VMPSP: Efficient Skyline Computation Using VMP-Based Space Partitioning

Kaiqi Zhang[1(✉)], Donghua Yang[2], Hong Gao[1], Jianzhong Li[1], Hongzhi Wang[1], and Zhipeng Cai[3]

[1] School of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
{zhangkaiqi,honggao,lijzh,wangzh}@hit.edu.cn
[2] Academy of Fundamental and Interdisciplinary Sciences, Harbin Institute of Technology, Harbin, China
yang.dh@hit.edu.cn
[3] Department of Computer Science, Georgia State University, Atlanta, USA
zcai@gsu.edu

Abstract. The skyline query returns a set of interesting points that are not dominated by any other points in the multi-dimensional data sets. This query has already been considerably studied over last several years in preference analysis and multi-criteria decision making applications fields. Space partitioning, the best non-index framework, has been proposed and existing methods based on it do not consider the balance of partitioned subspaces. To overcome this limitation, we first develop a cost evaluation model of space partitioning in skyline computation, propose an efficient approach to compute the skyline set using balanced partitioning. We illustrate the importance of the balance in partitioning. Based on this, we propose a method to construct a balanced partitioning point VMP whose ith attribute value is the median value of all points in ith dimension. We also design a structure RST to reduce dominance tests among those subspaces which are comparable. The experimental evaluation indicates that our algorithm is faster at least several times than existing state-of-the-art algorithms.

1 Introduction

The skyline query [1] returns a set of interesting points that are not dominated by any other points in the multi-dimensional data sets. A point q is dominated by a point p if and only if p is not worse than q in all dimensions and strictly better in at least one dimension. Without loss of generality, here we assume that lower value is better to users in all dimensions.

The skyline query is important in database community and has already been considerably studied over last several years in preference analysis, multi-criteria decision making applications, and so on. It aims to reduce search space when there is not existing a scoring function.

H. Gao et al. (Eds.): DASFAA 2016 Workshops, LNCS 9645, pp. 179–193, 2016.
DOI: 10.1007/978-3-319-32055-7_16

The existing algorithms can be summarized into two categories, index-based algorithms and index-independent algorithms, respectively. Index-based algorithms [8–11] strive primarily to avoid scanning the entire data sets by utilizing pre-construct index structure. Some points are usually checked to be a skyline point or a non-skyline point by index structure early on. The main issues are that it needs a great deal of time and space to pre-construct the data points index structures and store them respectively and it can't deal with dynamic or stream data sets.

The other category is index-independent algorithms, which compute skyline results without any pre-construct structures. The early algorithms [2,6,7] are mainly based on sorting. They first sort the data sets by a monotone function to guarantee that previous points are never dominated by their following points. These approaches strive to find out early the skyline points which can prune out most of the non-skyline points. [3] first proposed the schema of space partitioning to compute skyline set, and implemented two algorithms which both outperform significantly than prior methods. This schema divides recursively the entire space into several disjoint subspaces and compute local skyline results in their subspace according to the relationship of these subspaces. We collect all the local skyline points as the results of skyline computation. It reduces dramatically the dominance tests. While it randomly selects a skyline point as partitioning point, this method cannot guarantee the stability. Based on the schema, BSkyTree [4] makes some progress by selecting the skyline point with the closest value to the main diagonal as the partitioning point. They both do not consider the importance of the balance in partitioning.

This paper focuses mainly on developing a cost evaluation model of space partitioning in skyline computation and designing an efficient approximated approach based on it. We illustrate the importance of the balance in partitioning and provide a low-cost balanced partitioning point VMP whose ith attribute value is the median value of all points in ith dimension. Finally, we utilize a structure named RST to reduce the dominance tests among comparable subspaces. We also implement an algorithm VMPSP using our VMP partitioning point and RST structure. Our experiments indicate that our algorithm VMPSP is faster at least several times than existing state-of-the-art algorithms.

In this paper, the key contributions are summarized as follows:

- We analyze the defections of existing sorting-based and space-partitioning-based algorithms.
- We develop a cost evaluation model of space partitioning in skyline computation, and illustrate the importance of balance of partitioned subspaces.
- We propose a new more balanced partitioning point Virtual Median Point (VMP) whose ith attribute value is the median value of all points in ith dimension. It can reduce the cost of partitioning. We also design a structure named Recursive Search Tree(RST) to reduce the dominance tests among comparable subspaces.
- We implement an algorithm VMPSP by constructing VMP as the partitioning point and utilizing RST structure.

– We evaluate our proposed algorithm VMPSP by comparing it with state-of-the-art algorithms in dimensionality and cardinality over real and synthetic data sets.

The rest of this paper is organized as follows. Section 2 introduces some key proposed algorithms based on sorting or space partitioning. Section 3 presents existing definitions and properties about skyline and space partitioning schema. Section 4 develops a cost evaluation model of space partitioning in skyline computation and illustrates the importance of balance in partitioning. A construction of balanced partitioning point VMP and a structure of reducing dominance tests among comparable subspaces RST are presented in this section. Section 5 illustrates our algorithm VMPSP using our constructed partitioning point VMP and the structure RST. Section 6 evaluates our proposed algorithm with existing approach in dimensionality and cardinality scalability for real and synthetic data sets. Finally, our conclusion is summarized in Sect. 7.

2 Related Work

The skyline query mainly resolves the problem that how to reduce the search space by discarding the non-interesting objects from multi-dimensional databases when the scoring function is not existing. It is important in multi-criteria decision making and analysis in database community. There are a great many of existing literatures in this field, while we just summarize sketchily most proposed key main-memory algorithms into two categories as follows.

2.1 Index-Based Algorithms

Index-based skyline computation techniques utilize the pre-construct indexes on all data sets to accelerate the skyline query by avoiding scanning the entire data sets. [1] first roughly imagines the algorithms utilizing B-tree or R-tree index structures which can prune out non-skyline points immediately without accessing the entire data sets. Since then a set of techniques [8–11] by using indexes are proposed to improve the efficiency of the skyline computation. [9] exploits bitmap to represent data sets and perform a series simple of bitwise operations to get skyline results. By the observation which the nearest neighbor point to the origin must be a skyline point, [8] develops NN algorithm. NN first computes the nearest neighbor point to the origin which must be a skyline point from the entire data sets by utilizing R-tree. Then divides remaining points except those dominated by the nearest neighbor point into several overlapping subsets. Next, keeps on computing NNP from these subsets recursively. Finally it is necessary to eliminate duplicate skylines from these NNP. Based on NN, an improvement version named BBS is presented in [11]. It adopts branch-and-bound strategy and avoids retrieving duplicates to improve performance. The index-based state-of-the-art algorithm is ZSearch [10]. It improves significantly the overall performance according to ZBtree in which data points is organized.

Obviously, the disadvantage of index-based algorithms is need a great deal of time and space to pre-construct the data points index structures and store them respectively. When the dimension is high, it is unfeasible since the time cost of preconstruction is dramatically more than the skyline query response time using indexes. Also, it is unfeasible to construct indexes for stream or dynamic data sets. So it is urgent to find out a method that efficiently computes the skyline results without preconstruction.

2.2 Index-Independent Algorithms

The other type of skyline computation is to retrieve the skyline results straightly without any pre-prepare structures. The first algorithm is proposed along with the skyline problem named block-nested-loops(BNL) in [1]. It uses a memory buffer *window* to contain the *candidate* skyline points which have been yet not dominated by others. The point accessed by the stored order in the data sets must compare with *candidates* one by one until it is dominated. If it is dominated by any one of the *candidates*, discard the point and validate the next point in the data sets. Otherwise insert the point into the *window* and remove those *candidates* which are dominated by it. The *candidates* in the *window* are definitely all skyline points until the data sets in store have been all accessed. BNL is much inefficient since the *candidates* in the buffer may be not the skyline points, conducts a great extremely many of unnecessary dominance tests among non-skyline points in the *window*.

Based on this, Sort-First Skyline(SFS) algorithm [2] is presented. It guarantees that the points added into the *window* are certainly skyline points by sorting the data sets using a monotone function(such as *entropy function* and *sum function*). In the sorted data sets, a point must certainly not dominate its previous points. So, we are sure that the *candidates* in the *window* must be the skyline points of the data sets. In the light of this, SFS algorithm accelerates significantly the query response time to BNL. Later, Some important approaches, such as LESS [7] and SaLSa [6], have been developed to improve the performance of SFS based on the framework of sorting the data sets first. LESS conducts the elimination by maintaining a small memory buffer when the points are sorting. SaLSa selects monotone functions carefully and adopts a stop point to terminate the algorithm early such that not all points in the sorted list need to be accessed.

Although these methods, especially SaLSa, have a better performance, the greatest disadvantage of above sorting-based algorithms is sensitive to the size of skyline set since every accessed point must compare with all *candidates* until it is dominated. The skyline size becomes larger as the cardinality or dimensionality higher such that the performance of above algorithms deteriorates severely. The total numbers of dominance tests are unimaginable huge, as well as the cost of sorting is not negligible when data sets are large.

In order to overcome the aforementioned problems, a crucial algorithm OSPS is devised in [3] which first proposes a new framework named *space partitioning* to organize already found skyline points as a skyline tree. Any accessed point need not compare with all the found skylines instead do the dominance tests

with only a small part of skylines in ROSP even if the point belongs to skyline. This framework is also effective to other studies [5,12] which are related with skyline. [3] proposes a method that selects randomly a skyline point as the *partitioning point*. Although avoid the occurrence of the worst case like SFS, it is failed to guarantee the stability. Also the random partitioning point cannot partition the space into balanced subspaces. BSkyTree [4] both consider dominance and incomparability relationships which reduces substantially the numbers of dominance tests and have become state-of-the-art algorithm of the skyline computation in main-memory environment. It selects the skyline point with the closest value to the main diagonal and makes the subpartitions more balanced than the random selection method. While only the first partitioned subspaces are relatively balanced, the subsequent partitioning conduct severely unbalanced situation. Besides, BSkyTree cannot organize yet found skylines as a recursive search tree to accelerate dominance tests.

3 Preliminaries and Observation

In this section, we introduce some definitions and properties which are all given in prior works [1,3,4] as well as present our observations that can be used to improve the overall performance of skyline computation. Given a data set S in a d-dimensional positive real space R^d, for any point p in S, we denote the value of point p in ith dimension by p_i.

3.1 Skyline

Definition 1. (Dominate) *Given two distinct points p and q in the data set S, p dominates q, denoted by $p \succ q$ iff $\forall\ i \in [1,d]$, $p_i \leq q_i$ and $\exists i \in [1,d]$, $p_i < q_i$. Otherwise, p doesn't dominate q, denoted by $p \not\succ q$. We call point p as the first check point and q as the second check point from here on.*

Definition 2. (Skyline) *For any point p in S, p is a skyline point iff $\nexists\ q \in S$ and $q \neq p$, such that $q \succ p$. The skyline set consists of all skyline points.*

Definition 3. (Incomparable) *Given any two points $p, q \in S$, if $p \not\succ q$ and $q \not\succ p$, then p and q are incomparable each other, denoted by $p \sim q$.*

3.2 The Framework of Space Partitioning

Before the framework of space partitioning is proposed, most prior algorithms focus heavily on *point-wise* dominance tests as well as how to fast check out whether a point is dominated or not. We can know nothing but whether the *second check point* is dominated by the *first check point* from one *point-wise* dominance test. If not, we must repeat the same operation for the next point. While the schema of space partitioning can provide more information even if the *second check point* is not dominated by the *first check point*. And it promotes *point-wise* dominance tests to *space-wise* dominance tests which reduces significantly dominance tests among incomparable point pairs.

Definition 4. (Partitioning Point *[3]) Given a point* $\widehat{p}$ *in* R^d, $\forall\ i \in [1,d]$, $\widehat{p}_i$ *partitions* R^d *into 2 disjoint complementary subspaces* $S_i^+(\widehat{p})$ *and* $S_i^-(\widehat{p})$ *respectively.* $\forall\ p \in S$, *if* $p_i < \widehat{p}_i$, $p \in S_i^+(\widehat{p})$, *otherwise,* $p \in S_i^-(\widehat{p})$. *Therefore, the space* R^d *is partitioned into* 2^d *non-overlapping subspaces by* $\widehat{p}$ *which is formally named as partitioning point. We define a subspace set* $C_{\widehat{p}}^{R^d}$ *consists all of aforementioned* 2^d *subspaces partitioned by* $\widehat{p}$ *in* R^d. *Any point in* S *except* $\widehat{p}$ *is mapped into unique one space of* $C_{\widehat{p}}^{R^d}$. *We name a d-bit vector as the address of any subspace* V, *denoted by* $A_{\widehat{p}}^V$, $\forall\ i \in [1,d]$, *if* $V \subset S_i^+(\widehat{p})$, *set the ith bit value of* $A_{\widehat{p}}^V$ *be 0, i.e.,* $A_{\widehat{p}}^V[i]$*=0; Otherwise* $A_{\widehat{p}}^V[i]$*=1.*

Based on Definition 4, any *partitioning point* in R^d can divide S into 2^d separate subsets corresponding to 2^d disjoint subspaces of R^d of which the *addresses* range from 0 to 2^d-1. We can also represent a subspace by its *address.*

Definition 5. (Space-wise Dominate *[3]) For any two subspaces* $V, W \in C_{\widehat{p}}^{R^d}$, *if* $\forall\ i \in [1,d]$, $A_{\widehat{p}}^V[i] \leqslant A_{\widehat{p}}^W[i]$ *then* V *space-wise dominates* W, *for brevity of representation, replace it with dominate, denoted by* $V \succ W$. *Otherwise,* V *cannot dominate* W, *denoted by* $V \not\succ W$.

Definition 6. (Incomparable *[3] and Comparable)For any two subspaces* V, $W \in C_{\widehat{p}}^{R^d}$, *if* V *and* W *cannot dominate each other, we call that they are incomparable, denoted by* $V \sim W$. *Otherwise,* W *and* V *are comparable. Specifically, subspace* V *is comparable with itself. The space set of all spaces dominating* V *is named as the dominating space set, denoted by* $D_{\widehat{p}}^V = \{W \in C_{\widehat{p}}^{R^d} \mid W \succ V\}$, *the space set of all spaces dominated by* V *is named as the dominated space set, denoted by* $U_{\widehat{p}}^V = \{W \in C_{\widehat{p}}^{R^d} \mid V \succ W\}$, *and the space set of all spaces being incomparable with* V *is named as the incomparable space set, denoted by* $I_{\widehat{p}}^V = \{W \in C_{\widehat{p}}^{R^d} \mid V \sim W\}$.

For the definitions of space-wise dominate and incomparable, there are a few differences with prior works. We describe some key properties about space partitioning in the following.

Property 1. For any two different subspace $V, W \in C_{\widehat{p}}^{R^d}$, if V and W are incomparable, then $\forall\ v \in V$, $\forall\ w \in W$, v and w must be incomparable, here v and w are both points.

Property 2. For any two different subspace $V, W \in C_{\widehat{p}}^{R^d}$. $V \succ W$ iff $(A_{\widehat{p}}^V \mid A_{\widehat{p}}^W) = A_{\widehat{p}}^W$.

Property 3. For any two different subspace $V, W \in C_{\widehat{p}}^{R^d}$. $V \sim W$ iff $(A_{\widehat{p}}^V \mid A_{\widehat{p}}^W) \neq A_{\widehat{p}}^V$ and $(A_{\widehat{p}}^V \mid A_{\widehat{p}}^W) \neq A_{\widehat{p}}^W$.

We ignore detailed process of proof about above properties, which have already been proofed in [3]. Property 1 can promote *point-wise* dominance tests

to *space-wise* dominance tests. We need first check out whether space V and W are incomparable or not, if it is, all dominance tests between the points in V and the points in W can be safely skipped through only one space-wise dominance tests. If not, *i.e.*, they are comparable with each other, then must do point-wise dominance tests between the points in the two spaces. Based on this, space partitioning can significantly reduce dominance tests among incomparable point pairs. Specifically, if the space $V_{00\ldots0}$ is not empty, then all the points in space $V_{11\ldots1}$ can be discarded immediately since they must be dominated by any point in space $V_{00\ldots0}$. Given a space V belonging to $C_{\widehat{p}}^{R^d}$, and any other space except V in $C_{\widehat{p}}^{R^d}$ can be summarized into 3 types, $D_{\widehat{p}}^V$, $U_{\widehat{p}}^V$ and $I_{\widehat{p}}^V$, respectively. For any point p in V need to be checked to judge whether it is a skyline point, obviously, those points in $U_{\widehat{p}}^V$ can be immediately ignored since they can certainly not dominate p according to partitioning definitions. Also, above properties imply that it is also correct to skip the dominance tests with all the points in $I_{\widehat{p}}^V$. We combine $U_{\widehat{p}}^V$ and $I_{\widehat{p}}^V$ as the space set which can be ignored dominance tests of space V, denoted by IT_V=$\{W \in C_{\widehat{p}}^{R^d} \mid W \in U_{\widehat{p}}^V$ or $W \in I_{\widehat{p}}^V\}$. While it is necessary to compare with the points in $D_{\widehat{p}}^V$ and the other points in itself space V.

We combine $D_{\widehat{p}}^V$ and space V as the space set which cannot be ignored dominance tests of space V, denoted by NIT_V=$\{W \in C_{\widehat{p}}^{R^d} \mid W \in D_{\widehat{p}}^V$ or $W = V\}$. It is easy to imply that if space V and W are comparable with each other, then $V \in NIT_W$ or $W \in NIT_V$. Here, we can claim that any accessing point p in V, the points in IT_V is safely skipped based on above analysis. While it is necessary to check out with the points in NIT_V.

4 Constructing Virtual Median Point as Partitioning Point

This section introduces the importance of partitioning points and illustrates the schema of space partitioning is a kind of typical divide and conquer algorithm that sub-problems are non-independent. To simplify the complexity of selecting optimized partitioning points, we first illuminate the importance of balance in partitioning, in addition provide a method to construct balanced partitioning points. Then, we adopt a structure name recursive search tree to reduce dominance tests among those subspaces which are comparable.

4.1 The Cost Evaluation Model of Space Partitioning in Skyline Computation

The schema of space partitioning needs to divide recursively current space into several disjoint subspaces until their sizes are small enough, and compute the local skyline points in all subspaces. Finally, all local skyline points are combined as the global skyline set, *i.e.*, the results of skyline computation. The schema

is a typical kind of divide and conquer algorithm that sub-problems are non-independent. Now we design a cost evaluation model of space partitioning in skyline computation as follows:

$$Cost(\mathcal{A}) = C_p(A) + C_c(A)$$

$\mathcal{A}$ is any algorithm of utilizing the schema of space partitioning, $C_p(A)$ and $C_c(A)$ represent the cost in partitioning the spaces and in dealing with comparable subspaces, respectively. And $C_c(A)$ relies primarily on the method of partitioning.

Theorem 1. *Using divide and conquer method to solve a problem, if it is divided into several independent disjoint sub-problems and the cost of dividing and conquering is proportional to the size of data set. Then the optimized dividing method is that all the sub-problems have the same size.*

Proof. For any partitioning method, suppose that the problem can be divided into k sub-problems, the time complexity is $T(kn) = T(n + \alpha_1) + T(n + \alpha_2) + \cdots + T(n + \alpha_k) + kn$ and $\forall\ i \in [1, k]$, $\alpha_i \in [-n, n]$ as well as $\Sigma_{i=1}^{k} \alpha_i = 0$. The cost of every iteration is $T(kn)$, the optimized partitioning has the least iteration times. Thus it is easy to conclude that the times of partitioning is minimum when $\alpha_1 = \alpha_2 = \cdots = \alpha_i = 0$. □

In general, the lowest cost of partitioning the points is proportional to the size of data set because all points must compare with partitioning point at least once even though ignore the cost of determining partitioning point. Based on aforementioned analysis, the optimized space partitioning approach makes that all subspaces have the same size and can compute their local skylines independently. While this partitioning is certainly not existing because there must be comparable partitioned subspaces.

Considering the complexity of this cost evaluation model, we propose a approximated method. We separate the process of computing skyline using space partitioning schema into two phases. Firstly, we maximize the balance of subspaces, *i.e.* the difference of subspaces' size is as small as possible. And then we strive to weak the influence among subspaces with comparable relationship, in other word, reduce the times of dominance tests among points in those subspaces which are comparable.

4.2 Virtual Median Point

The balanced degree of partitioned subspaces depends solely on the position of partitioning point. We now discuss where the partitioning point is can maximize the balance of subspaces.

Theorem 2. *Given a data set S, suppose that all points are distributed uniformly and independently [2, 7] in d-dimensional $[0, m]^d$ space, the point p whose every attribute value is $\frac{m}{2}$ can maximize the balance of partitioned 2^d subspaces.*

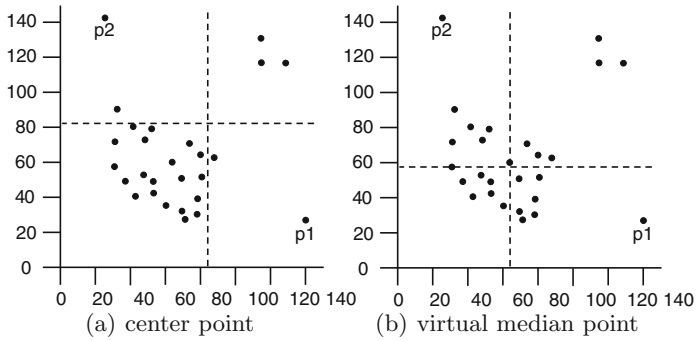


Fig. 1. Selecting partitioning point

Proof. The partitioning point p divides S into 2^d subspaces. Under uniform and independent condition, they have the same number of points since p is the center point of $[0, m]^d$ space. □

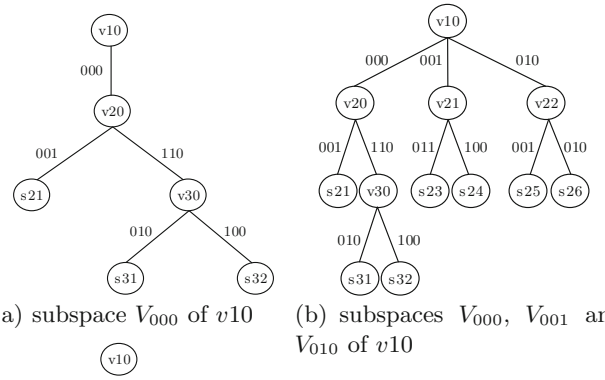
Based on Theorem 2, the center point of space can maximize the balance of subspaces under uniform and independent condition. While in some real environment, it is difficult to get exact distributed region, for example in Fig. 1(a), $p1(140,25)$ and $p2(28,141)$ decide the boundary, if we regard straightly the data set as a uniform and independent distribution, the center of the boundary is $\alpha(84,83)$ and all points are partitioned in a extremely unbalanced method. To solve this problem, We construct the *Virtual Median Point*(VMP) as the partitioning point. The ith attribute value of VMP is the median value of all points in ith dimension. The VMP is (65,57) in Fig. 1(b) and the partitioned subspaces are maximum balance. Besides, VMP is infinitely close to the center point when data set is distributed uniformly and independently.

4.3 Recursive Search Tree

The computations in subspaces based on the framework of space partitioning are non-independent as we have stated before. And we only need to consider the influence among subspaces which are comparable. Now we discuss how to reduce the dominance tests among comparable subspaces as much as possible.

As aforementioned descriptions, for two comparable subspaces V and W, it is either $V \in NIT_W$ or $W \in NIT_V$. So we aim to reduce the dominance tests between any partitioned subspace V and these subspaces which are in NIT_V.

The naive method of judging whether a point p in V is dominated by any point in space set NIT_V such that eliminating p is to do the dominance tests sequentially with p one by one until the occurrence of point q which can dominate p [4]. This method is extremely expensive since p will compare with all points in space set NIT_V if p is a local skyline point. In order to raise the efficiency of NIT tests of p, we introduce a structure recursive search tree(RST) which is similar to *skyline tree* in [3]. The root and inner nodes of RST are the virtual



(a) subspace V_{000} of $v10$ (b) subspaces V_{000}, V_{001} and V_{010} of $v10$

Fig. 2. An example of recursive search tree

median points constructed by respective subspaces. And the leaf nodes are local skyline sets of their lying subspaces. The partitioning is recursively conducted, it is necessary to terminate it when space size is less than or equal to a threshold β.

Now we describe the detailed process of maintaining the RST by an example. As well as discuss how to reduce the compared times using it. Given a 3D data set S which contains 32 points and set threshold $\beta = 2$. We construct the first level VMP $v10$ and these points are partitioned into 8 subspaces, V_{000}:{$p1$, $p2$, $p3$, $p4$}, V_{001}:{$p5$, $p6$, $p7$, $p8$}, V_{010}:{$p9$, $p10$, $p11$, $p12$}, V_{011}:{$p13$, $p14$, $p15$, $p16$}, V_{100}:{$p17$, $p18$, $p19$, $p20$}, V_{101}:{$p21$, $p22$, $p23$, $p24$}, V_{110}:{$p25$, $p26$, $p27$, $p28$} and V_{111}:{$p29$, $p30$, $p31$, $p32$}. Then we recursively partition the new produced 8 subspaces in depth-first by their address values(That is the address value presents that $|V_{000}|$=0, $|V_{001}|$=1, $|V_{010}|$=2, and so on. The sorted order is V_{000}, V_{001}, V_{010}, V_{011}, V_{100}, V_{101}, V_{110}, and V_{111}). We recursively divide the subspace V_{000} first until the sizes of partitioned subspaces are all not more than β. The result of partitioning space V_{000} is shown in Fig. 2(a), $v20$ is the VMP of subset {$p1$, $p2$, $p3$, $p4$}. Suppose that $s21$={$p1$}, $s31$={$p2$}, $s32$={$p3$, $p4$}, terminate the partitioning of $s1$ since $1 < \beta$. Then we continue to partition the subspace V_{110} of $v20$, $v30$ is constructed and the other 3 points are put into subspace V_{010} and V_{100} of $v30$. Here, V_{010} and V_{100} of $v30$ both satisfy threshold β. Then we partition the subspace V_{001} of $v10$.

The aforementioned descriptions mainly introduce the process of partitioning. Now we discuss how to reduce the dominance tests using RST. The status of RST now is reported in Fig. 2(b), and suppose that $s21$={$p1$}, $s31$={$p2$}, $s32$={$p3$, $p4$}, $s23$={$p5$, $p6$}, $s24$={$p7$, $p8$}, $s25$={$p9$, $p10$}, $s26$={$p11$, $p12$}.We start to partition the subspace V_{011} of $v10$. Before partition these points, we first eliminate those which are dominate by some points in space set $NIT_{V_{011}}$, *i.e.*, subspaces V_{000}, V_{001}, and V_{010} of $v10$. We suppose p is lied in V_{011} of $v10$, now we check whether p is dominated. First, we compare p with $v20$, suppose that p lies in subspace V_{011} of $v20$. Then p only needs to compare with point $p1$ in $s21$ and skips all the points in subspace V_{110} of $v20$. If p is not dominate by $p1$,

we continue this tests in subspace V_{001} and V_{010} of $v10$. Suppose that p lies in subspace V_{010} of $v21$ and in V_{001} of $v22$, p only needs to compare with points in $s25$ and save 6 times dominance tests with points in $s23$, $s24$ and $s26$. If p is not dominated by points in $s25$, keep it and continue this tests for next point in subspace of V_{011} of $v10$. Otherwise, drop p immediately. Finally, we collect all the leaf nodes as the results of skyline computation.

Based on above introduction about RST, it is easy to observe that the more balanced the partitioned subspaces, the less average dominance tests the RST conducts.

5 Algorithms

This section proposes our algorithm VMPSP which adopts constructed virtual median point as the *partitioning point* and utilizes a RST structure to reduce the dominance tests among these subspaces which are comparable. VMPSP strives to keep the balance of partitioned subspaces which can reduce the cost of partitioning and the cost of comparisons in RST. VMPSP algorithm is given in Algorithm 1.

Algorithm 1. VMPSP(S)

Input: A d-dimensional positive numerical data set S
Output: The RST of S

```
1: create a recursive search tree node RST.
2: p̂ ← ConstructVirtualMedianPoint(S).
3: max ← 2^d − 1
4: create Space Subsets {S_0, ··· ,S_max} as well as all are empty.
5: RST.value ← p̂. // Assign partitioning point p̂ to current RST node
6: for ∀p ∈ S do
7:    i ← Compare(p, p̂). // i represents the address of p locating subspace w.r.t. p̂
8:    Add p into S_i. // S_i contains the ith space subset's points.
9: for i ← 0 to max do
10:   if |S_i| > 0 then
11:      for ∀j < i, (j | i) = i do
12:         for ∀p ∈ S_i do
13:            flag ← CheckByRST(p,RST[j]).
14:            if flag == true then
15:               S_i.discard(p)
16:      if |S_i| > β then
17:         T_i ← VMPSP(S_i). // Recursively conduct partition.
18:         RST[i].add(T_i). // Add T_i as the RST's ith subtree.
19:      else if |S_i| > 0 then
20:         localSkyline ← BNL(S_i).
21:         RST[i].skyline ← localSkyline.
22: return RST
```

Algorithm 1 depicts the pseudo code of VMPSP. We first construct a virtual median point $\widehat{p}$ as partitioning point by **ConstructVirtualMedianPoint** function as shown in line 2. And then in lines 3-8 partition all data points into 2^d subsets $\{S_0, \cdots, S_{max}\}$ corresponding to 2^d subspaces $\{V_0, \cdots, V_{max}\}$, and the subscript i of S_i represents the address of subpartitions. **Compare** function in line 7 computes the subspace in which p lies w.r.t $\widehat{p}$. For any non-empty subset S_i belonging to V_i, we can immediately skip subsets in IT_{V_i} and solely check remaining subsets in NIT_{V_i} as described in line 11, all subspaces V_j are in NIT_{V_i} if $(j \mid i) = i$. Next, for the dominance tests in NIT_{V_i}, we only check a small part of them using RST maintained by yet found local skyline sets and constructed virtual median points and eliminate dominated points in S_i. After above checking, if the remaining points in S_i is still larger than threshold β, then recursively partition S_i until it is not more than β. Otherwise, we could compute the local skyline sets for the remaining points using basic method BNL [1], it is not inefficient since the threshold β is small. We create a RST at the beginning of the algorithm, the root and inner nodes are the virtual median points constructed by respective subspaces and the leaf nodes are local skyline points of their lying subspaces.

Algorithm 2. CheckByRST(p,RST)

Input: A d-dimensional positive numerical data set S
Output: Whether point p is dominated by any point in recursive search tree RST or not

```
1: if RST.isLeaf() then
2:    if DOMINATE(p,RST.skyline) then
3:       return ture
4: else
5:    i ← Compare(p,RST)
6:    for ∀T ∈ RST.child, (T.address | i) = i do
7:       if CheckByRST(p,T) then
8:          return true
9: return false
```

RST is used to reduce the dominance tests among these subspaces which are comparable as described in Algorithm 2. First, we need to judge whether current RST node is a leaf. If it is true, check p with local skyline set in it. Otherwise, compare p with it and recursively recall function until check out whether p is dominated by some points. Finally return the result of tests.

6 Experiments

In this section, we evaluate the performance of our proposed algorithm VMPSP in dimensionality and cardinality in detail, by comparing it with state-of-the-art main-memory algorithms SFS [2], OSPS [3], and BSkyTree [4] using both synthetic and real data sets.

6.1 Experimental Settings

We generate two types of synthetic data sets that are Independent(IND) and Anti-correlated(ANT) according to the instructions in [1], respectively. The synthetic data sets dimensionality ranges from 6 to 16 and the data sets cardinality ranges from 200 K to 1M. All attributes values are positive real number in [0,1000]. Specifically, we claim here that the default dimensionality and cardinality are 12 and 200K, respectively. We also collect two real data sets ColorMoments[1] and IPUMS[1]. ColorMoments has 68404 tuples with 9 attributes. It describes the image features extracted from an image collection. IPUMS has 74954 tuples with 23 attributes. It describes unweighted PUMS census data from the Los Angeles and Long Beach areas for the year 1980. All algorithms are implemented by C++ languages and run on Intel Core-Q8300 CPU at 2.5GHz, with 4GB of RAM. Assuming that memory can contain all the origin data sets.

We compare our algorithm VMPSP with three state-of-the-art main-memory algorithms SFS [2], OSPS [3] and BSkyTree [4]. The monotone function of SFS adopts the *sum function* that the sum of all the attributes values. OSPS has two versions and we adopt the better OSPSOnPartitioningFirst method as OSPS and BSkyTree has two versions and we adopt the better BSkyTree-P method as BSkyTree here. We propose a complete algorithm of skyline computation using VMP named VMPSP which also utilizes our designed structure RST. It is better than state-of-the-art algorithms by extensive experiments.

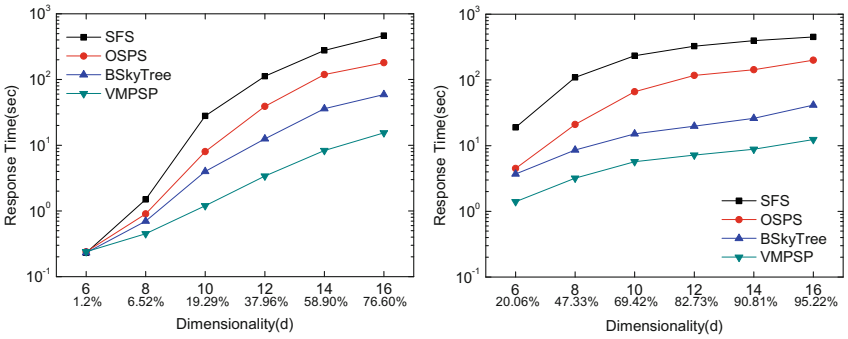


Fig. 3. Performance over dimensionality variation

6.2 Scalability

Varing Dimensionality. This section describes the performance of above algorithms with the dimensionality variation. Figure 3 reports the response time

[1] The data set is collected from https://kdd.ics.uci.edu.

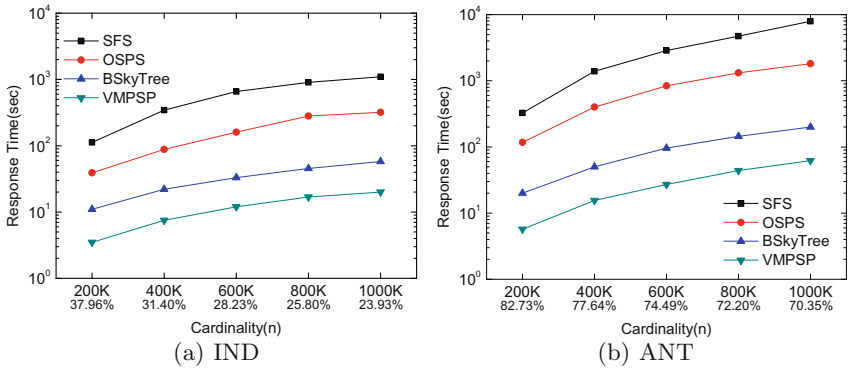


Fig. 4. Performance over cardinality variation

of the four algorithms. The two types distributions data sets that IND and ANT with dimension from 6 to 16 and 200k cardinality are used to conduct the experiments. VMPSP outperforms the other three algorithms in every dimensionality. Experiments validate space-based algorithms is better than sort-based algorithms and our algorithm is the best in existing space-based algorithms. Specifically, VMPSP is about faster 3 to 5 times over state-of-the-art space-partitioning algorithm BSkyTree, and outperforms SFS by up to one to two orders of magnitude.

Varing Cardinality. This section mainly describes the performance of algorithms with cardinality variation. We also generate two types data sets like above with cardinality from 200 K to 1M and 12 dimensionality. As Fig. 4 reports, our proposed algorithm VMPSP is always better than others.

6.3 The Performance on Real Data Sets

This section describes the performance of all algorithms on real data sets ColorMoments and IPUMS. Table 1 presents the response time which it takes to execute all algorithms in ColorMoments. We observe that our algorithm VMPSP is the best algorithm. The ColorMoments has 68040 tuples, 9 attributes, and 1533 of them are skyline results.

Table 1. Performance on real data set.

Algorithm	ColorMoments	IPUMS
	n=68,040; d=9	n= 74,954; d=23
	skyline=1533; 2.25s%	skyline=23190; 30.94 %
SFS	0.17s	35.17s
OSP	0.16s	1.92s
BSkyTree	0.15s	1.37s
VMPSP	0.08s	0.90s

7 Conclusion

In this paper, we illustrated the development of existing non-index main-memory algorithms of skyline computation. Compared two popular framework of being based on sorting and based on space partitioning, respectively. And concluded that space-partitioning-based schema is better. Then, the defections of existing algorithms based on it were introduced which they do not consider the importance of balance in partitioning. To overcome this limitation, we studied the process of partitioning and develop a cost evaluation model of space partitioning in skyline computation. Also, we analyzed the importance of balance in partitioning and proposed an approach to construct the balanced partitioning point VMP. In addition, a structure RST was designed to accelerate dominance tests among comparable subspaces. Finally, the evaluation indicates that our algorithm is faster several times than existing state-of-the-art algorithms.

References

1. Borzsony, S., Kossmann, D., Stocker, K.: The skyline operator. In: ICDE (2001)
2. Chomicki, J., Godfrey, P., Gryz, J., et al.: Skyline with presorting. In: ICDE (2003)
3. Zhang, S., Mamoulis, N., Cheung, D.W.: Scalable skyline computation using object-based space partitioning. In: SIGMOD (2009)
4. Lee, J., Hwang, S.: BSkyTree: Scalable skyline computation using a balanced pivot selection. In: EDBT (2010)
5. Bøgh, K.S., Chester, S.: Assent I. Work-efficient parallel skyline computation for the GPU. In: VLDB (2015)
6. Bartolini, I., Ciaccia, P., Patella, M.: Efficient sort-based skyline evaluation. ACM TODS **33**(4), 1–45 (2008)
7. Godfrey, P., Shipley, R., Gryz, J.: Maximal vector computation in large data sets. In: VLDB (2005)
8. Kossmann, D., Ramsak, F., Rost, S.: Shooting stars in the sky: An online algorithm for skyline queries. In: VLDB (2002)
9. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: VLDB (2001)
10. Lee, K.C.K., Zheng, B., Li, H., et al.: Approaching the skyline in Z order. In: VLDB (2007)
11. Papadias, D., Tao, Y., Fu, G., et al.: Progressive skyline computation in database systems. ACM TODS **30**(1), 41–82 (2005)
12. Lee, J., Hwang, S.: QSkycube: Efficient skycube computation using point-based space partitioning. In: VLDB (2010)