# Subspace Global Skyline Query Processing

Mei Bai
College of Information Science
& Engineering
Northeastern University
P.R. China
baimei1221@gmail.com

Junchang Xin
College of Information Science
& Engineering
Northeastern University
P.R. China
xinjunchang@gmail.com

Guoren Wang
College of Information Science
& Engineering
Northeastern University
P.R. China
wanggr@mail.neu.edu.cn

## ABSTRACT

Global skyline, as an important variant of skyline, has been widely applied in multiple criteria decision making, business planning and data mining, while there are no previous studies on the global skyline query in the subspace. Hence in this paper we propose subspace global skyline (SGS) query, which is concerned about global skyline in ad hoc subspace. Firstly, we propose an appropriate index structure RB-tree to rapidly find the initial scan positions of query. Secondly, by making analysis of basic properties of SGS, we propose a single SGS algorithm based on RB-tree (SSRB) to compute SGS points. Then an optimized single SGS algorithm based on RB-tree (OSSRB) is proposed, which can reduce the scan space and improve the computation efficiency in contrast to SSRB. Next, by sharing the scan space of different queries, a multiple SGS algorithm based on RB-tree (MSRB) is proposed to compute multiple SGS (MSGS). Finally, the performances of our proposed algorithms are verified through a large number of simulation experiments.

## Categories and Subject Descriptors

H.2.4 [**Databases**]: Query processing

## General Terms

Algorithms, Performance

## Keywords

global skyline, subspace global skyline, RB-tree, query optimization

## 1. INTRODUCTION

The skyline operator [3] and its variations [7, 18, 5] have been widely applied in many applications involving multiple criteria decision making, business planning and data mining. Given a data set $D$ in the space $S$, the *skyline* of $D$ is the subset of data points that are not dominated by any other ones [3]; if $x, y \in D$, $x$ dominate $y$ (denoted as $x \prec y$), it means $x$ is as good as or better than $y$ in all dimensions, and better in at least one dimension. In this paper, suppose the small value is better, $x.i$ denotes the value of $x$ in dimension $i$, then $x \prec y$ if $\forall i \in S, x.i \leq y.i$ and $\exists j \in S, x.j < y.j$.

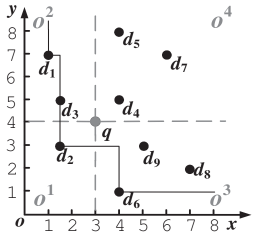


**Figure 1: Example of Skylines**

As variations of skyline, *global skyline* [7] and *dynamic skyline* [7, 18, 5] are concerned about the potential interesting points when given a query point, which are more challenging than skyline and have more applications. Global skyline is more comprehensive than dynamic skyline, which considers the directions of the query. Global skyline query can return all the points which are not global-dominated by other points; given a data set $D$ of $|S|$-dimensional points and a query point $q$ in data space $S$, $D$ can be divided into $l = 2^{|S|}$ octants with regard to (w.r.t.) $q$, the set of octants $O_q = \{o^1, o^2, ..., o^l\}$. If $x$ global-dominate $y$ w.r.t. $q$ (denoted as $x \preceq_q y$), it means $x$ and $y$ are in the same octant $o^j$ and the distance between $x$ and $q$ is as long as or shorter than the distance between $y$ and $q$ in all dimensions, and shorter in at least one dimension, then $x \preceq_q y$ if $\exists m, x, y \in o^m$, $\forall i \in S, |x.i - q.i| \leq |y.i - q.i|$ and $\exists j \in S, |x.j - q.j| < |y.j - q.j|$.

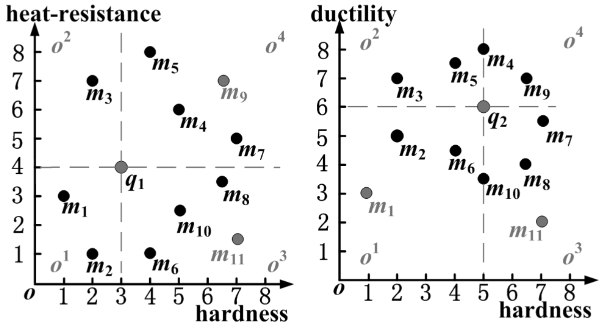
*Example 1.* (*Skyline and Global Skyline*). As illustrated in Figure 1, the data set $D = \{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8, d_9\}$ in 2-dimensional space and the query point $q = \{3, 4\}$. $d_2 \prec d_3$ because $d_2.x = d_3.x$ and $d_2.y < d_3.y$, the skyline of $D$ is $SKY(D) = \{d_1, d_2, d_6\}$. $d_4 \preceq_q d_5$ because $d_4, d_5 \in o^4$, $|d_4.x - q.x| = |d_5.x - q.x|$ and $|d_4.y - q.y| < |d_5.y - q.y|$, the global skyline w.r.t. $q$ is $GSKY_q(D) = \{d_2, d_3, d_4, d_6, d_9\}$.

When given an interesting query point $q$, global skyline query can get all the potential interesting points, which are close to $q$ in all the dimensions. In general, people do not pay attention to the information in all the dimensions, they may be concerned about the information in some subdimensions. So we introduce a new skyline variation which

*EDBT/ICDT '13* March 18 - 22 2013, Genoa, Italy

| Material | Hardness | Heat-resistance | Ductility |
|---|---|---|---|
| $m_1$ | 1 | 3 | 3 |
| $m_2$ | 2 | 1 | 5 |
| $m_3$ | 2 | 7 | 7 |
| $m_4$ | 5 | 6 | 8 |
| $m_5$ | 4 | 8 | 7.5 |
| $m_6$ | 4 | 1 | 4.5 |
| $m_7$ | 7 | 5 | 5.5 |
| $m_8$ | 6.5 | 3.5 | 4 |
| $m_9$ | 6.5 | 7 | 7 |
| $m_{10}$ | 5 | 2.5 | 3.5 |
| $m_{11}$ | 7 | 1.5 | 2 |

(a) Example of Material Library



(b) Query of $q_1 = \{3, 4, \_\}$ (c) Query of $q_2 = \{5, \_, 6\}$

**Figure 2: Example of queries in Material Library**

is concerned about global skyline in ad hoc subspace, called *subspace global skyline* (SGS), which can get the potential concerned points w.r.t. the query point in ad hoc subspace. SGS query is very useful for many applications. Next, we give the motivation of SGS query.

*Example 2.* (*Motivation of Subspace Global Skyline*). Figure 2(a) is a material library $M$, for each material $m_i \in M$, there are three properties, hardness, heat-resistance and ductility. If user$_1$ wants to obtain a material to produce automotive parts, the material which meets the conditions: hardness=3, heat-resistance=4 is the best suitable material, then s/he can do subspace global skyline query w.r.t. $q = \{3, 4, \_\}$ in $M$, the returned materials will be user$_1$'s best candidate materials. As illustrated in Figure 2(b), materials $m_1$-$m_8$ and $m_{10}$ are the candidate materials for user$_1$. User$_2$ wants to find a material to make car doors, the best suitable material meets: handness=5 and ductility=6, $m_2$-$m_{10}$ are the best candidate materials for user$_2$ in Figure 2(c).

To the best of our knowledge, this is the first attempt to compute global skyline in ad hoc subspace. Our principal contributions can be summarized as follows.

1. We introduce a novel variation of skyline, called subspace global skyline, which aims to provide a minimum set of candidates to the query point $q$, and there is no restrictions of $q$ about its dimensions and values.

2. We propose a new index RB-tree, on the basis of this index, a single SGS algorithm (SSRB) is proposed. Then an optimized algorithm OSSRB which can reduce the scan space and improve the computation efficiency is proposed to efficiently evaluate single SGS.

3. We extend single SGS query to multiple SGS queries. Firstly propose a scheduling strategy of the queries, and then propose multiple SGS algorithm MSRB.

The rest of the paper is organized as follows. Section 2 overviews the related work. Section 3 formally defines the problems. Section 4 proposes a new index RB-tree and two single SGS algorithms, SSRB and OSSRB. Section 5 presents the multiple SGS queries (MSRB) in details. Experiment results are presented in Section 6. Finally, Section 7 concludes the paper.

# 2. RELATED WORK

The skyline problem was firstly attempted by Kung et al. [10] in 1975 under the maximum vector problem. Many efficient algorithms about skyline and its variations have been proposed.

## 2.1 Traditional Skyline Processing

There are many existing classic algorithms for computing skyline points.

BNL and D&C are proposed in [3], BNL computes skyline by scanning the data file and keeping a list of candidate skyline points in main memory. D&C approach divides the data set into several partitions so that each partition fits in memory. Then the partial skyline points of every partition are computed using a main-memory algorithm, and the final skyline is obtained by merging the partial ones. SFS [6] sorts the entire data first according to some monotonic function of the skyline dimensions. The skyline points are output to a window. If the window is large enough, all the skyline points can be computed through once scan. Otherwise, the final result can be obtained through several iterations.

The above algorithms do not rely on any index, several index-based algorithms have been proposed.

In Bitmap [21], each point is mapped to a $m$-bit vector, the approach can quickly return the first few skyline points according to their insertion orders. However, the algorithm is not suitable for dynamic databases where insertions may alter the rankings of attribute values. NN [9] computes the skyline using nearest neighbor search, and prunes the search space using the newly found nearest neighbor object. BBS is proposed by Papadias et al. in [17], which is a progressive algorithm based on nearest neighbor search using R-tree structure. BBS can only access those R-tree nodes that may contain skyline points. Furthermore, it does not retrieve duplicates. Lee et al. [11] utilized the close relationship between Z-order curve and constructed a novel index structure called ZBtree, then the skyline search can be conducted over ZBtree based on the pruning property of Z-order curve. Liu et al. [16] proposed an efficient index for skyline computation, called ZINC, which is based on ZBtree.

There have also been many papers concerning the skyline query in some specific settings. Balke et al. [2] solved the skyline query in distributed environment and there are some works about skyline query in uncertain data set [19, 8, 13].

## 2.2 Skyline Variants

There are many variants of the traditional skyline query. Top-k skyline computations are proposed based on specific dominance definitions [1, 4, 14, 22]. Tao et al. [23] gave an efficient algorithm to calculate skylines in a specific subspace, Pei et al. [20] and Yuan et al. [24] presented methods

**Table 1: The Summary of Notations**

| Notation | Definition |
|---|---|
| $D$ | data set in date space $S$ |
| $q$, $Q$ | query point $q$, the query point set $Q$ |
| $S_q$ | the ad hoc subspace of $q$ |
| $x \preceq_q^{S_q} y$ | $x$ global-dominate $y$ w.r.t. $q$ in $S_q$ |
| $GSKY^{S_q}(q, D)$ | SGS w.r.t. $q$ of $D$ in $S_q$ |
| $O_q$ | the set of octants w.r.t. $q$ |
| $GSKY^{S_q,i}(q, D)$ | SGS w.r.t. $q$ in octant $o^j$ |
| $RB_i$ | index RB-tree in dimension $i$ |

to compute skylines in all possible subspace. Chen et al. [5] proposed dynamic skyline computation when a query point is given. Dellis et al. [7] proposed methods to compute reverse skyline, which is defined on basis of global skyline.

In the context of uncertain databases, Pei et al. [19] presented the probabilistic skyline over uncertain data, which can return all the probabilistic skyline points that are expected to be skyline with probability higher than a given threshold. Lin et al. [15] et al. proposed a new skyline operator over multi-dimensional uncertain data, namely stochastic skyline, which can guarantee to provide the minimum set of candidates for the optimal solutions over all possible monotonic multiplicative utility functions.

The most relevant problems to our work are the global skyline [7], subspace skyline [23], dynamic skyline in metric spaces [5] and group subspace skyline [12]. Specifically, global skyline was first proposed by Dellis et al. [7] and they applied R-tree structure to retrieve the data points in all octants, then get the global skyline. Tao et al. [23] proposed SUBSKY aims at finding the skyline in an arbitrary subspace with a low dimensionality, which settles the problem using a single B-tree, and converts multi-dimensional data to 1-dimensional values, and enables several effective pruning heuristics. Chen et al. [5] proposed technique to compute dynamic skyline in metric space, metric skyline is a variant of skyline whose dynamic attributes are defined in the metric space, and proposed an efficient and effective pruning mechanism to answer metric skyline queries through a metric index. Lian et al. [12] proposed PGSS query, which retrieves the uncertain objects not dynamically dominated by other objects w.r.t a group of queries in ad hoc subspaces. They proposed some effective pruning methods to reduce PGSS search space, then in light of the cost model, uncertain data are pre-processed and indexed, and get the final results.

In summary, most of previous studies on skyline variants are limited to get the dynamic skyline in the full space, or return the skyline in subspace w.r.t. origin. Group subspace skyline query [12] can return probabilistic dynamic skyline in ad hoc subspace. In contrast, our work focus on the global skyline query in subspace.

## 3. PROBLEM DEFINITION

Table 1 summarizes the mathematical notations used in this paper.

Specifically, given a data set $D$ in full space $S$, a query point $q$ in its subspace $S_q$. Let $x.i$ denote the $i$th coordinate value of $x$. Firstly, we define *subspace global dominance*, then give the definition of *single subspace global skyline*.

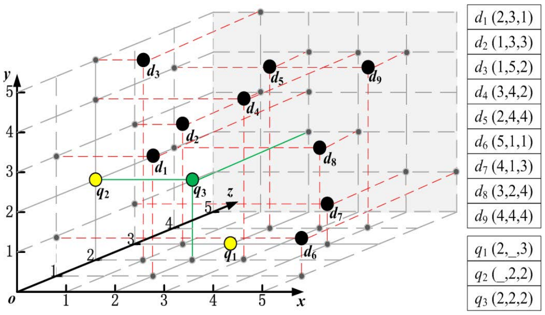


**Figure 3: Example of Subspace Global Skyline**

*Definition 1.* (*Subspace Global Dominance*). Given a set $D$ of $|S|$-dimensional points in data space $S$, $q$ is a query point in its subspace $S_q$, $S_q \subseteq S$, $x$, $y \in D$, $x$ subspace-global-dominate (sg-dominate) $y$ w.r.t. $q$ in $S_q$ (denoted as $x \preceq_q^{S_q} y$) if it holds that: 1) $\forall i \in S_q$, $|x.i - q.i| \leq |y.i - q.i|$, $(x.i - q.i)(y.i - q.i) \geq 0$; 2) $\exists j \in S_q$, $|x.j - q.j| < |y.j - q.j|$.

*Definition 2.* (*Single Subspace Global Skyline*). Given data set $D$ in data space $S$, $q$ is a query point in its subspace $S_q$, $S_q \subseteq S$, $x \in D$ is a SGS point w.r.t. $q$ if there is no point $y \in D$ such that $y \preceq_q^{S_q} x$. The set of SGS w.r.t. $q$ of $D$ (denoted as $GSKY^{S_q}(q, D)$) is the result of SGS query. $GSKY^{S_q}(q, D) = \{x | x \in D, \nexists y \in D, y \preceq_q^{S_q} x\}$.

According to the above definitions, given a query point $q$ in subspace $S_q$, $q$ can divide $D$ into $l = 2^{|S_q|}$ octants, $O_q = \{o^1, o^2, ..., o^l\}$. The subspace global dominance relationship can exist between the data points in the same octant. For each $o^j$, we can get its SGS $GSKY^{S_q,j}(q, D)$. And the whole SGS is the sum of the SGS in each octant, $GSKY^{S_q}(q, D) = \sum_{o^j \in O_q} GSKY^{S_q,j}(q, D)$.

*Example 3.* (*Single SGS*). In Figure 3, the full space $S = \{x, y, z\}$, the subspace of $q_2$ is $S_{q_2} = \{y, z\}$. $q_2$ can divide $D$ into 4 octants, $o^1 = \{d_6\}$, $o^2 = \{d_1, d_3, d_4\}$, $o^3 = \{d_7, d_8\}$ and $o^4 = \{d_2, d_3, d_4, d_5, d_8, d_9\}$. In subspace $S_{q_2}$, $d_4 \preceq_{q_2} d_3$, then we can get SGS for each octant, $GSKY^{S_{q_2},1}(q_2, D) = \{d_6\}$, $GSKY^{S_{q_2},2}(q_2, D) = \{d_1, d_4\}$, $GSKY^{S_{q_2},3}(q_2, D) = \{d_7, d_8\}$ and $GSKY^{S_{q_2},4}(q_2, D) = \{d_2, d_4, d_8\}$. The whole SGS is the sum of SGS in each octant $GSKY^{S_{q_2}}(q_2, D) = \{d_1, d_2, d_4, d_6, d_7, d_8\}$.

Next, we define *multiple subspace global skyline*.

*Definition 3.* (*Multiple Subspace Global Skyline*). Given data set $D$ in data space $S$, $Q$ is a set of query points $Q = \{q_1, q_2, ..., q_n\}$. Multiple subspace global skyline w.r.t. $Q$ can return all the single SGS results w.r.t. $q_i \in Q$.

In the above definition, multiple SGS is equivalent to computing the single SGS w.r.t. each query in $Q$.

*Example 4.* (*Multiple SGS*). In Figure 3, given $D$ in data space $S = \{x, y, z\}$, $Q = \{q_1, q_2, q_3\}$, $S_{q_1} = \{x, z\}$, $S_{q_2} = \{y, z\}$ and $S_{q_3} = \{x, y, z\}$. Multiple SGS queries can return all the single SGS results w.r.t. $q_i \in Q$. $GSKY^{S_{q_1}}(q_1, D) = \{d_1, d_2, d_4, d_5, d_7\}$, $GSKY^{S_{q_2}}(q_2, D) = \{d_1, d_2, d_4, d_6, d_7, d_8\}$ and $GSKY^{S_{q_3}}$ $(q_3, D) = \{d_1, d_2, d_4, d_5, d_6, d_7, d_8\}$.

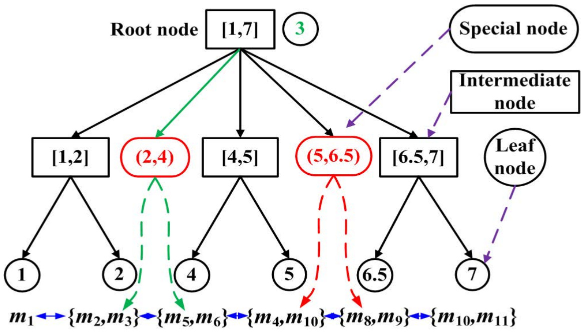


**Figure 4: RB-tree of Material Library (Figure 2(a)) in Dimension Hardness**

# 4. SINGLE SGS QUERY PROCESSING

In this section, we propose a new index structure RB-tree to rapidly find the initial scan positions of query point. Then we propose a single SGS algorithm based on RB-tree (SSRB). Finally, we present two optimization methods, then propose an optimized single SGS algorithm (OSSRB) which can reduce the scan space and improve the computation efficiency in contrast to SSRB.

## 4.1 RB-tree Index

Given data set $D$ in $S$, the query point $q$ can be in any arbitrary subspace $S_q$, and there are $2^{|S|}$ choices for $S_q$. So the indexes in the full space (e.g. R-tree, M-tree) are not appropriate for SGS problem. In order to facilitate the subspace query, we establish a separate index for each dimension.

For each dimension $i$, we build a new index based on B-tree, called Region Balance Tree (RB-tree), which can rapidly find the initial scan positions of query point. Then we build $|S|$ RB-trees for data set $D$, the RB-tree set of $D$, $RBSet(D) = \{RB_1, RB_2, ..., RB_{|S|}\}$. $RB_i$ can rapidly find the initial scan positions of $q.i$, $i \in S_q$, then begin to scan the data points from the positions to the both sides.

Given RB-tree in dimension $i$ $RB_i$, all the values of $D$ in dimension $i$ are stored in the leaf node in ascending order, and there exist a bi-pointer between two adjacent leaf nodes. The intermediate node stores the pointers and the bound of its leaf nodes. Specially, there is a special node between two adjacent intermediate nodes, which stores the vacancy interval of the two intermediate nodes, and there are two pointers in each special node. The left pointer of special node points to the last leaf node in its left intermediate node, while the right pointer points to the first leaf node in its right intermediate node.

Given $RB_i$ and $q.i$, we can easily get the initial scan positions of $q.i$, that is to say get the two values which are the closest to $q.i$ in $RB_i$, the maximum value smaller than $q.i$ and the minimum value larger than $q.i$. Then we can begin to scan the data points from the initial positions.

*Example 5.* (*Region Balance Tree RB-tree*). Figure 4 describes a RB-tree in dimension hardness of material library (Figure 2(a)), there are 6 leaf nodes sorted in the ascending order according to their values. Between the intermediate nodes $[1, 2]$ and $[4, 5]$, there exists a special node $(2, 4)$, its left pointer points to leaf node 2, which is the last leaf node in intermediate node $[1, 2]$, while its right pointer points to leaf node 4.

Given $q.hardness = 3$, we can rapidly find its initial positions of $q.hardness$, obtain the values in leaf nodes which are the closest to 3, are 2 and 4. 2 is the maximum value smaller than 3 in $RB_i$, while 4 is the minimum value larger than 3. Then we can scan the data points from the positions.

## 4.2 SSRB Algorithm

In this section, we propose some basic properties of SGS query, and apply round robin scheduling method to scan the data points in different dimensions, then present the single SGS algorithm based on RB-tree (SSRB).

Given a query point $q$ in subspace $S_q$, the data set can be divided into $2^{|S_q|}$ octants, $O_q = \{o^1, o^2, ..., o^{2^{|S_q|}}\}$, we have known that subspace global dominance relationship can exist between the points in the same octant, $GSKY^{S_q,j}(q, D)$ denote the SGS points in $o^j$.

We propose some basic properties of SGS. Subspace global dominance has transitivity, which is an important property.

THEOREM 1. *If $x \preceq_q^{S_q} y$ and $y \preceq_q^{S_q} z$, then $x \preceq_q^{S_q} z$.*

PROOF. We can easily get it by Definition 1. □

We can use RB-trees to find the initial scan positions of $q$ for all the dimensions (introduced in Section 4.1). Then we scan the data points from the initial positions to both sides, the scan direction is from near to far according to the distance to $q$. Given $RB_i$ and query point $q$, $i \in S_q$, we can obtain the theorems below.

THEOREM 2. *In $RB_i$, $x$ and $y$ are in the same octant w.r.t. $q$, and $x$ is scanned before $y$ in $RB_i$, then $x$ cannot be sg-dominated by $y$ ($y \not\preceq_q^{S_q} x$).*

PROOF. Supposed that $y \preceq_q^{S_q} x$, then $|y.i - q.i| < |x.i - q.i|$. Because $x$ is scanned before $y$ in $RB_i$, we know that $|x.i - q.i| < |y.i - q.i|$, this is contradictory to the assumption. Then we can get the conclusion in Theorem 2. □

Given the query point $q$ and data points $x, y$ in the same octant, we can know the subspace global dominance relationship between $x$ and $y$ through their scan sequences in the dimensions. Then we can get the following lemma.

LEMMA 1. *Given $x \in o^j$, it is scanned in $RB_i$, if $x \notin GSKY^{S_q,j}(q, D)$, so must exist $y \in GSKY^{S_q,j}(q, D)$ which can sg-dominate $x$, and $y$ is scanned before $x$ in all $RB_i$.*

PROOF. If $x \notin \notin GSKY^{S_q,j}(q, D)$, there must exist a SGS point $y \preceq_q^{S_q} x$, then we know that $|y.i - q.i| < |x.i - q.i|, \forall i \in S_q$, so $y$ is scanned before $x$ in all $RB_i$. □

From the above theorems, we can get that when we scan a point $x$ in $RB_i$, we can determine whether $x$ is a subspace global skyline by judging whether $x$ is sg-dominated by the points scanned before $x$ in $RB_i$.

First of all, we propose the dimension scan strategy of SSRB, which apply round robin scheduling method to scan each dimension of $q$, each time we scan the data points in both sides in dimension $i$, then we scan the data points in next dimension $i + 1$.

Given $RB_i$, we can get the initial positions of $q_i$, then there are two directions for $q_i$, we define the data points whose values in dimension $i$ are smaller than $q.i$ as direction 0, while the direction larger than $q_i$ is 1. Each time we scan

$RB_i$, we should read the data points in both directions 0 and 1. When we first scan dimension $i$, we get the data points with maximum value at $i$ in direction 0 and the data points with minimum value in direction 1. If we scan dimension $i$ the second time, we get the next data points along direction 0, and the next points along direction 1. For example in Figure 4, given $q.hardness = 3$, the first time we read $m_2, m_3$ in direction 0 while $m_5, m_6$ in direction 1. The second time we read $m_1$ in direction 0, while $m_4, m_{10}$ in direction 1.

We apply round robin scheduling to scan dimensions, and each time read data points of two directions in the current scan dimension. Theorem 3 presents the end conditions of scan in one octant.

Theorem 3. *For any $x \in D$ and $x$ belongs to octant $o^j$, if $x$ has been scanned for $|S_q|$ times, then all the SGS points in $o^j$ have been scanned. The calculation of $o^j$ is finished.*

Proof. Suppose that $y \in GSKY^{S_q}(q, D)$ and $y$ has not been scanned, then $x \not\preceq_q^{S_q} y$ and $\exists i \in S_q, |y.i - q.i| < |x.i - q.i|$, so $y$ must be scanned before $x$ in $RB_i$, this is contradictory to the assumption, then get the conclusion. □

From Theorem 3, given $x \in o^j$ has been scanned for $|S_q|$ times, then all the SGS points in $o^j$ have been found, the calculation about $o^j$ is finished. We call the data point $x$ which is scanned for $|S_q|$ times as ***finished scan point*** in $o^j$. If we find the finished scan points in every octant, the algorithm SSRB ends, all the SGS points have been found. The data points which have been scanned constitute the scan space of $q$ (denoted as $ScanSpace(q)$). Algorithm 1 presents the details of SSRB algorithm.

**Algorithm 1** SSRB Algorithm

```
Input:   data set $D$ in date space $S$;
         query point $q$ in subspace $S_q$;
Output:  the subspace global skyline $GSKY^{S_q}(q, D)$;
 1: for each dimension $i \in S_q$ do
 2:   get the initial scan positions of $q.i$ in RB-tree $RB_i$;
 3: end for
 4: while $\exists o^j \in O_q$ do not finish do
 5:   $D'$=get points by scan strategy of round robin;
 6:   for each $d \in D'$ do
 7:     $d.times{+}{+}$;
 8:     if $d.times == 1$ then
 9:       $O_d$=get the octant set $d$ belongs to;
10:       for each $o^j \in O_d$ do
11:         if $GSKY^{S_q,j}(q, D) \not\preceq_q^{S_q} d$ then
12:           add $d$ into $GSKY^{S_q,j}(q, D)$;
13:         end if
14:       end for
15:     else if $d.times == |S_q|$ then
16:       the calculations of $O_d$ ends;
17:     end if
18:   end for
19: end while
20: for each $o^j \in O_q$ do
21:   $GSKY^{S_q}(q, D) += GSKY^{S_q,j}(q, D)$;
22: end for
23: return;
```

As described in Algorithm 1, we locate the query point $q$ to find its initial scan positions in all the dimensions of $S_q$ (lines 1-3). Then we get the data point set $D'$ which should be computed according to the scan strategy of SSRB (line 5). When scanning the points in $RB_i$, we scan all the points with the same value in dimension $i$ at a time, then delete the data points which are sg-dominated by others, the remaining data points form the set $D'$. If we scan only one data point at one time, there is only one point in $D'$, do not need to be deleted. Then we deal with each data point in $D'$ (lines 6-18). If $d$ is scanned at first time (lines 8-15), we should judge which octants $d$ belongs to (line 10), then compute whether $d$ is sg-dominated by the data points in $GSKY^{S_q,j}(q, D), o^j \in O_d$, if not, $d$ is a SGS point (lines 11,12). If $d$ has been scanned for $|S_q|$ times, then the calculations of octants $d$ belongs to are finished (lines 15,16). If the calculations of all the octants are finished, we can get the final SGS points (lines 20, 21).

Finally, we give the detailed description of SSRB by the example below.

*Example 6.* (*SSRB Algorithm*). As illustrated in Figure 2(b), we firstly locate the query point $q = \{3, 4, \_\}$ in dimension hardness and heat-resistance. Then begin to scan dimension hardness the first time, get $m_2, m_3$ in direction 0 and $m_5, m_6$ in direction 1. There is no global dominance relationship between $m_2$ and $m_3$, $m_5$ and $m_6$, and we should deal with these points. Next we scan dimension heat-resistance and get $m_8$ in direction 0, $m_7$ in direction 1. Continue to scan dimension hardness the second time, we get data points $m_1$ in direction 0 and $m_4, m_{10}$ in direction 1, when scan dimension heat-resistance the second time, we get $m_1$ and $m_4$. Now $m_1.times = 2$ and $m_4.times = 2$, the calculations of the octants $m_1$ and $m_4$ belong to are finished, so $GSKY^{S_q,1}(q, D) = \{m_1, m_2\}$ and $GSKY^{S_q,4}(q, D) = \{m_4, m_5, m_7\}$. Repeat this process until the calculations of all the octants are finished, we will get the final SGS points, $m_1$-$m_8$ and $m_{10}$.

## 4.3 OSSRB Algorithm

In this section, we propose an optimized algorithm OSSRB, which adds the scan optimization and calculation optimization to SSRB.

### *4.3.1 Scan Optimization*

In SSRB, we scan the RB-trees by turns until finding the finished scan points in each octant. The scan strategy of SSRB may scan many invalid points before finding the finished scan points, which wastes a lot of calculations. So we establish a *histogram* for each dimension to optimize the scan strategy of SSRB. In the $|S|$-dimensional space, we need build $|S|$ histograms $H = \{h_1, h_2, ..., h_{|S|}\}$.

Figure 5 describes an example about scan optimization. The query point $q = \{5, 4.5\}$ in subspace $S_q = \{x, y\}$, if applying scan strategy of SSRB, we need to scan 15 data points, obtain $GSKY^{S_q,1}(D, q) = \{d_9\}$, $GSKY^{S_q,2}(D, q) = \{d_3\}$, $GSKY^{S_q,3}(D, q) = \{d_5, d_6, d_8\}$ and $GSKY^{S_q,4}(D, q) = \{d_8, d_{12}, d_{14}\}$. The scan space of SSRB is $d_1$ and $d_3$-$d_{16}$ (denoted as BScanSpace in Figure 5). Next we introduce an optimized scan strategy which can reduce the scan space.

For each octant, the finished scan point must be the SGS point, when we select different points in $GSKY^{S_q,j}(D, q)$ as finished scan points, their scan spaces are different. In Figure 5, $d_3$ and $d_9$ are the unique SGS points in their octants, they must be the finished scan points in their octants. Their scan spaces are $ScanSpace(d_3) = \{d_3, d_5, d_6, d_7, d_9\}$

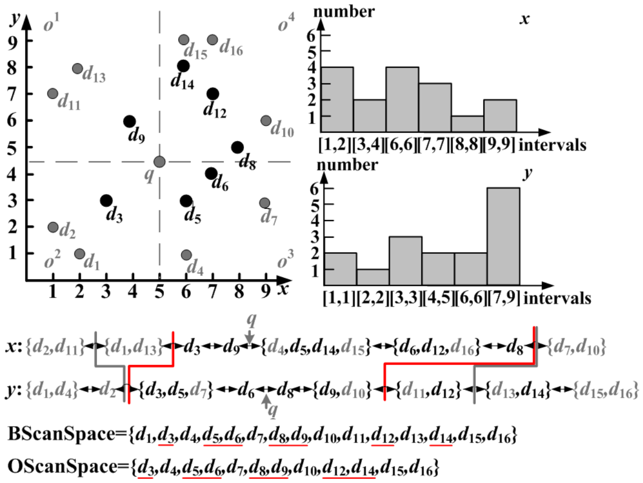

y
x
$o^1$
$o^2$
$o^3$
$o^4$
$d_1$
$d_2$
$d_3$
$d_4$
$d_5$
$d_6$
$d_7$
$d_8$
$d_9$
$d_{10}$
$d_{11}$
$d_{12}$
$d_{13}$
$d_{14}$
$d_{15}$
$d_{16}$
q
1 2 3 4 5 6 7 8 9
number
x
[1,2][3,4][6,6][7,7][8,8][9,9] intervals
number
y
[1,1][2,2][3,3][4,5][6,6][7,9] intervals
q
$x$: $\{d_2,d_{11}\}$ ↔ $\{d_1,d_{13}\}$ ↔ $d_3$ ↔ $d_9$ ↔ $\{d_4,d_5,d_{14},d_{15}\}$ ↔ $\{d_6,d_{12},d_{16}\}$ ↔ $d_8$ ↔ $\{d_7,d_{10}\}$
$y$: $\{d_1,d_4\}$ ↔ $d_2$ ↔ $\{d_3,d_5,d_7\}$ ↔ $d_6$ ↔ $d_8$ ↔ $\{d_9,d_{10}\}$ ↔ $\{d_{11},d_{12}\}$ ↔ $\{d_{13},d_{14}\}$ ↔ $\{d_{15},d_{16}\}$
q
BScanSpace=$\{d_1,\underline{d_3},d_4,\underline{d_5,d_6},d_7,\underline{d_8,d_9},d_{10},d_{11},\underline{d_{12}},d_{13},\underline{d_{14}},d_{15},d_{16}\}$
OScanSpace=$\{\underline{d_3},d_4,\underline{d_5,d_6},d_7,\underline{d_8,d_9},d_{10},\underline{d_{12},d_{14}},d_{15},d_{16}\}$

**Figure 5: Scan Optimization**

and $ScanSpace(d_9) = \{d_8, d_9, d_{10}\}$. For octants $o^3$ and $o^4$, there are some candidate points as the finished scan points, $d_5, d_6$ for $o^3$ and $d_8, d_{12}, d_{14}$ for $o^4$. We should select appropriate points as the finished scan points of $o^3$ and $o^4$ which can guarantee that the combination their scan spaces is minimal. If we choose $d_5$ (or $d_6$) and $d_8$ as the finished scan points, the scan space combination of $\{d_3, d_5(d_6), d_8, d_9\}$ is $d_3$-$d_{10}$, $d_{12}$ and $d_{14}$-$d_{16}$, which has 12 points. When we select $d_5$ (or $d_6$) and $d_{12}$ as the finished scan point in $o^4$, the scan space is $d_3$-$d_{12}$ and $d_{14}$-$d_{16}$ which has 13 points. If the finished scan points are $d_5$ and $d_{14}$, the scan space combination of $\{d_3, d_5, d_9, d_{14}\}$ is $d_3$-$d_{15}$, which has 13 points. While if the finished scan points are $d_6$ and $d_{14}$, the scan space combination of $\{d_3, d_6, d_9, d_{14}\}$ is $d_3$-$d_{16}$ which has 14 points. so $d_5$ (or $d_6$) and $d_8$ are the best finished scan points in $o^3$ and $o^4$, which can guarantee the whole scan space of all the octants is minimal. And the best finished scan points in Figure 5 are $\{d_3, d_5, d_9, d_8\}$ (or $\{d_3, d_6, d_9, d_8\}$). Next we propose an optimized scan strategy using histograms to find the best combination of these finished scan points.

Firstly, we should maintain a histogram for each dimension. For example, given histogram $h_i$ in dimension $i$, the data values in dimension $i$ are divided into several intervals, interval $r_j$ stores the total number of data points fall in its scope (denoted as $r_j.totalnum$) and its boundary values, $r_j.min$ denotes the low-bound value while $r_j.max$ denotes the up-bound value, then we can compute the density of $r_j$, $\rho(r_j) = \frac{r_j.totalnum \times unit}{|r_j.max - r_j.min| + unit}$, where $unit$ is the smallest unit of difference in interval $r_j$. For example in Figure 5, the density of interval [7,9] in histogram $h_y$ is $\frac{6 \times 1}{|9-7|+1} = 2, unit = 1$, it means the number of data points whose values in $y$-axis equal to 7 (or 8, 9) is 2.

In each interval, we hope the number of data is evenly distributed, in accordance with this principle, we divide the intervals. In Figure 5, because the number of data points whose values in $y$-axis equal to 7 is 2, also there are 2 data points whose values in $y$-axis are 8 and 9, we divide 7, 8 and 9 to the same interval [7,9]. Using these histograms, we can estimate the number of data points between two values in each dimension, then using function $h_i(x.i, y.i)$ to estimate the number of data points between $x.i$ and $y.i$ in dimension $i$, suppose that $x.i \leq y.i, x.i \in r_m, y.i \in r_n, m \leq n$.

In Equation 1, $\lfloor 4.5 \rfloor$ denotes getting the integer 4. For example in Figure 5, given two points $q.y = 4.5$ and $d_{13}.y = 8$, we can estimate the number of data points between them, $h_y(4.5, 8) = 1 \times (\lfloor \frac{(5-4.5)}{1} \rfloor + 1) + 2 \times (\frac{(8-7)}{1} + 1) + 2 = 1 + 4 + 2 = 7$, they are $d_8$-$d_{14}$ in dimension $y$.

$$h_i(x.i, y.i) = \begin{cases} \rho(r_m)(\lfloor \frac{|y.i - x.i|}{unit} \rfloor + 1), & m = n \\ \\ \rho(r_m)(\lfloor \frac{|r_m.max - x.i|}{unit} \rfloor + 1) & \\ +\rho(r_n)(\lfloor \frac{|y.i - r_n.min|}{unit} \rfloor + 1) & m \neq n \\ + \sum_{j \in (m,n)} r_j.totalnum & \end{cases} \tag{1}$$

Before describing the details of optimized scan strategy, we give some representations. $cnum_{i,0(1)}$ denotes the number of data points we have scanned in direction 0 (or 1) of dimension $i$. Given a data $x$ has been scanned in dimension $i$, we can use Equation 1 to estimate the number of data points between the initial scan position $q.j$ and $x.j, \forall j \in S_q, j \neq i$, which is the number should be scanned before scanning $x$ in dimension $j$. $x.hnum_{j,0(1)}$ denotes the number of data points which should be scanned at direction 0 (or 1) in dimension $j$ until scanning $x$. The difference between $cnum$ and $x.hnum$ in corresponding direction 0 (or 1) of dimension $i$ is denoted as $x.dif_{i,0(1)} = x.hnum_{i,0(1)} - cnum_{i,0(1)}$, and $x.Alldif = \sum_{i \in S_q} x.dif_{i,0(1)}$ denotes the sum of $x.dif$ in all dimensions. For example in Figure 5, $q = \{5, 4.5\}$ we scan $d_9 = \{4, 6\}$ in direction 0 of $x$-axis, then $cnum_{x,0} = 1$, and we can estimate $d_9.hnum_{y,1} = h_y(4.5, 6) = 3$. $d_9.dif_{y,1} = 3 - 0 = 3$, and $d_9.Alldif = 3$.

For each octant, we select the data point with minimum value $Alldif$ as the *finished scan point* of corresponding octant. From these finished scan points in all the octants, we choose one with minimum $Alldif$ as the ***fastest finished scan point***.

Here we give the detailed description of optimized scan strategy. Firstly we randomly select dimension $i$ and begin to scan the data points in both sides from the initial scan positions of $q.i$. Secondly we deal with the initial scanned data points, and find the *fastest finished scan point* $d_{fast}$. Next we choose the direction with minimum value $d_{fast}.dif_{i,0(1)}$ to scan, and deal with the new scanned data points, update the corresponding parameters, and select new direction with minimum value $d_{fast}.dif_{i,0(1)}$ to scan. Repeat this process until all SGS points are found.

*Example 7.* (*Optimized scan strategy*). As illustrated in Figure 5, starting from $x$-axis, we scan the data point $d_9$ in direction 0, $\{d_4, d_5, d_{14}, d_{15}\}$ in direction 1, $cnum_{x,0} = 1$, $cnum_{x,1} = 4, cnum_{y,0} = 0, cnum_{y,1} = 0$. Because $d_4, d_{15}$ are sg-dominated by $d_5, d_{14}$, we only need to deal with $d_9, d_5$, $d_{14}$. $GSKY^{S_q,1} = \{d_9\}$ and $d_9.Alldif = d_9.dif_{y,1} = 3$, $GSKY^{S_q,3} = \{d_5\}$ and $d_5.Alldif = 4$, and $GSKY^{S_q,4} = \{d_{14}\}$, $d_{14}.Alldif = 7$. So $d_9$ is the fastest finished scan point, we select the direction of $d_9$ (direction 1 of $y$-axis) to scan. Next we get $d_8$, update $cnum_{y,1} = 1$, $GSKY^{S_q,4} = \{d_{14}, d_8\}$, $d_8.Alldif = 4$ which is smaller than $d_{14}.Alldif = 6$, $d_8$ replace $d_{14}$ as the finished scan point of $o^4$. And $d_9$ is still the fastest finished scan point with $d_9.Alldif = 2$, continue to scan direction 1 of $y$-axis, and get points $d_9, d_{10}$, $d_9 \preceq_q^{S_q} d_{10}$, only to deal with $d_9$. Now $d_9$ has been scanned twice, the calculation of octant $o^1$ has been finished,

$GSKY^{S_q,1} = \{d_9\}$, update $cnum_{y,1} = 3$. We select the fastest finished scan point in $d_5, d_8$, because $d_5.Alldif = d_8.Alldif = 4$, we randomly select $d_5$ as the fastest finished scan point and scan direction 0 of $y$-axis, then get $d_6$, update $cnum_{y,0} = 2$. $GSKY^{S_q,3} = \{d_5, d_6\}$, $d_6.Alldif = 3$ which is not smaller than $d_5.Alldif = 3$, $d_5$ is still the fastest finished scan point, continue to scan direction 0 of $y$-axis and get $d_3, d_5, d_7$, update $cnum_{y,0} = 5$. Because $d_5 \preceq_q^{S_q} d_7$, we need to deal with $d_3$ and $d_5$, $d_5$ has been scanned twice, the calculation of $o^3$ is finished, $d_3$ belongs to $o^2$ and $GSKY^{S_q,2} = \{d_3\}$, $d_3.Alldif = 2$ which is the fastest finished scan point. Repeat this process, next we select $d_8$ as the fastest finished scan point. At last, we get the final SGS set $GSKY^{S_q} = \{d_3, d_5, d_6, d_8, d_9, d_{12}, d_{14}\}$.

### 4.3.2 Calculation Optimization

In Section 4.2, if $x \in o^j$, when we scan $x$ the first time, we must consider whether $x$ is sg-dominated by the points in $GSKY^{S_q,j}(D,q)$ w.r.t. $q$, then we can determine whether $x \in GSKY^{S_q,j}(D,q)$. If the number of data points in $GSKY^{S_q,j}(D,q)$ is large, it requires a lot of calculations. In this section, we propose two pruning strategies, which can reduce amount of redundant calculations and improve the computation efficiency.

For each set $GSKY^{S_q,j}(q,D)$, we maintain two points, *Low Filter Point* ($LFP^j$) and *High Filter point* ($HFP^j$).

Suppose the current scan dimension is $i$, the value of $LFP^j$ in dimension $i$ $LFP^j.i$ is the current scan value. The value of $LFP^j$ in the other dimension $l$ is the nearest value to $q.l$ in $GSKY^{S_q,j}(q,D)$. $LFP^j.l = argmin_{x.l}(|x.l - q.l|), x \in GSKY^{S_q,j}(q,D), l \neq i$. The value of $HFP^j$ in dimension $i$ is also the current scan value. The value of $HFP^j$ in other dimension $l$ is the farthest value to $q.l$ in $GSKY^{S_q,j}(q,D)$. $HFP^j.l = argmax_{x.l}(|x.l - q.l|), x \in GSKY^{S_q,j}(q,D), l \neq i$. Then we can get two pruning rules.

THEOREM 4. *Given $x \in o^j$ and $x$ is scanned the first time, if $x$ is sg-dominated by $HFP^j$, $x \notin GSKY^{S_q}(q,D)$.*

PROOF. If $HFP^j \preceq_q^{S_q} x$, $\forall y \in GSKY^{S_q,j}(q,D), y \preceq_q^{S_q} x$, $x$ cannot be the SGS point. □

THEOREM 5. *Given $x \in o^j$ and $x$ is scanned the first time, if $x$ is not sg-dominated by $LFP^j$, $x \in GSKY^{S_q,j}(q,D)$.*

PROOF. If $LFP_i^j \not\preceq_q^{S_q} x$, $\forall y \in GSKY^{S_q,j}(q,D), y \not\preceq_q^{S_q} x$, $x$ can add to $GSKY^{S_q,j}(q,D)$. □

We can use Theorem 4 and 5 to rapidly judge whether $x$ belongs to $GSKY^{S_q}(D,q)$. The details of the filter rules are described in Algorithm 2.

**Algorithm 2** Filter(d)

```
Input: point $d \in o^j$ which is scanned the first time;
       $LFP^j$ and $HFP^j$ corresponding to $GSKY^{S_q,j}(D,q)$;
Output: updated $LFP^j$, $HFP^j$ and $GSKY^{S_q,j}(D,q)$;
1: if $HFP^j \preceq_q^{S_q} d$ then
2:     $d$ is marked as non-result point;
3:     continue deal with next point;
4: else if $LFP^j \not\preceq_q^{S_q} d$ then
5:     add $d$ into $GSKY^{S_q,j}(D,q)$;
6:     update $LFP^j$, $HFP^j$;
7: end if
8: return;
```

In Algorithm 2, we can know that $LFP^j$ and $HFP^j$ need to be updated when there is a new point added to $GSKY^{S_q,j}(D,q)$ (line 6). And we can add Algorithm 2 to Algorithm 1 between lines 10 and 11, then calculation optimization can be achieved.

The OSSRB proposed in this section has better performance than SSRB by using the optimized scan strategy and filter points, which can reduce the scan space and improve the computation efficiency.

## 5. MULTIPLE SGS QUERIES PROCESSING

In this section, we propose an algorithm to process multiple SGS queries. When given multiple queries, we can execute these queries one by one using OSSRB, which would waste a lot of calculations. Therefore, we propose a scheduling strategy to divide some queries into one subset, then present an algorithm MSRB to compute these queries together through one pass scan.

### 5.1 Division Scheduling Strategy of Queries

In this section, we introduce a division scheduling strategy of queries. Given a query set $Q$, the scan spaces of some queries have overlap, we except to compute these queries together to reduce the scan space, the division scheduling can divide some similar queries into the same subset, and the queries in a subset can be computed together.

First of all, we introduce the *similarity* of different queries. Given two query points $q_1$ and $q_2$, when we calculate their single SGS, the more overlap of their scan space, the more similar they are. We cannot get their exact scan space until finishing their SGS queries, but we know some rules to estimate their *similarity*.

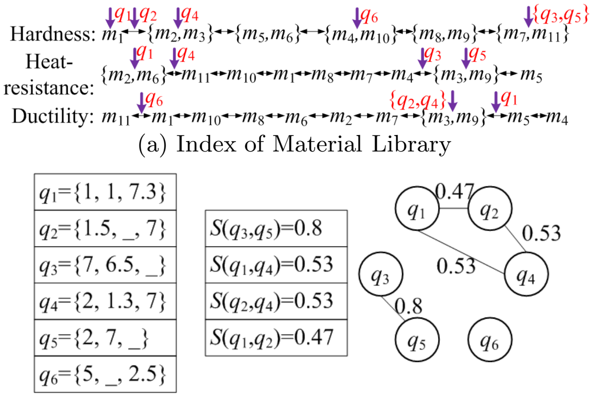
Given two query points $q_1$, $q_2$, and they share dimension $i$, if the difference between $q_1.i$ and $q_2.i$ is too lager, it means the number of data points between $q_1.i$ and $q_2.i$ is large, their scan spaces in dimension $i$ may have no overlap, the *dimension similarity* of $q_1$ and $q_2$ is 0. If the difference between $q_1.i$ and $q_2.i$ is very small, there must be some overlap of their scan spaces in dimension $i$. Based on the rules above, we design function $S_i(q_1, q_2)$ $(0 \leq S_i(q_1, q_2) \leq 1)$ to estimate the ***dimension similarity*** of two queries in dimension $i$ if they share dimension $i$.

$$S_i(q_1, q_2) = \begin{cases} 1, & q_1.i = q_2.i \\ 0, & q_1.i \neq q_2.i, |\frac{h_i(q_1.i, q_2.i)}{N}| > \varepsilon \\ 1 - \frac{h_i(q_1.i, q_2.i)}{\varepsilon \times N}, & q_1.i \neq q_2.i, |\frac{h_i(q_1.i, q_2.i)}{N}| \leq \varepsilon \end{cases} \quad (2)$$

$h_i(q_1.i, q_2.i)$ denotes the number of data points between $q_1.i$ and $q_2.i$, which can be calculated using Equation 1. $N$ denotes total number of data points in data set $D$. $\varepsilon$ is a minimum value $(0 < \varepsilon << 1)$, which is used to limit the maximum number of data points between two similar queries. If $\varepsilon$ is too large, the similar queries may have no overlap of their scan spaces, while if $\varepsilon$ is too small, the queries whose scan spaces have overlap will not be determined as similar queries, so we should select an appropriate $\varepsilon$.

We can calculate the similarity of two queries in all their shared dimensions using Equation 2, for other dimensions which can not be shared, the dimension similarity is 0. Then we use $S(q_1, q_2)$ to denote the similarity of $q_1$ and $q_2$.

$$S(q_1, q_2) = \frac{\sum_{i \in (S_{q_1} \cap S_{q_2})} S_i(q_1, q_2)}{|S_{q_1} \cup S_{q_2}|} \quad (3)$$


Hardness: $m_1$ ↔ $\{m_2,m_3\}$ ↔ $\{m_5,m_6\}$ ↔ $\{m_4,m_{10}\}$ ↔ $\{m_8,m_9\}$ ↔ $\{m_7,m_{11}\}$
$q_1$ $q_2$ $q_4$ $q_6$ $\{q_3,q_5\}$
Heat-resistance: $\{m_2,m_6\}$ ↔ $m_{11}$ ↔ $m_{10}$ ↔ $m_1$ ↔ $m_8$ ↔ $m_7$ ↔ $m_4$ ↔ $\{m_3,m_9\}$ ↔ $m_5$
$q_1$ $q_4$ $q_3$ $q_5$
Ductility: $m_{11}$ ↔ $m_1$ ↔ $m_{10}$ ↔ $m_8$ ↔ $m_6$ ↔ $m_2$ ↔ $m_7$ ↔ $\{m_3,m_9\}$ ↔ $m_5$ ↔ $m_4$
$q_6$ $\{q_2,q_4\}$ $q_1$
$q_1$={1, 1, 7.3}
$q_2$={1.5, _, 7}
$q_3$={7, 6.5, _}
$q_4$={2, 1.3, 7}
$q_5$={2, 7, _}
$q_6$={5, _, 2.5}
$S(q_3,q_5)$=0.8
$S(q_1,q_4)$=0.53
$S(q_2,q_4)$=0.53
$S(q_1,q_2)$=0.47
$q_1$ $q_2$ $q_3$ $q_4$ $q_5$ $q_6$
0.47
0.53
0.53
0.8

(a) Index of Material Library

(b) Queries Division ($\varepsilon$=0.5)

**Figure 6: Division Scheduling of Queries**

Given any two queries, we can get their similarity using Equation 2 and 3. The larger the similarity, the more overlap of their scan spaces. If their similarity is larger than 0, they are called *similar queries*.

Given a query point set $Q = \{q_1, q_2, ..., q_m\}$, we divide $Q$ into some subsets $Q = \{Q_1, Q_2, ..., Q_l\}$, each subset stores the similar queries. The queries in each subset can be calculated together. Therefore, each query in $Q_i$ should have high similarity with the other queries in $Q_i$. Then we use $S(Q_i)$ to denote the similarity of the queries in $Q_i$.

$$S(Q_i) = \begin{cases} \frac{\sum_{\forall q_i, q_j \in Q_i} S(q_i, q_j)}{C^2_{|Q_i|}} & |Q_i| \geq 2 \\ 0, & |Q_i| = 1 \end{cases} \tag{4}$$

In Equation 4, $C^2_{|Q_i|}$ is the number of combinations of any two queries in $Q_i$, $C^2_{|Q_i|} = \frac{|Q_i| \times (|Q_i - 1|)}{2}$. We can use Equation 4 to measure the whole similarity of $Q_i$. Next we use function $f(Q)$ to score the division of $Q$.

$$f(Q) = \frac{\sum_{Q_i \in Q} S(Q_i)}{|Q_{subset}|} \tag{5}$$

In Equation 5, $|Q_{subset}|$ denotes the number of subsets of $Q$, the more the function $f(Q)$, the better the division scheduling of $Q$. Then we give an example about queries division scheduling as follows.

Figure 6 describes an example of queries division, Figure 6(a) is the index of Material in Figure 2(a), meanwhile gives the initial scan positions of 6 queries in all the dimensions. In this example we suppose $\varepsilon = 0.5$, then we can calculate the similarity between two queries using Equation 2 and 3, and get the results in Figure 6(b). From Figure 6(b), we divide six queries into three subsets $Q_1 = \{q_1, q_2, q_4\}, Q_2 = \{q_3, q_5\}, Q_3 = \{q_6\}$, and $S(Q_1) = 0.51, S(Q_2) = 0.8, S(Q_3) = 0$, the function of this division scheduling is $f(Q) = 0.44$, $f(Q)$ of other divisions must be less than 0.44. The best division is $Q = \{\{q_1, q_2, q_4\}, \{q_3, q_5\}, \{q_6\}\}$.

Given a query set $Q$, we can easily get the similarity between any two queries in $Q$ using Equation 2 and 3. All the similarity values are stored in similarity queue $SQ$, and then sort $SQ$ in descending order according to $S(q_i, q_j)$. Then we propose algorithm 3 to divide $Q$ rapidly.

As described in Algorithm 3, we apply greedy algorithm to divide the queries in $Q$, every time we process the two queries with greatest similarity in the remaining $SQ$ (line 2). Then we assign the two queries to appropriate subset, and guarantee the function $f(SUB)$ w.r.t. the current division $SUB$ with the greatest value (lines 3-14). Algorithm 3 can provide a good division of $Q$ rapidly.

**Algorithm 3** Division Scheduling of Queries

```
Input: query point set $Q$;
       similarity queue $SQ$ in descending order;
Output: the division $SUB$ of $Q$;
 1: while $SQ$ is not empty do
 2:   get the first element of $SQ$ to $S(q_i, q_j)$;
 3:   if $q_i$ and $q_j$ have not been assigned to a subset then
 4:     assign $q_i$ and $q_j$ to a new subset $sub_j$;
 5:     add $sub_j$ to $SUB$;
 6:   else if $q_i$ has been assigned to subset $sub_i$ then
 7:     compute $f(SUB)$ suppose $q_j$ is assigned to $sub_i$;
 8:     compute $f(SUB)'$ suppose $q_i$ and $q_j$ are assigned
        to a new subset $sub_j$;
 9:     if $f(SUB) > f(SUB)'$ then
10:       assign $q_j$ to $sub_i$;
11:     else
12:       assign $q_i$ and $q_j$ to a new subset $sub_j$;
13:       add $sub_j$ to $SUB$;
14:     end if
15:   end if
16: end while
```

## 5.2 MSRB Algorithm

In this section, we propose the details of MSRB algorithm, given a similar query set $Q$, how to compute the multiple SGS results for each query in $Q$ together.

Given a subset $Q$ of similar queries (divided in Section 5.1), we have known the similarity between these queries in $Q$. We use $S(q_i, Q)$ to measure the similarity between $q_i$ and other queries in $Q$. Then we sort the queries in $Q$ by descending order according to the value of $S(q_i, Q)$, and select the query point $q_i$ with maximum $S(q_i, Q)$ as the ***main query point*** (denoted as $q_{main}$).

$$S(q_i, Q) = \sum_{\forall q_j \in Q, q_j \neq q_i} S(q_i, q_j) \tag{6}$$

Equation 6 gives how to compute the $S(q_i, Q)$. For example in Figure 6(b), $Q_1 = \{q_1, q_2, q_4\}$, and $S(q_1, Q_1) = 1, S(q_2, Q_1) = 1, S(q_4, Q_1) = 1.06$, so sort $Q_1 = \{q_4, q_1, q_2\}$ and $q_4$ is the *main query point*.

Then we can get the queries which are similar to $q_{main}$ in all its dimensions, the queries which are similar to $q_{main}.i$ can be saved in $Set_i(q_{main})$. Next we can get the initial scan positions of $q_{main}$ in all dimensions, and use optimized scan strategy to scan the data points.

When we scan a data $x$ in dimension $i$, $x$ is not only in the scan space of $q_{main}$, but also in the scan space of the queries in $Set_i(q_{main})$. So we deal with $x$ for all the queries in $Set_i(q_{main})$. Because the scan direction of $q_{main}$ is different from the queries in $Set_i(q_{main})$. So we apply different computation methods.

For query point $q$ and the current scan data point $x \in o^j$, if the scan direction is from near to far, we detect whether $x$ is sg-dominated by the data points which have been scanned before in $GSKY^{S_q, j}(q, D)$. Conversely, if the direction is from far to near, we detect whether $x$ can sg-dominate previous scanned data points in $GSKY^{S_q,j}(q,D)$.

We use OSSRB to compute the SGS points of $q_{main}$, meanwhile we deal with the data points in $ScanSpace(q_{main})$ for other queries in $Set_i(q_{main}), \forall i \in S_{q_{main}}$. If the computation of $q_{main}$ is finished, we select the next main query point in the remaining $Q$. Repeat this process until all the queries in $Q$ have been finished. Next we give the details of MSRB algorithm.

**Algorithm 4** MSRB Algorithm

```
Input: sorted similar query set $Q$;
Output: $GSKY(q_i, D)$ for all $q_i \in Q$;
1: while $Q$ is not empty do
2:     Remove the first element of $Q$ to $q_{main}$;
3:     while OSSRB($q_{main}, D$) is not finished do
4:         $d$ = the data point which is obtainned by optimized
           scan strategy of $q_{main}$ in dimension $i$;
5:         for each query point $q_j$ in $Set_i(q_{main})$ do
6:             deal with $d$ w.r.t. $q_j$;
7:         end for
8:     end while
9: end while
```

In Algorithm 4, for every main query point $q_{main}$, we apply OSSRB method to compute its $GSKY^{S_{q_{main}}}(q_{main}, D)$ (lines 3-8), meanwhile deal with the scanned data point w.r.t. the queries which are similar to $q_{main}$ (lines 4-7). When all the query points in $Q$ have been finished, MSRB ends. Then we give an example about MSRB.

*Example 8.* (*MSRB Algorithm*). As illustrated in Figure 6, the sorted $Q_1 = \{q_4, q_1, q_2\}$, $q_4$ is the main query point. And we begin to scan the data points according to the optimized scan strategy of $q_4$. Firstly we scan $m_2, m_3$ in dimension hardness and $q_1, q_2$ are the similar queries with $q_4$ in dimension hardness, then get $GSKY^{S_{q_4}}(q_4, D) = \{m_2, m_3\}$, $GSKY^{S_{q_1}}(q_1, D) = \{m_2, m_3\}$, $GSKY^{S_{q_2}}(q_2, D) = \{m_3\}$ because $m_3 \preceq^{S_{q_2}}_{q_2} m_2$. Now $m_2$ is the fastest finished scan point of $q_4$, then we scan $m_2, m_6$ at direction 0 in dimension heat-resistance and $q_1$ is the similar query, because $m_2$ sg-dominate $m_6$ w.r.t. $q_1, q_4$, there is no change about SGS w.r.t. $q_1, q_4$. Continue to scan $m_3, m_9$ in dimension ductility and $q_1, q_2$ are the similar queries, because $m_3$ sg-dominate $m_9$ w.r.t. $q_1, q_2, q_4$, there are no change about SGS w.r.t. $q_1, q_2, q_4$. Then scan $m_7$ at direction 0 in ductility and get $GSKY^{S_{q_4}}(q_4, D) = \{m_2, m_3, m_7\}$, $GSKY^{S_{q_1}}(q_1, D) = \{m_2, m_3, m_7\}$. Next scan $m_2$ at direction 0 in ductility, $m_2$ has been scanned 3 times, the calculation of octants $m_2$ belongs to are finished. And we select $m_3$ as the fastest finished scan point, then scan $m_{11}, m_{10}, m_1, m_8$, $m_7, m_4, m_3, m_9$ at direction 1 in heat-resistance while $m_3$ is the fastest finished scan point, get $GSKY^{S_{q_4}}(q_4, D) = \{m_2, m_3, m_7, m_{11}, m_{10}, m_1, m_8, m_4\}$ and $GSKY^{S_{q_1}}(q_1, D) = \{m_2, m_3, m_7, m_1, m_4\}$. Now $m_3$ has been scanned 3 times, the calculation of octants $m_3$ belongs to are finished, and the calculations of all the octants of $q_4$ are finished.

Then we begin to deal with the remaining queries $q_1, q_4$, now $q_1$ is the $q_{main}$. Get the initial scan positions of $q_1$, because the scan spaces in some dimensions have been scanned, we can skip these spaces. We scan $m_1$ in dimension hardness and $q_2$ is the similar query, then get $GSKY^{S_{q_2}}(q_2, D) = \{m_3, m_1\}$. Because the scan direction is from far to near for $q_1$, we detect whether $m_1$ sg-dominate $m_2, m_3, m_7$ which are in the same octants with $m_1$. Because there is no data points along direction 0 in hardness, the calculations of some octants are finished for $q_1, q_2$. Therefore, the calculations of all the octants of $q_2$ are finished, we get $GSKY^{S_{q_2}}(q_2, D) = \{m_3, m_1\}$. And the calculations of only one octant of $q_1$ ($m_4$ belongs to) has not been finished. Next we scan $m_5, m_4$ at direction 1 in ductility, because there is no data along the direction 1 in ductility, the calculations of all the octants of $q_2$ are finished. Finally we get $GSKY^{S_{q_4}}(q_4, D) = \{m_1, m_2, m_3, m_4, m_7, m_8, m_{10}, m_{11}\}$ and $GSKY^{S_{q_1}}(q_1, D) = \{m_1, m_2, m_3, m_4, m_7\}$, $GSKY^{S_{q_2}}(q_2, D) = \{m_1, m_3\}$

# 6. EXPERIMENTAL EVALUATION

In this section, through extensive experiments, we demonstrate the efficiency and effectiveness of our proposed approaches to answer SGS queries. We have developed a simulator to evaluate the performance of our proposed approaches with C++ programming language. A PC with Intel Core i3-2120 CPU at 3.3GHz, 4GB Ram is used for all experiments.

In particular, we test our methods using both real and synthetic data sets. Real data set applies forest environmental monitoring data obtained by a sensor network. In the real data set, there are 30000 tuples and each tuple has 4 attributes, including humidity, temperature, light and voltage. Table 2 records average scan space number (the number of data points in scan space) and total CPU time of random 100 SGS queries for algorithms SSRB, OSSRB, MSRB($\varepsilon = 0.003$), B-tree and R-tree. B-tree algorithm applies the scan strategy of SSRB and B-tree index, while R-tree algorithm uses BBS to compute SGS based on R-tree index. Because the total number of tuples is 30000, we set $\varepsilon = 0.003$, then $\varepsilon \times 30000 = 90$, which means if the number of data points between two queries is larger than 90, they are not the similar queries.

**Table 2: Experimental Results of Real Date**

| Algorithms | CPU time ($s$) | Scan Space Number |
|---|---|---|
| SSRB | 306 | 7357 |
| OSSRB | 262 | 5723 |
| MSRB ($\varepsilon = 0.003$) | 214 | 4013 |
| B-tree | 320 | 7357 |
| R-tree | 612 | 9221 |

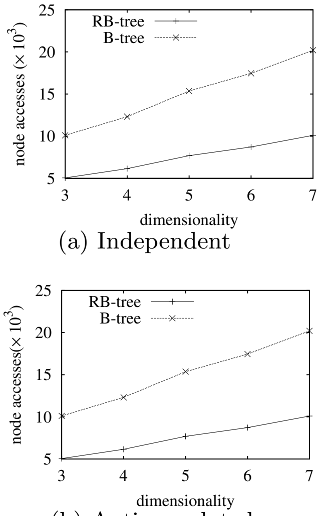
In Table 2, the average result number of 100 SGS queries is 819. MSRB can get the SGS results just scan 4013 points, which is the best algorithm to compute multiple SGS results, and meanwhile its CPU time is the least. OSSRB has the best performance to compute these SGS queries one by one.

Next we verify the performance of our algorithms using synthetic data sets. For synthetic data sets, we employ independent(uniform) and anti-correlated data sets with dimensionality $d$ in the range [3,7] and cardinality $N$ in the range [100K,1.6M]. Data sets are indexed by RB-trees using a page size of 4kbytes, resulting in node capacities is 254 which is not affected by dimensionality.

In particular, Section 6.1 and 6.2 study the effect of dimensionality and cardinality, respectively.

## 6.1 The effect of dimensionality

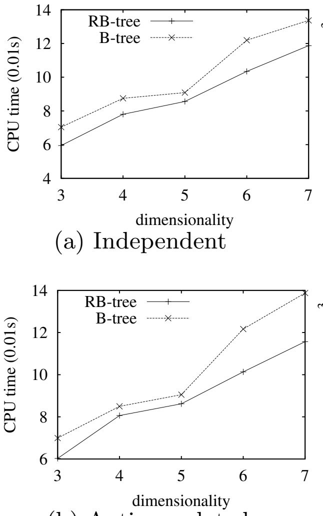
In order to study the effect of dimensionality, we use the data sets with cardinality $N$=100K and vary $d$ between 3



(a) Independent

(b) Anti-correlated

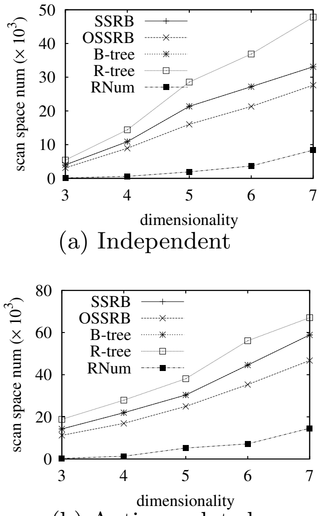
**Figure 7: Node accesses vs. $d(N = 100\mathbf{K})$**



(a) Independent

(b) Anti-correlated

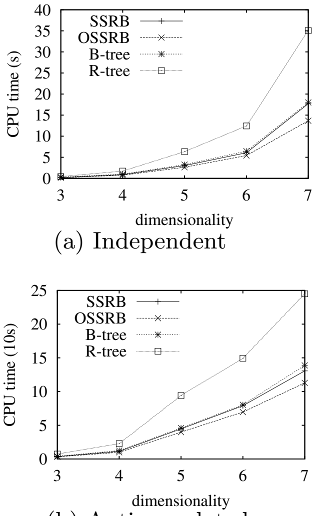
**Figure 8: CPU-time vs. $d(N = 100\mathbf{K})$**



(a) Independent

(b) Anti-correlated

**Figure 9: Scan space number vs. $d(N = 100\mathbf{K})$**



(a) Independent

(b) Anti-correlated

**Figure 10: CPU-time vs. $d(N = 100\mathbf{K})$**

and 7. Given any dimension $d \in [3, 7]$, we randomly generate 1000 query points, and the dimension of each query point $d'$ is between 2 and $d$. We randomly select arbitrary $d'$ dimensions in the full data space.

### 6.1.1 Index Performance

In this section, we demonstrate the efficiency and effectiveness of our proposed index structure RB-tree. B-tree and RB-tree are both able to scan the data points sequentially from the initial scan positions to both sides, so we compare the ability of finding the initial scan positions of the 1000 queries using two indexes. We record the total number of node access and CPU time for locating 1000 consecutive queries.

Figure 7 shows the number of node access as a function of dimensionality, for independent (7(a)) and anti-correlated (7(b)) data sets. Figure 8 illustrates a similar experiment that compares the indexes in terms of CPU-time under the same settings. With the growth of dimensionality, the number of node access and CPU-time increase. And the experimental results are consistent in the two data sets. Because with the growth of dimensionality, the dimension number of query points is larger, and more query locations will come, so the number of node access and CPU-time will increase. We can see that the number of node access of RB-tree is much larger than that of B-tree under the same conditions and the differences increase with the dimensionality. Thus, the CPU-time of RB-tree is shorter than that of the B-tree. From the above phagomania, we can get the conclusion that RB-tree has better performance than B-tree regardless of the change of dimensionality.

### 6.1.2 Single query algorithms

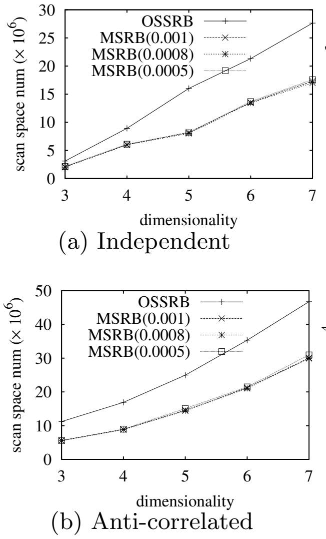
In this section, we demonstrate the efficiency and effectiveness of our proposed single SGS query algorithms. We record the performance of our proposed algorithms SSRB and OSSRB. In addition, we compare our algorithms to B-tree algorithm and R-tree algorithm. B-tree algorithm applies the scan strategy of SSRB and B-tree index, while R-tree algorithm applies the variant BBS algorithm using R-tree index. We randomly generate 1000 subspace query points, and record the average scan space number and CPU time of 1000 queries.

Figure 9 shows the number of scan points as a function of dimensionality, over independent (9(a)) and anti-correlated (9(b)) data sets. Figure 10 illustrates a similar experiment compared with the algorithms in terms of CPU-time under the same settings. With the growth of dimensionality, the scan space and CPU time increase in all the algorithms. We can see that OSSRS has better performance than any other algorithm, while scan space is the smallest and CPU time is the fastest. All the algorithms have better performance in independent data set because of the distribution characteristics.

### 6.1.3 Multiple queries results

In this section, we demonstrate the efficiency and effectiveness of algorithm MSRB. We record the performances of OSSRB and MSRB. We randomly generate 1000 query points and record the total scan space numbers and CPU time of 1000 queries. OSSRB deals with these queries one by one. The value after MSRB is the value of $\varepsilon$. The number of the data set is $N = 100000$. If the number of data points between two queries is larger than 100, the overlap of their scan space must become very small, so the value of $\varepsilon$ should be smaller than 0.001. If $\varepsilon$ is small enough, the similar queries will be fewer, so the value of $\varepsilon$ should larger than 0.0005. We give the experimental results of MSRB when $\varepsilon = 0.001$, $\varepsilon = 0.0008$ and $\varepsilon = 0.0005$.

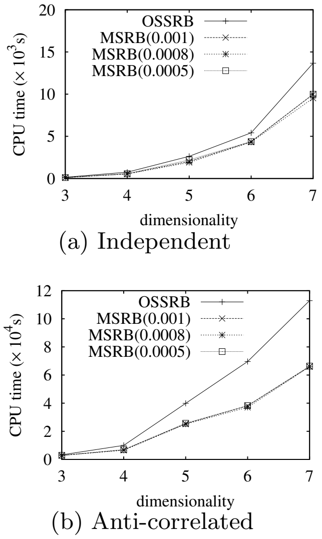
Figure 11 shows the total number of scan points for all the 1000 queries as a function of dimensionality, for independent (11(a)) and anti-correlated (11(b)) data sets. Figure 12 illustrates a similar experiment that compares the algorithms in terms of CPU-time under the same settings. With the growth of dimensionality, the scan space number and CPU time increase, the performances of the algorithms are better in independent data set than anti-correlated data set. We



(a) Independent

(b) Anti-correlated

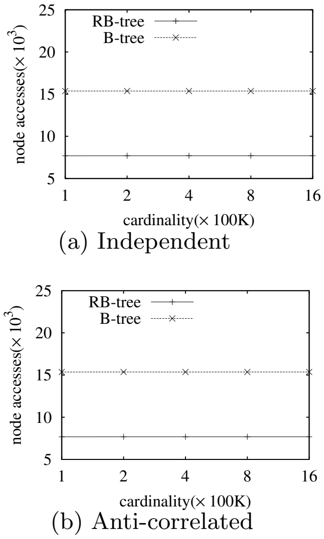
**Figure 11: Scan space number vs. $d(N = 100\mathbf{K})$**



(a) Independent

(b) Anti-correlated

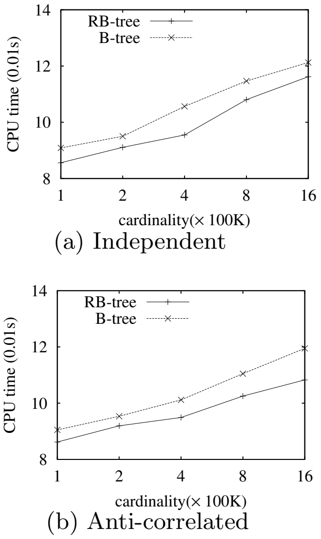
**Figure 12: CPU-time vs. $d(N = 100\mathbf{K})$**



(a) Independent

(b) Anti-correlated

**Figure 13: Node accesses vs. $N(d = 5)$**



(a) Independent

(b) Anti-correlated

**Figure 14: CPU-time vs. $N(d = 5)$**

can see that MSRB has much better performance than OSSRB to process multiple queries. If vary $\varepsilon$ between 0.0005 and 0.001, there is little effect on MSRB.

## 6.2 The effect of cardinality

In order to study the effect of cardinality, we use the data sets with dimensionality $d$=5 and vary $N$ between $100K$ and $1.6M$. We randomly generate 1000 query points, the dimension number $d$ of each query point is between 2 and 5, and we randomly select arbitrary $d$ dimensions in the full data space.

### 6.2.1 Index Performance

Figure 13 and 14 show the number of node access and CPU time, respectively, versus the cardinality for 5-dimensional data sets. We can see that with the growth of cardinality, the number of node access maintains stable and CPU time slowly increases. In both data sets, the performance of RB-tree is better than B-tree.

### 6.2.2 Single query algorithms

In this section, we record the performance of the algorithms SSRB, OSRB, algorithm with B-tree index and algorithm with R-tree index.

Figure 15 shows the number of scan points as a function of cardinality, for independent (15(a)) and anti-correlated (15(b)) data sets. Figure 16 illustrates a similar experiments that compare the algorithms in terms of CPU-time under the same settings. With the growth of cardinality, the computations will increase, so the number of scan space becomes more and CPU time becomes longer. The performance of OSSRB is better than other algorithms. The algorithms in independent data set have better performance than that in anti-correlated data set.

### 6.2.3 Multiple queries results

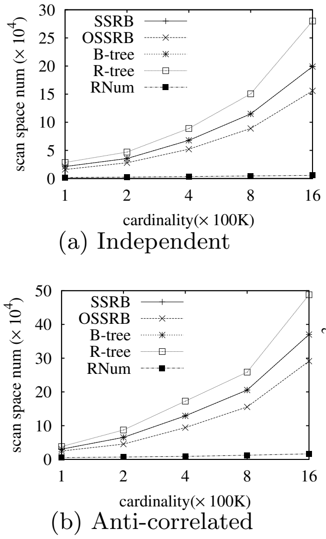
In this section, Figure 17 shows the total number of scan points for all the 1000 queries as a function of cardinality, for independent (17(a)) and anti-correlated (17(b)) data sets. Figure 18 illustrates a similar experiment that compares the algorithms in terms of CPU-time under the same settings. MSRB has better performance than OSSRB, the effect of $\varepsilon$ is small with the growth of cardinality.

# 7. CONCLUSIONS

Skyline plays an important role in many applications, including business planning, multi-criteria decision making and so on. In this paper, we present a variant of skyline query, subspace global skyline query, which can process the global skyline query with a query point in arbitrary subspace and have more extensive application prospect. To solve S-GS query, we firstly propose a new index RB-tree, which can rapidly find the initial scan positions. Secondly, we analyze the properties of SGS, and propose SSRB algorithm to compute single SGS points through one pass scan of RB-trees. On the basis of SSRB, we put forward two optimization strategies, and get OSSRB algorithm which has better performances in scan space and processing time. Then we propose a method to deal with multiple SGS queries, using the division scheduling strategy, we assign the similar queries to the same subset, then employ algorithm MSRB to solve the queries in subset through one pass scan. MSR-B use the shared scan space of similar queries and it can rapidly return the results of multiple SGS queries. Finally, a large number of simulation experiments verify the performance of our proposed algorithms, OSSRB and MSRB show stable performances and are better than other methods, so they are appropriate for practical applications.

# 8. ACKNOWLEDGEMENT

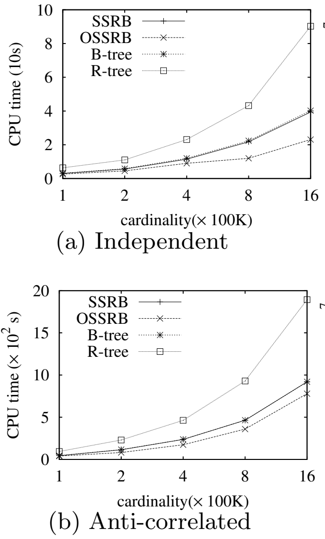
This research was partially supported by the National Natural Science Foundation of China under Grant No. 60933 001, 61025007, 61100022 and 61202087; the National Basic Research Program of China under Grant No. 2011CB302200-G; the 863 Program under Grant No. 2012AA011004, the Public Science and Technology Research Funds Projects of Ocean Grant No. 201105033, and the Fundamental Re-



(a) Independent

(b) Anti-correlated

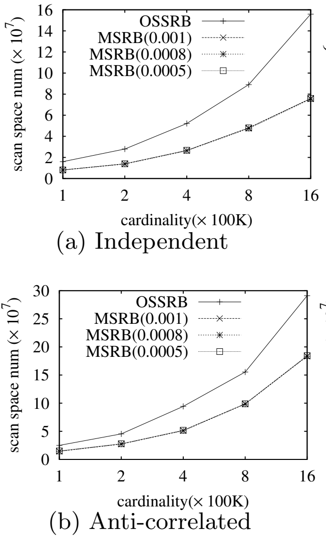
Figure 15: scan space number vs. $N(d = 5)$



(a) Independent

(b) Anti-correlated

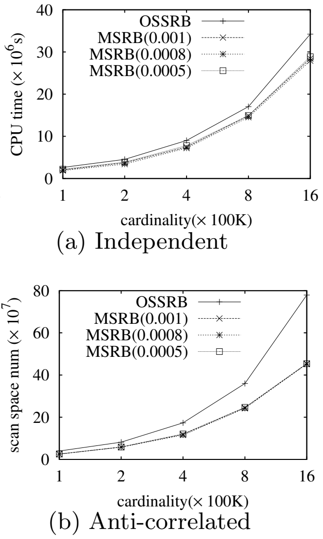
Figure 16: CPU-time vs. $N(d = 5)$



(a) Independent

(b) Anti-correlated

Figure 17: scan space number vs. $N(d = 5)$



(a) Independent

(b) Anti-correlated

Figure 18: CPU-time vs. $N(d = 5)$

search Funds for the Central Universities under Grant No. N110404009.

## 9. REFERENCES

[1] W.-T. Balke and U. Gĺńntzer. Multi-objective query processing for database systems. In *VLDB*, pages 936–947, 2004.

[2] W.-T. Balke, U. Gĺńntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT*, pages 256–273, 2004.

[3] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[4] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD Conference*, pages 503–514, 2006.

[5] L. Chen and X. Lian. Dynamic skyline queries in metric spaces. In *EDBT*, pages 333–343, 2008.

[6] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.

[7] E. Dellis and B. Seeger. Efficient computation of reverse skyline queries. In *VLDB*, pages 291–302, 2007.

[8] M. E. Khalefa, M. F. Mokbel, and J. J. Levandoski. Skyline query processing for uncertain data. In *CIKM*, pages 1293–1296, 2010.

[9] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[10] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. pages 469–476, 1975.

[11] K. C. K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in z order. In *VLDB*, pages 279–290, 2007.

[12] X. Lian and L. Chen. Efficient processing of probabilistic groupsubspace skyline queries in uncertain databases. In *Information Systems(2012), http://dx.doi.org/10.1016/j.is.2012.08.006*.

[13] X. Lian and L. Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD Conference*, pages 213–226, 2008.

[14] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.

[15] X. Lin, Y. Zhang, W. Zhang, and M. A. Cheema. Stochastic skyline operator. In *ICDE*, pages 721–732, 2011.

[16] B. Liu and C.-Y. Chan. Zinc: Efficient indexing for skyline computation. pages 197–207, 2010.

[17] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD Conference*, pages 467–478, 2003.

[18] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.

[19] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *VLDB*, pages 15–26, 2007.

[20] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB*, pages 253–264, 2005.

[21] K. L. Tan, P. K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.

[22] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-based representative skyline. In *ICDE*, pages 892–903, 2009.

[23] Y. Tao, X. Xiao, and J. Pei. Subsky: Efficient computation of skylines in subspaces. In *ICDE*, page 65, 2006.

[24] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB*, pages 241–252, 2005.