G-skyline query over data stream in wireless sensor network

Leigang Dong^{1,2} · Guohua Liu³ · Xiaowei Cui² · Tianyu Li⁴

Published online: 9 July 2018
© The Author(s) 2018

Abstract

There are much data sampled continuously by sensors in the wireless sensor network. Storing and mining these data can find more potential information and provide help for decision making. As an important technology for data mining and multi-criteria decision, skyline computation can identify the interesting single points for user. In order to analyze the groups of points, the group-based skyline is proposed to query all the Pareto Optimal groups which are not g-dominated by other groups with the same number of points. Existing algorithms about g-skyline can just compute static data. However, data stream is very common in many applications, and it is very important to design algorithm go query g-skyline over data stream. In this paper, we propose new algorithms to compute g-skyline over a data stream. We present sharing strategy and then present two efficient algorithms: point-arriving algorithm and point-expiring algorithm. The experimental results on three kinds of synthetic data and a real stock data show that our algorithms perform efficiently over a data stream.

Keywords Wireless sensor network · Group-based skyline · Data stream · Sharing strategy

1 Introduction

The Internet of things is the inter-networking of physical devices, vehicles and other items embedded with different information sensors. These wireless sensors can collect much data from the terminals. The most importance is how to dig useful information from these mass data for special purpose.

As one of the important means of multi-decision making, skyline query plays an important role in the applications of sensor network, data mining and so on. The skyline of a data set includes all the points which are not worse than any other points. Given a data set D with d -dimension, every point q can be written as $(q[1], q[2], \dots, q[d])$ where

$q[i]$ is the i th attribute value of q . Assume that there are two points $p = (p[1], p[2], \dots, p[d])$ and $q = (q[1], q[2], \dots, q[d])$ in $\mathbb{R}^d$, we say q dominates p , if $q[j] \leq p[j]$ for each j and there is at least one $j (1 \leq j \leq d)$, $q[j] < p[j]$. The skyline of D consists of all the points which are not dominated by any other points in D . So the skyline query identifies all the best individual points.

For example, in the application of forest fires monitoring, the wireless sensor nodes can perceive nearby temperature, humidity and smoke density. When the fire happens, the nearby temperature will increase and the humidity will decrease, and the nearby sensors can perceive these changes. So a wireless sensor network can be arranged to monitor fire. For convenience, we assume that the sensors sample data at the same time in this paper. As shown in Fig. 1(a), there is a data set $D = \{p_1, p_2, \dots, p_{11}\}$, and each point represents a sensor node with two attributes: the inverse-temperature and the humidity. Doing the skyline query on the sensor nodes can return the skyline points with lower inverse-temperature and lower humidity, as shown in Fig. 1(b), and these indicate the dangerous areas. We find that the point p_6 dominates point p_3 because the inverse-temperature and humidity of p_6 are smaller than that of p_3 . The skyline of the dataset D consists of p_1, p_6 and p_{11} . Therefore, the firemen can quickly identify these dangerous areas and take an earlier action.

✉ Leigang Dong
lgdong010@163.com

¹ School of Information Science and Technology, Donghua University, Shanghai 201620, China

² School of Computer Science and Information Technology, Daqing Normal University, Daqing 163712, China

³ School of Computer Science and Technology, Donghua University, Shanghai 201620, China

⁴ School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

sensor	inverse-temperature	humidity	sensor	inverse-temperature	humidity
p_1	4	400	P_7	40	200
p_2	24	380	P_8	20	180
p_3	14	340	P_9	34	140
P_4	36	300	p_{10}	28	120
P_5	26	280	p_{11}	16	60
P_6	8	260			

(a)

(b)

Fig. 1 An example of skyline. **a** Dataset, **b** skyline

However, the fire forces are limited and we cannot check each area at the same time, so if we intend to select 2 areas to examine, the traditional skyline query will not return the result directly. In order to solve such problem in a better way, the group-based skyline query was proposed. The group-based skyline (G-Skyline for short) query is to identify the best groups not g-dominated by any other groups with the same group size, and paper [1] proposed two algorithms for computing G-Skyline. Different from the traditional skyline, the G-Skyline presents much more useful information in more complexity phenomenons such as wireless sensor network, multi-decision and data mining. In the above example, the G-Skyline groups with 2 points include $\{p_1, p_6\}$, $\{p_1, p_{11}\}$, $\{p_6, p_{11}\}$, $\{p_6, p_3\}$, $\{p_{11}, p_8\}$, $\{p_{11}, p_{10}\}$, then the fire force could consider selecting one group from the result.

Although the G-Skyline is very useful, the existing algorithms focus on the static data set. In fact, in the wireless network, each sensor node maybe send the perceived data to receiver at intervals, and the environmental intrusion or node-fault maybe affect the perceived data to be send, therefore the data received is dynamic. We can regard these dynamic data as data stream. In the data stream, each data has its life cycle, and it is only effective in its life cycle, so when a data arrives or expires, the current active data set will changes. Based on the data in

Fig. 1, we give an example of data stream in Fig. 2. At first, there are 3 points $\{p_1, p_2, p_3\}$, at moment t , a new point p_4 arrives, the active points are $\{p_1, p_2, p_3, p_4\}$, and at moment $2t$, an old point p_2 expires, now the active points are $\{p_1, p_3, p_4\}$. With the dataset changing, the groups set will also change, for example, at first, the groups with 2 points are $\{p_1p_2, p_2p_3, p_1p_3\}$, when the point p_4 arrives, the groups with 2 points change to $\{p_1p_2, p_1p_3, p_1p_4, p_2p_3, p_2p_4, p_3p_4\}$, the groups containing p_4 appear, when the point p_2 expires, the groups with 2 points change to $\{p_1p_3, p_1p_4, p_3p_4\}$, the groups containing p_2 are removed.

With the groups set changing, the corresponding G-Skyline maybe also change. For example, at first, the G-Skyline groups are $\{p_1p_3, p_2p_3\}$ because p_1p_2 is dominated by p_2p_3 , while p_1p_3 and p_2p_3 are not be g-dominated by any other groups. When p_4 arrives, the G-Skyline groups are $\{p_1p_3, p_1p_4, p_3p_4, p_2p_3\}$, we can see that the new point arriving affect the G-Skyline result. When p_2 expires, the G-Skyline changes to $\{p_1p_3, p_1p_4, p_3p_4\}$, the result shows the old point expiring affect the G-Skyline too. So in order to keep the G-Skyline effective all the time in the data stream, we should update the G-Skyline when the new data arrives or an old data expires.

The naive method to find the G-Skyline over a data stream is to use the existing algorithm Pwise in paper [1] directly when the data set changes. However, the data maybe change quickly in the data stream, under this circumstance, repeating Pwise computation for the whole data set will need much time cost and much redundant computation. Because when a new data arrives or an old data expires, some G-Skyline groups may not change their status. For example, in Fig. 2, the groups p_1p_3 and p_2p_3 are G-Skyline groups at the moment t and $2t$. That is to say, in a data stream, when a point arrives or expires, we do not compute all the groups again. So the naive method is not a good idea.

In this paper, we present two algorithms to efficiently find G-Skyline over a data stream in the wireless sensor network. The point-arriving-algorithm can compute the new G-Skyline when a new point arrives, and the point-

Fig. 2 An example of data stream

expire-algorithm will get the new G-Skyline result while an old point expires. In order to improve this two algorithms effectively, we present some pruning theorems to remove the groups which will not affect the new G-Skyline.

We summarize our main contributions in brief as follows.

- We present the problem of finding the G-Skyline with k points over a data stream in the wireless sensor network. This query will provide much more useful information.
- We present a sharing strategy to make us compute the new G-Skyline based on the existing result. According to the pruning theorems, lots of groups will be pruned without computation.
- We propose two algorithms to compute the new G-Skyline over a data stream.
- The experiments are performed based on three kinds of synthetic data and a real stock data.

Organization The rest of this paper is organized as follows. In Sect. 2, we review the related work to our research work. In Sect. 3, we firstly give the definition and existing theorems of G-Skyline, then we define the problem of G-Skyline query over a data stream formally. Section 4 presents the two algorithms of G-Skyline over a data stream, and gives relevant examples. In Sect. 5, we show the experimental results and evaluations of our algorithms. At last, we conclude our research work in this paper and propose the future work.

2 Related work

Since Borzsonyi et al proposed the skyline operator [2] in 2001, the skyline has been researched in many filed, and there are many algorithms of skyline have been proposed for different problems. Next, we describe the related work of skyline query.

BNL algorithm and D&C algorithm were proposed in paper [2]. BNL computed the skyline by scanning the whole data set and maintaining a candidate set, D&C returned the final skyline set by computing each sub-set skyline. In the algorithm SFS [3], the skyline was returned after all the data being sorted according to the monotone function. Bitmap [4] firstly mapped each tuple to a m-bit vector, then got the skyline by computing the vectors, but this algorithm was only suitable for the static data set. NN [5] algorithm returned the skyline result by filtering the nearest neighbor points. BBS managed the data set by R-tree, and only the nodes containing result points would be visited.

In addition, there are some new skyline algorithms for the specific environment. The sub-space skyline [6–9] can compute the skyline by dividing the data set to some sub-space. K -dominant skyline [10, 11] can return the points

which are not k -dominated by any other points, it can find much more potential information. Top- k skyline [12–15, 30] will find the points ranking in the top k position, this algorithm was only suitable for the query within a set limit in volume. In recent years, the skyline query on uncertain data has been studied, in this query, a threshold q was given at first, then the points whose probabilities are larger than q would be returned as probabilistic skyline [16–18]. [31, 32] introduced the MapReduce technology to compute the skyline efficiently. [19] firstly proposed the skyline query in a data stream, the data in the sliding window were managed in R-tree, and the skyline set was maintained by the interval tree. [20] presented algorithm LOOK-OUT to compute skyline in a data stream. In paper [21], the data in the sliding window were managed based on a multi-layer grid structure. [22] proposed a parallel algorithm for window-based skyline targeting multicores.

Paper [23] returned the top- k composition skyline, however, this paper did not propose the composition skyline formally. [24–26] defined and researched the group skyline query, they calculated the value of the same attribute of k points to form a group, then compared the dominance relation between the groups using the traditional dominance. The calculate functions commonly used in these work were some aggregate functions, such as, SUM, MAX, and MIN. [27] defined the group dominance concept based on uncertain data. In fact, which aggregate function should be used in practical application is difficult to select, so paper [1] proposed the Pareto optimal groups and group-base skyline which can return all the Pareto optimal solutions. [28, 41] proposed efficient skyline group algorithms based on the algorithms in paper [24, 25] in a data stream, but the skyline group in their algorithms focus on some functions such as SUM, MAX and MIN, and the result of skyline groups under these functions is a subset of our G-skyline groups. The reason is that if group G is dominated by G' according to G-skyline definition, this relation is right under SUM function, but the reverse is not true. So these algorithms are not suitable for our G-skyline groups. Paper [42] proposed the skyline algorithms over data stream, but they focus on individual subjects rather than groups. [33–40] discussed different skyline applications in wireless sensor networks, such as continuous reverse skyline, spatial skyline, distributed dynamic skyline and probabilistic skyline query in the wireless sensor networks.

3 Preparations

In this section, we will present the foundation work, including the relevant definitions, basic theorems and the G-Skyline algorithm in static data that will be used in our paper, then we propose our problem.

3.1 Definitions and theorems

Definition 1 (Skyline) There are two different points p and q coming from the same data set D , we can say q dominates p , denoted by $q \prec p$, if $q[j] \leq p[j]$ for every j ($1 \leq j \leq d$) and at least one j , $q[j] < p[j]$ ($1 \leq j \leq d$), where $p[j]$ is the j th attribute value. The skyline consist of all the points which are not dominated by any other point in D .

Definition 2 (G-Dominante) Given a data set D , there are two different groups $G_1 = \{p_1, p_2, \dots, p_k\}$ and $G_2 = \{p'_1, p'_2, \dots, p'_k\}$, where each point coming from D , we can say G_1 g-dominates G_2 , if two arrays with k points for G_1 and G_2 are found, $G_1 = \{p_{n1}, p_{n2}, \dots, p_{nk}\}$ and $G_2 = \{p'_{m1}, p'_{m2}, \dots, p'_{mk}\}$, and $p_{ni} \leq p_{mi}$ for each i ($1 \leq i \leq k$) and at least one i , p_{ni} dominates p_{mi} .

Definition 3 (G-Skyline) The G-Skyline consist of all the groups with k points which are not g-dominated by any other group with the same size.

Example 1 The G-Skyline is different from skyline, and it is also not same as skyline groups [24–26]. We take the data in Fig. 1 as an example. Let $G_1 = \{p_6, p_8, p_{11}\}$ and $G_2 = \{p_2, p_3, p_{10}\}$, we can say G_1 g-dominates G_2 because there are two arrays $G_1 = \{p_6, p_{11}, p_8\}$ and $G_2 = \{p_3, p_{10}, p_2\}$ such that $p_6 \prec p_3$, $p_{11} \prec p_{10}$, and $p_8 \prec p_2$. So G_2 is not the G-Skyline group, but G_1 belongs to G-Skyline because there is no other groups g-dominates G_1 with the same size.

Definition 4 (Skyline Layers) The data set D can be divided to some layers, the $layer_i$ is composed by skyline points of $(D - \bigcup_{j=1}^{i-1} layer_j)$, such as $layer_i = \text{skyline}(D - \bigcup_{j=1}^{i-1} layer_j)$ which is computed recursively until all the points of D are in layers, where $layer_1$ is the traditional skyline of D .

Theorem 1 Given a data set D and the group size k , each point of G-Skyline groups must be in the first k skyline layers.

Theorem 2 If there is a non G-Skyline group G with k points, when another point from G 's tail set is added to it, this new group with $k + 1$ point also does not belong to the G-Skyline.

Theorem 3 For each point p of a G-Skyline group G , all of p 's parents must be in G too.

3.2 Algorithm in static data

To compute the G-Skyline groups with k points from the given n points, the crude method is to enumerate all the $\binom{n}{k}$ groups, and then do the query based on the g-dominance. Obviously doing it like this needs much time cost and storage cost, so paper [1] proposes the Pwise (Point-

Wise) algorithm to efficiently compute the G-Skyline groups. Next, we will introduce this algorithm in brief.

1. **Skyline Layers.** Firstly, all the points in D with 2-dimension are sorted with increasing x-coordinate value, then all the points in this order are processed by binary search to compute which layer each point belongs to. Because the Pwise algorithm only compute the G-Skyline for the given group size k , so just the first k skyline layers need to be constructed. The point with minimum y-coordinate in $layer_i$ is referred as the *tail point* of $layer_i$. An example of skyline layers is shown in Fig. 3, and p_{11} is the tail point of $layer_1$.
2. **Construct Directed Skyline Graph (DSG).** The DSG is a data structure which reflects the dominance relations between the first k layers. It is constructed based on skyline layers: all the points in D are calculated according to the increasing layers. For every point p , it should be compared with all the points in the previous layers and get their dominance relations, for the points which dominate p , p will be added to their children list, and these points will be added as p 's parents list. An example of DSG is shown in Fig. 4 based on the data in Fig. 1. In order to look clarity, all the indirect dominant relations are omitted, such as $p_{11} \prec p_2$.
3. **Compute G-Skyline.** Based on the skyline layers and DSG, the algorithm performs by the classic set enumeration tree search framework. According to Theorem 2, the algorithm firstly prunes the non-G-Skyline groups as soon as possible, because if a group is not the G-Skyline group, it should not be expanded further, then according to Theorem 3, the algorithm prunes the point from the tail set of each node. Finally, the G-Skyline is returned.

Fig. 3 The skyline layers

Fig. 4 Directed skyline graph

3.3 Our problem

The points in the static data set are stable, but the points in the data stream are dynamic, each point has its life cycle, and the point is only valid in its life cycle. Thus, the active data set of the data stream is not static, and it will change when a new point arrives or an old point expires. Aiming at this problem, we propose an algorithm to find G-Skyline in the data stream. To the best of our knowledge, this problem is the first time to be considered here, and there has been no algorithm can solve it.

In this paper, we use sliding window to manage the data in the stream. There are two type sliding windows [29]: the one based on time, and another based on count. We focus on the time-based sliding window.

Definition 5 (Sliding Window) For a time window W , and t is a random moment, when the point p arrives at t , its life cycle can be written as $[t, t + W]$, and the point is only valid in this period, that is to say the point p is added to the active data set at t moment and is deleted from the active data set at $t + W$ monent.

Theorem 4 Given a group G with k points coming from the dataset D , for each point $p \in G$, if all of its parents are in G , or it is the traditional skyline point of D , we can say G is G-Skyline group.

Prove Assume a group $G = \{p_1, p_2, \dots, p_k\}$, each point in G and its parents are in G . If there is another group $G' = \{q_1, q_2, \dots, q_k\}$ can g-dominate G , we can find two permutations that for $i \in [1, k]$, $q_i \prec p_i$, so q_i is p_i 's parent. From the known condition, we conclude that each q_i is in G , G and G' have the same items. So there is not such a

group which g-dominates G , according to the concept of G-Skyline, we can say G is a G-Skyline group. $\square$

4 G-skyline query over a data stream

In this section, we elaborate the algorithm to compute G-Skyline in the data stream. For convenience, we maintain that: (1) the layer of point p , denoted by $p.l$, indicating which skyline layer the point p belongs to; (2) each point has the constant life cycle, denoted as $[p.t_{arr}, p.t_{exp}]$, while $p.t_{arr}$ means when the point p arrives and $p.t_{exp}$ indicates when the point p expires, $p.t_{exp} = p.t_{arr} + W$.

In order to compute G-Skyline effectively, we present the sharing strategy, and based on which we propose two algorithms to find G-Skyline groups in the data stream.

4.1 Sharing strategy

When the point arrives or expires in a data stream, we can update the G-Skyline based on the existing G-Skyline.

Proof The dynamic of the data stream reflects in two aspects: new point arriving and old point expiring. Here we take point p as an example. Both cases will result in the active data set changing, and the G-Skyline of active data set will also change. But not all of the dominance relationships between the points are affected in these two cases, and only such relations are affected: the dominance relationships between p and its parents, and the dominance relationships between p and its children. According to Theorems 1 and 2, we find that when a new point p arrives, the non-G-Skyline groups are still non-G-Skyline groups, and the G-Skyline groups not containing p 's children are still G-Skyline groups, we should only check the other existing G-Skyline groups and the new groups containing p . When an old point p expires, the existing G-Skyline groups not containing p are still G-Skyline groups, and the existing G-Skyline groups containing p should be deleted, so we should check the status of some non-G-Skyline groups.

That is to say, when the active data set changes, we do not have to computing all the active data, we can compute the new G-Skyline based on the existing G-Skyline. $\square$

In the G-Skyline processing over the data stream, this sharing strategy will prune most of groups which will not affect the new G-Skyline, and help us to compute the G-Skyline quickly in the data stream.

4.2 Computing G-skyline for point arriving

When a new point p arrives, we should firstly check which layer the point p belongs to, then we update the DSG to construct the new relationships between all the points,

finally, we compute the G-Skyline based on the sharing strategy. In order to compute the G-Skyline continuously in the data stream, we should compute the skyline layers and the DSG for all the active points rather than the first k skyline layers and the DSG in the PWISE [1].

Update the skyline layers For the existing active points, their skyline layers have been constructed, and the points in each layer have been sorted increasingly by x coordinate. When a new point p arrives, by computing p and the tail point of each layer, we can execute the bin-search to find which layer this new point belongs to, then if $layer_i.tail$ does not dominate p and $layer_{i-1}.tail$ dominates p , we can say p belongs to $layer_i$. If p is dominated by the tail point of the last layer, it will belong to a new layer. Then we can compare p with all the points in this layer to determine which position the point p locates.

Algorithm 1: update skyline layers when new point arrives

Input: the skyline layers of n active points, a new arriving point p

Output: the new skyline layers of active points

```

1  for  $i=1$  to  $maxlayer$  do
2    if the  $layer_1$ 's tail point does not dominate  $p$  then
3       $p.layer=1$ ;
4    else if  $layer_{max}$ 's tail point dominates  $p$  then
5       $p.layer=maxlayer+1$ ;
6    else
7      calculate bin-search to find which layer  $p$ 
      belongs to
8    compare  $p$  with the points in  $p.layer$  to determine  $p$ 's
      position in the layer
9    if there is a point in  $p.layer$  is dominated by  $p$ 
10     Denote all the points in  $p.layer$  dominated by  $p$ 
      as  $D_I$ 
11     While(  $D_I \neq null$  )
12       {   For each point  $q$  in  $D_I$ 
13         Copy all the points in
14          $layer_{(q.layer+1)}$  dominated by  $q$  to  $D_2$ 
15          $q.layer=q.layer+1$ ;
16         delete  $q$  from  $D_I$ ;
17          $D_I=D_2$ ;
18       }
19  return the new skyline layers

```

Example We show an example of Algorithm 1 in Fig. 5 based on Fig. 1. Assume the active points in the data stream are these 11 points, at this moment, a new point p arrives, so the active data set will change. By updating the skyline layers, we firstly execute the bin-search to find that p locates between $layer_1$ and $layer_2$, then we construct the new skyline layers as shown in Fig. 4. From the new skyline layers, we find that the layers of some points have changed. For example, p_8 is in $layer_2$ previously, when p arrives, the point in $layer_2$ dominated by p is only p_8 , so p_8 moves to $layer_3$, at the same time, as the children of p_8 , p_2 and p_5 also move to higher $layer_4$, similarly, as the child of p_5 , p_4 moves to $layer_5$.

Update the directed skyline graph (DSG) When the new point p arrives, it changes the skyline layers, because the DSG is built based on the skyline layers, so we should also update the DSG of the active points.

According to the DSG concept, we know that when a new point p arrives, it does not affect other points except for its parents and children. So the DSG updating can be finished in two steps. Firstly, to find p 's parents, we can compare p with each point whose layer is smaller than p 's layer, if a point q is p 's parent, we should not compare p with q 's parents. Secondly, to find all of p 's children, we can compare p with each point whose layer is larger than p 's layer, if a point q is p 's child, we should not compare p with q 's children.

Example We show an example of DSG updating in Fig. 6 based on Fig. 1. When a new point p arrives, we firstly update the skyline layers, then we can update the DSG to reflect the dominance relationships in real time. Because p lies between $layer_1$ and $layer_2$ in the old skyline layers, we can find p is dominated by p_{11} in the $layer_1$, p_8 in the $layer_2$ is dominated by p , and p_8 is the child of p_{11} previously, so we set p_{11} as p 's parent, and p_8 as p 's child, then the children (p_2, p_5, p_4, p_7) of p_8 will not be compared with p , and we also find p_9 is also p 's child. The new DSG is shown in Fig. 6. We can see that when a new point

Fig. 5 New skyline layers

Fig. 6 Updating DSG when a new point arrives

p arrives, there are some dominance relationships about p added to DSG, but other dominance relationships irrelevant to p will remain unchanged.

Compute G-skyline for a point arriving After updating the skyline layers and DSG, we can compute the new G-Skyline based on the existing G-Skyline. According to theorem 1, we infer that which layer p belongs to will generate different effect on the new G-Skyline. So we present different solutions on the basis of p 's location as follows.

1. If $p_{\text{layer}} > k$. According to Theorem 1, the point of G-Skyline groups must in the first k layers, so if $p_{\text{layer}} > k$, p will not affect the G-Skyline result.
2. If $p_{\text{layer}} < k$. In this case, p has no effect on the non-G-Skyline groups and the G-Skyline groups which do not contain all of p 's parents. However, the point p may only affect the G-Skyline groups which containing all of p 's parents, here we denote these groups as candidate groups.

Prove For the non-G-Skyline groups G_1 , there must be a group G_2 dominating it, when p arrives, G_2 still dominates G_1 , so we can easily find p has no effect on such kind of groups.

For each G-Skyline group (such as G_3) not containing all of p 's parents, according to Theorem 3, there must be no child of p existing in G_3 , so there is no point dominated by p , and p has no effect on the G-Skyline groups not containing all of p 's parents.

For the G-Skyline groups not containing any parent of p , we can easily prove p has no effect on such kind of groups. Finally only the G-Skyline groups containing all of p 's parents should be re-evaluated. $\square$

We call this kind of groups the candidate groups, and we divide the candidate groups into two kinds, and give the different solutions for them. If there is not any G-Skyline group containing all of p 's parents, p will not affect the query result.

Solution 1 For each G-Skyline group G which contains all parents of p except for p 's children, p will not affect its status. We can replace the leaf point of G by p to compose the new group which is G-Skyline group, while this leaf point can not be p 's parent. However, G is still G-Skyline group.

Prove Assume G is a G-Skyline group not containing p 's children, that is to say, there is not any point in G dominated by p , and there will be no group containing p can dominate G , so p does not affect G 's status, and G is still G-Skyline group. On the other hand, if we replace the leaf point of G by p to compose the new group G' , we can not find another group which can dominate G' because all of p 's parents are already in G , so G' is G-Skyline group too. $\square$

Solution 2 For each G-Skyline group containing p 's children, p will affect the its status. We replace the leaf point of G by p to compose the new group which will be G-Skyline group. But, G is not G-Skyline group yet.

Prove If the G-Skyline group contains p 's children, such as $G = \{g_1, g_2, \dots, g_i, \dots, g_k\}$ and g_i is p 's child, we can find $G' = \{g_1, g_2, \dots, p, \dots, g_k\}$ can g-Dominates G because $p \prec g_i$, so the group G will not be G-Skyline. At the same time, if we replace the leaf point of G by p to compose a new group G' , then each of the point in G' and all of its parents are in the G , so according to concept of G-Skyline and Theorem 4, we can see that there is no group which can g-Dominate G' , so G' is G-Skyline group. $\square$

According to the above solutions, when a new point arrives we can quickly find the new G-Skyline based on the existing G-Skyline groups. The process is shown in Algorithm 2 as follows.

Algorithm 2: Point-arriving-Algorithm

Input: a data set D with n active points, the corresponding skyline layers and DSG, group size k , a new arriving point p

Output: New G-Skyline groups

- 1 update the skyline layers
- 2 update the DSG
- 3 for each group G in $G\text{-Skyline}$
- 4 if G contains all of p 's parents
- 5 if G does not contain p 's child
- 6 $G' \leftarrow$ replace the leaf point of G by p
- 7 insert G' to $G\text{-Skyline}$
- 8 if G contain p 's child
- 9 $G' \leftarrow$ replace the leaf point of G by p
- 10 insert G' to $G\text{-Skyline}$
- 11 remove G from $G\text{-Skyline}$
- 12 return New $G\text{-Skyline}$

Example An example of Algorithm 2 is shown in Fig. 7 based on Fig. 1. Assume at present the active points in the data stream are these 11 points which compose the data set D . The k -item G-Skyline groups are shown in Fig. 7 where $1 \leq k \leq 4$. When a new point $p(18, 30)$ arrives, after updating the skyline layers and DSG, we can begin to update the relevant groups. The parent of p is only p_{11} , so

each G-Skyline group containing p_{11} should be expanded. For visualization clarity, we omit the reduplicate new G-Skyline groups coming from the existing G-Skyline groups. The groups in the dotted box are new groups born from the existing G-Skyline groups. At level $ISl_p = 1$, we can easily find that p 's arriving has no effect on the 1-item G-Skyline. At level $ISl_p = 2$, among the existing 2-item G-Skyline groups, we find the group $\{p_{11}, p_8\}$ contains the parent and the child of p , according to solution 2, we should replace p_8 by p to form the new G-Skyline group $\{p_{11}, p\}$, and this new group is G-Skyline group, but the group $\{p_{11}, p_8\}$ will be no longer the G-Skyline group because it is g-dominated by the new group $\{p_{11}, p\}$. Similarly, for the group $\{p_6, p_{11}, p_8\}$, it also contains the parent and the child of p , so we get the new G-Skyline group $\{p_6, p_{11}, p\}$ instead of $\{p_6, p_{11}, p_8\}$. As a result, level $ISl_p = 4$ shows all the 4-item G-Skyline groups without checking.

4.3 Updating G-skyline for point expiring

Each point in the data stream has its life cycle, when an active point expires, the active data set will change too, so we should compute the new G-Skyline groups of the new active dataset based on the existing result. In this section, we firstly update the skyline layers, then reconstruct the DSG, finally compute the G-Skyline for point expiring.

Update the skyline layers Different from the point arriving, when a point expires, we can easily update the skyline layers. Assume the point p in $layer_L$ expires, if p is the tail point of $layer_L$, each tail point of $layer_{L'}$ ($L' > L$) will be moved to the tail of $layer_{L-1}$, at the same time, any other points still locate in their previous layer. If p is not the tail point of $layer_L$, we will not only delete p , but also change the layers of some points. If the point q is in $layer_{L+1}$ and its parent in $layer_L$ is only p , then we can change q 's layer to $layer_{L-1}$. Then we will similarly change the layers of some points one layer by one layer. The procedure of updating skyline layers is shown in Algorithm 3.

Fig. 7 Finding G-Skyline when a new point arrives

Algorithm 3: update skyline layers for point expiring

Input: the skyline layers of n active points, a point p to expire

Output: the skyline layers of active points

```

1  if  $p$  is the tail point of  $layer_L$ 
2    for  $i=L$  to  $maxlayer-1$  do
3       $(layer_i).tail = (layer_{i+1}).tail$ ;
4  else
5    Initialize a set  $S = \text{null}$ ;
6     $L' = L$ ;
7    copy  $p$  to  $S$ ;
8    while( $S \neq \text{null}$ )
9      { for each point  $p'$  in  $S$ 
10       { for each child  $q$  in  $layer_{L'+1}$  of  $p'$ 
11        { if  $S$  contains all parents of  $q$  in  $layer_{L'}$ 
12          $q.layer = q.layer - 1$ ;
13         copy  $q$  to  $S$ ;
14       }
15        $L'++$ ;
16       delete  $p'$  from  $S$ ;
17     }
18   }
19   delete  $p$ 
20   return the new skyline layers

```

Example An example of Algorithm 3 is shown in Fig. 8 based on Fig. 1. For the 11 active points, if the tail point expires, we should firstly delete this tail point, then change the tail point of $layer_{i+1}$ to become the tail point of $layer_i$, other points keep their layers unchanged. As shown in Fig. 8(a), when the tail point p_{10} expires, p_9 becomes the tail of $layer_2$, and p_7 becomes the tail of $layer_3$, where in the previous layers p_9 was the tail of $layer_3$ and p_7 was the tail of $layer_4$. However, when point p_8 expires, we find p_8 is p_5 's single parent in $layer_2$, so we move p_5 to lower $layer_2$, but p_4 still lie in its original layer because the parents of p_4 in $layer_3$ are p_5 and p_9 while only p_5 is in S , the new skyline layers is shown in Fig. 8(b).

Update the directed skyline graph (DSG) When an old point expires, it may affect the existing skyline layers, and it maybe also change the DSG.

The DSG reflect the dominance relationships of each point, so when point p expires and is removed, there will be no relationships between it and its parents, and between it and its children. In the DSG, the directed edge indicates the

Fig. 8 Updating skyline layers. **a** Point p_{10} expires. **b** Point p_8 expires

Fig. 9 The change of edges when a point expires

dominance relationship, so when p expires, the edges between p and its parents and the edges between p and its children should be deleted, at the same time, the new directed edges from p 's parents to p 's children will be added to DSG to update it, the procedure is shown in Fig. 9. In fact, the dominant relationships indicated by new directed edges have been existed already, just because we omit them for visualization clarity. The most importance is p 's expiration will not change the dominance relationships between other points.

Example We give an example to show the DSG updating when an old point expires in Fig. 10 based on Fig. 1. Assume that these 11 points in Fig. 1 are active points in data stream, when the expiration time of p_3 arrives, the point p_3 will be deleted. Then the dominance relationships about p_3 will be removed from DSG, at the same time, the new directed edges between its parents (such as p_6) and its children (such as p_2) will be added to DSG.

Compute G-skyline for a point expiring Similar to the G-Skyline computing for point arriving, after updating the skyline layers and the DSG, we can compute the new G-Skyline groups based on the existing G-Skyline. According to the G-dominate concept and Theorem 3, we find that the expiration of point p only has an effect on the G-Skyline groups which contain p or p 's children, it will not affect any other groups. Next, we analyze whether these groups are new G-Skyline groups or not.

1. For the G-Skyline groups

If G is a G-Skyline group and does not contain p , according to G-Skyline concept, there is not a groups G' can G-dominate G , we can easily know that p 's expiration does not affect such groups. $\square$

If G is a G-Skyline groups and contains p , we can get the new k -item G-Skyline group by deleting p from the old $k + 1$ -point G-Skyline groups.

Prove From the algorithm for the static data set, we know that the G-Skyline groups with $k + 1$ points come from G-Skyline with k points. Assume that $G = \{q_1, q_2, \dots, q_k, p\}$ is G-Skyline groups with $k + 1$ points, and it maybe contains p 's children, we know that each point in G and all of its parents are in G . When p expires and is deleted from G , $G' = G - p = \{q_1, q_2, \dots, q_k\}$, and each point in G' and all of its

parents are still in G' , according to Theorem 4, we can say G' is G-Skyline groups with k points when p expires. $\square$

2. For the non-G-Skyline groups

If G is a non-G-Skyline group and contains p , it can be deleted safely.

Prove We can easily prove this conclusion because G will not exist when p expires.

If G is a non-G-Skyline group and does not contain p , it maybe become G-Skyline group when p expires.

Prove: When point p expires, it will not dominate its children any more, so some of non-G-Skyline groups which does not contain p but contains p 's children maybe become G-Skyline. According to Theorem 4, these groups must satisfy the condition: for such group G , each of point in G and all of its parents except p must be in G . Because p 's children's parents contain p 's parents, so if we add p to G to form G' with $k + 1$ points, G' must be $k + 1$ -point G-Skyline group of the active data set before p expires. Then we can get these candidate groups by deleting p from the $k + 1$ -item G-Skyline groups, and these groups have been returned in (1). However, any other non-G-Skyline groups which do not meet this condition will not belong to G-Skyline, and they can be pruned safely. $\square$

When a point p expires, we can quickly compute the new G-Skyline groups based on the existing G-Skyline groups. Based on the above strategy, our key idea of the algorithm is shown in Algorithm 4.

Algorithm 4: Point-expiring-algorithm

Input: a data set D with n active points, the corresponding skyline layers and DSG, group size k , a point p to expire.

Output: New G-Skyline groups NGS

- 1 update the skyline layers
- 2 update the DSG
- 3 for each group G_k in G-Skyline
- 4 if G does not contain p
- 5 Copy G_k to NGS
- 6 for each group G_{k+1} containing p in G-Skyline
- 7 $G' \leftarrow$ remove p from G_{k+1}
- 8 insert G' to NGS
- 9 return NGS

Example Now we show an example of Algorithm 4 in Fig. 11 based on the data in Fig. 1. Assume the active points of data stream at present time are these 11 points. When an old point p_6 expires, p_6 will be useless, and the

Fig. 10 Updating DSG

Fig. 11 Finding G-Skyline when point p_6 expires

active points data set will change, so we should compute the new G-Skyline. At level $|S_p| = 1$, we firstly delete p_6 , then get the new 1-point G-Skyline group p_3 by removing p_6 from $\{p_6, p_3\}$ which is 2-point old G-Skyline group, and we set a dashed box to denote the new G-Skyline group. At the same level, we can also remove p_6 from $\{p_1, p_6\}$ and $\{p_6, p_{11}\}$ to get the new skyline point p_1 and p_{11} , but these two points already belong to skyline, for visualization clarity, we do not show this kind of indications. Similarly, at the level $|S_p| = 2$, we can get the new 2-point G-Skyline groups $\{p_1, p_3\}$ and $\{p_{11}, p_3\}$ by removing p_6 from the old 3-point G-Skyline groups $\{p_1, p_6, p_3\}$ and $\{p_6, p_{11}, p_3\}$. As a result, level $|S_p| = 3$ shows all the new G-Skyline groups with 3 points. Based on our pruning strategies, we can easily get the new G-Skyline groups at each level without much computation.

5 Experiments

In this section, we present experimental evaluation about our algorithms.

5.1 Experiment preparation

We simulate a data stream and evaluate the algorithms when the new data arrives or the old data expires. For each condition, we firstly evaluate the skyline layers updating, and then perform the comprehensive experiments to test the G-Skyline algorithm based on the synthetic data. To examine the extendibility of our algorithms, we generate three critical types of data: the correlated data (COR), the independent data (IND) and the anti-correlated data (ANTI-COR). The example of each type of data with 2-dimension is shown in Fig. 12.

Fig. 12 The synthetic dataset. **a** COR, **b** ANTI-COR, **c** IND

For the correlated dataset and the anti-correlated dataset, the points are generated by selecting a plane perpendicular to the line from $(0, \dots, 0)$ to $(1, \dots, 1)$ using a normal distribution, while for the independent dataset, all attribute values of points are generated independently using a uniform distribution. For each type of data, we simulate the data stream in such way, a new point is generated randomly at regular intervals to simulate a new point arriving in the data stream, similarly, a point will be deleted at regular intervals to simulate an old point expiring in the data stream, and the point which generated earlier will be deleted earlier.

We also use the real stock data to evaluate the efficiency of our algorithms.

Because this is the first time to compute G-Skyline over the data stream, our examine evaluation was conducted against the existing algorithm for static dataset. All the experiments are performed on a PC with 1.7 GHz Intel Core i7 processor running Windows 7 operation system with 8 GB memory and 1TB hard drive. The algorithms to be examined in the experiments are as follows.

PAA Computing G-Skyline groups for a new point arriving.

PEA Computing G-Skyline groups for an old point expiring.

PWise Point-Wise algorithm of G-Skyline for static dataset in paper [1].

5.2 Updating skyline layers

Firstly we examine our algorithms for updating skyline layers when the new point arrives or an old point expires. The PWise algorithm is to rebuild all the skyline layers by binary searching for the new active dataset, while our algorithm can update the skyline layers directly based on the existing skyline layers.

Figure 13 shows the running time cost of updating skyline layers in the PWise algorithm and our algorithms on the different datasets. When the group size k varies from 2 to 6, we find that the PWise algorithm is affected by the different datasets and the growth of running time is flat from correlated dataset to independent dataset, and to anti-

Fig. 13 Updating skyline layers. **a** COR, **b** IND, **c** ANTI-COR

correlated dataset. The reason is that the PWISE algorithm only considers the points in the first k skyline layers while other points will not be considered. Different from PWISE, our algorithms perform better. The reason is that when a new point p arrives in the data stream, based on the existing skyline layers, we should only use binary search to find where p will locate. In order to compute the G-Skyline continuously, the skyline layers in our algorithm must contain all of the active points, so no matter what value the

group size k is, for the same dataset, the running time of updating skyline layer is same, and it is much less than PWISE. However, when an old point p expires, to update the skyline layers, our PEA can directly delete the p and change the layers of some points dominated by p . This work is more easy and the running time is the most least.

According to the distribution of each dataset, we find that the average layer number follows $COR.ln > IND.ln > ANTI-COR.ln$. Then the running time of our algorithm shows little growth. Finally, our algorithms perform better than PWISE.

5.3 Performance with respect to the synthetic data

In this section, we show the experimental evaluation of algorithms on the synthetic dataset. Each dataset is generated following the seminal work in paper [2].

Figure 14 shows the running time of algorithms on each synthetic dataset with different dataset size n , while $d = 2$, $k = 4$. When n is more than 10^3 , adding a new point to the active dataset or deleting an old point from the active dataset has no effect on the total number of points to be computed in PWISE, so the running time of PWISE for this two cases approximately equal, and we can use the same time value in the figure. The varying n has a certain effect on the PWISE algorithm because it should compute the points in the first k layer, and the total number increases with n increasing, then the running time shows little growth on the COR dataset and IND dataset, while PWISE need much time in ANTI-COR dataset because every layer has more points than other two dataset. However, our algorithm perform better. When a new point arrives in the data stream, based on the existing G-Skyline groups, PAA only need to check the groups expanded from the existing G-Skyline groups which contain all of p 's parents, the number of these candidates will not be large, so the running time of PAA is less than PWISE. Similarly, when an old point p expires, PEA only need to check the existing G-Skyline groups which contain p , this is very easy and the running time is very little.

Figure 15 shows the running time of algorithms on each synthetic dataset with different dimension size d , while $n = 1000$, $k = 3$. The varying d has much effect on the PWISE algorithm because the total number of the points in the first k layers increases sharply with d increasing. However, the running time of our algorithms is less and increase smoothly. The reason is that our algorithms can get the new G-Skyline based on the existing G-Skyline, although the number of points in the first k layers increases sharply, the number of candidates to be checked in our algorithm keeps little growth.

Fig. 14 Finding G-Skyline with different dataset size n . **a** COR, **b** IND, **c** ANTI-COR

Figure 16 show the running time of algorithms on synthetic dataset with different group size k , while $n = 1000$, $d = 2$. The running time of PWise increases sharply with k increasing, the reason is that the number of points in the first k layers increases quickly. PAA needs a little more time than PEA, this is due to their different solution approach, PAA needs to check more candidates than PEA.

Fig. 15 Finding G-Skyline with different dimension size d . **a** COR, **b** IND, **c** ANTI-COR

5.4 Performance with respect to the real stock data

In order to evaluate the algorithms' efficiency on a real data set, we do the experiments on the real stock data from www.finance.yahoo.com. The real data contain 3×10^5 records of stock, and each record has 3 attributes: change, volume and price.

Fig. 16 Finding G-Skyline with different group size k . **a** COR, **b** IND, **c** ANTI-COR

Figure 17 shows performance of algorithms on real data with different dataset size n . We find that the dataset size has little impact on the algorithms, and our algorithms are better, the reason is that the dataset size is not very large. Figure 18 shows performance of algorithms on real data with different group size k . The group size has much impact on the algorithms, however, our algorithms are better and efficient. As a result, our algorithms do better in

Fig. 17 Finding G-Skyline with different dataset size n in real data

Fig. 18 Finding G-Skyline with different group size k in real data

G-skyline query over real data stream, the reason is that our algorithms can compute the new G-skyline based on existing result, and there will be fewer points to be used to form the candidate groups when a new point arriving or an old point expiring.

6 Conclusions and future work

Processing dynamic data or data stream from the wireless sensor network will provide important information for users. In this paper, we proposed the problem of finding G-Skyline groups over the data stream in the wireless sensor network. In order to compute the G-Skyline groups efficiently, we firstly presented the sharing strategy, and then based on which, we proposed two algorithms PAA and PEA to compute the new G-Skyline groups when a new point arrive or an old point expires. The experiment results based on the synthetic data and real data show our algorithms' benefit. In the future, we will consider how to compute the G-Skyline groups in wireless network if different sensors sample data at different time.

Acknowledgements The authors will thank all the members in database laboratory of Donghua University. This work is supported by Natural Science Foundation of China (No. 61672151), Youth fund of Daqing Normal University (No. 15ZR07).

Compliance with ethical standards

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

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

- Liu, J., Xiong, L., & Pei, J. (2015). Finding pareto optimal groups: Group-based skyline. In *VLDB* (pp. 2086–2097).
- Borzsonyi, S., Kossmann, D., & Stocker, K. (2001). The skyline operator. In *ICDE* (pp. 421–430).
- Chomicki, J., Godfrey, P., & Gryz, J., et al. (2003). Skyline with presorting. In *ICDE* (pp. 717–719).
- Tan, K., Eng, P., & Ooi, B. (2001). Efficient progressive skyline computation. In *VLDB* (pp. 301–310).
- Kossmann, D., Ramsak, F., & Rost, S. (2002). Shooting stars in the sky an online algorithm for skyline queries. In *VLDB* (pp. 275–286).
- Pei, J., Jin, W., Ester, M., & Tao, Y. (2005). Catching the best views of skyline: Asemantic approach based on decisive subspaces. In *VLDB* (pp. 253–264).
- Lee, J., & Hwang, S. W. (2014). Toward efficient multidimensional subspace skyline computation. *VLDB Journal*, 23(1), 129–145.
- Xia, T., & Zhang, D. (2006). Refreshing the sky: the compressed skycube with efficient support for frequent updates. In *SIGMOD* (pp. 491–502).
- Li, Y., Li, Z. Y., & Dong, M. X. (2015). Efficient subspace skyline query based on user preference using MapReduce. *Ad Hoc Networks*, 35, 105–115.
- Chan, C. Y., Jagadish, H. V., & Tan, K. L. et al. (2006). Finding k-dominant skylines in high dimensional space. In *SIGMOD* (pp. 503–514).
- Miao, X. Y., Gao, Y., et al. (2016). k-dominant skyline queries on incomplete data. *Information Sciences*, 367, 990–1011.
- Lee, J., You, G., & Hwang, S. (2008). Personalized top-k skyline queries in high-dimensional space. *Information Systems*, 1, 45.
- Jiang, T., Zhang, B., & Lin, D. (2015). Incremental evaluation of top-k combinatorial metric skyline query. *Knowledge-Based Systems*, 74, 89–105.
- Zhang, W., Lin, X., Zhang, Y., et al. (2010). Threshold based probabilistic top k dominating query[J]. *The VLDB Journal*, 19(2), 283–305.
- Jiang, T., Zhang, B., Gao, Y., et al. (2013). Efficient top k query processing on mutual skyline. *Journal of Computer Research and Development*, 50(5), 986–997. (In Chinese).
- Le, T. M. N., Cao, J., & He, Z. (2016). Answering skyline queries on probabilistic data using the dominance of probabilistic tuples. *Information Sciences*, 340–341, 58–85.
- Pei, J., Jiang, B., Lin, X., & Yuan, Y. (2007). Probabilistic skylines on uncertain data. In *VLDB* (pp. 15–26).
- Pujari, A. K., Kagita, V. R., & Garg, A. (2015). Efficient computation for probabilistic skyline over uncertain preferences. *Information Sciences*, 324, 146–162.
- Lin, X., Yuan, Y., Wang, W., & Lu, H. (2005). Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE* (pp. 502–513).
- Morse, M., Patel, J.-M., & Grosky, W.-I. (2007). Efficient continuous skyline computation. *Information Sciences*, 177(17), 3411–3437.
- Li, H., Yoo, J. (2014). An efficient scheme for continuous skyline query processing over dynamic data set. In *Proceedings of the international conference on big data and smart computing* (pp. 54–59). Bangkok, Thailand.
- Tiziano, D. M., Salvatore, D. G., & Gabriele, M. (2015). A multicore parallelization of continuous skyline queries on data streams. *LNCS*, 9233, 402–413.
- Su, I.-F., Chung, Y.-C., & Lee, C. (2010). Top-k combinatorial skyline queries. In *Proceedings of the 15th international conference on database systems for advanced applications (DASFAA 2010)*.
- Im, H., & Park, S. (2011). Group skyline computation. *Information Sciences*, 188, 151–169.
- Li, C., Rajasekaran, S., Zhang, N., et al. (2014). On skyline groups. *TKDE*, 4(26), 942–956.
- Chung, Y.-C., Su, I.-F., & Lee, C. (2013). Efficient computation of combinatorial skyline queries. *Information System*, 38, 369–387.
- Magnani, M., & Assent, I. (2013). From stars to galaxies: Skyline queries on aggregate data. In *EDBT* (pp. 477–488).
- Guo, Xi, Li, Hailing, Wulamu, Aziguli, et al. (2016). Efficient processing of skyline group queries over a data stream. *Tsinghua Science and Technology*, 21(1), 29–39.
- Babcock, B., Babu, S., & Datar, M., et al. (2002). Models and issues in data stream systems. In *Proceedings of the ACM SIGACT-SIGMOD symposium on principles of database systems*. Wisconsin.
- Son, W., Stehn, F., & Knauer, C. (2017). Top-k manhattan spatial skyline queries. *Information Processing Letters*, 123(1), 27–35.
- Lee, K. H., Kim, J., & Kim, M. H. (2017). Simultaneous processing of multi-skyline queries with mapreduce. In *IEICE transactions on information and systems* (Vol. E100-D, No. 7).
- Zaman, A., & Siddique, M. A. (2017). Annisa. Finding key persons on social media by using MapReduce skyline. *International Journal of Networking and Computing*, 7(1), 86–104.
- Zhu, H., Zhu, P., & Li, X. (2017). Computing skyline groups: an experimental evaluation. In *ACM turing celebration conference-China* (p. 48).
- Zhu, H., Zhu, P., & Li, X. (2017). Parallelization of group-based skyline computation for multi-core processors. *Concurrency and Computation Practice and Experience*, 3, e4195.
- Yin, B., Zhou, S., & Zhang, S. (2017). On efficient processing of continuous reverse skyline queries in wireless sensor networks. *KSII Transactions on Internet and Information Systems*, 11(4), 1931–1953.
- Wang, Y., Song, B., & Wang, J. (2016). Geometry-based distributed spatial skyline queries in wireless sensor networks. *Sensors*, 16(4), 454.
- Ahmed, K., Nafi, N., & Gregory, M. (2016). Enhanced distributed dynamic skyline query for wireless sensor networks. *Journal of Sensor and Actuator Networks*, 5(1), 2.
- Yin, B., Zhou, S., & Zhang, S. (2017). On efficient processing of continuous reverse skyline queries in wireless sensor networks. *KSII Transactions on Internet and Information Systems*, 11(4), 1931–1953.

39. Wang, Y., Wei, W., & Deng, Q. (2016). An energy-efficient skyline query for massively multidimensional sensing data. *Sensors*, 16(1), 83.
40. Example, F., & Quality, C. (2015). A environment. Alternative tuples based probabilistic skyline query processing in wireless sensor networks. *Mathematical Problems in Engineering*, 2015, 1–10.
41. Wulamu, A., & Li, H., et al. (2016). Processing skyline groups on data streams. ubiquitous intelligence and computing and 2015. In *2015 IEEE 12th international conference on autonomic and trusted computing and 2015, IEEE 15th international conference on scalable computing and communications and its associated workshops (UIC-ATC-ScalCom)* (pp. 935–942).
42. Nagendra, Mithila. (2014). *Efficient processing of skyline queries on static data sources, data streams and incomplete datasets*. Arizona: Arizona State University.

Leigang Dong is currently a doctoral graduate student at DongHua University, China. His principal research interests include data analysis and data mining.

Xiaowei Cui received the graduate degree in Computer Application Technology from YanShan University, China, in 2009. Her research interests include data mining and information processing.

Tianyu Li is currently a post-graduate student at Shanghai Jiaotong University, China. His research interest is electronic information.

Guohua Liu is currently a professor of the Department of Computer Science and Technology at Donghua University, China. He received his Ph.D. degree from FuDan University. In recent years, he successively presided over a number of research programs of National Natural Science Foundation of China.