Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 375630, 16 pages
http://dx.doi.org/10.1155/2015/375630

Research Article

Efficient Reverse Skyline Processing over Sliding Windows in Wireless Sensor Networks

Jun-Ki Min

School of Computer Science and Engineering, Korea University of Technology and Education, Byeongcheon-myeon, Cheonan, Chungnam 330-708, Republic of Korea

Correspondence should be addressed to Jun-Ki Min; jkmin@koreatech.ac.kr

Received 26 November 2014; Revised 13 March 2015; Accepted 24 March 2015

Academic Editor: Ibrahim Kamel

Owning to the proliferation of cost-effective sensors, there has been an increased growth in a number of applications of wireless sensor networks (WSNs). In addition, the skyline operator as well as its variants such as the dynamic skyline and reverse skyline operator has attracted increasing attention since those are useful for multicriteria decision making applications. Since the energy efficiency is utmost important issue to prolong the network lifetime, in this paper, we proposed efficient algorithms to process a reverse skyline query over a sliding window in WSN environments. We first devise our algorithm for the data stream environments and extend it to WSN environments. To compute the reverse skyline, we partition the data space into several orthants with respect to a query point. And, in each orthant, we compute the reverse skyline independently using two buffers. In our experiment study, we demonstrate that our algorithm is much better than other algorithms.

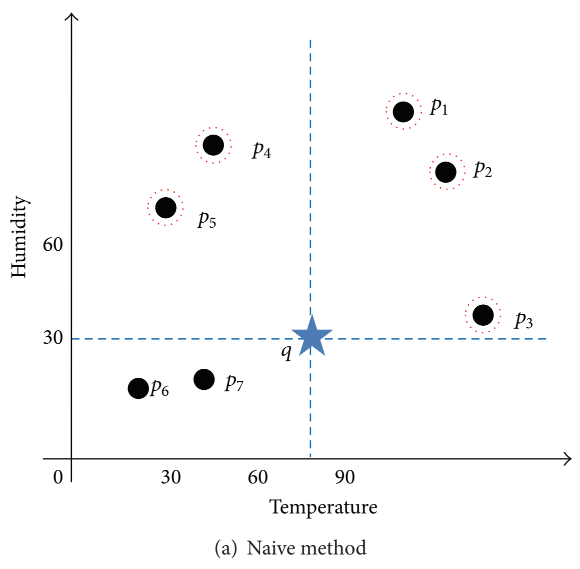
1. Introduction

Since being introduced in the database community, the skyline operator [1] and its variants such as dynamic skyline [2] and reverse skyline [3] operators have attracted increasing attention in multicriteria decision making applications such as product recommendations [4, 5], querying wireless sensor networks [6], and graph analysis [7].

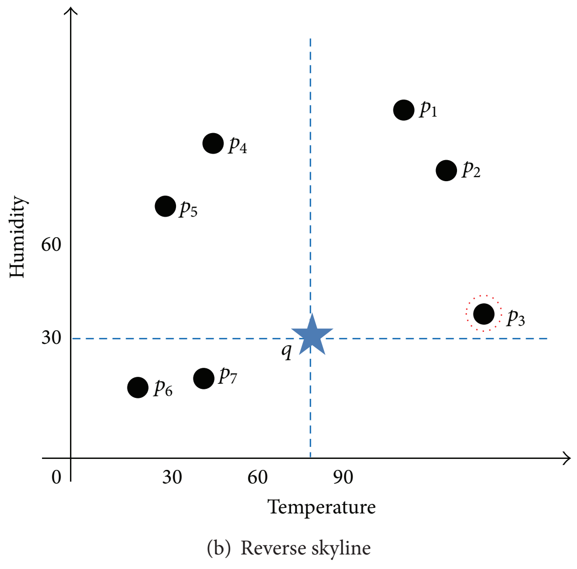
Given a d-dimensional point set P, a point $p_i \in P$ *dominates* another point $p_j \in P$ if p_i is smaller than p_2 in at least one dimension and not greater than p_2 in all other dimensions. The skyline on P comprises all points that are not dominated by any other points. Papadias et al. [2] proposed the dynamic skyline which is a set of points in P not to be *dynamically dominated* by any other point with respect to (wrt) coordinate-wise distances to a given query point q. Another interesting skyline variant is the reverse skyline operator which returns a set of every point in P, denoted as $RSL(q, P)$, whose dynamic skyline contains a query point q [3].

In general, a wireless sensor network (WSN) is considered as a cost effective platform to monitor environments. A WSN consists of spatially distributed devices with various sensors and a powered base station which serves as an access point for users to pose ad hoc queries. The research for diverse types of queries over WSNs, for example data gathering [8], aggregation queries [9, 10], join queries [11, 12], and skyline queries [6, 13, 14], has been conducted to satisfy the diverse application demands. Among the diverse types of queries, a reverse skyline query is very useful for environmental monitoring applications. For example, in an application of monitoring the forest environment, a lot of sensors are deployed in a forest to collect sensor readings such as temperature and humidity. Assume a query point q represents the thresholds of a possible fire disaster on different attributes.

A naive method to detect a forest fire with a query point q is that only the sensor nodes with sensor readings exceeding thresholds report their sensor reading. For instance, let each point in Figure 1(a) represent the sensor reading of each sensor node. Since many sensor readings, represented by dotted circles in Figure 1(a), exceed the thresholds, many sensor nodes consume much energy to transmit a lot of sensor readings. Because each sensor node is battery-powered and located in hazardous or hard-to-reach place, it is impossible or very difficult to change the batteries of sensor nodes. Thus,



(a) Naive method



(b) Reverse skyline

Figure 1: An example for a forest fire.

in WSN environments, the energy efficiency is the utmost important issue to prolong the network lifetime.

In contrast to the naive method, the reverse skyline operator considers the dominance relationship for attributes with respect to a query point q which indicates a potential fire disaster as shown in Figure 1(b). The reverse skyline points are represented by dotted circles in Figure 1(b). A point p_i is a reverse skyline point when q is a dynamic skyline point wrt p_i. In other words, q is not dynamically dominated by other points wrt p_i. Informally, it means that q and p_i are close to each other at least in one dimension (i.e., q and p_i are similar to each other compared to other points at least in one dimension). Thus, the reverse skyline can represent the sensor nodes having sensor readings highly following the fire pattern for at least one attribute compared with others. Therefore, this can save much time and quickly locate the most dangerous places.

In this paper, we investigate the problem of energy-efficient in-network reverse skyline computation in WSN environments. In particular, WSN can be considered as a source of data streams. The data stream can be broken into possibly overlapping partitions by specifying a window and computation can be carried out in each partition. While efficient processing techniques for window queries have been proposed in the area of data streams, most of the previous work on data stream processing assumes that query processing is conducted at a centralized server. On contrary, in-network processing is commonly used in sensor network where each sensor calculates a partial result. Therefore, in our work, we devise an energy efficient algorithm to compute the reverse skyline considering sliding window queries which return repeatedly reverse skyline points during a given time interval. In this paper, we consider the sensor readings that arrived in a sliding window with size w. Specifically, a sensor reading generated at time t_s is *alive* during $[t_s, t_e) = [t_s, t_s + w)$.

Our Contributions. Our work has the following combination of contributions to perform the reverse skyline operator over a sliding window.

(i) To make an efficient algorithm of the reverse skyline, we analyze the properties of the reverse skyline theoretically. At first, we divide the d-dimensional data space into 2^d orthants wrt a query point q. Then, we prove that every reverse skyline point wrt q is also a dynamic skyline point wrt q on each orthant and any dynamic skyline point dominated by a midpoint of another point in each orthant is not a reverse skyline point.

(ii) Each sensor node can be regarded as a source of stream data since each sensor node measures its environment repeatedly. Thus, we first proposed an effective algorithm which computes reverse skyline for a sliding window over a data stream. The devised algorithm is running on each sensor node to generate partial result. To compute the reverse skyline progressively, our algorithm maintains two buffers $o.Buff_{dsky}$ and $o.Buff_{rest}$ on each orthant o which keep the dynamic skyline points and the dynamic skyline candidates, respectively.

(iii) We devise in-network reverse skyline processing technique in WSN environments. Each node in a WSN only transmits small number of points to its parent node when the points become newly dynamic skyline points or the states of them are changed. Accordingly, the energy consumption of each node decreases.

To evaluate our proposed algorithms, we implemented our algorithms. In our experiments, we use the synthetic data set and the real-world data set to show the effectiveness of our algorithms in data stream environments and WSN environments. In data stream environments, we measure the processing time of each algorithm and, in WSN environments, we measure the total energy consumption of every node. Our comprehensive empirical evaluation demonstrates that our algorithm delivers the best performance in all situations.

The rest of the paper is organized as follows. Section 2 introduces the various skyline operators and wireless sensor networks. Section 3 contains related work. We present the basic features of the reverse skyline and propose our basic algorithm to process the reverse skyline in Section 4. In Section 5, we present the energy efficient in-network processing technique to compute reverse skyline over sliding window in WSN environments. Section 6 presents the empirical evaluation results and Section 7 summarizes the paper.

2. Preliminaries

2.1. Various Skyline Operators. Formally, given a d-dimensional data set $P = \{p_1, \ldots, p_{|P|}\}$, a point $p_i \in P$ is represented as $p_i = \langle p_i \cdot x_1, p_i \cdot x_2, \ldots, p_i \cdot x_d \rangle$, where $p_i \cdot x_k$ is the value of p_i's kth coordinate. A point p_i *dominates* another point p_j, denoted as $p_i \prec p_j$, if the two conditions hold: (1) $\forall k \in \{1, \ldots, d\}$, $p_i \cdot x_k \leq p_j \cdot x_k$ and (2) $\exists k \in \{1, \ldots, d\}$, $p_i \cdot x_k < p_j \cdot x_k$. Based on the dominance relationship, the *skyline* of P is defined as follows.

Definition 1 (skyline). Given a d-dimensional data set $P = \{p_1, \ldots, p_{|P|}\}$, the skyline of P, represented by $\mathrm{SL}(P)$, is largest subset of P where every point in $\mathrm{SL}(P)$ is not dominated by any other point in P. In other words, $\mathrm{SL}(P) = \{p_i \in P \mid \nexists p_j\ (\neq p_i) \in P \text{ s.t. } p_j \prec p_i\}$.

Given a d-dimensional query point q, the dominance relationship extended the dynamic dominance relationship. We say that a point p_i *dynamically dominates* p_j with respect to (wrt) q, denoted as $p_i \prec_q p_j$, if $\forall k \in \{1, \ldots, d\}$, $|p_i \cdot x_k - q_k| \leq |p_j \cdot x_k - q \cdot x_k|$, and $\exists k \in \{1, \ldots, d\}$, $|p_i \cdot x_k - q_k| < |p_j \cdot x_k - q \cdot x_k|$.

Definition 2 (dynamic skyline). Given a d-dimensional data set P and a query point q, the dynamic skyline of P with respect to q is represented by $DSL(q, P)$ such that $DSL(q, P) = \{p_i \in P \mid \nexists p_j\ (\neq p_i) \in P \text{ s.t. } p_j \prec_q p_i\}$.

Based on the dynamic skyline, the notion of the *reverse skyline* was proposed in [3]. The reverse skyline is defined as the follows.

Definition 3 (reverse skyline). Given a d-dimensional data set P and a query point q, the reverse skyline, represented by $RSL(q, P)$, is the set of every point p_i in P satisfying $q \in DSL(p_i, P \cup \{q\} - \{p_i\})$.

Example 4. Consider the data set $P = \{p_1, \ldots, p_7\}$ in Figure 2(a). Given a query point q represented by $\star$, a point p_4 is a dynamic skyline point wrt q since p_4 is not dynamically dominated by the other points. However, since a point p_5 is dynamically dominated by p_7, p_5 is not a dynamic skyline. The dynamic skyline of P wrt q is $DSL(q, P) = \{p_3, p_4, p_7\}$ in Figure 2(a). Figure 2(b) shows the dynamic skyline wrt p_7. Since q is not a dynamic skyline point wrt p_7, p_7 is a reverse skyline point (i.e., $p_7 \notin RSL(q, P)$). As shown in Figure 1(b), $RSL(q, P) = \{p_3\}$.

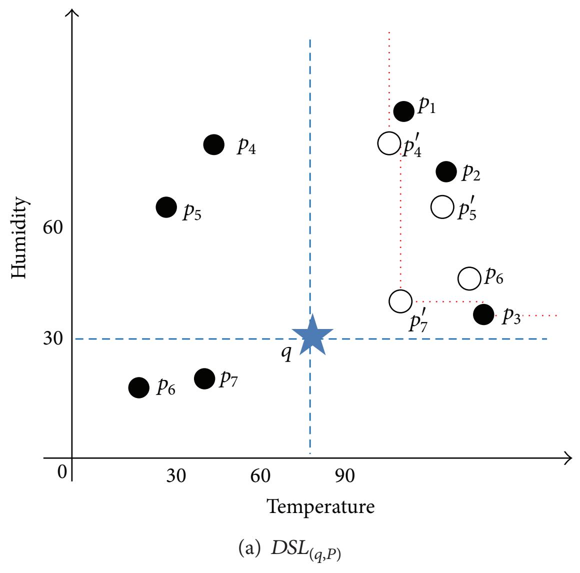
2.2. Wireless Sensor Networks. We consider a sensor network consisting of n stationary sensor nodes $\{s_1, s_2, \ldots, s_n\}$ deployed in a field of interest and a powered base station serving as an access point for users to pose ad hoc queries. As a basic primitive to collect sensing data in WSNs, we use an ad hoc spanning tree, such as TinyDB [15] and SNEE [16] as the basic routing structure from each sensor node to the base station. Figure 3 illustrates an example of simple sensor network consisting of eight sensor nodes.

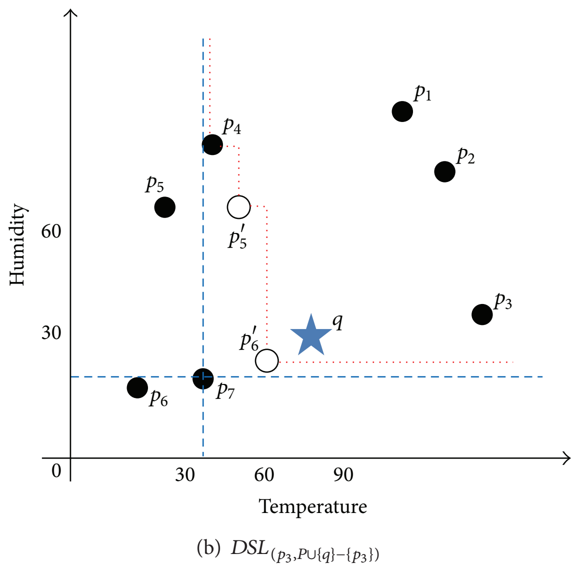
To form a routing tree, the base station first sends a request message which contains a hop-counter indicating the hop distance from the base station. When each node s_c receives a request message from another node s_p, if s_c does not have a parent node yet, node s_p becomes the parent node of s_c, and, then, s_c forwards the request message with the hop-counter increased by 1 to the other nodes. If s_c already has a parent, s_c simply ignores the request message. When s_c receives several request messages from the other nodes, s_c picks the one which has the smallest hop-counter as the parent. To break ties, the heuristics such as signal strength and arrival time can be applied. A sensor reading consists of several attributes each of which is associated with a sensor module. A sensor node may be equipped with several sensor modules. Sensor nodes generate their readings periodically and synchronously. To synchronize the sampling time, every sensor node executes a global time synchronization protocol [17].

3. Related Work

To reduce the energy consumption of WSNs, research on diverse types of queries such as aggregation, data gathering, join, and skyline has been conducted. One of the well-known approaches to reduce the energy consumption of WSNs is in-network processing. In the in-network processing techniques, the partial results are progressively merged at intermediate nodes on their way to the base station according to the tree routing.

3.1. Aggregation. The pioneering TAG work by Madden et al. in [9] studied in-network aggregation for reducing communication overhead using summary data (e.g., SUM) and/or exemplary data (e.g., MIN and MAX). In TAG, as climbing up a routing tree from leaf nodes to the base station, partial aggregation values are computed. Approximate aggregation techniques have been also proposed to reduce the energy consumption. The work of Considine et al. [18] was based on the FM sketch. Shrivastava et al. [19] developed the *q-digest* structure to support approximate processing for quantile queries such as MEDIAN. In [20], an effective aggregation technique for the situation that sensor nodes can detect an object duplicately was presented. To identify the duplicates





(a) $DSL_{(q,P)}$ (b) $DSL_{(p_3, P\cup\{q\}-\{p_3\})}$

Figure 2: An example of the dynamic skyline.

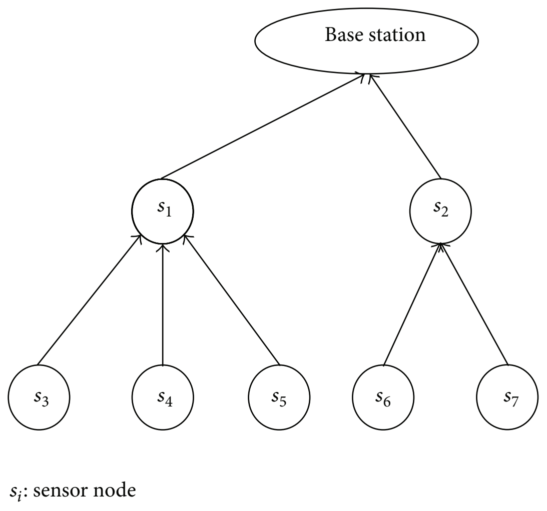


Figure 3: A simple sensor network.

eagerly, a variant of bloom filers was utilized in [20]. Refer to [21] for the summary of in-network aggregation.

3.2. Data Gathering. For the situations that require the sensor readings rather than an aggregate value, some approximate sensor data gathering techniques have been proposed since most applications of sensor networks do not require highly accurate data. Some correlations appear among sensor readings. Such correlations can be captured by standard techniques like the linear regression and statistical distribution functions. Basically, each sensor estimates its readings independently with its own model. And the mirror model for each sensor is in the base station. Thus, if a sensor node does not transmit a sensor reading, the base station can obtain an approximate reading using the mirror model.

BBQ [22] uses the multivariate Gaussian to model the sensor readings instead of data interval. Chu et al. [23] extends BBQ by partitioning the sensor field to cliques in order to utilize the spatial correlation. Since the optimal partitioning is NP-hard, Ken uses the greedy heuristics. Jain et al. suggested dual Kalman filter [24] which is based on the Kalman filter. In addition, Min and Chung proposed EDGES [25] based on a variant of the Kalman filter, that is, multimodel Kalman filter. In [8], by utilizing the spatial correlation such that the change patterns of sensor readings of the neighbor sensors are the same or similar, an effective data gathering technique was presented.

3.3. Join. In some applications, a user wants to identify the relationship between sensor readings in different regions. This regional correlation can be expressed as a join query of sensor readings in two regions. Thus, recently, research on in-network join processing has been proposed to reduce the communication overhead. Some works [26, 27] study how to find an optimal join location using the cost models. In these works, the optimal join location is near to the weighted centroid of three points: the center points of two regions and the base station.

Some in-network join techniques utilize a semijoin operator which filters out one of the relations based on the join attribute values of the other relation. However, due to a large number of join attribute values, a lot of energy is consumed. To alleviate this overhead, some work utilizes the synopsis of join attribute values. In [28], a histogram based semijoin approach is proposed. Stern et al. propose the method called SENS-Join [29], which is similar to that of [28], in order to avoid shipping tuples through the network that do not participate in the joins. As the compact representation, they use pointless quadtree representation.

3.4. Skyline. After Börzsönyi et al. [1] proposed the skyline operator, various techniques [30, 31] have been presented to improve the performance of skyline queries. The sort-filter skyline (SFS) algorithm [30] improves BNL using presorted data set according to the scores computed by a monotone function. By exploiting R*-tree, Kossmann et al. [31] presented an improved algorithm, called NN, based on the nearest neighbor search. The dynamic skyline was introduced by Papadias et al. [2]. Later on, the reverse skyline was proposed by Dellis and Seeger [3].

Since the skyline operator as well as its variants is useful to detect interesting events, there are some studies for in-network skyline processing in WSN environments. In [13], a filtering technique was proposed to reduce the energy consumption of WSNs in which some filter points are broadcasted to every sensor node and the data points dominated by the filter points are not transmitted since they cannot be in the skyline. Recently, a multiple filter-based algorithm called *SKYFILTER* was proposed to processing skyline over the sliding window in [14]. However, to compute filter points, every sensor node wastes its energy. The most related literature to our work is [6]. To obtain the reverse skyline points in WSNs, the *2-Skyband* query that retrieves every point which is dominated by at most one point wrt q is used. However, this technique calculates the reverse skyline with the currently generated points only. In other words, the reverse skyline processing over a sliding window is not supported. In contrast to previous work, we investigate effective reverse skyline processing techniques over a sliding window in data stream environments as well as WSN environments.

4. Reverse Skyline Processing over Sliding Windows

Before the presentation of the overall behavior of our proposed in-network reverse skyline processing, we first present the properties of the reverse skyline in Section 4.1. Since each sensor node in WSNs generates its readings continuously, each sensor node can be considered as a source of stream data. Thus, in Section 4.2, we present our basic algorithm in the context of stream data. Our in-network processing technique based on the basic algorithm will be presented in Section 5.

4.1. Properties of Reverse Skyline. In this section, we present the properties of the reverse skyline. Park et al. [32] showed that, when the d-dimensional space is divided into 2^d orthants with respect to a query point q as shown in Figure 4, the reverse skyline can be computed with each subset $P_o \subset P$ independently, where P_o denotes the set of points located in each orthant o.

Lemma 5. *Given a data set P, a query point q, and an orthant o, if and only if a point $p_i \in P_o \subseteq P$ is not in $RSL(q, P)$, then there exists another p_j in P_o which dynamically dominates q with respect to p_i (i.e., $p_j \prec_{p_i} q$).*

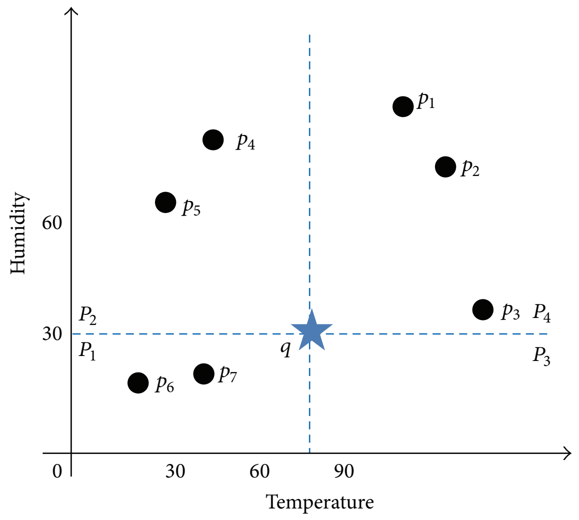


FIGURE 4: Subsets $P_{(o=1\cdots4)}$ in P.

Proof. (:⇐) For $p_i, p_j \in P_o$, if $p_j \prec_{p_i} q$, q is not a dynamic skyline with respect to p_i (i.e., $q \notin DSL(p_i, P \cup \{p\} - \{q_i\})$). By Definitions 2 and 3, p_i is not in $RSL(q, P)$.

(⇒:) If a point $p_i \in P_o$ is not in $RSL(q, P)$, there exists p_j such that $p_j \prec_{p_i} q$ by Definition 3. Then we have $|q \cdot x_k - p_i \cdot x_k| \geq |p_j \cdot x_k - p_i \cdot x_k| \forall k \in \{1, \ldots, d\}$. Squaring both sides, we get $0 \geq (p_j \cdot x_k - p_i \cdot x_k)^2 - (q \cdot x_k - p_i \cdot x_k)^2$. Rearranging terms, we have $0 \geq (p_j \cdot x_k + q \cdot x_k - 2p_i \cdot x_k) \cdot (p_j \cdot x_k - q \cdot x_k) = -2 \cdot (p_i \cdot x_k - q \cdot x_k)(p_j \cdot x_k - q \cdot x_k) + (p_j \cdot x_k - q \cdot x_k)^2$. Since $2 \cdot (p_i \cdot x_k - q \cdot x_k) \cdot (p_j \cdot x_k - q \cdot x_k) \geq (p_j \cdot x_k - q \cdot x_k)^2 \geq 0 \forall k \in \{1, \ldots, d\}$, $(p_i \cdot x_k - q \cdot x_k)$ and $(p_j \cdot x_k - q \cdot x_k)$ have the same sign. Thus, p_j is also in P_o. □

By Lemma 5, we have $RSL(q, P) = \cup_{\forall P_o} RSL(q, P_o)$. Now, for brevity, we explain our algorithm on a single orthant o and the corresponding data set $P_o \subseteq P$.

The following lemma addresses that every reverse skyline point wrt q is also a dynamic skyline point wrt q but not vice versa.

Lemma 6. *Given an orthant o and a query point q, $RSL(q, P_o) \subseteq DSL(q, P_o)$.*

Proof. For the purpose of contradiction, we assume $p_i (\in P_o) \notin DSL(q, P_o)$. Thus, there exists $p_j \in P_o$ such that $p_j \prec_q p_i$, and, hence, $|p_j \cdot x_k - q \cdot x_k| \leq |p_i \cdot x_k - q \cdot x_k|\ \forall k \in \{1, \ldots, d\}$ (and $\exists k \in \{1, \ldots\}$, $|p_j \cdot x_k - q \cdot x_k| < |p_i \cdot x_k - q \cdot x_k|$). It means that $p_i \cdot x_k$ is located farther than $p_j \cdot x_k$ from $q \cdot x_k$ in the orthant o. It implies that $p_j \cdot x_k$ is closer to $p_i \cdot x_k$ than $q \cdot x_k$ is. Obviously, $|p_j \cdot x_k - p_i \cdot x_k| \leq |q \cdot x_k - p_i \cdot x_k|\ \forall k$ (and $\exists k$, $|p_j \cdot x_k - p_i \cdot x_k| < |q \cdot x_k - p_i \cdot x_k|$). Then, we have $p_j \prec_{p_i} q$. Thus, $p_i \notin RSL(q, P_o)$. Therefore, $RSL(q, P_o) \subseteq DSL(q, P_o)$. □

From Lemma 6, in order to compute the reverse skyline of each P_o, a nonreverse skyline point in $DSL(q, P_o)$ should be eliminated. For this purpose, we utilize the idea of midpoints introduced in [3, 6, 32]. The midpoint m_i of a point p_i with

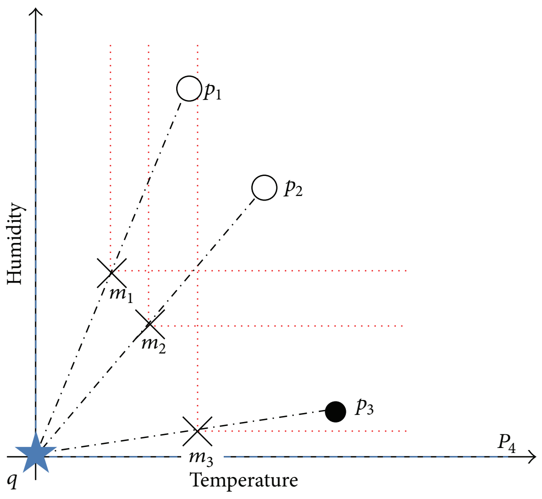


FIGURE 5: Midpoints in an orthant.

respect to a query point q is defined as $m_i = \langle (p_i \cdot x_1 + q \cdot x_1)/2, (p_i \cdot x_2 + q \cdot x_2)/2, \ldots, (p_i \cdot x_d + q \cdot x_d)/2 \rangle$.

Lemma 7. *Given an orthant o and a query point q, $p_i \in DSL(q, P_o)$ is not a reverse skyline if there exists another point $p_j \in P_o$ whose midpoint dynamically dominates p_i with respect to q (i.e., $m_j \prec_q p_i$).*

Proof (by contradiction). Assume that $p_i \in RSL(q, P_o) \subset DSL(q, P_o)$. Since $p_i \in RSL(q, P_o)$, there does not exist $p_j \in P_o$ s.t.; $p_j \prec_{p_i} q$. Then, in the proof of Lemma 5, we infer that there does not exist $p_j \in P_o$ satisfying $2 \cdot (p_i \cdot x_k - q \cdot x_k) \cdot (p_j \cdot x_k - q \cdot x_k) \geq (p_j \cdot x_k - q \cdot x_k)^2$ $\forall k \in \{1, \ldots, d\}$.

However, when $m_j \prec_q p_i$, we have $|(p_j \cdot x_k + q \cdot x_k)/2 - q \cdot x_k| = |p_j \cdot x_k - q \cdot x_k|/2 \leq |p_i \cdot x_k - q \cdot x_k|$ $\forall k \in \{1, \ldots, d\}$. By multiplying $2 \cdot (p_j \cdot x_k - q \cdot x_k)$ to both sides, we get $(p_j \cdot x_k - q \cdot x_k)^2 \leq 2 \cdot (p_j \cdot x_k - q \cdot x_k)(p_i \cdot x_k - q \cdot x_k)$. Therefore, by contradiction, $p_i \notin RSL(q, P_o)$ if $m_j \prec_q p_i$. □

Example 8. Consider a data set P in Figure 4. By Lemma 5, we can compute the reverse skyline on each orthant o independently.

Given a data set $P_4 = \{p_1, p_2, p_3\} \subset P$, as shown in Figure 5, since each point in P_4 is not dynamically dominating each other wrt q, every point is a dynamic skyline point (i.e., $DSL(q, P_4) = P_4$). However, a midpoint m_2 of p_2 dynamically dominates p_1 wrt q. Thus, by Lemma 7, p_1 is not a reverse skyline point (i.e., $p_1 \notin RSL(q, P_4)$). Similarly, p_2 does not belong to $RSL(q, P)$, either. In this example, p_3, denoted as a bold circle, is a reverse skyline point since p_3 is not dynamically dominated by any midpoint wrt q except its midpoint m_3.

4.2. Computing $RSL(q, P_o)$ over Sliding Windows. In this section, we present our algorithm, called *RSPW*, to compute reverse skyline over sliding windows in WSNs by utilizing the properties of reverse skyline presented in Section 4.1. Basically, *RSPW* is working on each sensor node to generate partial reverse skyline. In Section 5, we will present how to integrate the partial reverse skyline generated by each sensor node.

To compute the skyline over a sliding window, Tao and Papadias [33] proposed an effective method. Similarly, we need to keep the dynamic skyline based on Lemma 6 to obtain the reverse skyline. Thus, we adapt the sliding window skyline processing technique (denoted as *SWSP*) proposed in [33] to our reverse skyline processing over sliding windows.

Lemma 9 (see [33]). *Let p_i be a point in DB. If p_i is dominated by a newly generated point p_j, then p_i can be safely discarded from DB; that is, p_i will not be part of the skyline in the future.*

Since *SWSP* is for the skyline processing, *SWSP* considers a single data space. In addition, *SWSP* handles the database DB (i.e, the set of points which are alive) based on Lemma 9. Meanwhile, by Lemma 5, we can compute the reverse skyline on each orthant independently. For each orthant o, two buffers $o.Buff_{dsky}$ and $o.Buff_{rest}$ are maintained in our work. In addition, although *SWSP* maintains DB efficiently based on Lemma 9, Lemma 9 does not hold in our work since we need to prune out the nonreverse skyline points from $DSL(q, P_o)$ by Lemma 7.

For instance, as shown in Figure 6(a), p_1 and p_2 were generated where a point p_i is generated at time i. Let a window size w be 3. As shown in Figure 6(a), when time t is 2, p_2 dynamically dominates p_1 wrt q. Thus, p_1 is not a dynamic skyline point nor a reverse skyline point either. Meanwhile, p_2 is a dynamic skyline point but is not a reverse skyline point since the midpoint m_1 of p_1 dynamically dominates p_2. To indicate whether a dynamic skyline point is not a reverse skyline, we assign a *mark* to the point which is not a reverse skyline point. Note that, even though we use a mark, we cannot discard p_1 simply in this example.

As shown in Figure 6(b), p_1 expires when time $t = 4$ since $w = 3$. If other points are not generated within the time interval [13, 30], p_2 should become a reverse skyline point at $t = 4$ since no midpoint dominates p_2. In this case, if we discard p_1 at $t = 2$, we do not have a time information for p_2 being a reverse skyline point. In other words, a mark itself is not sufficient to preserve the dominance relationship with respect to time. To keep such information, each mark has an expiry time. In Figure 6(a), a mark with an expiry time $t_{\exp}$ is represented by "$*, t_{\exp}$."

In our work, $o.Buff_{dsky}$ keeps the dynamic skyline in the orthant o at the current time. Every nonreverse skyline point among the dynamic skyline points has a mark with its expire time. $o.Buff_{rest}$ maintains the dynamic skyline candidates which will be a part of the dynamic skyline in the future. To maintain $o.Buff_{dsky}$ and $o.Buff_{rest}$, we devise the following proposition and lemma.

Proposition 10. *Given a query point q and an orthant o, every point p_i in $o.Buff_{dsky} \cup o.Buff_{rest}$ dynamically dominated by the midpoint of a newly generated point p_j could not be a reverse skyline point for p_j's lifespan.*

Proof. It trivially holds by Lemma 7. □

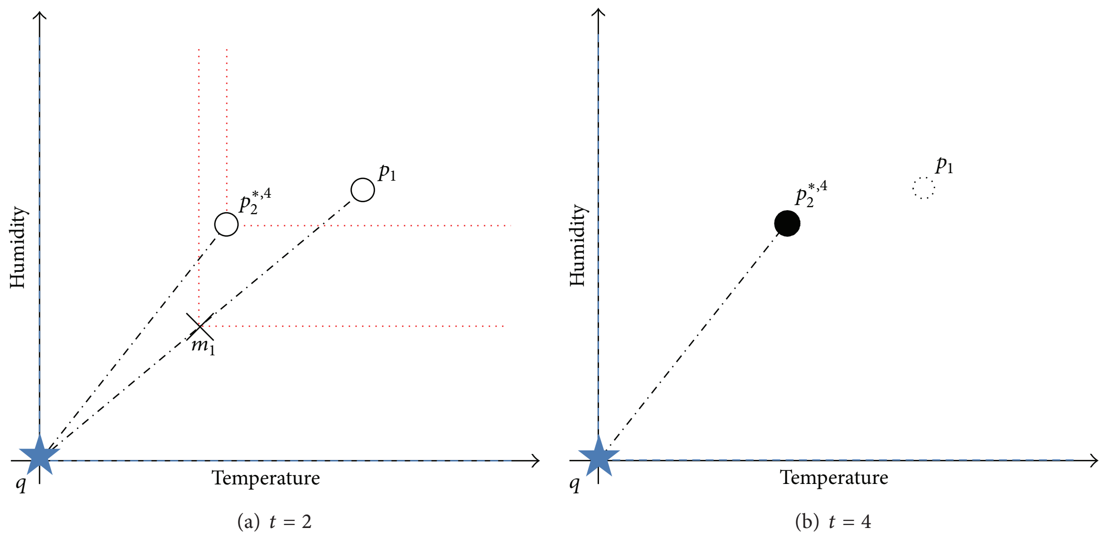


Figure 6: An example for a sliding window ($w = 3$).

By Proposition 10, when a point p_j appears in an orthant o at time j, every point $p_i \in (o.Buff_{dsky} \cup o.Buff_{rest})$ dynamically dominated by m_j of p_j is assigned a mark with expiry time (i.e., t_{exp}) as $p_j \cdot t_e$.

Lemma 11. *Given a query point q and an orthant o, every point p_k in $o.Buff_{dsky}$ and $o.Buff_{rest}$ dynamically dominated by a newly generated point p_j (i.e., $p_j \prec_q p_k$) can be discarded. In addition, if there is a point p_i such that p_i has the largest expiry time among the points in $o.Buff_{dsky} \cup o.Buff_{rest}$ whose midpoints dynamically dominate p_j, then p_j cannot be a reverse skyline point within p_i's lifespan.*

Proof. By Definition 2, given a new point p_j, every $p_k \in (o.Buff_{dsky} \cup o.Buff_{rest})$ such that $p_j \prec_q p_k$ cannot be a dynamic skyline point. In addition, since p_j is newly generated, every p_k will expire before p_j. Thus, every p_k cannot be a reverse skyline point within its lifespan due to p_j. In addition, by definition of midpoints, if $p_j \prec_q p_k$, then $m_j \prec_q m_k$. Thus, since every point dynamically dominated by m_k is also dynamically dominated by m_j, we can discard p_k. Meanwhile, by Lemma 7, since the point p_i is the point whose expire time is largest among the points whose midpoints dynamically dominate p_j, p_j cannot be a reverse skyline during p_i's lifespan. □

By Lemma 11, $p_j \in P_o$ has a mark with expiry time as $\max_{p_i \in P_o^j}(p_i \cdot t_e)$ where $P_i^j = \{p_i \in (o.Buff_{dsky} \cup o.Buff_{rest}) \mid m_i \prec_q p_j\}$. In addition, we can remove every point p_i in $o.Buff_{dsky}$ and $o.Buff_{rest}$ if p_i is dynamically dominated by the incoming point p_j.

The pseudocode of our proposed algorithm, denoted by *RSPW*, is presented in Pseudocode 1. The algorithm *RSPW* computes the reverse skyline over a sliding window. *RSPW* consists of two parts. The first one is for processing a newly created point p_j at the current time j (lines 1–12 in Pseudocode 1). The other one is for managing expired points and expired marks at the current time j (lines 14–20).

Recall that we maintain two buffers for each orthant o: $o.Buff_{dsky}$ and $o.Buff_{rest}$. Given a sliding window sized w, $o.Buff_{dsky}$ maintains the dynamic skyline points in the orthant o. The nonreverse skyline points in $o.Buff_{dsky}$ are annotated with marks. The buffer $o.Buff_{rest}$ keeps the dynamic skyline candidates which can be a reverse skyline in the future.

When a new point p_j is generated at the current time j in an orthant o (line 1), *RSPW* investigates whether p_j is a dynamic skyline point or not by comparing with every point p_i in $o.Buff_{dsky}$ and $o.Buff_{rest}$. If p_j is dynamically dominated by p_i, since p_j cannot be a dynamic skyline, the flag is_*dsky* sets to *false* (line 5). If there is a point p_i such that the midpoint m_i of p_i dynamically dominates p_j, p_j should have a mark with an expiry time. Thus, based on Lemma 11, *RSPW* maintains the largest expiry time for p_j's mark in mark_time (line 6). In addition, *RSPW* removes p_i if p_i is dynamically dominated by Lemma 11. When p_i is not dynamically dominated by p_j, p_i can become a dynamic skyline point. But if p_i is dynamically dominated by the midpoint m_j of p_j, since p_i will not be a reverse skyline point, we assign a mark with an expiry time as $p_j \cdot t_e$ to p_i due to Proposition 10 (line 8). After iterating all points in $o.Buff_{dsky}$ and $o.Buff_{rest}$, *RSPW* assigns a mark with mark_time to p_j if it is required (line 10). And p_j is inserted into $o.Buff_{dsky}$ or $o.Buff_{rest}$ with respect to the flag is_*dsky* (lines 11-12).

When a point p_i is expired at the current time j (i.e., $p_i \cdot t_e = j$) (line 15), p_i should be eliminated. By elimination of p_i, every point p_k in $o.Buff_{rest}$ which is dynamically dominated by p_i exclusively becomes dynamic skyline point at time j. Thus, the algorithm *RSPW* moves such p_k in $o.Buff_{rest}$ to $o.Buff_{dsky}$ and removes p_i (lines 15–18). In addition, *RSPW* unmarks every point p_i in $o.Buff_{dsky}$ whose mark's expiry time is j (line 19).

```
Procedure $RSPW(P_o)$
//$P_o$ is an input stream and this is invoked at each time $j$
//the midpoint of a point $p_i$ is denoted as $m_i$
begin
(1)  Let $p_j$ be a newly generated point at the current time $j$ and located on an orthant $o$;
(2)  mark_time = 0
(3)  is_$dsky = true$
(4)  for each point $p_i \in (o.Buff_{dsky} \cup o.Buff_{rest})$ do {
(5)        if $p_i \prec_q p_j$ then is_$dsky = false$
(6)        if $m_i \prec_q p_j$ and mark_time < $p_i \cdot t_e$ then mark_time = $p_i \cdot t_e$   //Lemma 11
(7)        if $p_j \prec_q p_i$ then remove $p_i$                                     //Lemma 11
(8)        else if $m_j \prec_q p_i$ then mark $p_i$ where $t_{\exp} = p_j \cdot t_e$          //Proposition 10
(9)  }
(10) if mark_time $\neq 0$ then mark $p_j$ where $t_{\exp}$ = mark_time              //Lemma 11
(11) if is_$dsky = true$ then insert $p_j$ into $o.Buff_{dsky}$
(12) else insert $p_j$ into $o.Buff_{rest}$
(13)
(14) for each point $p_i \in o.Buff_{dsky}$ do {
(15)       if $p_i \cdot t_e = j$ then {//$p_i$ expires at this time
(16)             move every $p_k \in o.Buff_{rest}$ which is exclusively dominated by $p_i$ to $o.Buff_{dsky}$
(17)             remove $p_i$ from $o.Buff_{dsky}$
(18)       }
(19)       else if $p_i$ has a mark and $t_{\exp} = j$ then unmark $p_i$
(20) }
(21) return $o.Buff_{dsky}$
end
```

Pseudocode 1: The pseudocode of *RSPW*.

Finally, the set of dynamic skyline point (i.e., $o.Buff_{dsky}$) is returned (line 21). Recall that every nonreverse skyline point has a mark. Thus, we can easily identify reverse skyline points from the dynamic skyline points in $o.Buff_{dsky}$.

The following example illustrates the behavior of our proposed algorithm *RSPW* within a single orthant o.

Example 12. Let the size of window w be 2 and each point p_t be generated at time t. Figure 7(a) shows the states of the points generated when t is 1 to 4. When $t = 1$, since there is p_1 only in an orthant o, p_1 is a dynamic skyline point (and a reverse skyline point), and, hence, p_1 is in $o.Buff_{dsky}$. When $t = 2$, since p_2 is dynamically dominated by p_1, p_2 is in $o.Buff_{rest}$. In addition, since the midpoint m_2 of p_2 dynamically dominates p_1, p_1 is annotated with a mark $*, 4$. Since p_2 is also dynamically dominated by m_1, p_2 has a mark $*, 3$. When $t = 3$, since p_3 is not dynamically dominated by any other point as well as the other midpoints, p_3 becomes a reverse skyline point and is in $o.Buff_{dsky}$. In addition, since p_1's expiry time is 3, p_1 is removed, and, then, p_2 becomes a dynamic skyline point. Thus, p_2 moves to $o.Buff_{dsky}$. Furthermore, the expiry time of p_2's mark is 3, and p_2 becomes a reverse skyline point.

As shown in Figure 7(b), since $p_2 \cdot t_e = 4$, p_2 is expired when $t = 4$. Since p_4 dominates $p_3 \in o.Buff_{dsky}$, p_4 becomes a dynamic skyline point and p_3 is discarded. Since m_3 does not dynamically dominate p_4, p_4 has no mark (i.e., p_4 is a reverse skyline point at $t = 4$). In addition, when $t = 5$, p_5 is newly generated. Since p_5 and p_4 do not dynamically dominate each other, p_5 and p_4 are dynamic skyline points as well as p_4 is not removed. However, since their midpoints m_4 and m_5 dynamically dominate p_5 and p_4, respectively, p_4 and p_5 have marks.

Up to now, we present our algorithm to compute the reverse skyline over a sliding window in data stream environments. In the next section, we will describe how to calculate the reverse skyline in WSN environments.

5. Energy Efficient *RSPW* for WSNs

As mentioned above, the energy efficiency is the utmost important in WSN environments. A brute-force algorithm, denoted as $RSPW_{bf}$, to compute reverse skyline over a sliding window in WSNs is that every sensor node transmits its sensor readings to the base station along the routing path and the base station computes dynamic skyline using the algorithm *RSPW* presented in Section 4.2 and extracts reverse skyline points having no mark. However, since each sensor node blindly sends its readings to the base station, each sensor node consumes much energy.

Based on the following lemma, we can apply *RSPW* to each sensor node in WSNs.

Lemma 13. *Given an orthant o, a query point q, and two sensor nodes s_1 and s_2, let P_o^1 and P_o^2 be the set of points located in o and generated by s_1 and s_2, respectively. Then,*

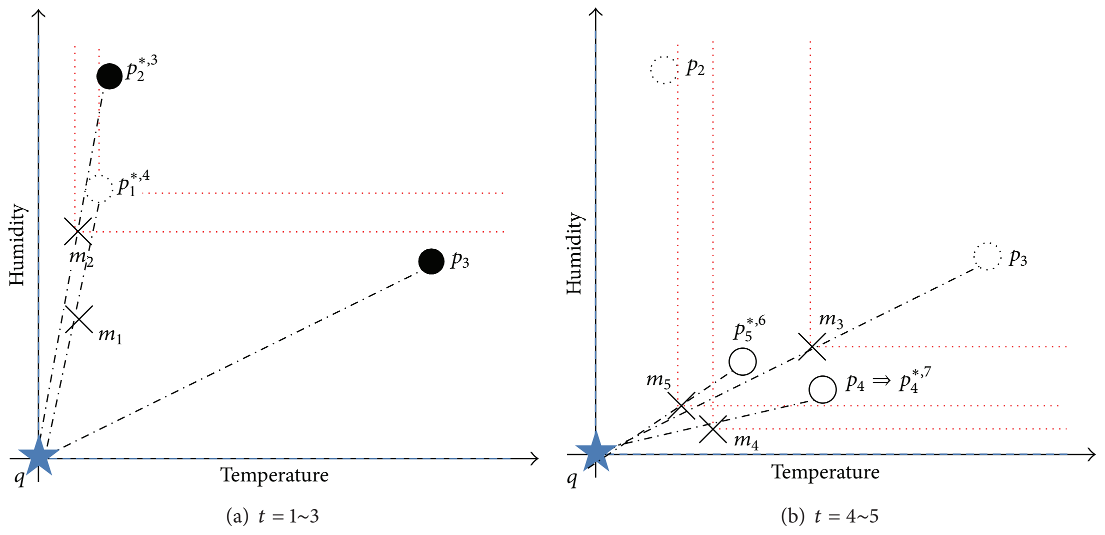


Figure 7: The behavior of *RSPW*.

p_i is in $RSPW(P_o^1 \cup P_o^2)$ iff p_i is in $RSPW(P_o^1 \cup RSPW(P_o^2))$, where $RSPW(P_o^k)$ returns $o.Buff_{dsky}$ of P_o^k. In addition, p_i in $RSPW(P_o^1 \cup P_o^2)$ has a mark iff p_i in $RSPW(P_o^1 \cup RSPW(P_o^2))$ has a mark.

Proof. Assume that $p_i \notin RSPW(P_o^1 \cup P_o^2)$ (i.e, p_i is not a dynamic skyline point in $(P_o^1 \cup P_o^2)$). Then there is another point p' in $(P_o^1 \cup P_o^2)$ such that $p' \prec_q p_i$. Let the result of $RSPW(P_o^2)$ be $o^2.Buff_{dsky}$ for brevity. If $p' \in P_o^1$, p_i is not in $RSPW(P_o^1 \cup RSPW(P_o^2))$ trivially. Otherwise, if $p' \in P_o^2$ and $o^2.Buff_{dsky}$, p_i cannot be in $RSPW(P_o^1 \cup RSPW(P_o^2))$ either. If $p' \in P_o^2$ but not in $o^2.Buff_{dsky}$, since p' is not a dynamic skyline point in P_o^2, there exists p'' in $o^2.Buff_{dsky}$ such that $p'' \prec_q p' (\prec_q p_i)$. Consequently, $p_i \in RSPW(P_o^1 \cup P_o^2)$ iff $p_i \in RSPW(P_o^1 \cup RSPW(P_o^2))$.

Now, we assume that p_i in $RSPW(P_o^1 \cup P_o^2)$ has a mark and p_i in $RSPW(P_o^1 \cup RSPW(P_o^2))$ does not have mark. Since p_i has a mark, there is a point p_j in $P_o^1 \cup P_o^2$, where the midpoint m_j of p_j dynamically dominates p_i. If p_j is in P_o^1, p_i in $RSPW(P_o^1 \cup RSPW(P_o^2))$ has also a mark trivially. Otherwise (i.e., $p_j \in P_o^2$), if p_j is in $o^2.Buff_{dsky}$, $p_i \in RSPW(P_o^1 \cup RSPW(P_o^2))$ also have a mark. Thus, in order not to have a mark, p_j should not be in $o^2.Buff_{dsky}$. It implies that $\exists p_j' \in o^2.Buff_{dsky}$ s.t. $p_j' \prec_q p_j$. By definition of midpoints, $m_j' \prec_q m_j$, and, hence, we have $m_j' \prec_q p_i$. Therefore, $p_i \in RSPW(P_o^1 \cup RSPW(P_o^2))$ must have a mark. Since the proof for the case that p_i in $RSPW(P_o^1 \cup P_o^2)$ does not have a mark and p_i in $RSPW(P_o^1 \cup RSPW(P_o^2))$ has mark is similar to the above, we omit it for brevity. □

By Lemma 13, a simple extension of *RSPW* to WSN environments is that, at each time, a sensor node performs *RSPW* with its sensor reading and the dynamic skyline points coming from its child nodes to maintain its $o.Buff_{dsky}$ and $o.Buff_{rest}$ as well as transmiting $o.Buff_{dsky}$ to its parent node. And, then, the parent node performs *RSPW* and so on. In this way, the base station obtains the complete $o.Buff_{dsky}$ for each orthant o. We denote the simple extension of *RSPW* to WSNs as $In\text{-}NetRSPW_{sim}$.

Since each sensor node transmits $o.Buff_{dsky}$ only in $In\text{-}NetRSPW_{sim}$, each sensor node can reduce its energy consumption. However, in $In\text{-}NetRSPW_{sim}$, a dynamic skyline point in a sensor node's $o.Buff_{dsky}$ can be transmitted redundantly (at most w times) within a window sized w. It incurs energy waste. Thus, we present an enhanced algorithm, referred to as $In\text{-}NetRSPW_{enh}$, which is also based on Lemma 13. The pseudocode of $In\text{-}NetRSPW_{enh}$ is presented in Pseudocode 2.

The intuition of $In\text{-}NetRSPW_{enh}$ is that a sensor reading becoming a dynamic skyline point newly and/or a dynamic skyline point which has a mark recently is transmitted only rather than transmitting all dynamic skyline points at each time in order to reduce the energy consumption of each sensor node. To do this, each sensor node s_i has $o.Buff^{sent}$ which consists of the dynamic skyline points (i.e., sensor readings) sent to its parent previously.

At first, each sensor node s_i collects sensor readings coming from its child nodes into $o.Buff^{temp}$ (lines 2–5). Then, s_i conducts *RSPW* with its sensor readings and sensor readings coming from its child nodes and the result of *RSPW* is kept in *o.Result* (lines 6-7). Before eliminating the point sent previously from *o.Result*, we remove every expired point from $o.Buff^{sent}$ and unmark the point whose mark's expire time (i.e., t_{exp}) is this time (lines 8-9). Then, every point p_k in *o.Result* is evaluated on whether p_k was sent previously (lines 10–17). If p_k was sent (i.e., p_k is in $o.Buff^{sent}$) and the status of p_k is not changed, we do not need to send p_k (lines 11–15). Otherwise, the old p_k in $o.Buff^{sent}$ is removed and the new p_k is inserted into $o.Buff^{sent}$ to maintain $o.Buff^{sent}$ properly

```
Procedure $In\text{-}NetRSPW_{enh}()$
begin
//This is invoked at each time $j$
//$P^i_o$ is the input stream of this sensor node $s_i$
//Let $o.Buff^{sent}$ be the set of points sent previously
(1)  for each orthant $o$ do {
(2)      $o.Buff^{temp} = \emptyset$
(3)      for each child node $s_c$ do {
(4)          $o.Buff^{temp} = o.Buff^{temp} \cup$ receiveFromChild($s_c$);
(5)      }
(6)      $P^i_o = P^i_o \cup Buff^{temp}$
(7)      $o.Result = RSPW(P^i_o)$
(8)      remove every point $p_k \in o.Buff^{sent}$ where $p_k \cdot t_e = j$
(9)      unmark every point $p_k \in o.Buff^{sent}$ where $t_{\exp} = j$
(10)     for each point $p_k \in o.Result$ do {
(11)         if $p_k \in o.Buff^{sent}$ then {
(12)             if $p_k \in o.Result$ has not a mark and $p_k \in o.Buff^{sent}$ has not a mark then
(13)                 remove $p_k$ from $o.Result$
(14)             else if $p_k \in o.Result$ has a mark and $p_k \in o.Buff^{sent}$ has a mark then
(15)                 remove $p_k$ from $o.Result$
(16)             else {
(17)                 remove $p_k$ from $o.Buff^{sent}$
(18)                 $o.Buff^{sent} = o.Buff^{sent} \cup \{p_k\}$
(19)             }
(20)         } else $o.Buff^{sent} = o.Buff^{sent} \cup \{p_k\}$
(21)     }
(22)     sendToParent($o.Result$)
(23) }
end
```

Pseudocode 2: The pseudocode of $In\text{-}NetRSPW_{enh}$.

(lines 16–19). If p_k was not sent, p_k is inserted in $o.Buff^{sent}$ (line 20). Finally, a sensor node s_i sends $o.Result$ to its parent (line 22).

Since $In\text{-}NetRSPW_{enh}$ transmits new dynamic skyline points and the dynamic skyline points whose states are changed, the energy consumption of $In\text{-}NetRSPW_{enh}$ is much smaller than those of $In\text{-}NetRSPW_{sim}$ and the brute-force algorithm $RSPW_{bf}$. We will show the energy efficiency of $In\text{-}NetRSPW_{enh}$ by conducting experiments with a real-life data set.

6. Performance Study

We empirically evaluated the performances of our proposed algorithms in two environments: data stream environments and WSN environments. In data stream environments, we measured the processing time of our proposed algorithm *RSPW* and other algorithms with the synthetic data sets. On contrary, in WSN environments, we present the energy consumption of our algorithms $In\text{-}NetRSPW_{sim}$ and $In\text{-}NetRSPW_{enh}$ with a real data set to show the effectiveness of our algorithms. All experiments were conducted on Intel i5 platform with MS-Windows 7 and 4GB MBytes of main memory.

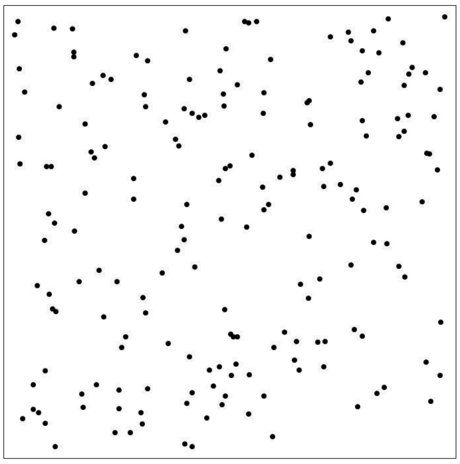
6.1. Experiments in Data Stream Environments

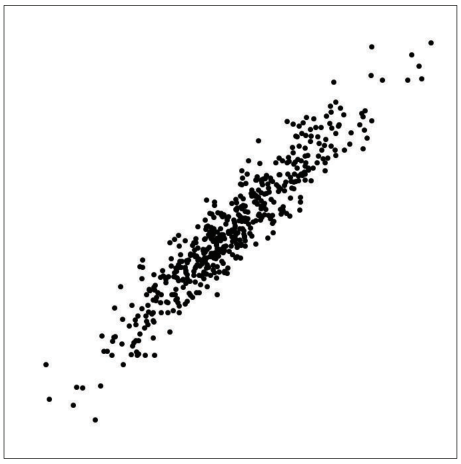
6.1.1. Experimental Environments. We performed this experiment to compare the execution time of *RSPW* with *Naive* and *2-Skyband* [6]. In *Naive* algorithm, each point in a window is compared with the other points in a window to check whether it is a reverse skyline point or not. In addition, since *2-Skyband* [6] did not consider the sliding window, we extended *2-Skyband* to the sliding window context which computed 2-skyband with recent w points at each time.

In order to evaluate the performance of each algorithm over diverse environments, we used three synthetic data sets which are generated by *independent*, *correlated*, and *anticorrelated* as shown in Figure 8. These three data sets are commonly used to evaluate the performance of the skyline operator as well as its variants [1, 32].

Table 1 shows the parameters used in this experiment. Each synthetic data set consists of 100,000 points. We ran all algorithms 10 times with different query points generated randomly and report the average execution times. We varied the number of data points' dimension from 2 to 10 as well as the windows size of a query from 2 to 10.

6.1.2. Experimental Results. Figure 9 shows the execution time of each algorithm according to the data sets with


(a) Independent


(b) Correlated

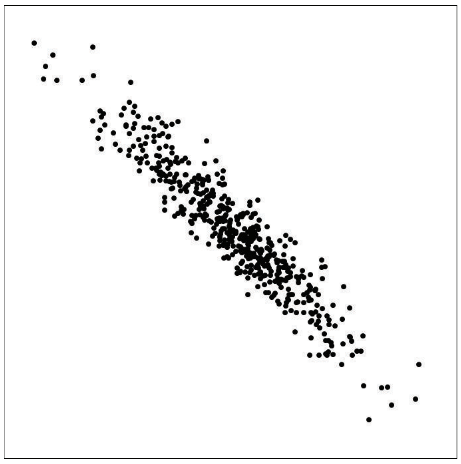

(c) Anticorrelated

Figure 8: Example of data sets (2-dimension).

Table 1: Parameters.

Parameter	Default	Range
Number of points	100,000	100,000
Number of queries	10	10
Number of dimensions (d)	6	2~10
Window size (w)	6	2~10

default values of the parameters. As shown in Figure 9, our proposed algorithm *RSPW* is the best performer. In average over all data sets, *RSPW* achieves up 9.23 times faster than *Naive* and 4.73 times faster than *2-Skyband*. In correlated and anticorrelated data sets, since the data distributions are skewed, a large number of points are dominated by a few points in each orthant and hence the number of reverse skyline points is small. Meanwhile, since the points are uniformly distributed in independent data set, the number of reverse skyline points is larger than those of the other data sets. Thus, the processing time for the independent data set is worse than those for the other data sets.

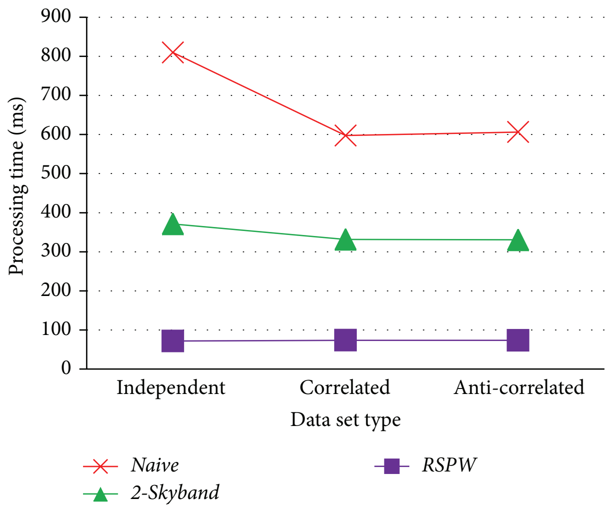
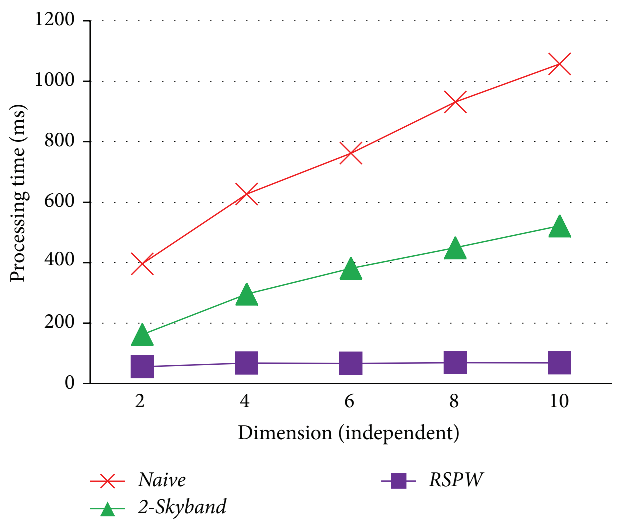
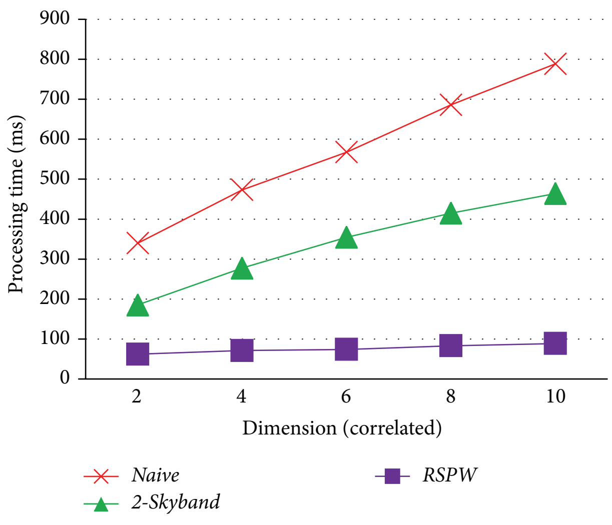


Figure 9: Execution time over each data set.

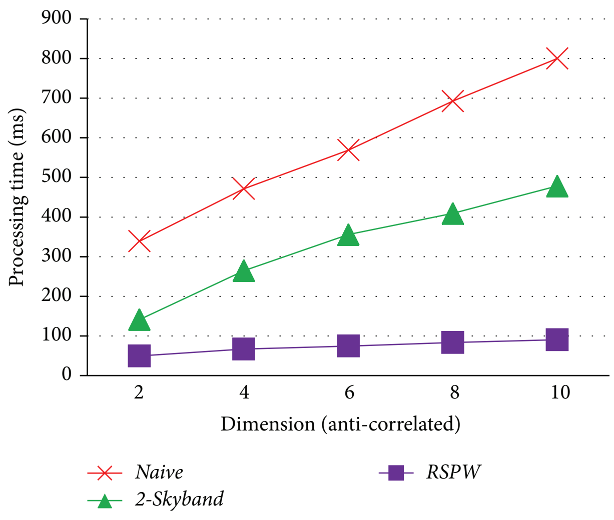
With varying d from 2 to 10, we plot the execution time of each algorithm in Figure 10. As shown in Figure 10, as



(a) Independent



(b) Correlated



(c) Anticorrelated

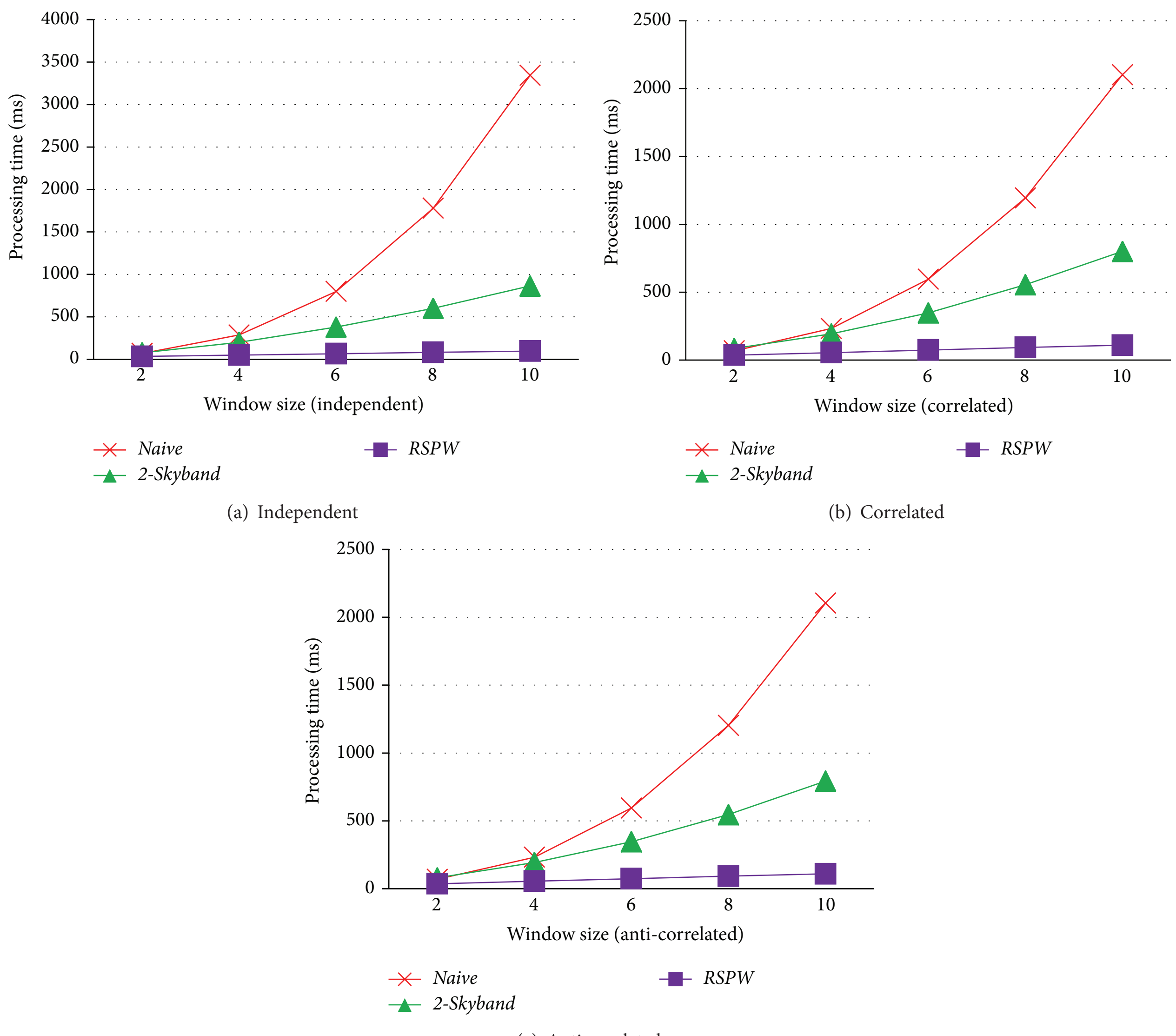
Figure 10: Varying d.

the number of dimensions d increases, the running time of each algorithm also increases since the overhead evaluating dominance relationship becomes increase with increasing d. However, the performance gap between *RSPW* and the other algorithms increases over all data sets as d increases since *RSPW* calculates the reverse skyline efficiently using two buffers.

We varied w from 2 to 10 and present the running times of the algorithms in Figure 11. As illustrated in Figure 11, when the window size w is small (i.e., $w = 2$), all algorithms show the similar performances. However, as w increases, the execution times of *Naive* and *2-Skyband* increase dramatically. In contrast, the execution time of *RSPW* increases slowly. This result indicates that *RSPW* computes reverse skyline efficiently over a sliding window.

6.2. *Experiments in WSN Environments*

6.2.1. Experimental Environments. We show show the effectiveness of our proposed algorithms for WSNs with a real-life data set. As a real-life data set, we used the data LUCE provided by Audiovisual Communications Laboratory [34]. A sensor network is composed of 89 nodes deployed on the EPFL campus as shown in Figure 12 and they measured key environmental quantities at high spatial and temporal resolution over a year. The data set consists of 9 attributes such as surface temperature, solar radiation, relative humidity, rain meter, and wind speed. The size of the sensing field is 277×430 meter^2 and the base station is located at the center of the sensing field. To make a routing tree, we set the communication distance to

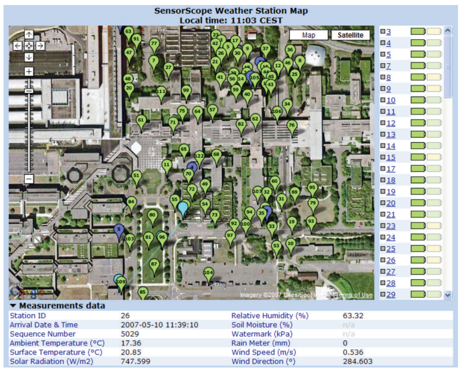


(a) Independent

(b) Correlated

(c) Anticorrelated

Figure 11: Varying w.


SensorScope Weather Station Map
Local time: 11:03 CEST
Map
Satellite
Imagery ©2007 Cnes/Spot Image, Terms of Use

▼ Measurements data

Station ID	26	Relative Humidity (%)	63.32
Arrival Date & Time	2007-05-10 11:39:10	Soil Moisture (%)	n/a
Sequence Number	5029	Watermark (kPa)	n/a
Ambient Temperature (°C)	17.36	Rain Meter (mm)	0
Surface Temperature (°C)	20.85	Wind Speed (m/s)	0.536
Solar Radiation (W/m2)	747.599	Wind Direction (°)	284.603

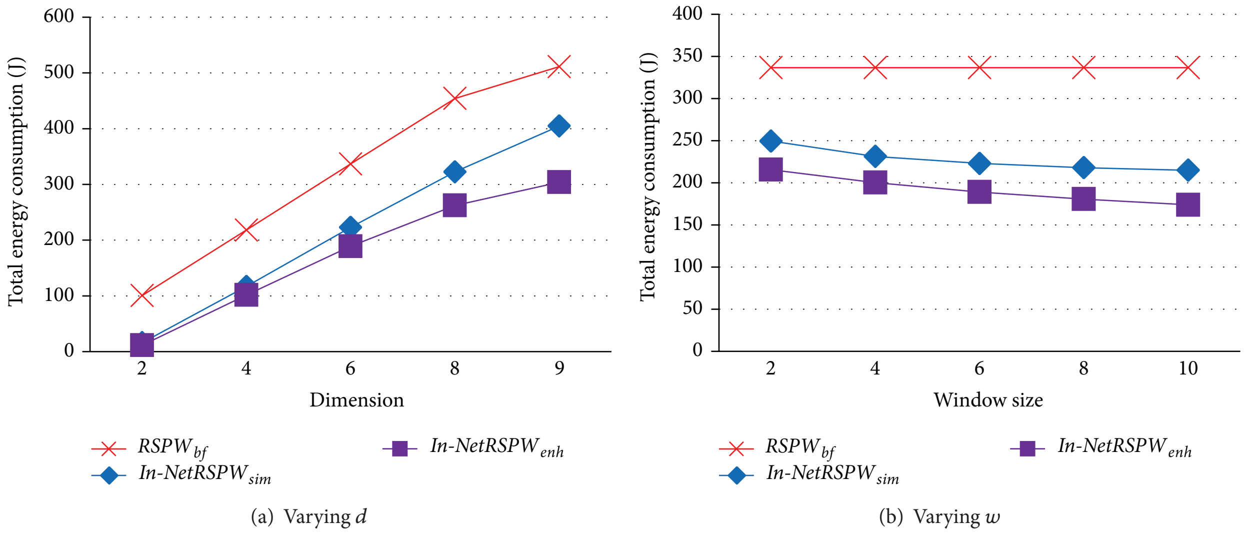
Figure 12: Placement of sensor nodes.

55 meter. The average depth (i.e., average number of child nodes) and the maximum width of the routing tree are 4.94 and 13, respectively. Since, in the real-life data set, the values of sensor readings are fixed, it is hard to make diverse configuration. Instead, to simulate diverse environments, we used some parameters. The parameters used for our experimental study are summarized in Table 2.

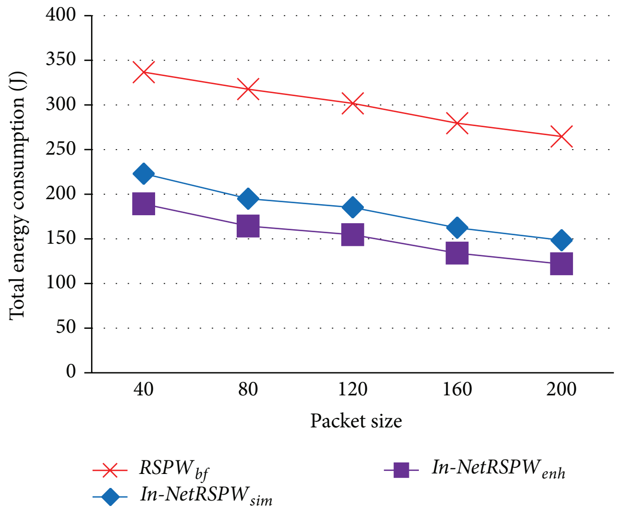
Table 2: Parameters.

Parameter	Default value	Range
Number of dimensions (d)	5	2, 3, 5, 7, 9
Window size (w)	12	4, 8, 12, 16, 20
Packet size (p)	40 bytes	40, 80, 120, 160, 200

For this experiment, we implemented $RSPW_{bf}$, $In\text{-}NetRSPW_{sim}$, and $In\text{-}NetRSPW_{enh}$ presented in Section 5. To compute the energy consumption of each algorithm, we used the free space channel model [35]. Under this model, to transmit a l-bits message and a distance



(a) Varying d (b) Varying w



(c) Varying p

Figure 13: The total energy consumption with the real-life data set.

c, a sensor expends $E_T(l, c) = E_{T-\text{elec}}(l) + E_{T-\text{amp}}(l, c) = l * E_{\text{elec}} + \xi_{\text{amp}} * l * c^2$. And, to receive this message, a sensor expends $E_R(l) = E_{R-\text{elec}}(l) = l * E_{\text{elec}}$. In this experiment, we set 50 nJ/bit to the electronic circuit constant (E_{elec}) and 100 pJ/bit/meter^2 to the transmit amplifier constant (ξ_{amp}). Like the experiments in data stream environments, we executed all algorithms 10 times with different query points generated randomly and report the average energy consumption of a network for 100,000 time units.

6.2.2. Experimental Results. We plotted the total energy consumption of the sensor network varying diverse parameter values in Figure 13. Figure 13(a) shows the consumed energy of each algorithm varying d. As the number of dimensions d increases, the energy consumption of each algorithm increases since the size of data to be transmitted increases. However, since our algorithms $In\text{-}NetRSPW_{sim}$ and $In\text{-}NetRSPW_{enh}$ transmit the dynamic skyline points only to the base station, the energy consumptions of $In\text{-}NetRSPW_{sim}$ and $In\text{-}NetRSPW_{enh}$ are less than that of $RSPW_{bf}$ in which every sensor node sends its readings to the base station blindly.

With varying the window size w from 2 to 10, we plot the energy consumption of each algorithm in Figure 13(b). Since, in $RSPW_{bf}$, each sensor sends its readings, the energy consumption of $RSPW_{bf}$ is not affected by the window size w. Interestingly, when w becomes large, the energy consumptions of our algorithms decrease. As w increases, the lifespan of each point also increases. Thus, when a point p_i becomes a dynamic skyline point, it will stay in $o.Buff_{dsky}$ for a long time and the number of points dynamically dominated by p_i increases as w increases. Therefore, the data volume to be transmitted decreases in our algorithms since

In-NetRSPW$_{sim}$ and *In-NetRSPW*$_{enh}$ transmit the dynamic skyline points in $o.Buff_{dsky}$. Furthermore, *In-NetRSPW*$_{enh}$ is better than *In-NetRSPW*$_{sim}$ since *In-NetRSPW*$_{enh}$ avoids redundant transmissions.

Figure 13(c) presents the consumed energy of each algorithm varying the packet size p. As the packet size p increases, the number of transmissions decreases since many points can be in a packet. Thus, the energy consumption of each algorithm decreases with increasing p. However, our enhanced algorithm *In-NetRSPW*$_{enh}$ shows the best performance.

7. Conclusion

In this paper, we present an algorithm *RSPW* to compute the reverse skyline over a sliding window. To calculate the reverse skyline, we divide d-dimensional data space into 2^d orthants. Basically, *RSPW* computes the reverse skyline in each orthant independently. If a dynamic skyline point in an orthant is dominated by the midpoint of another point, it is annotated with a mark since it cannot be a reverse skyline. To denote the valid time of a mark within a window, each mark has an expire time. We also extend *RSPW* to WSN environments. Since our enhanced algorithm *In-NetRSPW*$_{enh}$ transmits new dynamic skyline points and the dynamic skyline points which has a mark recently, the energy consumption of each sensor node is reduced. We implemented our algorithms and conducted an extensive evaluation with synthetic and real-life data sets. In our experiments, we demonstrated that the performance of our proposed algorithm is significantly better than other algorithms in data stream environments as well as WSN environments.

Conflict of Interests

The author declares that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work was supported by Defense Acquisition Program Administration and Agency for Defense Development under Contract UD140022PD, Republic of Korea.

References

[1] S. Börzsönyi, D. Kossmann, and K. Stocker, "The skyline operator," in *Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE '01)*, pp. 421–430, April 2001.

[2] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "An optimal and progressive algorithm for skyline queries," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '03)*, pp. 467–478, San Diego, Calif, USA, June 2003.

[3] 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, 2007.

[4] J. Lee, S. won Hwang, Z. Nie, and J.-R. Wen, "Navigation system for product search," in *Proceedings of the 26th International Conference on Data Engineering (ICDE '10)*, Long Beach, Calif, USA, March 2010.

[5] J. J. Levandoski, M. F. Mokbel, and M. E. Khalefa, "Preference query evaluation over expensive attributes," in *Proceedings of the 19th International Conference on Information and Knowledge Management and Co-Located Workshops (CIKM '10)*, pp. 319–328, Ontario, Canada, October 2010.

[6] 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.

[7] L. Zou, L. Chen, M. T. Özsu, and D. Zhao, "Dynamic skyline queries in large graphs," in *Proceedings of the 15th International Conference on Database Systems for Advanced Applications (DASFAA '10)*, pp. 62–78, Tsukuba, Japan, April 2010.

[8] J.-K. Min, "CMOS: efficient clustered data monitoring in sensor networks," *The Scientific World Journal*, vol. 2013, Article ID 704957, 11 pages, 2013.

[9] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tag: a tiny aggregation service for ad-hoc sensor networks," in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI '02)*, Boston, Mass, USA, December 2002.

[10] A. Silberstein, K. Munagala, and J. Yang, "Energy-efficient monitoring of extreme values in sensor networks," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '06)*, pp. 169–180, June 2006.

[11] D. J. Abadi, S. Madden, and W. Lindner, "REED: robust, efficient filtering and event detection in sensor networks," in *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB '05)*, pp. 769–780, September 2005.

[12] J.-K. Min, J. Kim, and K. Shim, "TWINS: efficient time-windowed in-network joins for sensor networks," *Information Sciences*, vol. 263, pp. 87–109, 2014.

[13] B. Chen and W. Liang, "Progressive skyline query processing in wireless sensor networks," in *Proceedings of the 5th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN '09)*, pp. 17–24, December 2009.

[14] Y. J. Roh, I. Song, J. H. Jeon, K. G. Woo, and M. H. Kim, "Energy-efficient two-dimensional skyline query processing in wireless sensor networks," in *Proceedings of the IEEE 10th Consumer Communications and Networking Conference (CCNC '13)*, pp. 294–301, January 2013.

[15] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TinyDB: an acquisitional query processing system for sensor networks," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 122–173, 2005.

[16] I. Galpin, C. Y. A. Brenninkmeijer, A. J. G. Gray, F. Jabeen, A. A. A. Fernandes, and N. W. Paton, "Snee: a query processor for wireless sensor networks," *Distributed and Parallel Databases*, vol. 29, no. 1-2, pp. 31–85, 2011.

[17] B. Sundararaman, U. Buy, and A. D. Kshemkalyani, "Clock synchronization for wireless sensor networks: a survey," *Ad Hoc Networks*, vol. 3, no. 3, pp. 281–323, 2005.

[18] J. Considine, F. Li, G. Kollios, and J. W. Byers, "Approximate aggregation techniques for sensor databases," in *Proceedings of the 20th International Conference on Data Engineering (ICDE '04)*, pp. 449–460, March-April 2004.

[19] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri, "Medians and beyond: new aggregation techniques for sensor networks," in *Proceedings of the 2nd International Conference on*

Embedded Networked Sensor Systems (SenSys '04), pp. 239–249, ACM, November 2004.

[20] J.-K. Min, R. T. Ng, and K. Shim, "Aggregate query processing in the presence of duplicates in wireless sensor networks," *Information Sciences*, vol. 297, pp. 1–20, 2015.

[21] E. Fasolo, M. Rossi, J. Widmer, and M. Zorzi, "In-network aggregation techniques for wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 14, no. 2, pp. 70–87, 2007.

[22] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB '04)*, pp. 588–599, Trondheim, Norway, August 2004.

[23] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *Proceedings of the 22nd International Conference on Data Engineering (ICDE '06)*, p. 48, April 2006.

[24] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive stream resource management using Kalman Filters," in *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '04)*, pp. 11–22, June 2004.

[25] J.-K. Min and C.-W. Chung, "EDGES: efficient data gathering in sensor networks using temporal and spatial correlations," *Journal of Systems and Software*, vol. 25, no. 5, pp. 933–944, 2010.

[26] A. Coman, M. A. Nascimento, and J. Sander, "On join location in sensor networks," in *Proceedings of the 8th International Conference on Mobile Data Management (MDM '07)*, pp. 190–197, May 2007.

[27] A. Pandit and H. Gupta, "Communication-efficient implementation of range-joins in sensor networks," in *Proceedings of the 11th International Conferenceon Database Systems for Advanced Applications (DASFAA '06), Singapore, April 2006*, vol. 3882 of *Lecture Notes in Computer Science*, pp. 859–869, Springer, Berlin, Germany, 2006.

[28] H. Yu, E.-P. Lim, and J. Zhang, "On in-network synopsis Join processing for sensor networks," in *Proceedings of the 7th International Conference on Mobile Data Management (MDM '06)*, p. 32, May 2006.

[29] M. Stern, E. Buchmann, and K. Böhm, "Towards efficient processing of general-purpose joins in sensor networks," in *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE '09)*, pp. 126–137, Shanghai, China, April 2009.

[30] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang, "Skyline with presorting," in *Proceedings of the 19th IEEE International Conference on Data Engineering (ICDE '03)*, pp. 717–719, March 2003.

[31] D. Kossmann, F. Ramsak, and S. Rost, "Shooting stars in the sky: an online algorithm for skyline queries," in *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB '02)*, pp. 275–286, 2002.

[32] Y. Park, J.-K. Min, and K. Shim, "Parallel computation of skyline and reverse skyline queries using mapreduce," *Proceedings of the VLDB Endowment*, vol. 6, no. 14, pp. 2002–2013, 2013.

[33] 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.

[34] Advanced Compliance Laboratory, *Luce Deployment*, Advanced Compliance Laboratory, Hillsborough Township, NJ, USA, 2006, http://lcav.epfl.ch/page-86035-en.html.

[35] W. R. Heinzelman, A. Chandrakasan, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks," in *Proceedings of the International Conference on System Sciences*, pp. 1–10, 2000.