generic metric skyline query, and the dynamic attributes of each point were defined in arbitrary metric space. Dellis and Seeger [7] considered the case that all the dynamic attributes were the absolute coordinate differences to the query point. For the sake of simplicity, in this paper, we adopt the definition of dynamic attributes the same as Dellis and Seeger [7].

Based on the concept of dynamic skyline, Dellis and Seeger [7] proposed the reverse skyline and presented an effective pruning method to reduce the search space of the reverse skyline computation. Lian and Chen [10, 11] formalized both monochromatic and bichromatic probabilistic reverse skyline query on uncertain data and proposed effective pruning methods to reduce the search space of query processing. Wu et al. [13] investigated the bichromatic reverse skyline on precise data and proposed several nontrivial heuristics that can optimize the access order of R-tree to reduce the I/O cost considerably. Wang et al. [20] and Min [21] investigated the reverse skyline queries in wireless sensor networks. Lim et al. [22–24] proposed a new reverse skyline query processing method that processed a query over moving objects or a continuous reverse skyline query efficiently. The proposed algorithm makes a verification range to guarantee the result of the reverse skyline query, whenever the new objects appear or moving objects move. A novel algorithm is proposed in [25], which is an efficient method to process the reverse skyline, by using an approach based on two pruning methods, the search-area pruning method and the candidate-objects pruning method, but in which it should refine the candidate list because there can be false positives in the candidate list after pruning phase. The reverse skyline processing techniques above cannot be used directly in rapid updated data stream applications.

There are some works that have been proposed to address the skyline query processing on data streams. Lin et al. [26] explored the problem of computing skyline against various different sliding windows. Tao and Papadias [27] studied the skyline computation in stream environments and developed efficient techniques to improve space/time efficiency. Morse et al. [28] introduced the continuous time-interval skyline operator to continuously compute the current skyline on a data stream and presented a LookOut algorithm for evaluating such queries efficiently. Sarkas et al. [29] studied the streaming categorical skylines and proposed some novel techniques for maintaining the skyline of categorical data in a streaming environment. Zhang et al. [30] investigated the problem of minimizing the communication overhead in client-server architectures, where the server continuously maintains the skyline of dynamic objects. Zhang et al. [31, 32] studied the problem of continuous, probabilistic skyline query and proposed some novel, efficient techniques to improve the efficiency. However, the existing works usually focus on skyline query processing on data streams, whereas ours concentrates on a more complex query (i.e., reverse skyline) processing on data streams.

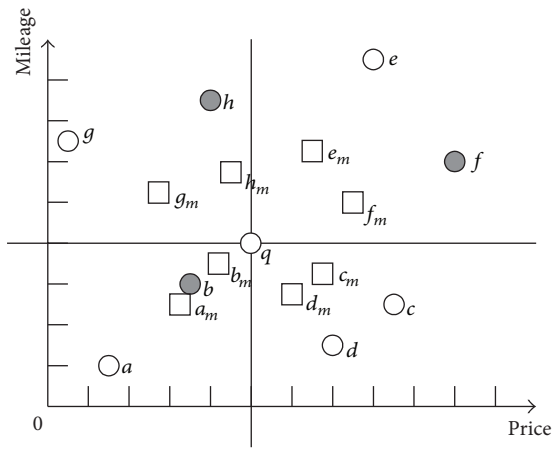


Figure 2: Full-dominance and semidominance.

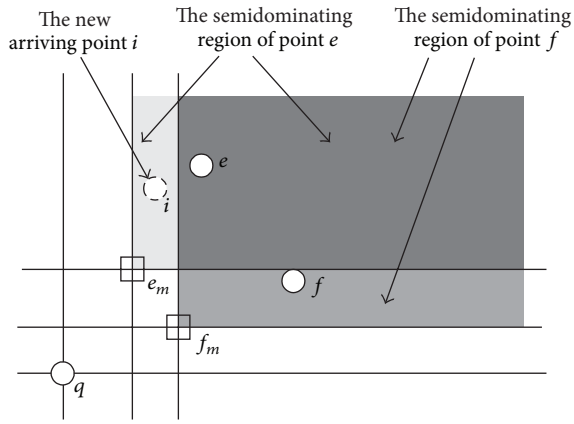


Figure 3: An example of false positive.

Bai et al. [33] proposed a probabilistic reverse skyline algorithm over the sliding window by using some probability pruning methods. Liu et al. [34] proposed an algorithm to process reverse k-skyband. The reverse skyline computation on data streams was first studied by Zhu et al. [16]. They proposed an efficient algorithm DCRS to process continuous reverse skyline query by employing an improved DC-Tree index [35]. Firstly, the dataset in sliding window is divided into 2^d pieces according to query point q, and a modified DC-Tree is constructed for each piece of dataset. Secondly, the modified DC-Tree recursively divides its dataset into 2^d equal pieces until the terminate condition is met. The intermediate nodes maintain the local first and second skylines by merging the corresponding results in their children, while the leaf nodes store the data points directly. Finally, DCRS algorithm runs a window query on each global skyline point to get the current reverse skyline when new arrival point belongs to global first or second skylines. Obviously, it is necessary to preserve all the data in sliding window for DCRS algorithm, which costs not only a lot of memory space but also a lot of CPU time. As a result, in this paper, we focus on reducing the number of the data points to be preserved and improve the space/time efficiency for computing continuous reverse skyline as a consequence. Moreover, an extension is also proposed to handle n-of-N and (n_1, n_2)-of-N reverse skyline queries.

3. Problem Statement

We first recall two important concepts called *full-dominance* and *semidominance* [20], respectively.

Definition 1 (full-dominance [20]). A point x full-dominates y with regard to query point q ($x \overline{\preceq}_q y$), if it holds that (1) $\forall i \in D(x[i] - q[i])(y[i] - q[i]) \geq 0 \wedge |x[i] - q[i]| \leq |y[i] - q[i]|$ and (2) $\exists j \in D(x[j] - q[j])(y[j] - q[j]) > 0 \wedge |x[j] - q[j]| < |y[j] - q[j]|$.

Definition 2 (semidominance [20]). A point x semidominates y with regard to query point q ($x \dot{\preceq}_q y$), if it holds that

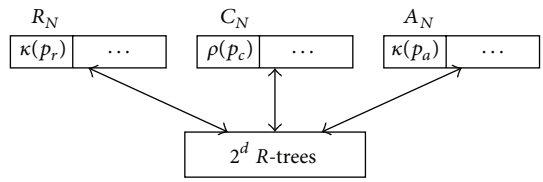


Figure 4: Data structure.

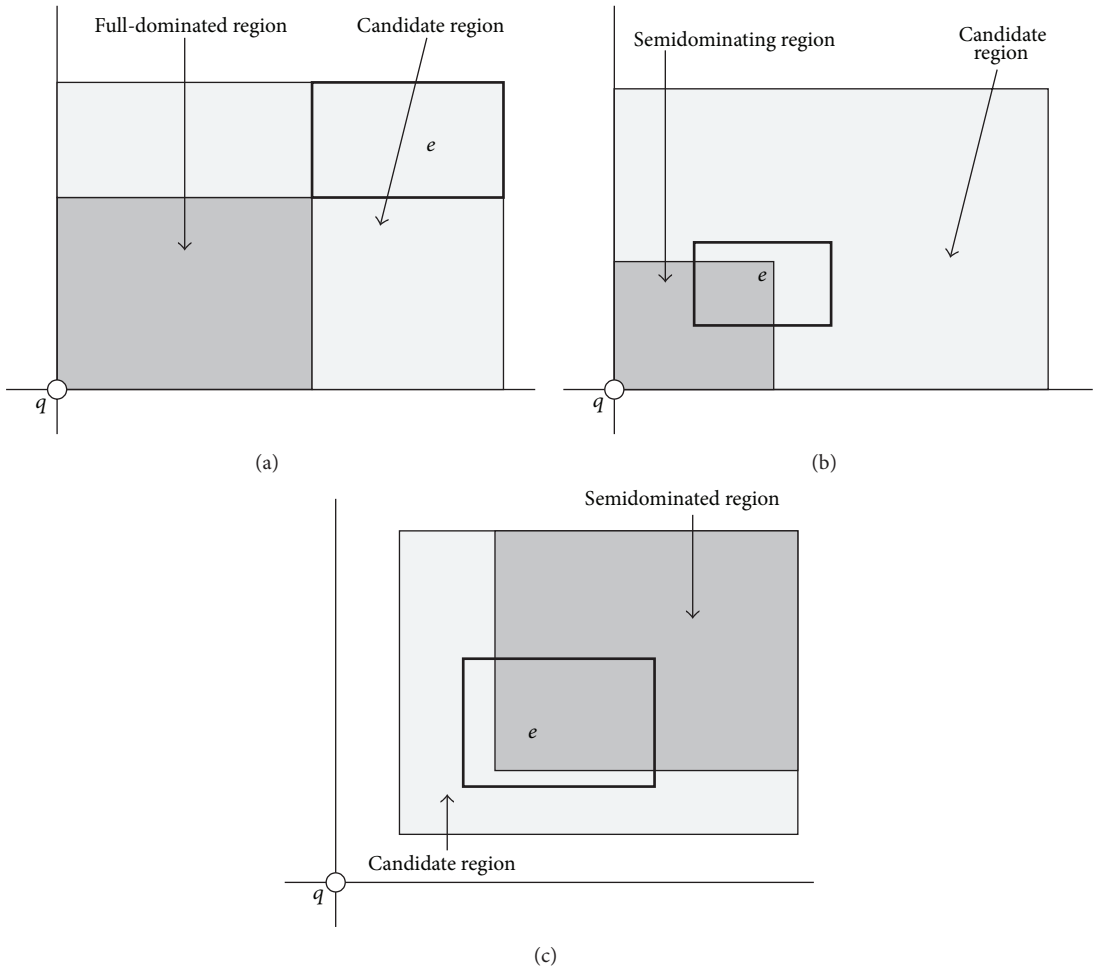
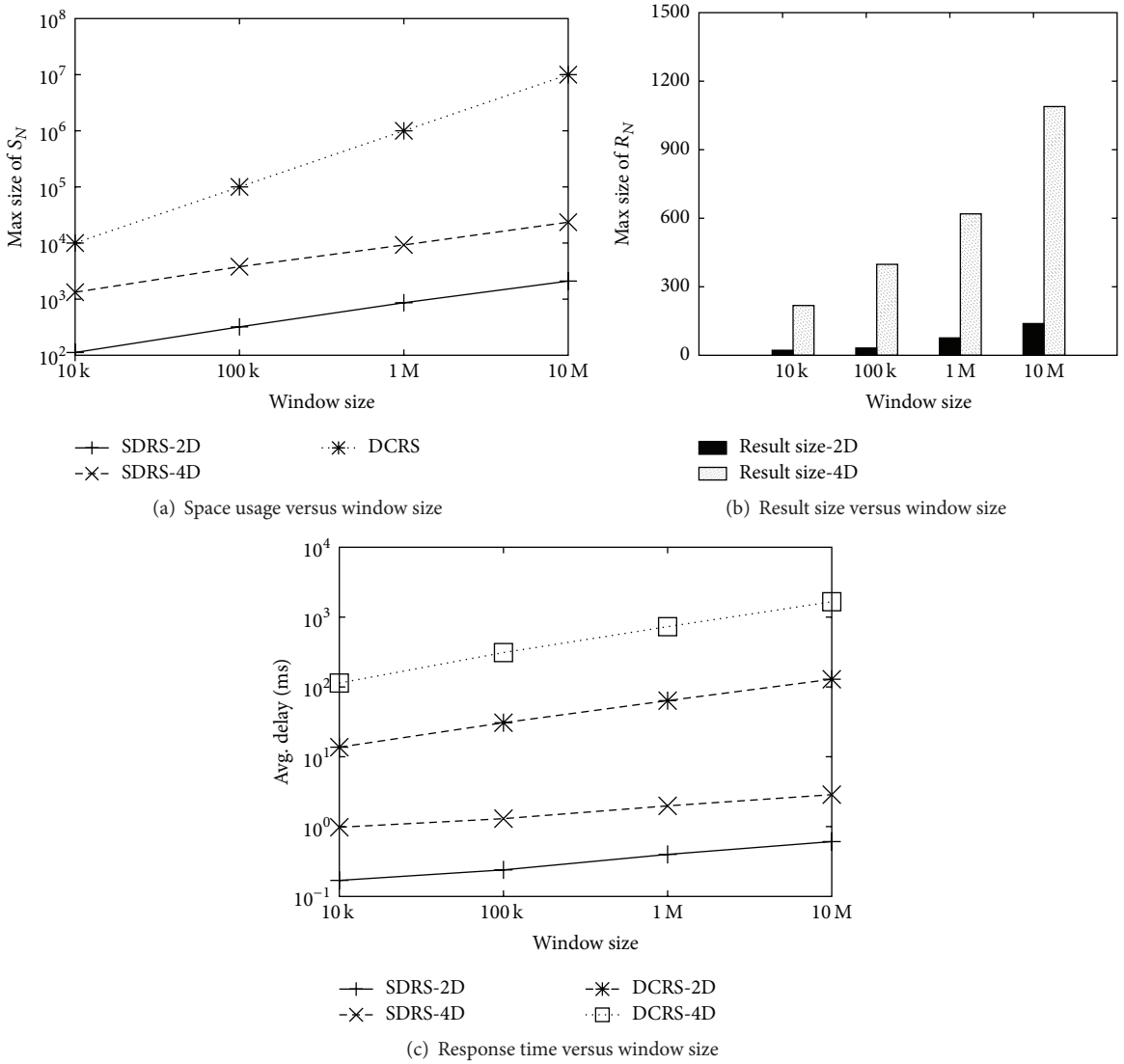


Figure 5: Dominance relationships.

(1) $\forall i \in D(x[i] - q[i])(y[i] - q[i]) \geq 0 \wedge |x[i] - q[i]| \leq 2|y[i] - q[i]|$ and (2) $\exists j \in D(x[j] - q[j])(y[j] - q[j]) > 0 \wedge |x[j] - q[j]| < 2|y[j] - q[j]|$.

Figure 2 illustrates an example of full-dominance and semidominance on dataset $\{a, b, c, d, e, f, g, h\}$ in a 2-D space. We can see that point b full-dominates a with regard to q, since a and b are in the same side of q, and meanwhile b has smaller distance to q than a on both dimensions. Though b has smaller distance to q than e on both dimensions, point b does not full-dominate e with regard to q, because they are not in the same side of q. Similarly, the points, that is, $b - h$, are not full-dominated by any others.

In order to explain semidominance more clearly, for each point $x \in \{a, b, c, d, e, f, g, h\}$, we use foursquare point x_m to stand for the midpoint between the original point x and query point q. We can see that point f semidominates e with regard to q, since the midpoint f_m between f and q full-dominates e. Similarly, point b semidominates a, point c semidominates d, point d semidominates c, and point h semidominates g, while the solid points b, f, and h are not semidominated by any others.



(a) Space usage versus window size

(b) Result size versus window size

(c) Response time versus window size

FIGURE 6: Experimental results of real dataset-stock.

It has been theoretically proved in [20] that we can make use of semidominance to conduct reverse skyline queries, as shown in Theorem 3.

Theorem 3 (see [20]). *Given a query point q and a dataset P, any point $p \in P$ is a reverse skyline point of q, if there does not exist any other point $p' \in P$ such that $p' \dot{\preceq}_q p$.*

According to Theorem 3, all the points that are not semidominated by others constitute the reverse skyline. We use P to present an append-only data stream in a d-dimensional space D (i.e., $d = |D|$), and each element (point) $x \in P$ has a label $\kappa(x)$ to indicate its position in P. Let P_N denote the most recent N elements in P. Then, the reverse skyline on data stream can be formally defined as in Definition 4.

Definition 4 (reverse skyline on data stream). Given a query point q and a data stream P, a *reverse skyline on data stream* according to query point q continuously retrieves all the points in the most recent N points P_N that are not semidominated by any other points in P_N.

4. Semidominance Based Reverse Skyline

In this section, we present the details of SDRS approach. Specifically, some important properties and query processing techniques are discussed in Sections 4.1 and 4.2, respectively.

4.1. Preliminaries. Suppose there are two points p and p' in data stream P such that p' semidominates p. According to Definition 4, point p is not a reverse skyline point if p' belongs to P_N. As a result, point p will never be a reverse skyline point if point p' is younger than p. Therefore, we can get Lemma 5 as follows.

Lemma 5. *A data point p will never be a reverse skyline point if it is semidominated by a younger point p'.*

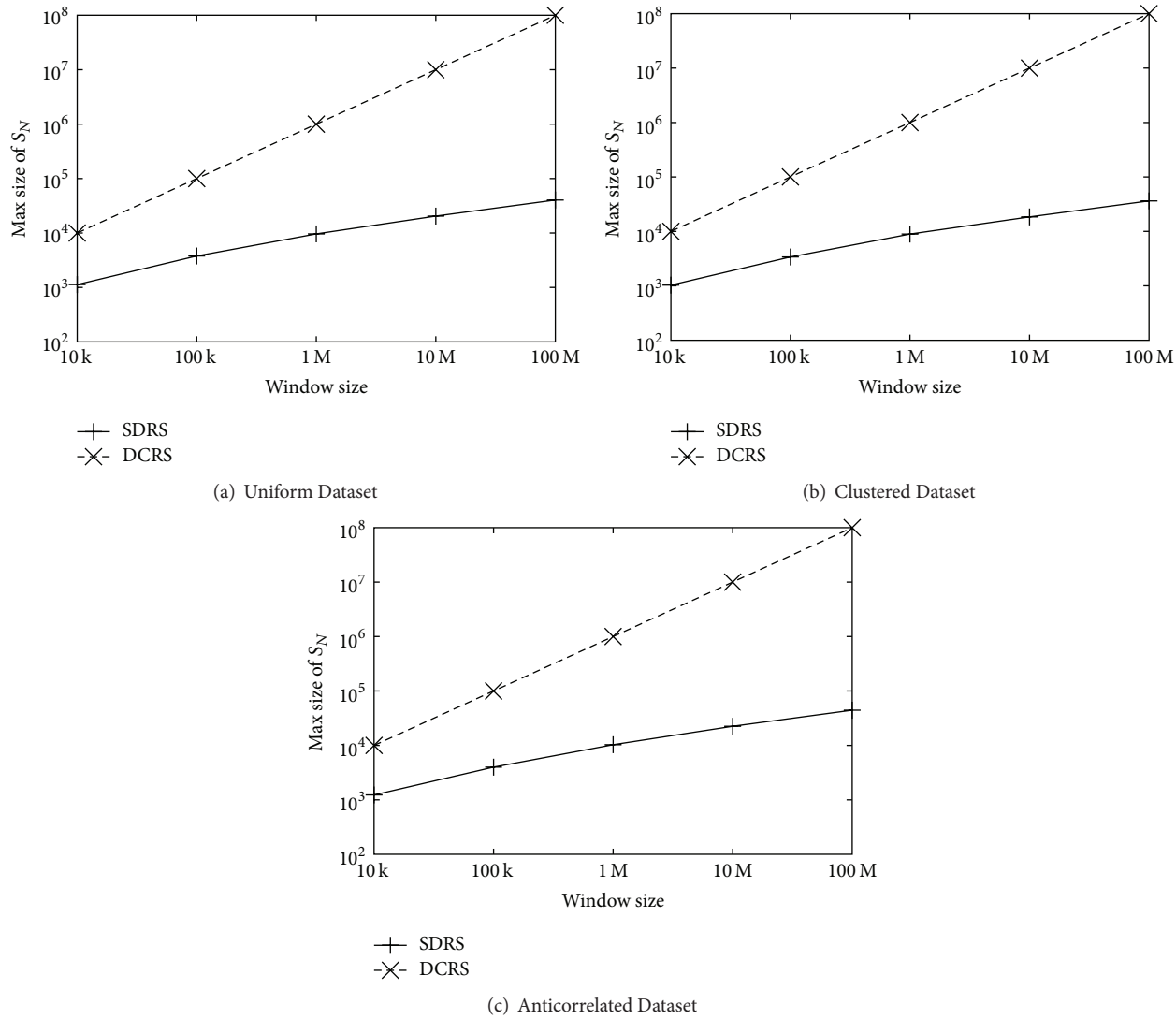


Figure 7: Space usage versus window size.

Proof. Since we have $p' \dot{\preceq}_q p$, point p is not a reverse skyline point when $p' \in P_N$. Moreover, since we also have $\kappa(p') > \kappa(p)$, p will expire earlier than p'. Therefore, point p will never be a reverse skyline point. □

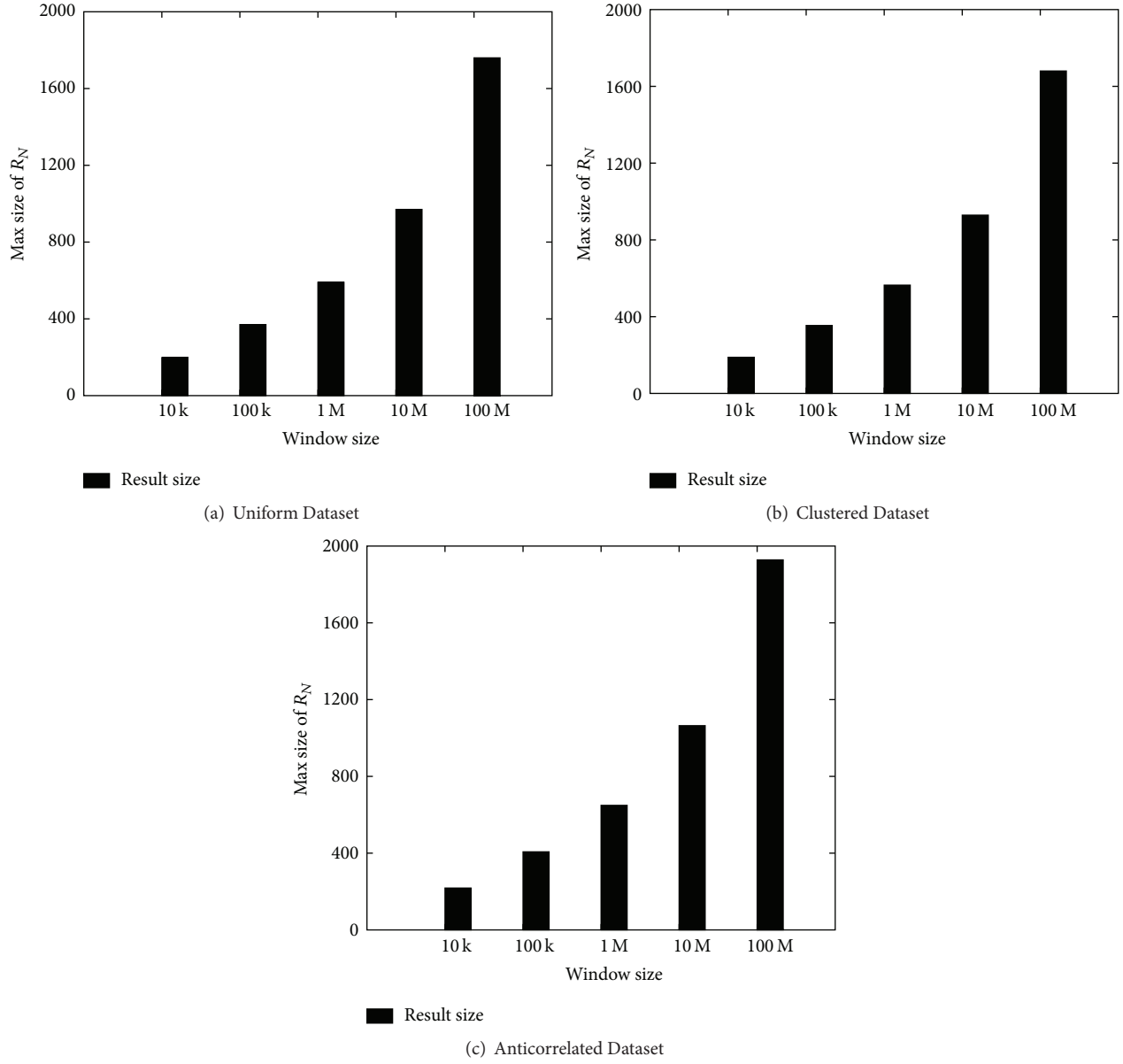
Consider the example in Figure 2 where points arrive according to their alphabetic ordering. Suppose point h just arrives and the length of sliding window is 8; then current $P_N = \{a, b, c, d, e, f, g, h\}$. Since it is semidominated by a younger point (i.e., point b), point a will never be a reverse skyline point on the basis of Lemma 5. Similarly, points c, e, and g are semidominated by some younger points; all of them cannot be the reverse skyline points. Obviously, it will not result in any false negative after abandoning any of a, c, e, and g but may lead to false positive if all of them are discarded.

As illustrated in Figure 3, now points e, f are in P_N; f can semidominate e and $\kappa(e) < \kappa(f)$. When a new point i arrives, point i would be judged as a reverse skyline point if we only preserve point f; f cannot semidominate i. However, the truth is quite opposite; point i is not a reverse skyline point as point e semidominates i. Moreover, point i cannot be a reverse skyline point before point e expires. The reason for this problem is that the preserved points cannot semidominate all the points which are semidominated by the discarded points. The new point would be considered as a reverse skyline point when it is semidominated by the discarded points but not semidominated by the preserved points.

Now, we will present a very important characteristic of semidominance which helps us to solve the above problem.

Lemma 6 (see [20]). *If $x \overline{\preceq}_q y$ and $y \dot{\preceq}_q z$, then $x \dot{\preceq}_q z$.*

Lemma 6 shows that if one point x can full-dominate another point y, all the points which are semidominated by y can also be semidominated by x. Therefore, the false positive problem could be avoided if all the points which are not full-dominated by a younger point are preserved. Let S_N denote



(a) Uniform Dataset

(b) Clustered Dataset

(c) Anticorrelated Dataset

Figure 8: Result size versus window size.

the set of points which are not full-dominated by a younger point; that is,

$$S_N = \left\{p \mid p \in P_N, \nexists p' \in P_N,\ \kappa\left(p'\right) > \kappa\left(p\right) \wedge p' \overline{\preceq}_q p\right\}. \quad (1)$$

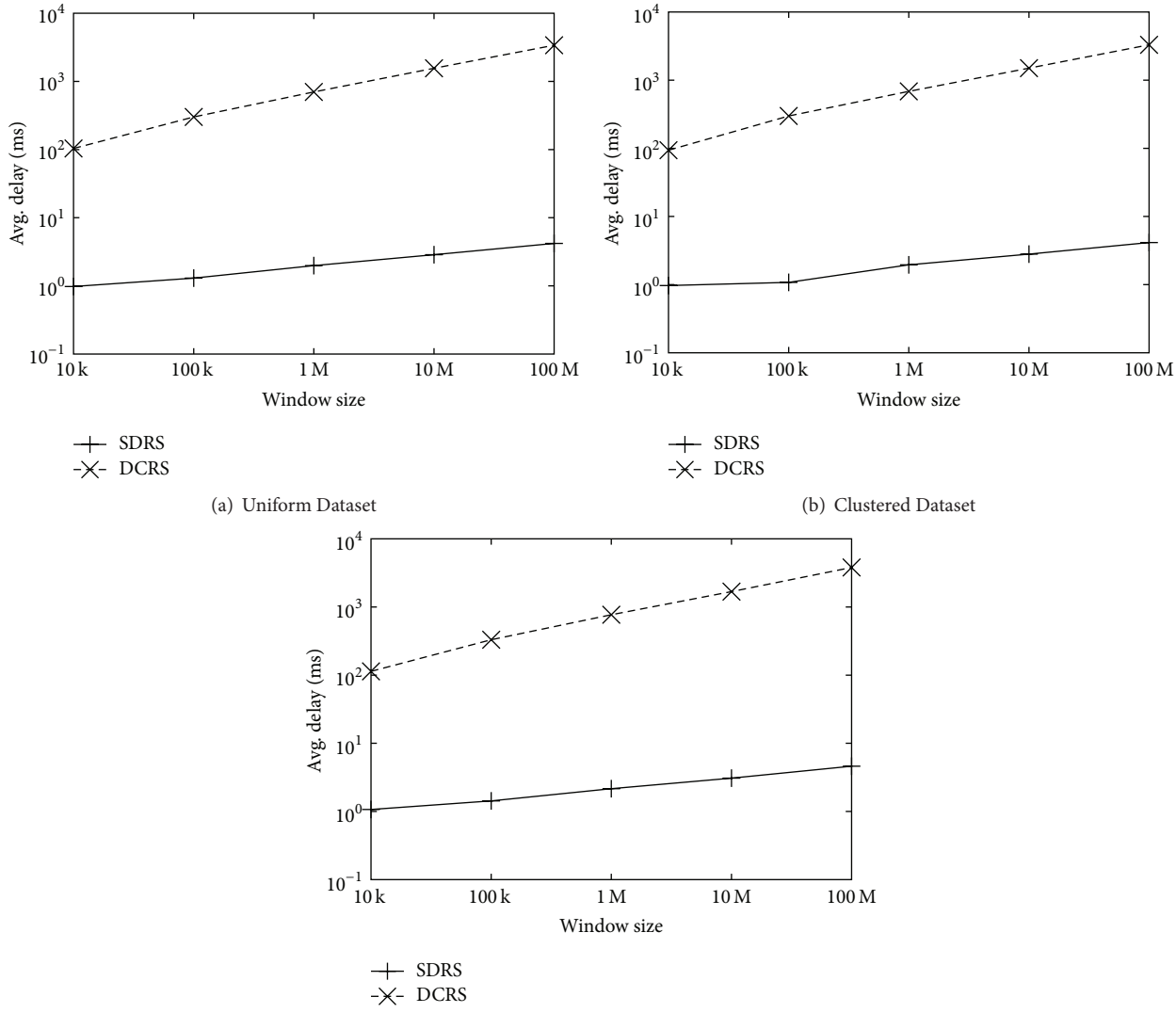
Then, the data points in S_N can be divided into three categories:

(i) *Reverse Skyline Point (RP)*. The data point which is not semidominated by any other point in P_N.

(ii) *Candidate Point (CP)*. The data point which is semidominated by an older point rather than a younger point in P_N.

(iii) *Assistant Point (AP)*. The data point which is semidominated by a younger point in P_N but not full-dominated by any younger point in P_N.

We use R_N, C_N, and A_N to denote the sets of reverse skyline points, candidate points, and assistant points, respectively. Continue the example in Figure 2, since only point a is full-dominated by the point which is younger than it (i.e., point b), $S_N = \{b, c, d, e, f, g, h\}$. Points b, f, and h are not semidominated by any other point; therefore they are reverse skyline points; namely, $R_N = \{b, f, h\}$. And point d is a candidate point because it is semidominated by the older point c; namely, $C_N = \{d\}$. The remaining c, e, and g



(a) Uniform Dataset

(b) Clustered Dataset

(c) Anticorrelated Dataset

Figure 9: Response time versus window size.

are assistant points as each of them is semidominated by a younger point; namely, $A_N = \{c, e, g\}$.

A candidate point p tends to be semidominated by several points, and it will still be a candidate point if any of them belongs to P_N. Apparently, the youngest one among them is the one who determines when point p can become a reverse skyline point, calling it the *predecessor* of p (denoted as pre(p)). Meanwhile, we set an attribute $\rho(p)$ for p to record the position of pre(p); namely, $\rho(p) = \kappa(\text{pre}(p))$.

Next, we will discuss the correctness of only keeping the data points in S_N.

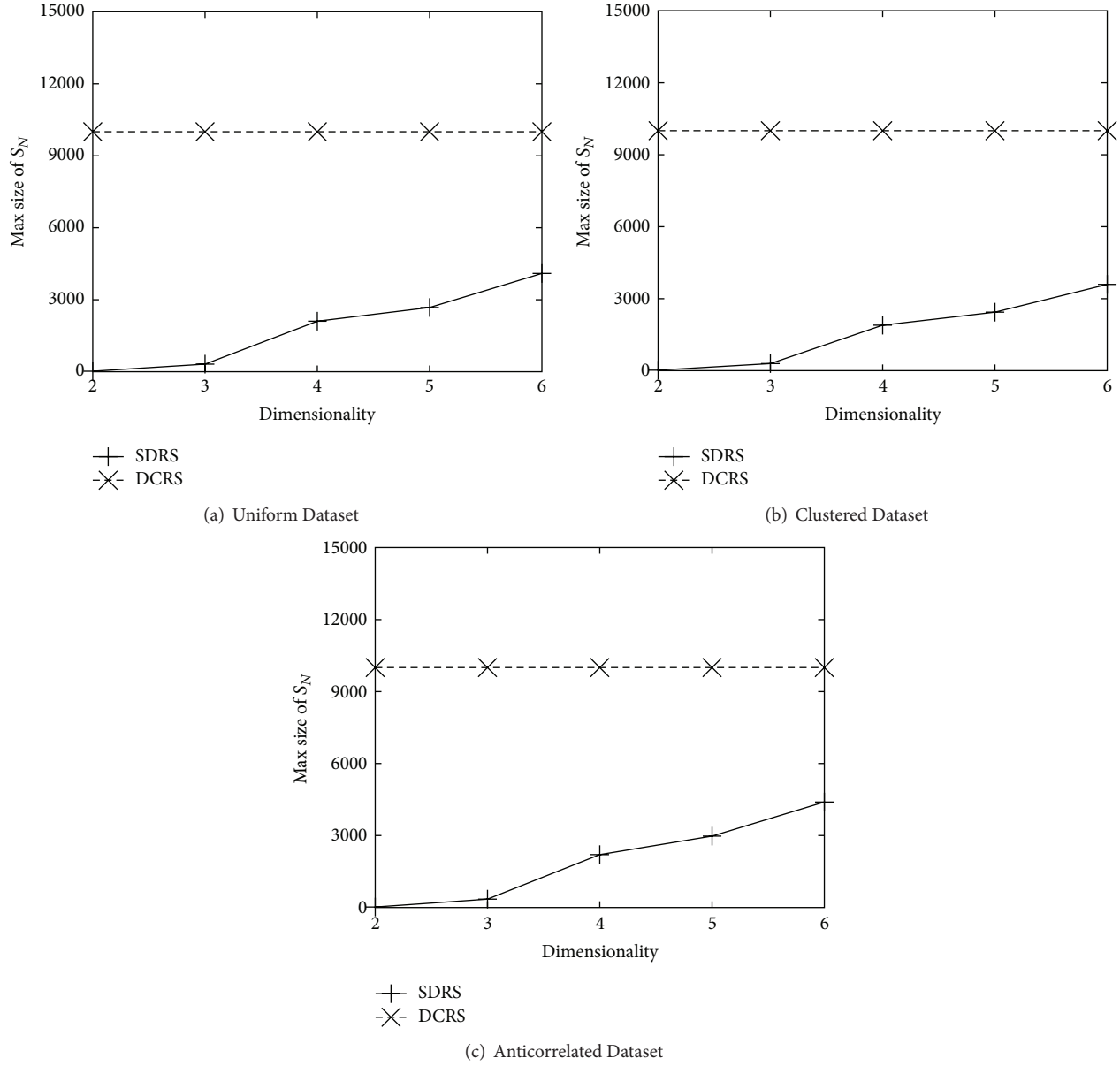
Lemma 7. *A data point p will never be a reverse skyline point if it is full-dominated by a younger point p'.*

Proof. It can be immediately deducted from Lemmas 6 and 5. □

Theorem 8. *A newly arriving point p must be a reverse skyline point or a candidate point. Moreover, if p is a candidate point, then its predecessor* pre(p) *must belong to S_N.*

Proof. Since p is a newly arriving point and the points which are younger than p do not exist in P_N, p is either a reverse skyline point or a candidate point.

We use apagoge to prove the second part. Suppose point p is a candidate point, while its predecessor pre(p) does not belong to S_N. According to the definition of S_N, there must be a point p' in P_N which is younger than pre(p) satisfying $p' \overline{\preceq}_q \text{pre}(p)$. According to Lemma 6, we can infer $p' \overline{\preceq}_q \text{pre}(p) \wedge \text{pre}(p) \dot{\preceq}_q p \Rightarrow p' \dot{\preceq}_q p$. We have $\kappa(p) > \kappa(p') > \kappa(\text{pre}(p))$ and $p' \dot{\preceq}_q p$, which contradict the prerequisite that pre(p) is the predecessor of point p. Therefore, the proof is completed. □



(a) Uniform Dataset

(b) Clustered Dataset

(c) Anticorrelated Dataset

Figure 10: Space usage versus dimensionality.

Lemma 7 shows that a discarded point will never be a reverse skyline point. Thus, false negative will not happen if we preserve all the points in S_N. Theorem 8 shows that whether the new arriving point is a reverse skyline point or a candidate point can be decided by the points in S_N. And if this point is a candidate point, we can get its predecessor correctly according to S_N and also know when it may become a reverse skyline point. Therefore, only preserving the points in S_N cannot result in any false positive.

Based on the analysis above, we can get Theorem 9 directly.

Theorem 9. *S_N is the minimum information that needs to be preserved to process a continuous reverse skyline query over sliding windows.*

Proof. It can be immediately deducted from Theorems 9 and 8. □

4.2. SDRS Algorithm. In this section, we will introduce the data structures and details of our SDRS approach successively.

Data Structure. The whole data space is divided into 2^d different regions. Full-dominance relationship and semidominance relationship only exist between two data points when they are in the same region according to Definitions 1 and 2. When a new data point arrives, we just need to figure out its full-dominance relationship and semidominance relationship with the points whose region is just the same one as the new arriving point's (when the new arriving point has the same values as the query point's in some dimensions, it

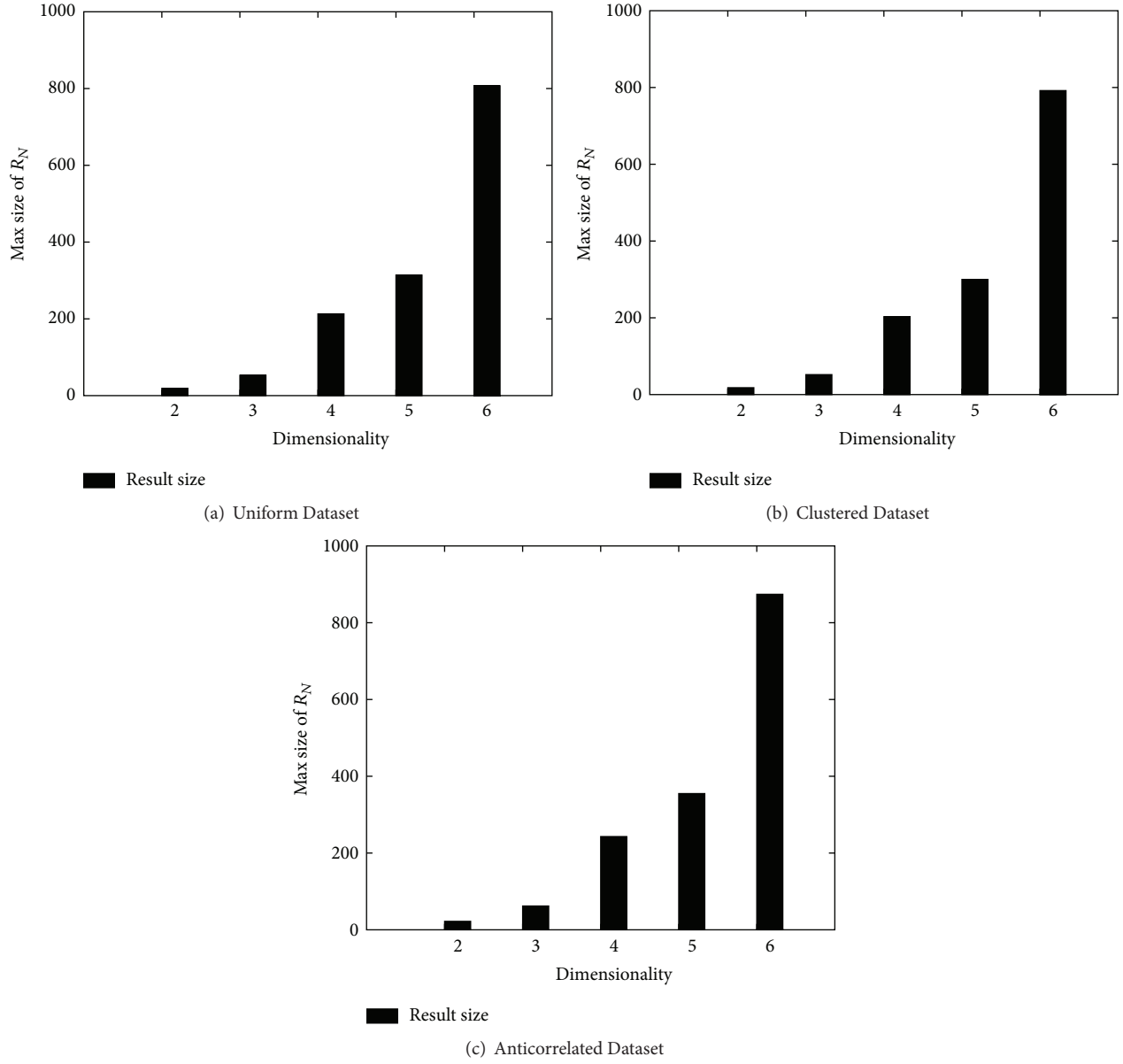


Figure 11: Result size versus dimensionality.

belongs to more than one region). Therefore, to shorten the time we need to figure that out, we can divide R_N, C_N, and A_N into 2^d regions depending on query point, respectively, and build up an in-memory R-tree for the points in each region, as showed in Figure 4.

Dominance Relationships. Since our techniques are based on R-trees, the relationship between point p and R-tree entry e must be involved. We use e.min and e.max to denote the lower-left corner and the upper-right corner of entry e; there are several main relationships being presented as follows.

If point p full-dominates e.min, all the points in e are full-dominated by p. If point p cannot full-dominate e.max, all the points in e are not full-dominated by p. Otherwise, point p full-dominates e.max but does not full-dominate e.min; there may be some points that are full-dominated by p. As showed in Figure 5(a), the dark shadow region is the full-dominated region of e; when point p is in this region, all the data points in e are full-dominated by p; while the light shadow region is candidate full-dominated region of entry e, we need to access each child of e to make sure which ones of these points are full-dominated by p.

If point p semidominates e.min, all the points in e are semidominated by p. If point p cannot semidominate e.max, all the points in e are not semidominated by p. Otherwise, point p semidominates e.max but does not semidominate e.min; there may be some points which are semidominated by p. As showed in Figure 5(b), the dark shadow region is the semidominated region of e; when point p is in this region, all the data points in e are semidominated by p; while the light

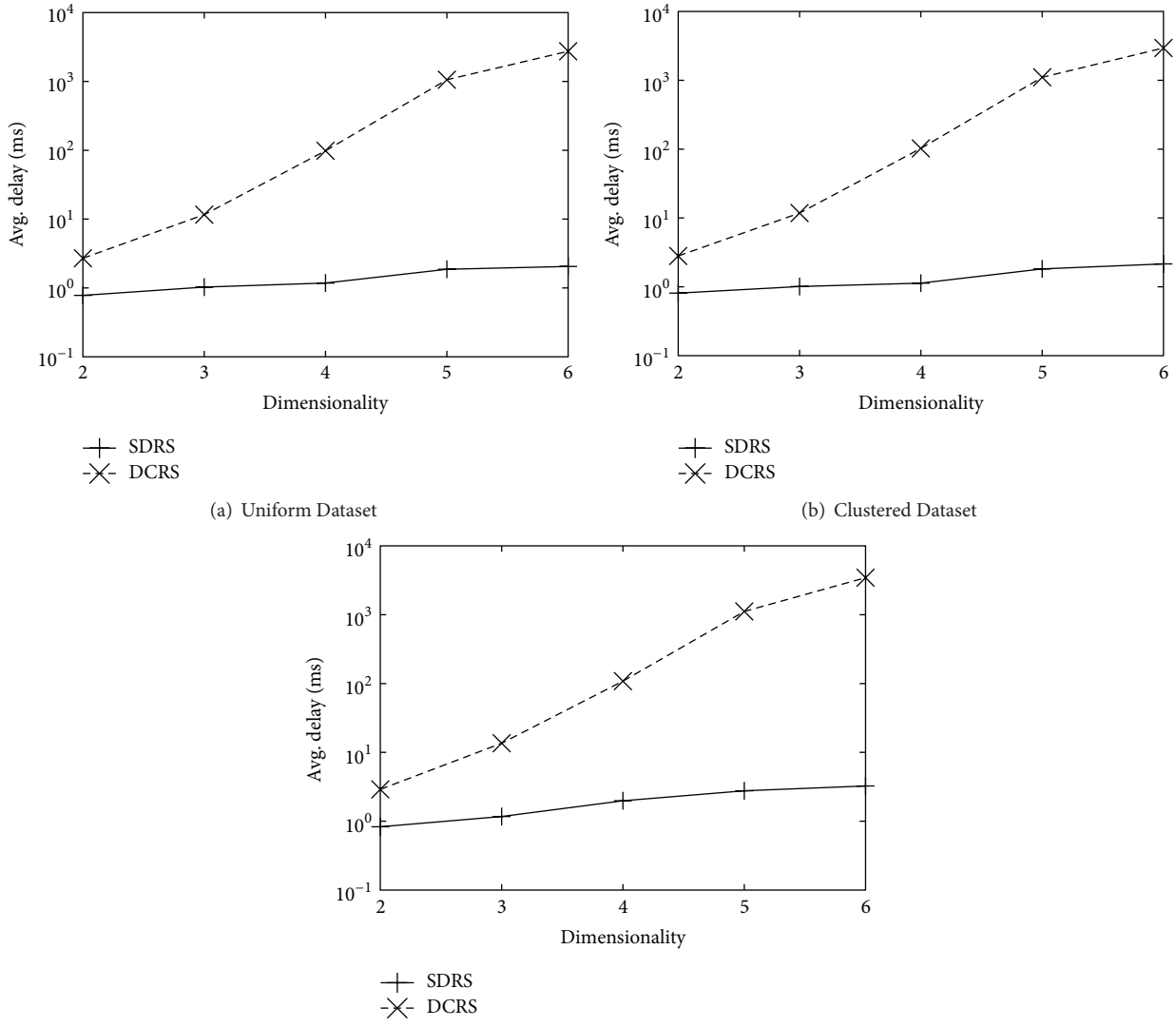


Figure 12: Response time versus dimensionality.

shadow region is candidate semidominated region of entry e, we need to access each child of e to make sure which ones of these points are semidominated by p.

If e.max semidominates p, all the points in e semidominate p. If e.min does not semidominate p, all the points in e cannot semidominate p. Otherwise, e.min semidominates p and e.max does not semidominate p; there may be some points in e that can semidominate p. In Figure 5(c), the dark shadow region is the semidominating region of entry e; when point p is in this region, each point in e semidominates p; while light shadow region is candidate semidominating region of entry e, we need to access each child of e to make sure which ones of these points can semidominate p.

For each entry e in R-tree, a label $\kappa(e)$ should be reserved. $\kappa(e)$ represents the largest label of its child node. For a leaf node e', its label is $\kappa(p)$ if point p corresponds to the leaf node e'.

Query Processing. Algorithm 1 describes the frame of SDRS. When a new point p arrives, if its label $\kappa(p) > N$, the oldest point o in S_N is expired if $\kappa(o) = \kappa(p) - N$ (Lines 2–4). Then find the set D_C of points whose predecessor is o, and remove D_C from C_N to R_N (Lines 5–7). Next, find the predecessor l_p of p. Find the set D_f of points which are full-dominated by p in S_N and delete D_f from S_N (Lines 8–10). Find the set D_s of points which are semidominated by p in S_N and remove D_s from R_N and C_N to A_N (Lines 11–14). Finally, deal with p according to $\kappa(l_p)$ (Lines 15–19).

Algorithm 2 describes how to find the predecessor of new point p. First, find the R-tree set S_R that p belongs to (Line 2). Then deal with all the entries of R-trees in S_R (Lines 3-4). If entry e is an intermediate entry, according to the dominance relationship from e to p, different methods are applied (Lines 7–13). If entry e is a leaf entry, the predecessor of p is found (Lines 14–17).

```
while a new data point p arrives do
  if κ(p) > N then
    if the oldest point o κ(o) = κ(p) − N then
      S_N = S_N − {o};
    find D_c ⊆ C_N whose predecessor is expired;
    C_N = C_N − D_c;
    R_N = R_N + D_c;
  find the predecessor l_p of point p;
  find D_f ⊆ S_N full-dominated by p;
  S_N = S_N − D_f;
  find D_s ⊆ S_N semi-dominated by p;
  R_N = R_N − D_s;
  C_N = C_N − D_s;
  A_N = A_N + D_s;
  if l_p exists in S_N then
    ρ(p) = κ(l_p);
    C_N = C_N + {p};
  else
    R_N = R_N + {p};
return;
```

Algorithm 1: Semidominance Based Reverse Skyline.

```
initialize κ = 0;
find the R-tree set S_R that point p belongs to;
for each R in S_R do
  insert all entries in the root of R into a heap H by descending order of κ(e);
while H is not empty do
  remove the top entry e from H;
  if e is an intermediate entry then
    if e.max ≺̇_q p then
      κ = κ(e);
      break;
    for each child e_i of e do
      if e.min ≺̇_q p then
        insert e_i into H by descending order of κ(e);
  else if e ≺̇_q p then
    κ = κ(e);
    break;
return κ;
```

Algorithm 2: Find predecessor of new point.

Algorithm 3 describes how to find the set D_s that contains all the points which are semidominated by the new point p but not full-dominated by p. First, find the R-tree set S_R that p belongs to (Line 2). Then deal with all the entries of R-trees in S_R (Lines 3-4). If entry e is an intermediate entry, according to the dominance relationship from p to e, different methods are applied (Lines 7–13). If e is leaf node, according to different dominance relationship from p to e, use different approach (Lines 14–17).

Algorithms 1, 2, and 3 can continuously maintain the reverse skyline queries efficiently. SDRS can prune some redundant data and minimize the number of points which are reserved in the sliding window, so it can reduce the time and space cost greatly.

5. Extensions

In this section, we extend the proposed SDRS algorithm to support n-of-N and (n_1, n_2)-of-N reverse skyline queries.

5.1. n-of-N Reverse Skyline. Firstly, we introduce the definition of n-of-N reverse skyline. P_n denotes the most recent n points in P_N.

```
initialize sets D_s = ∅;
find the R-tree set S_R that point p belongs to;
for each R in S_R do
  insert all entries in the root of R into a heap H;
  while H is not empty do
    remove the top entry e from H;
    if e is an intermediate entry then
      if p ⪯̄_q e.min then
        remove all points in e from S_N;
      else
        for each child e_i of e do
          if p ⪯̇_q e.max then
            insert e_i into H;
    else if p ⪯̄_q e then
      remove e from S_N;
    else if p ⪯̇_q e then
      remove e from D_s;
  balance R in a bottom-up fashion;
return D_s;
```

Algorithm 3: Remove D_f and compute D_s.

Definition 10 (n-of-N reverse skyline). Given the recent N points P_N in the sliding window, *n-of-N reverse skyline* computes the reverse skyline in the most recent n points which are not semidominated by the points in P_N.

According to Definition 10, we propose a new definition l_p to solve this problem. For a data point $p \in P_N$, we use l_p to denote the newest point which can semidominate p and arrive before p. In such a case, every point in P_N has a predecessor point l_p, and we use $\kappa(l_p)$ to denote l_p's label. If p has no l_p, $\kappa(l_p)$ equals 0.

For every point in P_N, we compute its l_p. Construct a 2D R-tree index on $\langle \kappa(l_p), \kappa(p) \rangle$. If the current label is t, we execute a range query $[0, t-n]$, $[t-n+1, t]$ on the 2D R-tree. For a point $p \in P_N$, if $\kappa(l_p)$ is in the range $[0, t-n]$ and $\kappa(p)$ is in the range $[t-n+1, t]$, p is the n-of-N reverse skyline. Through retrieving the whole 2D R-tree index, we can find all the n-of-N reverse skyline points.

5.2. (n_1, n_2)-of-N Reverse Skyline. In this section, we introduce the definition of (n_1, n_2)-of-N reverse skyline firstly. p_{n_1} denoted the most recent n_1 points in P_N and p_{n_2} denoted the most recent n_2 points in P_N.

Definition 11 ((n_1, n_2)-of-N reverse skyline). Given the recent N points P_N in the sliding window and $n_1 > n_2$, *(n_1, n_2)-of-N reverse skyline* computes the reverse skyline points in the set $P_{n_1} - P_{n_2}$.

According to Definition 11, we propose a new definition r_p to solve the problem. For a point $p \in P_N$, we can use r_p to denote the oldest point which can semidominate p and arrive after p. Specifically, (1) $r_p \dot{\preceq}_q p$, (2) $\kappa(r_p) > \kappa(p)$, and (3) $\nexists x \in P_N$, $\kappa(p) < \kappa(x) < \kappa(r_p)$ such that $x \dot{\preceq}_q p$. $\kappa(r_p)$ denotes the label of r_p. If a point p has no r_p, $\kappa(r_p)$ equals ∞.

For every point in P_N, we compute its l_p and r_p. Construct a 2D R-tree index on $\langle \kappa(l_p), \kappa(p) \rangle$. If the current label is t, we execute a range query $[0, t-n_1]$, $[t-n_1+1, t-n_2]$ on the 2D R-tree. For a point $p \in P_N$, if $\kappa(l_p)$ is in the range $[0, t-n]$ and $\kappa(p)$ is in the range $[t-n+1, t]$, we need further judgment. If $\kappa(r_p)$ is in the range $[t-n_2+1, \infty]$, p is the (n_1, n_2)-of-N reverse skyline. Through retrieving the whole 2D R-tree index and executing a further judgment, we can find all the (n_1, n_2)-of-N reverse skyline points.

6. Experimental Evaluation

In this section, we experimentally compare our proposed SDRS algorithm against the only existing DCRS algorithm [16] using real dataset-stock data from the Yahoo Financial website and some synthetic datasets. The stock data contains 10 M stock records. Each record contains 4 attributes: price, volume, growth, and capital size. We evaluate the performance of SDRS and DCRS in 2D stock dataset (containing attributes price and volume) and 4D stock dataset when the sliding window sizes are 10 K, 100 K, 1 M, and 10 M. The experimental results are shown in Figure 6.

Figure 6(a) shows the space usage in terms of the number of points kept in the sliding window against N. DCRS needs to preserve all the points in the sliding window. Our SDRS only preserves a small number of the points in the sliding window. When the dimensionality is increased, the number of the preserved points is also increased. Figure 6(b) records the number of the result against N. Figure 6(c) records the response time of the two algorithms for 2D and 4D stock datasets. The response time of SDRS is greatly shorter than that of DCRS because of the filtering policies. From Figure 6, we can see that our SDRS performs much better than DCRS in the real dataset-stock dataset.

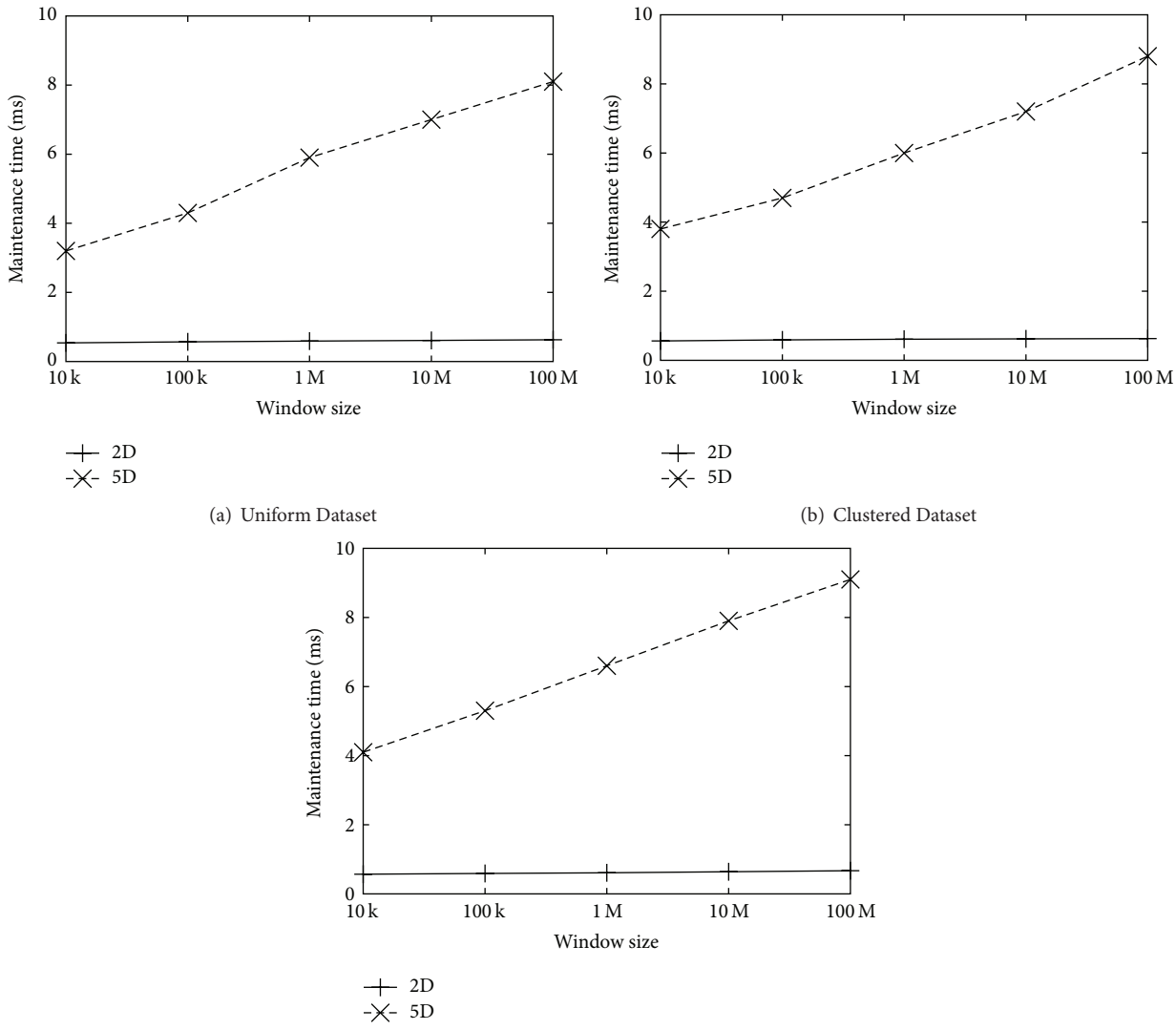


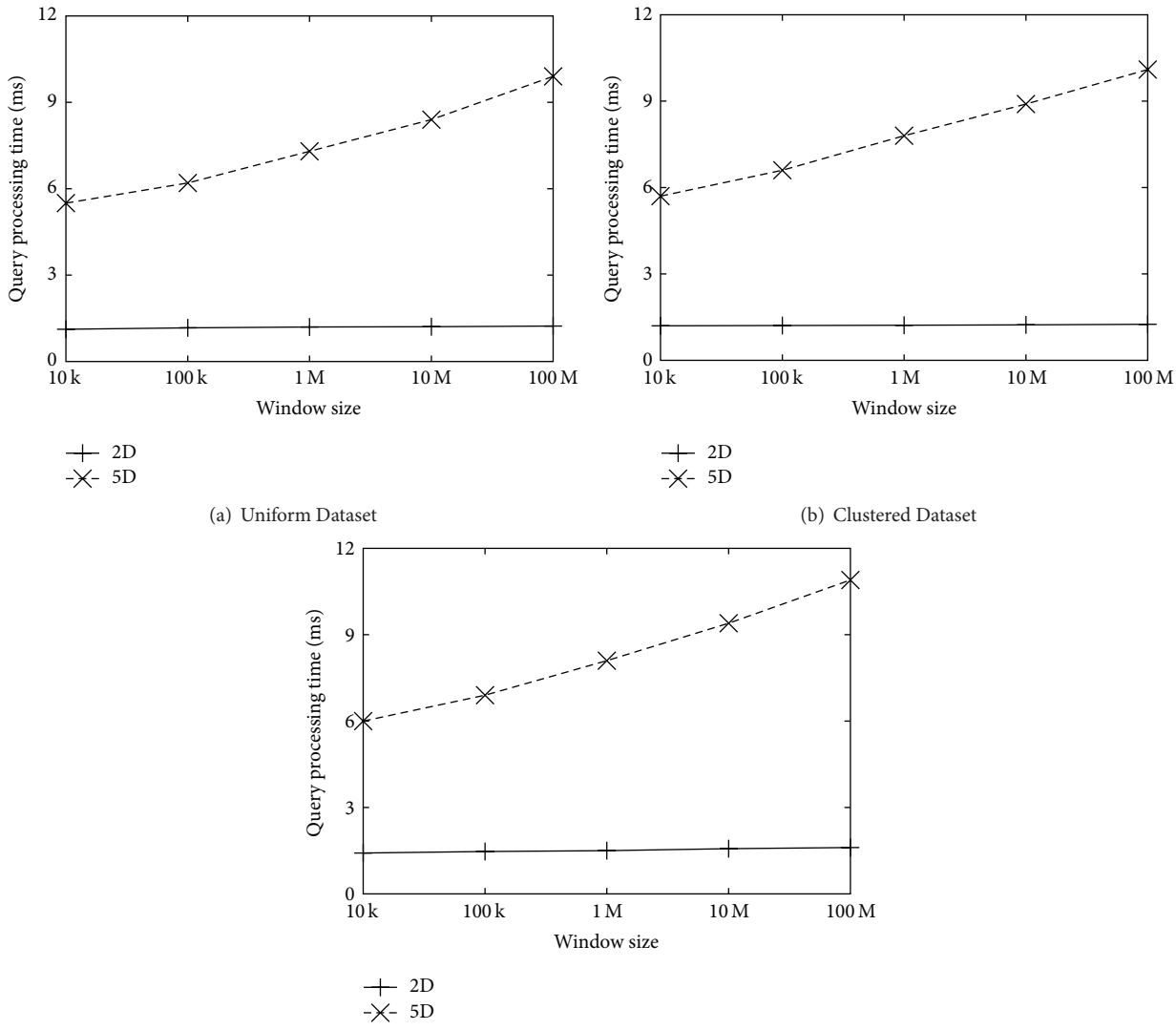
Figure 13: Maintenance time for n-of-N query.

The synthetic datasets contain three different distributions, including Uniformly Distributed, Clustered, and Anticorrelated Distributed Datasets [1, 7]. Uniformly Distributed Dataset consists of points randomly generated from a unit square. Anticorrelated Distributed Dataset consists of points which are around the antidiagonal. While the Clustered Dataset comprises ten randomly centered clusters, each of them consists of an equal number of points and follows a multivariate Gaussian distribution whose covariance is a 0.05-diagonal matrix and mean vector is equal to its associated centroid. All experiments are performed on a machine with Intel Core 2 1.86 GHz CPU, 1 GB memory, and 80 GB hard disk. Table 1 summarizes the parameters involved in our experiments.

Table 1: Experimental parameters.

Parameter	Range & default
Dimensionality	2, 3, **4**, 5, 6
Window size	**10 K**, 100 K, 1 M, 10 M, 100 M

6.1. Reverse Skyline on Data Stream. We first evaluate the impact of sliding window size N. The following is shown in Figures 7, 8, and 9. Figure 7 shows the space usage in terms of the maximal number of points kept in S_N against N. For three data distributions, DCRS must reserve all the points in the sliding window, while SDRS reserves only a small part of the points in the sliding window, since the filtering policies in SDRS can prune many redundant points. Figure 8 records the maximal number of the result against N in three data distributions. Figure 9 records the response time of the two algorithms against sliding window size N. The response time of SDRS is greatly shorter than DCRS in all the data distributions because of the filtering policies. From Figures 8 and 9, we can see that the number of the results is increased,



(a) Uniform Dataset

(b) Clustered Dataset

(c) Anticorrelated Dataset

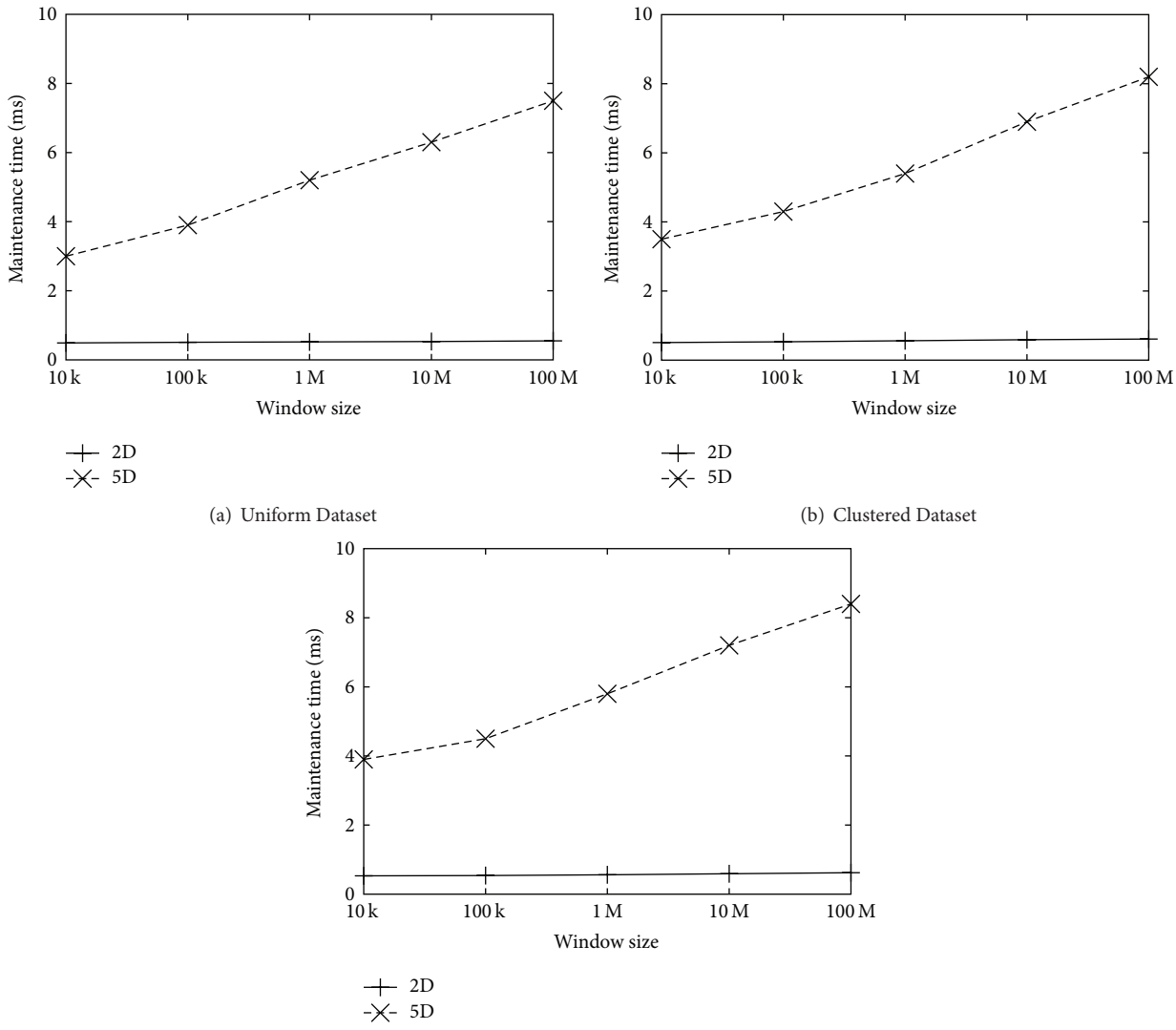
Figure 14: Processing time for n-of-N query.

and the response time of SDRS becomes longer with the increase of sliding window size N. Because, with the increase of N, more points in the window should be calculated, then result number will be more and the response time will be longer.

Next, we evaluate the impact of dimensionality d. The following is shown in Figures 10, 11, and 12. Figure 10 shows the space usage in terms of the maximal number of points kept in S_N against d. The points reserved in SDRS are only a small part of the whole points in the sliding window because the filtering policies can prune many redundant points, while DCRS must keep all the points in the sliding window. Figure 11 records the maximal number of the result against d in three data distributions. Figure 12 records the response time of the two algorithms against d; the response time of SDRS is greatly smaller than DCRS because of the filtering policies in SDRS. From Figures 11 and 12, the number of the results is increased, and the response time of SDRS becomes longer with the increase of dimensionality d. Because, with the increase of d, more calculations are needed to compute the dominance relationships between two points and the dominance relationship becomes harder, then result number will be more and the response time will be longer.

6.2. n-of-N Reverse Skyline. In this section, maintenance time and processing time are used to evaluate the algorithm performance of n-of-N reverse skyline query. Maintenance time is the time of constructing the 2D R-tree index, while the processing time is the query time of n-of-N reverse skyline query. In order to evaluate the algorithm performance stably, we generate 1000 random values as n and carry out 1000 continuously n-of-N reverse skyline queries. Finally we record the average value of 1000-query time as the processing time.

As shown in Figures 13 and 14, Figure 13 records the maintenance time in 2D and 5D against sliding window size N in



(a) Uniform Dataset

(b) Clustered Dataset

(c) Anticorrelated Dataset

Figure 15: Maintenance time for (n_1, n_2)-of-N query.

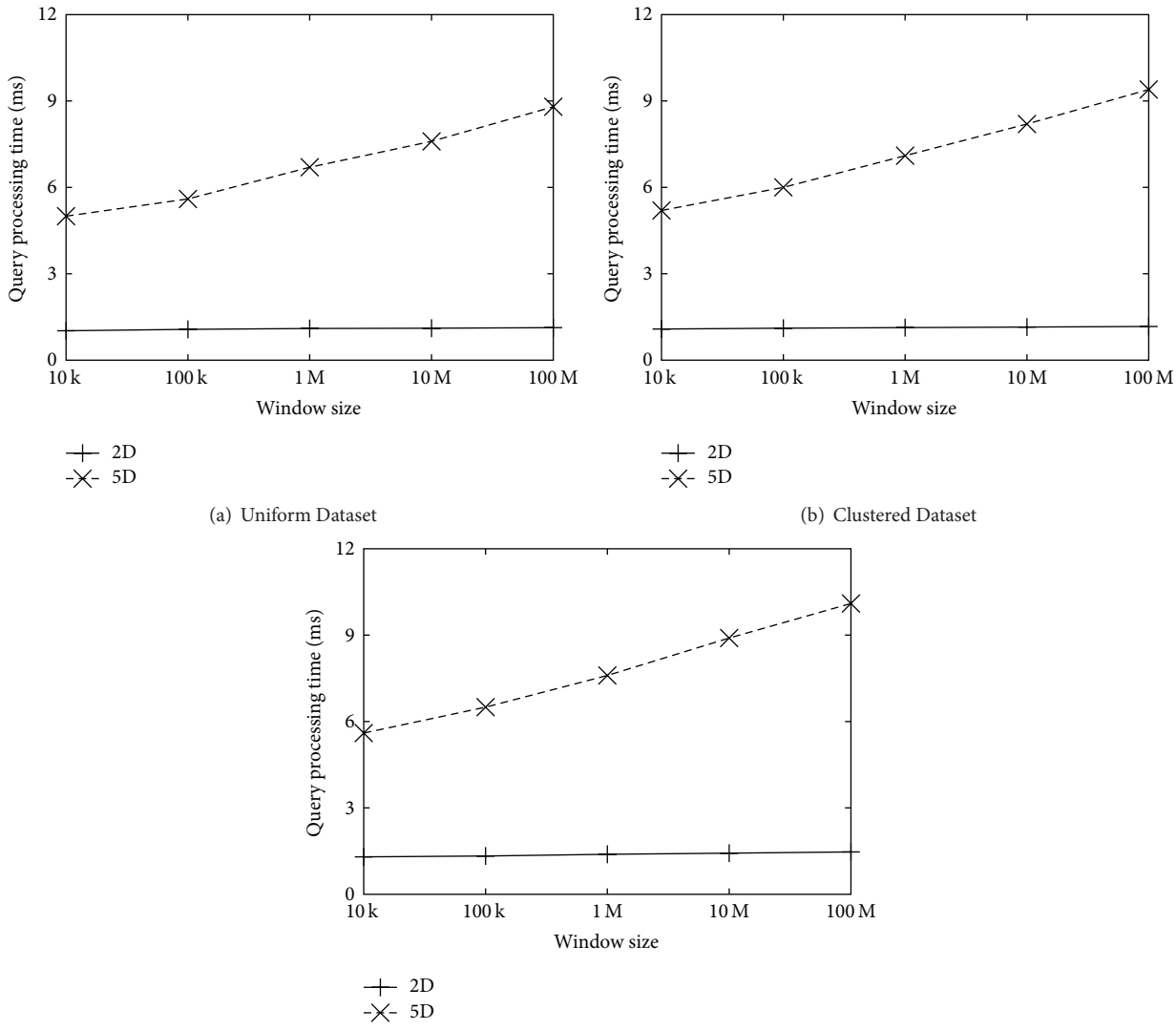
three data distributions and Figure 14 records the processing time in 2D and 5D against N in both data distributions. With the increase of N, maintenance time and processing time have a little increase. Because, with the increase of window size N, the number of points in R-tree is increased, maintenance time and processing time have a little increase. The maintenance time and processing time in 5D are much longer than those in 2D, because the calculation in 5D is more complex than that in 2D, and the reserved points in 5D are much more than those in 2D.

6.3. (n_1, n_2)-of-N Reverse Skyline. In this section, maintenance time and processing time are used to evaluate the algorithm performance of (n_1, n_2)-of-N reverse skyline query. Maintenance time is the time of constructing the 2D R-tree index, while the processing time is the query time of (n_1, n_2)-of-N reverse skyline query. In order to evaluate the algorithm performance stably, we generate 1000 random values as n_1, n_2 equal n_1 + 500 and then carry out 1000 continuously (n_1, n_2)-of-N reverse skyline queries. Finally, we record the average value of the 1000-query time as the processing time.

As shown in Figures 15 and 16, Figure 15 records the maintenance time in 2D and 5D against N in three data distributions and Figure 16 records the processing time in 2D and 5D against N in both data distributions. Maintenance time and processing time have a little increase with the increase of N, because the number of reversed points in R-tree has a little increase. The calculation in 5D is more complex than that in 2D, so maintenance time and processing time in 5D are much longer than those in 2D.

7. Conclusions

Despite its importance in real-world applications, reverse skyline computation on data streams has not been well



(a) Uniform Dataset

(b) Clustered Dataset

(c) Anticorrelated Dataset

Figure 16: Processing time for (n_1, n_2)-of-N query.

studied. Therefore, in this paper, we focus on the problem of efficiently computing reverse skyline against sliding windows over an append-only data stream. Specifically, we present an effective pruning approach to minimize the number of points to be kept in the sliding window and propose efficient semidominance based on approach SDRS for processing continuous reverse skyline queries. Moreover, we also propose an extension for handling n-of-N and (n_1, n_2)-of-N reverse skyline queries. Our extensive experiments have demonstrated the efficiency and effectiveness of our proposed SDRS approach under various experimental settings.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was partially supported by the National Natural Science Foundation of China under Grants nos. 61472069, 61402089, and 61100022 and the Fundamental Research Funds for the Central Universities under Grant no. N130404014.

References

[1] S. Borzsonyi, K. Stocker, and D. Kossmann, "The Skyline operator," in *Proceedings of the 17th International Conference on Data Engineering*, pp. 421–430, IEEE, Heidelberg, Germany, April 2001.

[2] Y. Tao, K. Xiao, and J. Pei, "Efficient skyline and top-k retrieval in subspaces," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 8, pp. 1072–1088, 2007.

[3] E. Dellis, A. Vlachou, I. Vladimirskiy, B. Seeger, and Y. Theodoridis, "Constrained subspace skyline computation," in *Proceedings of the 15th ACM Conference on Information and Knowledge Management (CIKM '06)*, pp. 415–424, ACM, November 2006.

[4] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proceedings of the 22th ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pp. 467–472, San Diego, Calif, USA, June 2003.

[5] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 41–82, 2005.

[6] K. Deng, X. Zhou, and H. Shen, "Multi-source skyline query processing in road networks," in *Proceedings of the 26th ACM SIGMOD International Conference on Management of Data*, pp. 796–805, Beijing, China, June 2007.

[7] E. Dellis and B. Seeger, "Efficient computation of reverse skyline queries," in *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB '07)*, pp. 291–302, September 2007.

[8] X. Ding, X. Lian, L. Chen, and H. Jin, "Continuous monitoring of skylines over uncertain data streams," *Information Sciences*, vol. 184, no. 1, pp. 196–214, 2012.

[9] Z. Huang, H. Lu, B. C. Ooi, and A. K. H. Tung, "Continuous skyline queries for moving objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 12, pp. 1645–1658, 2006.

[10] X. Lian and L. Chen, "Monochromatic and bichromatlc reverse skyline search over uncertain databases," in *Proceedings of the 27th ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, pp. 213–226, Vancouver, Canada, June 2008.

[11] X. Lian and L. Chen, "Reverse skyline search in uncertain databases," *ACM Transactions on Database Systems*, vol. 35, no. 1, article 3, 49 pages, 2010.

[12] P. M. Deshpande and P. Deepak, "Efficient reverse skyline retrieval with arbitrary non-metric similarity measures," in *Proceedings of the 14th International Conference on Extending Database Technology*, pp. 319–330, ACM, Uppsala, Sweden, March 2011.

[13] X. Wu, Y. Tao, R. C.-W. Wong, L. Ding, and J. X. Yu, "Finding the influence set through skylines," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT '09)*, pp. 1030–1041, Saint-Petersburg, Russia, March 2009.

[14] J. Xin, Z. Wang, M. Bai, L. Ding, and G. Wang, "Energy-efficient β-approximate skylines processing in wireless sensor networks," *Mathematical Problems in Engineering*, vol. 2015, Article ID 149513, 11 pages, 2015.

[15] L. Chen, J. Zhao, Q. Huang, and L. H. Yang, "Effective space usage estimation for sliding-window skybands," *Mathematical Problems in Engineering*, vol. 2010, Article ID 828035, 15 pages, 2010.

[16] L. Zhu, C. Li, and H. Chen, "Efficient computation of reverse skyline on data stream," in *Proceedings of the 2nd International Joint Conference on Computational Sciences and Optimization (CSO '09)*, pp. 735–739, IEEE, Hainan, China, April 2009.

[17] M. Sharifzadeh and C. Shahabi, "The spatial skyline queries," in *Proceedings of the 32nd International Conference on Very Large Data Bases (VLDB '06)*, pp. 751–762, Seoul, Republic of Korea, September 2006.

[18] L. Chen and X. Lian, "Dynamic skyline queries in metric spaces," in *Proceedings of the 11th International Conference on Extending Database Technology*, pp. 333–343, ACM, March 2008.

[19] L. Chen and X. Lian, "Efficient processing of metric skyline queries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 3, pp. 351–365, 2009.

[20] G. Wang, J. Xin, L. Chen, and Y. Liu, "Energy-efficient reverse skyline query processing over wireless sensor networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 7, pp. 1259–1275, 2012.

[21] J.-K. Min, "Efficient reverse skyline processing over sliding windows in wireless sensor networks," *International Journal of Distributed Sensor Networks*, vol. 2015, Article ID 375630, 16 pages, 2015.

[22] J. Lim, Y. Park, K. Bok, and J. Yoo, "A continuous reverse skyline query processing considering the mobility of query objects," in *Proceedings of the 5th International Conference on Internet and Distributed Computing Systems*, pp. 227–237, Fujian, China, November 2012.

[23] J. Lim, Y. Park, k. Bok, and J. Yoo, "An efficient continuous reverse skyline query processing method over moving objects," in *Ubiquitous Information Technologies and Applications: CUTE 2012*, vol. 214 of *Lecture Notes in Electrical Engineering*, pp. 323–331, Springer, Dordrecht, The Netherlands, 2013.

[24] J. Lim, H. Li, K. Bok, and J. Yoo, "A continuous reverse skyline query processing method in moving objects environments," *Data and Knowledge Engineering*, 2015.

[25] A. Han, Y. Park, and D. Kwon, "An efficient pruning method to process reverse skyline queries," *Journal of Information Science and Engineering*, vol. 30, no. 2, pp. 501–517, 2012.

[26] X. Lin, Y. Yuan, W. Wang, and H. Lu, "Stabbing the sky: efficient skyline computation over sliding windows," in *Proceedings of the 21st International Conference on Data Engineering (ICDE '05)*, pp. 502–513, IEEE, Tokoyo, Japan, April 2005.

[27] Y. Tao and D. Papadias, "Maintaining sliding window skylines on data streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 3, pp. 377–391, 2006.

[28] M. Morse, J. M. Patel, and W. I. Grosky, "Efficient continuous skyline computation," *Information Sciences*, vol. 177, no. 17, pp. 3411–3437, 2007.

[29] N. Sarkas, N. Koudas, G. Das, and A. K. H. Tung, "Categorical skylines for streaming data," in *Proceedings of the 27th ACM SIGMOD International Conference on Management of Data*, pp. 239–250, ACM, Vancouver, Canada, June 2008.

[30] Z. Zhang, R. Cheng, D. Papadias, and A. K. H. Tung, "Minimizing the communication cost for continuous skyline maintenance," in *Proceedings of the 28th International Conference on Management of Data*, pp. 495–507, ACM, Providence, RI, USA, July 2009.

[31] W. Zhang, X. Lin, Y. Zhang, W. Wang, and J. X. Yu, "Probabilistic skyline operator over sliding windows," in *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE '09)*, pp. 1060–1071, IEEE, Shanghai, China, April 2009.

[32] W. Zhang, X. Lin, Y. Zhang, W. Wang, G. Zhu, and J. X. Yu, "Probabilistic skyline operator over sliding windows," *Information Systems*, vol. 38, no. 8, pp. 1212–1233, 2013.

[33] M. Bai, J. Xin, and G. Wang, "Probabilistic reverse skyline query processing over uncertain data stream," in *Database Systems for Advanced Applications: 17th International Conference, DASFAA 2012, Busan, South Korea, April 15–19, 2012, Proceedings, Part II*, vol. 7239 of *Lecture Notes in Computer Science*, pp. 17–32, Springer, Berlin, Germany, 2012.

[34] Q. Liu, Y. Gao, G. Chen, Q. Li, and T. Jiang, "On efficient reverse k-skyband query processing," in *Proceedings of the 17th International Conference on Database Systems for Advanced Applications*, pp. 544–559, Busan, Republic of Korea, April 2012.

[35] J. Yang, B. Qu, C.-P. Li, and H. Chen, "DC-tree: an algorithm for skyline query on data streams," in *Advanced Data Mining and Applications: 4th International Conference, ADMA 2008, Chengdu, China, October 8–10, 2008. Proceedings*, vol. 5139 of *Lecture Notes in Computer Science*, pp. 644–651, Springer, Berlin, Germany, 2008.