Arab J Sci Eng (2016) 41:2801–2811
DOI 10.1007/s13369-015-2011-4

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Efficient Processing of the Skyline-CL Query

Zhenhua Huang[1] · **Juan Zhang**[2,3] · **Chunqi Tian**[1]

Received: 28 March 2015 / Accepted: 10 December 2015 / Published online: 24 December 2015

Abstract Given a set of k-dimensional objects, the skyline-CL query returns all clusters over skyline objects according to their cardinalities. A naïve solution to this problem can be implemented in two phases: (1) using existing skyline query algorithms to obtain all skyline objects and (2) utilizing the DBSCAN algorithm to cluster these skyline objects. However, it is extremely inefficient in real applications because phases 1 and 2 are all CPU-sensitive. Motivated by the above facts, in this paper, we present Algorithm for Efficient Processing of the Skyline-CL Query (AEPSQ), an efficient sound and complete algorithm for returning all skyline clusters. During the process of obtaining skyline objects, the AEPSQ algorithm organizes these objects as a novel k-ary tree $SI^{(k)}$-*Tree* which is first proposed in our paper, and employs several interesting properties of $SI^{(k)}$-*Tree* to produce skyline clusters fast. Furthermore, we present detailed theoretical analyses and extensive experiments that demonstrate our algorithm is both efficient and effective.

Keywords Skyline · Cluster processing · DBSCAN · Performance evaluation

✉ Zhenhua Huang
huangzhenhua@tongji.edu.cn

[1] Department of Computer Science, Tongji University, Shanghai 200092, China

[2] Open Fund of Robot Technology Used for Special Environment Key Laboratory of Sichuan, Mianyang 621010, Sichuan, China

[3] School of Information Engineering, Southwest University of Science and Technology, Mianyang 621010, Sichuan, China

1 Introduction

The skyline query technology has attracted much attention recently. This is mainly due to the importance of skyline results in many applications, such as multi-criteria decision making, data mining, and information recommender systems [1,2]. A skyline query over k dimensions selects the objects that are not dominated by any other objects restricted to those dimensions. However, in most real applications, the size of skyline result with k dimensions and $\wp$ input objects will exceed $(\ln \wp)^{k-1}/(k-1)!$ where "ln" and "!" denote the natural logarithm and the factorial, respectively [3]. It is not difficult to see that the number of skylines will grow exponentially as $\wp$ and k increase. Consequently, the result returned by the existing skyline query algorithms [4–12] is very huge and the users hardly efficiently analyze the whole skyline set.

In many real applications, we notice that the input objects are usually clustered, and hence in this paper, we present a novel extension of skyline, i.e., skyline-CL (skyline cluster), to produce similar objects within the skyline result. And in each skyline cluster, the users can locate their interested objects. Compared with skyline, skyline cluster has at least three advantages: (1) The interested optimal objects can be more quickly found within each cluster whose cardinality is smaller than that of the whole skyline set. (2) It can avoid wasting of much time in searching the similar optimal objects within the whole skyline set. (3) The interested optimal objects can tend to be as diversified as possible.

Motivation example (milk products selection) Assume there exists a milk marketing system which has 150 product items. Each item includes two attributes: price and expiration date. A milk product is better if its price is lower, and its expiration date is longer. Figure 1 shows this application scenario, in which there exists 42 skyline items represented by solid points. The user Jukie wants to buy some milk products

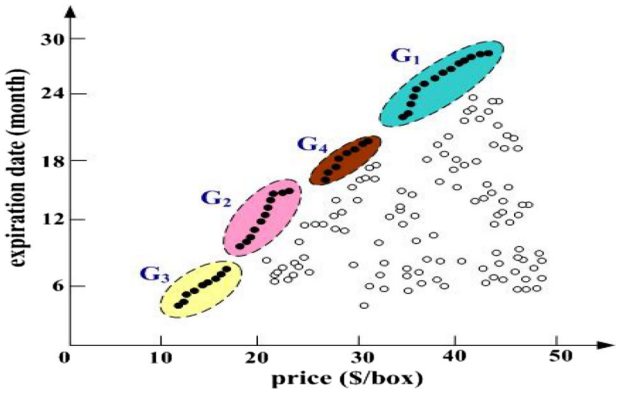

Fig. 1 Example for milk product selection

within these skyline items, but he hardly efficiently analyzes and locates his interested milk products. As an alternative in this application scenario, 42 skyline items can be clustered into four groups G_1-G_4, based on their two attributes, and then Jukie can locate the optimal milk products in each group.

Motivated by the above facts, in this paper, we integrate the density-based clustering methods [13,14] into the skyline query for efficient mining of skyline clusters. A straightforward solution consists of three key phases: (i) obtain the skylines from input objects, using any known skyline query approaches, such as the algorithms MR-GSC [9] and CDCA [10]; (ii) utilize the DBSCAN algorithm [13] to cluster these skyline objects; and (iii) Return all skyline clusters according to their cardinalities. However, it is inefficient because phases 1 and 2 are all CPU-sensitive, and their time complexities are: $\Omega(k\wp \lg \wp + (k-1)^{k-3}\wp)$ and $O\left(k\left(\frac{(\ln \wp)^{k-1}}{(k-1)!}\right)^2\right)$, respectively, where "Ω" denotes the asymptotic lower bound in time complexity, and "lg" denotes the denary logarithm [3]. In this paper, we present Algorithm for Efficient Processing of the Skyline-CL Query (AEPSQ), an efficient sound and complete algorithm for implementing the skyline-CL query. During the process of obtaining skyline objects, the AEPSQ algorithm organizes these objects as a novel k-ary tree $SI^{(k)}$-*Tree* which is proposed in our paper, and employs several interesting properties of $SI^{(k)}$-*Tree* to produce skyline clusters fast.

In summary, this paper makes the following contributions:

- We propose the concept of skyline-CL, which clusters the skyline objects and makes users zoom into different groups of skyline objects. We also propose to extend SQL with a SKYLINE-CL OF keyword, which can efficiently enhance the query engine functions of RDBMS.
- We investigate efficient implementation of the skyline-CL query. And an efficient approach (i.e., AEPSQ) is developed. It is based on the novel index tree $SI^{(k)}$-*Tree* which is proposed in our paper, and employs several interesting properties of $SI^{(k)}$-*Tree* to produce the skyline clusters fast.
- A performance study using both synthetic and real datasets is conducted to evaluate our approaches. We use the real dataset which is meaningful in practice. We also use the synthetic dataset with anti-correlated distribution. Our experimental evaluation indicates that the proposed approaches are both efficient and effective.

The rest of the paper is organized as follows: Sect. 2 formally describes the concept of skyline-CL. Section 3 presents and discusses the AEPSQ algorithm which can efficiently produce all skyline clusters. We present experimental study in Sect. 4. Finally, Sect. 5 concludes the paper with directions for future work.

2 Related Works

Borzsonyi et al. [4] first consider how to efficiently obtain skyline objects in the database community, and propose two feasible algorithms BNL and DC. The BNL algorithm essentially compares each object in the database with all the others and returns the objects that are not dominated by any others. The DC algorithm divides the input objects into several groups that can fit in memory. The skyline objects in all groups are computed separately using a memory-based algorithm and then merged to produce the final result. Kossmann et al. [5] propose a full space skyline computation algorithm based on the indexed dataset. It is based on the nearest neighbor query, which adopts the divide-and-conquer paradigm on the R-tree index. Papadias et al. [6,7] propose a branch and bound algorithm to progressively output skyline objects on dataset indexed by R-tree. One of the most important properties of this technique is that it guarantees the minimum I/O costs. Chomicki et al. [8] propose the SFS algorithm which sorts the input objects according to a preference function and then returns the skyline objects in another pass over the sorted list. Chan et al. [9] propose an effective approximate algorithm that is based on extending a Monte Carlo counting algorithm to fast return the skyline objects. Huang et al. [10] present an efficient cell-dominance computation algorithm (i.e., CDCA) for processing arbitrary single subspace skyline query. The CDCA algorithm uses the regular grid index and prunes all the cells which are dominated by any other ones. Li et al. [11] propose a system model that can support subspace skyline query in mobile distributed environment. This algorithm uses mapreduce and can obtain the meaningful subset of points from the full set of skyline points in any subspace. Huang et al. [12] focus on supporting concurrent and unpredictable subspace skyline queries over data streams. To balance the query cost and update cost, the authors only maintain the full space skyline and then propose an efficient and scalable two-phase algorithm to process the skyline queries in different subspaces based on the full space skyline.

Recently, several literatures study the extensions of skyline. Jin et al. [15] propose the concept of thick skyline, to

return not only skyline objects but also their nearby neighbors within ε distance. Chan et al. [16] present the concept of k-dominant skyline, which relaxes the idea of dominance to k-dominance and returns the k-dominant skyline set whose size is smaller than the skyline set. Huang et al. [17] propose a novel type of l-SkyDiv query which returns l skylines having maximum diversity. The authors prove that the l-SkyDiv query belongs to the NP-hard problem theoretically, and give three efficient heuristic algorithms whose time complexities are polynomial to fast implement the query. Lin et al. [18] propose to select k skyline objects so that the number of objects, which are dominated by at least one of these k skyline objects, is maximized. The authors first give an dynamic programming-based exact algorithm in a 2D space. Then, they prove that the problem is NP-hard for three or more. Tao et al. [19] give a new definition of representative skyline that minimizes the distance between a non-representative skyline object and its nearest representative. Like Lin et al. [18], Tao et al. present a dynamic programming-based exact algorithm in a 2D space, and prove that the problem is NP-hard for three or more. Lee et al. [20] discuss to support personalized skyline queries as identifying interesting objects based on user-specific preference and retrieval size k. The authors abstract personalized skyline ranking as a dynamic search over skyline subspaces guided by user-specific preference. Huang et al. [21] integrate K-means clustering into skyline computation and return K "representative" and "diverse" skyline objects to users. The authors propose an efficient evaluation approach which is based on the regular grid index to seamlessly integrate subspace skyline computation, K-means clustering and representatives selection.

Note that the concept of skyline-CL query proposed in our paper is different from the above extensions of skyline. The existing extensions mainly focus on the problem of selecting k representative skyline objects where k is the parameter and smaller than the number of skyline objects. However, they at least have two drawbacks: (1) They only return the k representative skyline objects, and these k objects may not be the ones that the users are interested in. (2) For three or more dimensions, the problem of selecting k representative skyline objects is NP-hard, and hence they need a great deal of time for the high-dimensional skyline applications.

Different from the existing extensions, our skyline-CL query fast produces several clusters of skyline objects and assists the users to locate their interested objects.

3 Related Concepts of Skyline-CL Query

In this section, we present several related concepts of skyline-CL query. Also, we propose to extend SQL with a SKYLINE-CL OF keyword, which can enhance the query engine functions of RDBMS.

Definition 1 *(dominance relationship)* Give two k-dimensional objects ϑ and λ. ϑ dominates λ if they satisfy the following conditions:

(1) $\forall i \in [1,k], \quad \vartheta[i] \leq \lambda[i]$;
(2) $\exists j \in [1,k], \quad \vartheta[j] < \lambda[j]$.

For simplicity, the relationship that ϑ dominates λ is denoted as $\lambda \prec \vartheta$.

Definition 2 *(skyline set)* Let $\wp$ be the set of k-dimensional objects. Then, the skyline set of $\wp$ can be expressed as: $\nabla(\wp)=\{\vartheta \,|\, \vartheta \in \wp \wedge \neg\exists\lambda \in \wp, \vartheta \prec \lambda\}$.

Definition 3 *(dominance region)* The dominance region $\vartheta.DR$ of a k-dimensional object ϑ is the area of the data space dominated by ϑ. Specifically, $\vartheta.DR$ is an axis-parallel rectangle whose major diagonal is decided by ϑ and the "max corner" of the data space (having the maximum coordinates on all dimensions).

Definition 4 *(non-dominance region)* [22] Given a data space D and the dominance region $\vartheta.DR$ of a k-dimensional object ϑ. Then the non-dominance region of ϑ denoted by $\overline{\vartheta.DR}$, is $D \backslash \vartheta.DR$(i.e., the minus space of D and $\vartheta.DR$).

Based on Definition 3, we define the dominance region of a k-dimensional object set $\wp$ as the union of the dominance regions of all objects in $\wp$ and denote it by $\wp.DR$. And based on Definition 4, its non-dominance region of $\wp$, denoted by $\overline{\wp.DR}$, is $D \backslash \wp.DR$.

Definition 5 *(skyline distance)* [22] Let ϑ and λ be the two k-dimensional objects. Then the skyline distance between these two objects is the volume of the distance area between them which can be specified by the following equation: $SKYDIST(\vartheta, \lambda) = \mathrm{VOL}((\vartheta.DR \backslash \lambda.DR) \cup (\lambda.DR \backslash \vartheta.DR))$.

Based on Definition 5, the skyline distance between 2 sets $\wp_1$ and $\wp_2$, denoted by $SKYDIST(\nabla(\wp_1), \nabla(\wp_2))$, is equal to $\mathrm{VOL}((\nabla(\wp_1).DR \backslash \nabla(\wp_2).DR) \cup (\nabla(\wp_2).DR \backslash \nabla(\wp_1).DR))$.

Definition 6 (ξ*-neighborhood of a skyline object)* Let $\wp$ and ξ be the set of k-dimensional objects and the distance parameter. For a skyline object $\vartheta[d_1, \ldots, d_k] \in \nabla(\wp)$, its ξ-neighborhood is defined by $NBH_\xi(\vartheta) = \{\eta[d_1, \ldots, d_k] \in \nabla(\wp) | SKYDIST(\vartheta, \eta) \leq \xi\}$.

Definition 7 *(core skyline object)* Let $\wp, \xi$ and $\Im$ be the set of k-dimensional objects, the distance parameter and the number threshold. Then, an object $\vartheta[d_1, \ldots, d_k]$ is called a core skyline object if it satisfies the following conditions:

(1) $\vartheta \in \nabla(\wp)$;
(2) $|NBH_\xi(\vartheta)| \geq \Im$.

Based on Definitions 6 and 7, for two skyline objects ϑ and λ, we say "λ is directly density-reachable from ϑ" if $\lambda \in NBH_{\xi}(\vartheta)$ and $|NBH_{\xi}(\vartheta)| \geq \Im$. Clearly, directly density-reachable is symmetric for pairs of core skyline objects.

Definition 8 *(density-reachable)* Let $\wp$ be the set of k-dimensional objects. A skyline object λ is density-reachable from the skyline object ϑ with respect to the distance parameter ξ and the number threshold $\Im$, if there exists a chain of skyline objects $c_1, c_2, \ldots, c_n$, $c_1 = \lambda$ and $c_n = \vartheta$ such that c_{i+1} is directly density-reachable from c_i with respect to ξ and $\Im$, for $1 \leq i \leq n$, $c_i \in \nabla(\wp)$.

Obviously, density-reachable is also symmetric for pairs of core skyline objects. For simplicity, the relation that the skyline object λ is density-reachable (density-unreachable) from ϑ is denoted as $\lambda \triangleright \vartheta$ ($\lambda \bar{\triangleright} \vartheta$).

Based on the following two lemmas in [13], we can give the concept of skyline cluster which is described in Definition 9.

Lemma 1 *Let $\wp$, ξ and $\Im$ be the set of k-dimensional objects, the distance parameter and the number threshold. If $NBH_{\xi}(\vartheta)| \geq \Im$, then $C = \{c | c \in \nabla(\wp)$ and c is density-reachable from $\vartheta\}$ is a cluster w.r.t. ξ and $\Im$.*

Lemma 2 *Let C be a cluster w.r.t. the distance parameter ξ and the number threshold $\Im$ and let $\vartheta \in \nabla(\wp)$ be any skyline object in C with $NBH_{\xi}(\vartheta)| \geq \Im$. Then C equals to the set $C' = \{c' | c' \in \nabla(\wp)$ and c' is density-reachable from $\vartheta\}$.*

Definition 9 *(skyline cluster)* Let $\wp$ and $\Im$ be the set of k-dimensional objects and the number threshold. If $SkyC \subseteq \wp$ satisfies the following conditions, then it is a skyline cluster:

(1) $SkyC \subseteq \nabla(\wp)$;
(2) $\forall \vartheta \in SkyC, |SkyC| \neq 1 \Rightarrow \exists \lambda \in SkyC \wedge \vartheta \triangleright \lambda$;
(3) $\forall \vartheta \in SkyC, \neg\exists \lambda \in SkyC \wedge \vartheta \bar{\triangleright} \lambda$;
(4) $\forall \vartheta \in SkyC, \neg\exists \lambda \notin SkyC \wedge \vartheta \triangleright \lambda$.

It is not difficult to see that the skyline cluster concept defines a series of maximal skyline groups that are based on "density-reachable."

In order to specify the skyline-CL query, we propose to extend SQL SELECT statement by an optional SKYLINE-CL OF clause as follows:

SELECT … FROM … WHERE …
GROUP BY … HAVING …
SKYLINE-CL OF [DISTINCT] Dis_param=ξ,

$$\begin{array}{c} \text{Thr\_param} = \Im, \\ d_1[\text{MIN}|\text{MAX}], \\ \ldots \\ d_m[\text{MIN}|\text{MAX}] \end{array}$$

ORDER BY …

Keywords Dis_param and Thr_param denote the distance parameter and the number threshold, respectively; $d_1, \ldots, d_m$ denote the dimensions of the skyline objects; MIN, MAX specify whether the value in that dimension should be minimized or maximized. The optional DISTINCT specifies how to deal with duplicates. The semantics of the SKYLINE-CL OF clause are straightforward. The SKYLINE-CL OF clause is executed after the SELECT … FROM … WHERE … GROUP BY … HAVING … part of the query, but before the ORDER BY clause and possibly other clauses that follow. The SKYLINE-CL OF clause captures all skyline clusters and returns them according to their cardinalities.

There are several advantages of the proposed SKYLINE-CL OF keyword and skyline cluster operator. First, users can calculate the skyline clusters in one concise and semantic-clear query. Second, they provide more optimization opportunities. As will be discussed in the rest of this paper, there are much more efficient algorithms to compute the skyline clusters than the naïve solution which handles skyline query and clustering process separately.

4 Efficient Implementation of the Skyline-CL Query

In this section, we present AEPSQ for processing the skyline-CL query. During the process of obtaining skyline objects, the AEPSQ algorithm organizes these objects as a novel k-ary tree $SI^{(k)}$-*Tree*, and employs several interesting properties of $SI^{(k)}$-*Tree* to produce skyline clusters fast. Table 1 summarizes some symbols that will be used frequently throughout the paper.

4.1 Constructing the Index Structure $SI^{(k)}$-Tree

In this subsection, we propose the efficient approach to construct the index structure $SI^{(k)}$-*Tree* for skyline objects.

Definition 10 ($SI^{(k)}$-*Tree*) The multiple-dimensional tree structure *MIS* is an index tree $SI^{(k)}$-*Tree* if it satisfies the following five properties (let $\chi^{(k)}$ be the set of nodes of *MIS*):

(1) $\forall \aleph \in \chi^{(k)}, \aleph^{(3)} \leq k$;
(2) $\forall \aleph \in \chi^{(k)}, \neg\exists \varpi \in \chi^{(k)}, \aleph^{(1)} \prec \varpi^{(1)}$;
(3) $\forall \aleph \in \chi^{(k)}, \forall \varpi \in Desc(\aleph), \aleph^{(2)} < \varpi^{(2)}$;
(4) $\forall \aleph, \varpi \in \chi^{(k)}, \aleph \in Anc(\varpi) \wedge \varpi \in Cht(\aleph, \gamma) \Rightarrow \begin{cases} \forall \delta \in Desc(\varpi) - Cht(\aleph, \gamma), skyED(\aleph, \delta) < skyED(\aleph, \varpi) \\ \forall \theta \in Cht(\varpi, \gamma), skyED(\aleph, \theta) > skyED(\aleph, \varpi); \end{cases}$
(5) $\forall \aleph, \varpi, \delta \in \chi^{(k)}, \aleph \in Anc(\varpi) \cap Anc(\delta) \wedge \varpi \in Cht(\aleph, \gamma) \wedge \delta \in Desc(\aleph)\text{-}Cht(\aleph, \gamma)$

$$\Rightarrow \begin{cases} skyED(\aleph, \varpi) < skyED(\varpi, \delta) \\ skyED(\aleph, \delta) < skyED(\varpi, \delta) \end{cases}.$$

Table 1 Frequently used symbols

Symbol	Description
$\vec{O}$	$\vec{O}$ is the origin
$\aleph$, $Par(\aleph)$	$\aleph$ and $Par(\aleph)$ are the node of $SI^{(k)}$-*Tree* and the parent of $\aleph$, respectively. $\aleph$ includes three components: (i) $\aleph^{(1)}$ is the corresponding object of $\aleph$; (ii) $\aleph^{(2)}$ is the skyline distance between $\aleph^{(1)}$ and $\vec{O}$, and (iii) $\aleph^{(3)}$ is $\aleph$ ' order number within the subnodes of $Par(\aleph)$
$Anc(\aleph)$	$Anc(\aleph)$ is the set of ancestor nodes of $\aleph$
$Desc(\aleph)$	$Desc(\aleph)$ is the set of offspring nodes of $\aleph$
$Cht(\aleph, \gamma)$	$Cht(\aleph, \gamma)$ is the γ-th substree of $\aleph$
$skyED(\aleph_1, \aleph_2)$ $= SKYDIST(\aleph_1^{(1)}, \aleph_2^{(1)})$	$skyED(\aleph_1, \aleph_2)$ is the node distance between the nodes $\aleph_1$ and $\aleph_2$

From the above definition, we can see that if *MIS* satisfies five key properties, then it becomes an index tree $SI^{(k)}$-*Tree*. Property (1) requires that *MIS* is the k-ary ree, i.e., each node in *MIS* has at most k children. Property (2) requires that for each $\aleph$ in *MIS*, the corresponding object of $\aleph$ (i.e., $\aleph^{(1)}$) is a skyline object in the set $\chi^{(k)}$. For each node $\aleph$ in *MIS* and its any offspring node ϖ, Property (3) requires that the skyline distance between $\aleph^{(1)}$ and $\vec{O}$ needs to be less than that between each offspring node of $\varpi^{(1)}$ and $\vec{O}$. Let ϖ be in the γ-th substree of $\aleph$, and θ (δ) be any offspring node of ϖ that is (not) in the γ-th substree of ϖ. Property (4) requires that the node distance between δ (θ) and $\aleph$ needs to be less (larger) than that between ϖ and $\aleph$. Assume that ϖ and δ are in the γ-th and λ-th substree of $\aleph$, respectively ($\gamma \neq \lambda$). Property (5) requires that the node distance between ϖ and δ needs to be larger than that between ϖ (δ) and $\aleph$.

Definition 11 *(NN-sphere)* [23] Let Q and NN be a query point and the nearest neighbor of Q. Then $NN\text{-}dist = SKYDIST(Q, NN)$ is the distance of the nearest neighbor and the query point. The NN-sphere $NNSP(Q, NN)$ of Q is defined as the sphere with center Q and radius $r = NN - dist$.

In the following part, we discuss how to organize skyline objects as an index tree $SI^{(k)}$-*Tree* during skyline computation. Inspired by [6,7], we present the CONST_$SI^{(k)}$-*Tree* algorithm to realize this task. The algorithm is based on the following theorem.

Theorem 1 (minimal skyline distance) *Let $\wp$ be the set of k-dimensional objects. Then $\vartheta[d_1,\ldots, d_k] \in \wp$ is bound to be a skyline object of $\wp$ if it satisfies the following condition:* $\neg\exists\lambda[d_1, d_2,\ldots, d_k] \in \wp$, $SKYDIST(\vartheta,\vec{O}) > SKYDIST(\lambda,\vec{O})$.

Proof Let Θ be the whole k-dimensional data space. Since $\neg\exists\lambda[d_1, \ldots, d_k] \in \wp$, $SKYDIST(\vartheta,\vec{O}) > SKYDIST(\lambda,\vec{O})$, we can get that $\forall\lambda[d_1,\ldots, d_k] \in \wp$, $\mathrm{VOL}((\Theta\backslash\delta.DR) \cup (\delta.DR\backslash\Theta)) \leq \mathrm{VOL}((\Theta\backslash\lambda.DR) \cup (\lambda.DR\backslash\Theta))$. Because Θ is the whole data space, $\delta.DR\backslash\Theta = \lambda.DR\backslash\Theta = \varnothing$. And therefore $\forall\lambda[d_1, \ldots, d_k] \in \wp$, $\mathrm{VOL}(\Theta\backslash\delta.DR) \leq \mathrm{VOL}(\Theta\backslash\lambda.DR))$. Then we can have $\forall\lambda[d_1, \ldots, d_k] \in \wp$, $\mathrm{VOL}(\delta.DR) \geq \mathrm{VOL}(\lambda.DR)$. That is, for any k-dimensional object λ in $\wp$, the volume of $\vartheta.DR$ is equal to or larger than that of $\lambda.DR$. Thus, we can get $\exists d_i$ ($i \in [1, k]$), and $\vartheta[d_i]$ is minimal among the objects in $\wp$. Then based on Definitions 1 and 2, we know that $\neg\exists\lambda[d_1, \ldots, d_k] \in \wp$, $\lambda \prec \vartheta$. Consequently, ϑ is a skyline object of $\wp$. □

The complete algorithm is shown below.

Algorithm 1: The CONST_$SI^{(k)}$-*Tree* algorithm
Input: the set of k-dimensional objects $\wp$;
Output: The tree structure *TS*;
Begin
1. $wDS \leftarrow <(\infty, \infty,\ldots, \infty)>$;
2. While ($wDS \neq \varnothing$) Do
3. $\overline{FR} \leftarrow$take the first region of wDS;
4. If ($\exists$NN-opt($\vec{O}, \wp, \overline{FR}$)) Then
   /* The NN-opt procedure is presented in Algorithm 2 */
5. $\delta \leftarrow$NN-opt($\vec{O}, \wp, \overline{FR}$);
6. For $i \leftarrow 1$ to k Do
7. $wDS \leftarrow wDS \cup <(\overline{FR}[1],\ldots, \overline{FR}[i\text{-}1], \delta[i], \overline{FR}[i\text{+}1],\ldots, \overline{FR}[k])>$;
8. Initialize a node $\aleph = \varnothing$;
9. $\aleph^{(1)} \leftarrow \delta$;
10. $\aleph^{(2)} \leftarrow SKYDIST(\delta, \vec{O})$;
11. $\aleph^{(3)} \leftarrow NULL$;
12. If $TS = \varnothing$ Then
13. $TS.Root \leftarrow \aleph$;
14. Else
15. $j \leftarrow$the latest modified position in $\overline{FR}$;
    /* we can use a flag to track the latest modified position in $\overline{FR}$ */
16. $\zeta \leftarrow$the object corresponding to the position j in $\overline{FR}$;
17. $\mathfrak{N} \leftarrow$the node whose first component $\mathfrak{N}^{(1)}$ is ζ;
18. Set $\aleph$ as the j-th subnode of $\mathfrak{N}$;
19. $\aleph^{(3)} \leftarrow j$;
20. Return *TS*;
End

Furthermore, in order to improve the efficiency of computing skyline distances, we utilize the approximation technology ATCSD which is proposed in [23]. ATCSD is based on the Monte Carlo sampling approach and approximates it by randomly sampling points and computing the ratio between samples that fall into the *SKYDIST* region defined by Definition 5 and the ones that do not. When we give the number of objects used in Monte Carlo sampling, the amount of sam-

ples which fall into the *SKYDIST* region is determined. The ratio between the samples located in the *SKYDIST* region and the ones that do not give an approximation of the skyline distances.

Algorithm 2: The NN-opt procedure [23]
Input: The query object $\vec{O}$, the set of objects $\wp$,the query region $\overline{FR}$;
Output: the nearest neighbor $\delta \in \wp$ of $\vec{O}$ within $\overline{FR}$;
Begin
1. Organize the objects in $\wp$ as a PMR-Quadtrees;
2. $\delta \leftarrow (\infty, \infty, \ldots, \infty)$;
3. *PList*←the list including subpartitions of the root-partition of PMR-Quadtrees;
4. For each partition *PAR* in *Plist* Do
5. Compute the minimal skyline distance *mSKD* from $\vec{O}$ to any object θ falling inside *PAR*;
/* $mSKD(PAR, \vec{O}) = \min_{\theta \in PAR} SKYDIST(\theta, \vec{O})$; */
6. Sort the list *PAR* by *mSKD*;
7. While (*PAR*≠∅) Do
8. If the first partition in *PAR* is a leaf Then
9. *Tp_NN*←the nearest object in leaf;
10. If $SKYDIST(Tp\_NN, \vec{O}) < SKYDIST(\delta, \vec{O})$ Then
11. Prune *PAR* with *Tp_NN*;
12. Else
13. Replace the first partition of *PAR* with its son nodes;
14. resort the list *PAR* by *mSKD*;
15. Return δ;
End

In the following part, we present a detailed proof that t *TS* which is produced by the CONST_$SI^{(k)}$-*Tree* algorithm is an index tree $SI^{(k)}$-*Tree*.

Theorem 2 *When the CONST_$SI^{(k)}$-Tree algorithm terminates, the tree structure TS satisfies the five properties of $SI^{(k)}$-Tree that are presented in Definition 10.*

Proof In the CONST_$SI^{(k)}$-*Tree* algorithm, when a node $\aleph$ is inserted into *TS*, *k* subregion is added into the list *wDS*, and for a subregion, there exists at most one node that becomes the subnode of $\aleph$. Hence, *TS* is a *k*-ary tree. Thus, *TS* satisfies Property (1). For any node $\aleph$ in*TS*, $\aleph^{(1)} = \delta$ is the nearest neighbor of$\vec{O}$ within $\overline{FR}$, and hence according to Theorem 1, $\aleph^{(1)}$ is a skyline object in the set $\wp$. That is, *TS* satisfies Property (2). Let $\aleph$ be the node whose first component $\aleph^{(1)}$ is the nearest neighbor of$\vec{O}$ within $\overline{FR}$. We can get that for each offspring node ϖ of $\aleph$, its first component $\varpi^{(1)}$ falls inside the region $\overline{FR}$, and the skyline distance between $\aleph^{(1)}$ and $\vec{O}$ is bound to be less than that between $\varpi^{(1)}$ and $\vec{O}$. That is, $\aleph^{(2)} < \varpi^{(2)}$. Therefore, *TS* satisfies Property (3). Assume that ϖ is the node whose first component $\varpi^{(1)}$ is the nearest neighbor of$\vec{O}$ within $\overline{FR}$, and locates in the γ-th substree of the node $\aleph$. When η locates in the i-th ($1 \le i \le k$, $i \neq \gamma$) substree of ϖ, we can get that $\overline{RR} \subset \overline{FR}$, where $\eta^{(1)}$ is the nearest neighbor of$\vec{O}$. Hence, $\eta^{(1)}$ falls inside the region determined by $\aleph^{(1)}$ and $\varpi^{(1)}$. Thus, $SKYDIST(\aleph^{(1)}, \eta^{(1)}) < SKYDIST(\varpi^{(1)}, \aleph^{(1)})$. On the other hand, When η locates in the i-th ($i=\gamma$) substree of ϖ, we can get that $\overline{RR} \cap \overline{FR} = \varnothing$. That is, $\eta^{(1)}$ falls outside the region determined by $\aleph^{(1)}$ and $\varpi^{(1)}$. Since $\aleph$ is the ancestor node of ϖ, η and $\aleph$ locate on the different sides of ϖ. Therefore, $SKYDIST(\aleph^{(1)}, \eta^{(1)}) > SKYDIST(\varpi^{(1)}, \aleph^{(1)})$. Consequently, *TS* satisfies Property (4). Let ϖ and η locate in the different subtrees of $\aleph$. We can get that $\varpi^{(1)}$ and $\eta^{(1)}$ locate on the different sides of $\aleph^{(1)}$. Hence, we can have the inequations $SKYDIST(\aleph^{(1)}, \varpi^{(1)}) < SKYDIST(\varpi^{(1)}, \eta^{(1)})$ and $SKYDIST(\aleph^{(1)}, \eta^{(1)}) < SKYDIST(\varpi^{(1)}, \aleph^{(1)})$. Thus, *TS* satisfies the Property (5). □

4.2 Efficiently Producing Skyline Clusters

In this subsection, we present an efficient algorithm, i.e GEN_SkyCLs, to fast produce all skyline clusters based on several interesting properties of the index structure $SI^{(k)}$-*Tree*. Without loss of generality, we assume that the distance parameter and the number threshold are ξ and $\Im$. Before formal description of the GEN_SkyCLs algorithm, we present an important concept, i.e., nearest node set (NNS).

Definition 12 *(nearest node set)* We define the nearest node set of $\aleph$ as follows: $NNS(\aleph) = \{\varpi |$ there exists the position i such that $\overline{FR} = (\overline{FR}[1], \ldots, \overline{FR}[i-1], \varpi^{(1)}[i], \overline{FR}[i+1], \ldots, \overline{FR}[k])$ and $q \neq \infty\}$, where $\overline{FR}$ is the region within which $\aleph^{(1)}$ is the nearest neighbor of$\vec{O}$.

Next, we give three key properties of NNS($\aleph$).

Theorem 3 (The properties of NNS) *Let Ω be the set of nodes of the index tree$SI^{(k)}$-Tree. For every node $\aleph$ in Ω, its nearest node set NNS($\aleph$) has the following three important properties: (1) the cardinality of NNS($\aleph$) is smaller than or equal to k; (2) NNS($\aleph$) is a subset of Anc($\aleph$); and (3) for each node $\varpi \in \Omega$-NNS($\aleph$), there exists a node $\eta \in$NNS($\aleph$) such that $\eta^{(1)}$ is nearer than $\varpi^{(1)}$ to $\aleph^{(1)}$.*

Proof (i) Since $\overline{FR}$ is the region within which $\aleph^{(1)}$ is the nearest neighbor of$\vec{O}$ and has *k* dimensions, the cardinality of NNS($\aleph$) does not exceed *k*. Hence, Property (1) is correct. (ii) The correctness of Property (2) is obvious because each node belonging to NNS($\aleph$) locates in the tree path *PT* from the root node to $\aleph$. And each node on *PT* is the ancestor node of $\aleph$. Hence, NNS($\aleph$)⊆*Anc*($\aleph$). (iii) Clearly, every node ϖ which does not belong to NNS($\aleph$) falls outside the region $\overline{FR}$. Therefore, for ϖ, there exists at least one node η belonging to NNS($\aleph$) such that $\angle \varpi^{(1)} \eta^{(1)} \aleph^{(1)} > 90^{\circ}$. That is, $\angle \varpi^{(1)} \eta^{(1)} \aleph^{(1)}$ is an obtuse angle. Hence, $SKYDIST(\eta^{(1)}, \aleph^{(1)}) < SKYDIST(\varpi^{(1)}, \aleph^{(1)})$. Thus, Property (3) is correct. □

In the following part, we describe and analyze the GEN_SkyCLs algorithm. The basic idea of GEN_SkyCLs can be described as follows. The algorithm visits each node

$\aleph$ of *TS* from the root in a breadth-first mode and computes the node distance $skyED(\aleph, \chi)$ between $\aleph$ and each node χ belonging to NNS($\aleph$). If it is not larger than the distance parameter ξ, GEN_SkyCLs adds χ into the $TC(\aleph)$. And then for the parent ϑ of χ, GEN_SkyCLs checks whether ϑ belongs to $TC(\aleph)$. If yes, GEN_SkyCLs adds all offspring nodes of $\aleph$ that do not locate in the γ-th child of $\aleph$ into $TC(\aleph)$. Else GEN_SkyCLs selects and adds each offspring node τ of $\aleph$ which does not locate in the γ-th child of $\aleph$ and the node distance $skyED(\chi, \tau)$ is not larger than ξ into$TC(\aleph)$. If the cardinality of $TC(\aleph)$ is not less than the number threshold $\Im$, GEN_SkyCLs inserts $\aleph$ into the list *csoL* which includes all core skyline objects after the algorithm terminates. Finally, according to Definition 9, GEN_SkyCLs divides *csoL* into Υ groups and returns these groups according to their cardinality.

The complete GEN_SkyCLs algorithm is shown below.

Algorithm 3: The GEN_SkyCLs algorithm
Input: the $SI^{(k)}$*-Tree* tree *TS* produced by Algorithm 1, the distance parameter ξ and the number threshold $\Im$;
Output: all skyline clusters;
Begin
1. *csoL*←NULL; /* the list of core skyline objects */
2. *POINTER*←1; /* the pointer of *csoL* */
3. For each node $\aleph$ of *TS* Do
   /* from the root in a breadth-first mode */
4. NNS($\aleph$)←$\varnothing$;
5. $\overline{F}$ ←the region that produces $\aleph$;
   /* $\aleph$ is the nearest neighbor of $\bar{O}$ in $\overline{F}$ */
6. NNS($\aleph$)={χ|$\exists i$, $\overline{FR}$=($\overline{FR}$[1] ,…, $\overline{FR}$[i-1] , χ[i], $\overline{FR}$[i+1] ,…, $\overline{FR}$[k])$\wedge\chi\neq\infty$};
7. $TC(\aleph)$←$\varnothing$;
8. For $\chi\in$ NNS($\aleph$) Do
9. If $skyED(\aleph, \chi)\leq\xi$ Then $TC(\aleph)$←$TC(\aleph)\cup\{\chi\}$;
10. ϑ←the parent of χ;
11. γ←the sequence number of χ in.the subnodes of ϑ;
    /* ϑ is the γ-th subnode of χ */
12. If $\vartheta\in TC(\aleph)$ Then
13. For each offspring node τ of χ which does not locate in the γ-th subtree of χ Do
14. $TC(\aleph)$←$TC(\aleph)\cup\{\tau\}$;
15. Else
16. For each offspring node τ of χ which does not locate in the γ-th subtree of χ Do
17. If $skyED(\chi, \tau)\leq\xi$ Then
18. $TC(\aleph)$←$TC(\aleph)\cup\{\tau\}$;
19. If $|TC(\aleph)|\geq\Im$ Then
20. Add the node $\aleph$ into *csoL*[*POINTER*];
21. *POINTER*←*POINTER* +1;
22. Divide *csoL* into Υ groups that each group $SC[i]$ ($1\leq i\leq\Upsilon$) satisfies the following conditions:
    Let *tempC*← $\bigcup_{\aleph\in SC[i]} TC(\aleph)$;
    (a) $\forall\vartheta\in tempC, \exists\lambda\in tempC\wedge\vartheta\rhd\lambda$;
    (b) $\forall\vartheta\in tempC, \neg\exists\lambda\in tempC\wedge\vartheta\overline{\rhd}\lambda$;
    (c) $\forall\vartheta\in tempC, \neg\exists\lambda\notin tempC\wedge\vartheta\rhd\lambda$.□□□
23. Return the Υ groups obtained in Step 22 according to their cardinality;
End

Theorem 4 (The correctness of algorithm) *Let* $\wp$, ξ *and* $\Im$ *be the set of k-dimensional objects, the distance parameter and the number threshold. When GEN_SkyCLs terminates,* Υ *skyline clusters produced by the algorithm satisfy the following conditions: (I) The skyline set* $\nabla(\wp)$ *just consists of the objects belonging to these* Υ *skyline clusters; (II) the cardinality of each skyline clusters is not less than* $\Im$*; (III) for any two skyline objects* ϑ *and* λ, λ *is density-reachable from* ϑ *with respect to* ξ*; and (IV) for any two skyline objects* ϑ *and* λ*, if* ϑ *and* λ *belong to different skyline clusters, then* λ *is density-unreachable from* ϑ *with respect to* ξ.

Proof The node set *ND* of the $SI^{(k)}$*-Tree* tree *TS* just includes the skyline objects of *AD*. That is, $ND= \nabla(\wp)$. On the other hand, the algorithm visits each node $\aleph$ of *TS* from the root in a breadth-first mode and puts $\aleph$ in a specific skyline cluster. So, $\cup_{i=1}^{\gamma} SC(i) = \nabla(\wp)$. And hence the condition (I) is satisfied. In the GEN_SkyCLs algorithm, Step 20 only inserts the node $\aleph$ whose corresponding set $TC(\aleph)$ has at least $\Im$ elements. Hence the condition (II) is satisfied. In the GEN_SkyCLs algorithm, for each node $\aleph$ in the $SI^{(k)}$*-Tree* tree *TS*, there

exist three places that can add a node p into $TC(\aleph)$: Step 9, Step 14 and Step 18. And according to the properties (4) and (5) of the $SI^{(k)}$-*Tree* tree, it is not difficult to see that in these three places, GEN_SkyCLs only adds each node p into $TC(\aleph)$ such that the node distance $skyED(\aleph, p)$ is not larger than the distance parameter ξ. Therefore, according to Definition 7, $\aleph^{(1)}$ is a core skyline object. Moreover, in the GEN_SkyCLs algorithm, Step 22 uses Definition 9 to divide the list *csoL* into Υ clusters. Hence, the conditions (III) and (IV) are satisfied. □

5 Experiments

This section conducts an empirical study of our methods using both synthetic and real datasets. We evaluate the efficiency and the scalability of the proposed algorithms. All our experiments are implemented in Java, running on a PC with PIV 2.4 GHz processor and 2G main memory.

The real dataset is European football players' technical statistics. It involves 21,649 records from 1920 to 2006. And each record has 8 attributes. Using the data generator [4], we generate the synthetic dataset with anti-correlated distribution where if a record is good in on dimension, it is unlikely to be good in other dimensions. It involves 170,000 records and each record has 8 attributes. For simplicity, we denote the real dataset and the synthetic dataset as RDS and SDS, respectively. Without loss of generality, all the values of objects are normalized in range [0, 1].

In the following experiments, we let the dimensionality vary in the range [2, 8]. The number of skylines on each dimensionality is shown in Table 2. On the other hand, the distance parameter ξ and number threshold $\Im$ can also determine the number of skyline clusters. The skyline distance between any two objects becomes larger with the increase in dimensionality. And the value of $\Im$ is inverse ratio with the number of skyline clusters. Hence, in order to balance of the number of skyline clusters on each dimensionality, the different range of ξ and the different value of $\Im$ are chosen for different dimensionality. Table 3 shows the details. For convenience, we use $Param_i (i = 1{-}4)$ to denote the values of two parameters ξ_i and $\Im_i$ in the following experiments.

Table 2 Number of skylines on each dimensionality

Datasets	Dimensionality			
	2	4	6	8
RDS	67	1294	5811	16659
SDS	168	6573	39539	130638

Table 3 Range of ξ and $\Im$

The parameters		Dimensionality			
		2	4	6	8
$Param_1$	ξ_1	0.2	0.5	0.9	1.3
	$\Im_1$	1	5	10	20
$Param_2$	ξ_2	0.4	0.8	1.2	1.6
	$\Im_2$	1	5	10	20
$Param_3$	ξ_3	0.6	1.1	1.5	1.9
	$\Im_3$	1	5	10	20
$Param_4$	ξ_4	0.8	1.4	1.8	2.2
	$\Im_4$	1	5	10	20

5.1 Evaluating the Number of Skyline Clusters

This subsection focuses on evaluating the number of skyline clusters. Figure 2 shows experimental results.

Comparing the number of skyline objects shown in Table 2 and the number of skyline clusters shown in Fig. 2, we can observe that multiple skyline objects correspond to the same skyline cluster. When we are unable to evaluate the whole set of skyline objects, we can reduce the range of evaluation and focus on a specific skyline cluster where the skyline objects have strong correlations. We can further observe that for the dataset RDS, when the dimensionality equals 8, the number of skyline objects is equal to 16659 (about 77 % of that of the whole dataset); however, the number of skyline clusters is 13 ($\xi = 2.2$ and $\Im = 20$). And for the dataset SDS, when the dimensionality equals 8, the number of skyline objects is equal to 130638 (about 79 % of that of the whole dataset); however, the number of skyline clusters is 28 ($\xi = 2.2$ and $\Im = 20$). On the other hand, for the experimental results, we can observe that the average number of skyline clusters becomes larger with the increase of ξ. This is mainly because more pairs of skyline objects are density-reachable with the increase of ξ.

5.2 Evaluating Top-3 Skyline Clusters

In this subsection, we focus on evaluating the top-3 skyline clusters w.r.t. the number of skyline objects. Figures 3 and 4 show the experimental results for these two datasets, respectively. For simplicity, we denote "top j ($j = 1{-}3$) of the real dataset" as "D1Tj", and "top j ($j = 1{-}3$) of the synthetic dataset" as "D2Tj".

From the experimental results, we observe that the number of objects in each top-3 skyline cluster becomes larger with the increase of ξ. This is because more pairs of skyline objects are density-reachable with the increase of ξ. For example, in Fig. 3d, when $\xi = 1.3$ and $\Im = 20$, the top-3 skyline clusters contain 2706, 1983 and 1491 skyline objects,

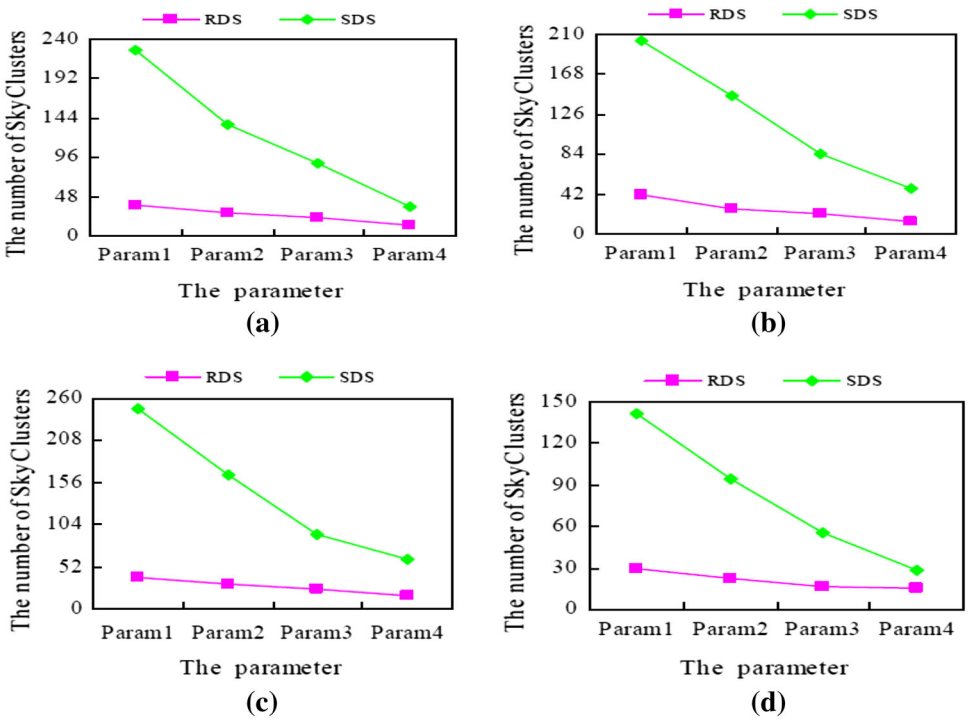

Fig. 2 Evaluation for the number of skyline clusters. **a** Dimensionality equals 2. **b** Dimensionality equals 4. **c** Dimensionality equals 6. **d** Dimensionality equals 8

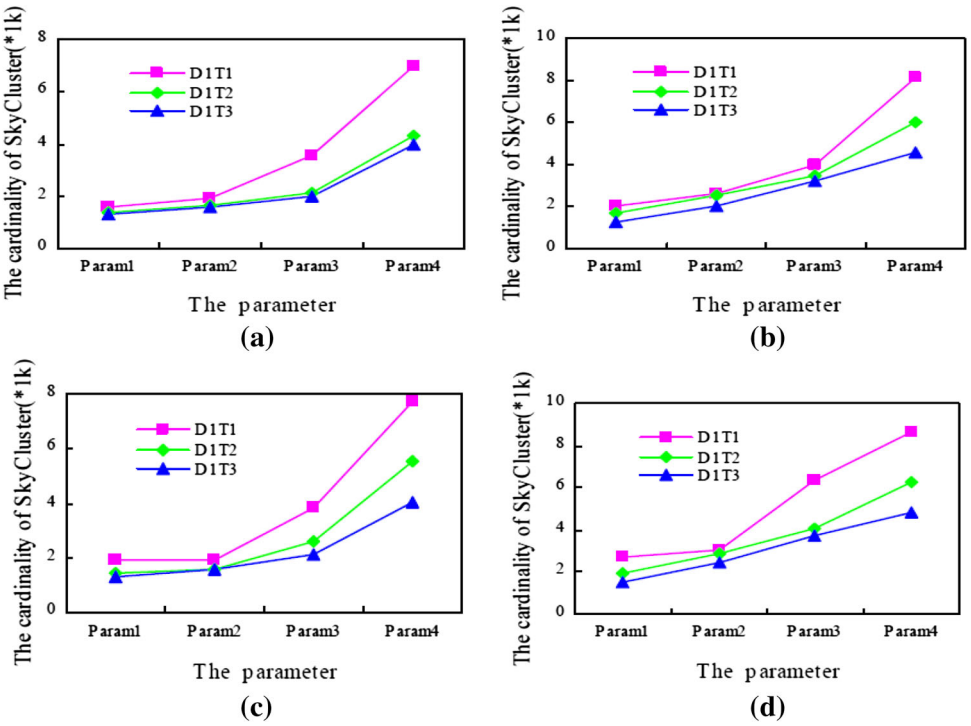

Fig. 3 Evaluation for the real dataset. **a** Dimensionality equals 2. **b** Dimensionality equals 4. **c** Dimensionality equals 6. **d** Dimensionality equals 8

respectively. However, when $\xi = 2.2$ and $\Im = 20$, the top-3 skyline clusters contain 8640, 6275 and 4834 skyline objects, respectively. We further find that in Fig. 3d, when $\xi = 2.2$ and $\Im = 20$, the first top skyline cluster consists of seven parts such that each part is a skyline cluster when $\xi = 1.3$ and $\Im = 20$; the second top skyline cluster consists of five parts such that each part is a skyline cluster when $\xi = 1.3$ and $\Im = 20$; and the third top skyline cluster consists of four parts such that each part is a skyline cluster when $\xi = 1.3$ and $\Im = 20$. On the other hand, from Fig. 4, we observe that the number of each top-3 skyline clusters increases markedly when ξ changes from the third value to the fourth value. For example, in Fig. 4d, when $\xi = 1.3$ and $\Im = 20$, the top-3 skyline clusters contain 9265, 7591 and 7082 skyline objects,

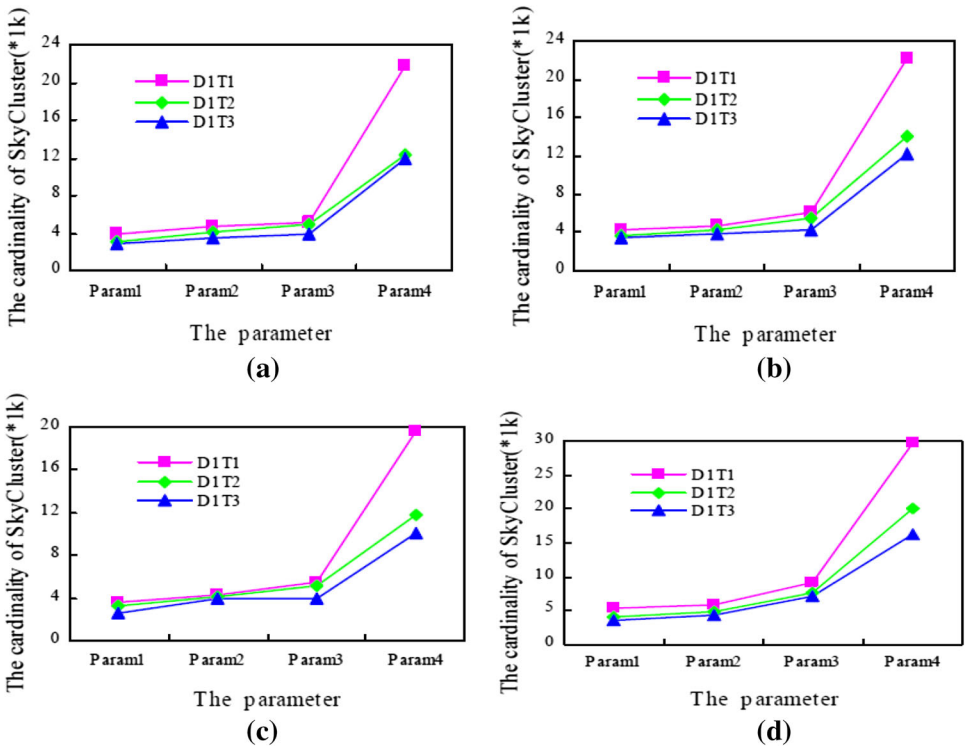

Fig. 4 Evaluation for the synthetic dataset. **a** Dimensionality equals 2. **b** Dimensionality equals 4. **c** Dimensionality equals 6. **d** Dimensionality equals 8

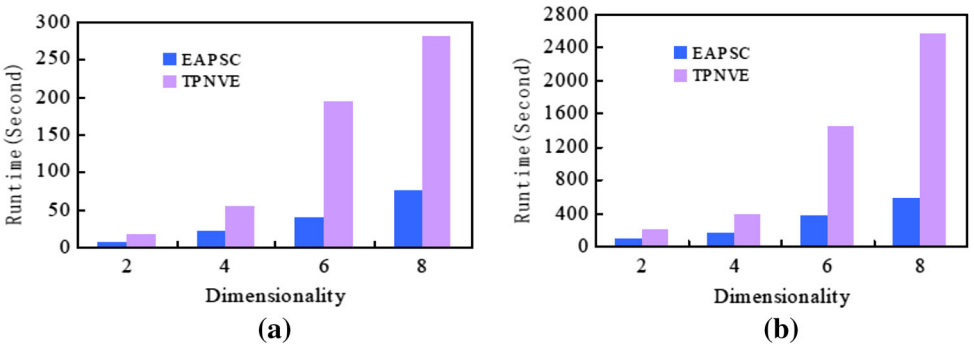

Fig. 5 Evaluation for the runtime. **a** The real dataset. **b** The synthetic dataset

respectively. However, when $\xi = 2.2$ and $\Im = 20$, the top-3 skyline clusters contain 29821, 20074 and 16255 skyline objects, respectively. This is mainly because the skyline distances between most pairs of objects do not exceed 2.2.

5.3 Evaluating the Time Cost

In this subsection, we focus on the time cost of AEPSQ which first uses the CONST_$SI^{(k)}$-*Tree* algorithm (Sect. 4.1) to organize the skyline objects as a $SI^{(k)}$-*Tree* tree and then utilizes the GEN_SkyCLs algorithm (Sect. 4.2) to obtain the skyline clusters fast. The compared solution consists of three separate procedures: (1) use the MR-GSC algorithm [9] to obtain all skyline objects; (2) utilize the DBSCAN algorithm [11] to cluster these skyline objects; and (3) return all skyline clusters according to their cardinalities. And we denote the compared solution as TPNVE. For simplicity, ξ and $\Im$ are set to (0.6, 2), (1.1, 6), (1.5, 10) and (1.9, 14) for 2D, 4D, 6D and 8D, respectively. Figure 5 shows the experimental results.

From Fig. 5, we can observe that the AEPSQ solution evidently outperforms the TPNVE solution in all cases. And the superiority of AEPSQ over TPNVE becomes more marked as the dimensionality increases. This is mainly because that the pruning ability of $SI^{(k)}$-*Tree* gradually enhances with the increase of the dimensionality. For instance, for the real dataset, when the dimensionality is equal to 2, the runtime of AEPSQ is about 42 % of that of TPNVE, and when the dimensionality is equal to 8, the runtime of AEPSQ is about 27 % of that of TPNVE. For the synthetic dataset, when the dimensionality is equal to 2, the runtime of AEPSQ is about 45 % of that of TPNVE, and when the dimensionality is equal to 8, the runtime of AEPSQ is about 22 % of that of TPNVE.

We can further observe that the effect of AEPSQ on the synthetic dataset is more evident than that on the real dataset. This is mainly since the skyline distances of most pairs of parent–child nodes do not exceed ξ in the synthetic dataset.

6 Conclusions

In this paper, we introduce the density-based clustering technology into real skyline query applications and first propose the concept of skyline-CL, which clusters the skyline objects and makes users zoom into different groups of skyline objects. Also, we propose to extend SQL with a SKYLINE-CL OF keyword, which can efficiently enhance the query engine functions of RDBMS. On the other hand, an efficient solution (i.e., AEPSQ) is developed for efficiently processing the skyline-CL query. Our solution is based on the novel index tree $SI^{(k)}$*-Tree* which is first proposed in our paper, and employs several interesting properties of the index tree to obtain the skyline clusters fast. Furthermore, we present detailed theoretical analyses and extensive experiments that demonstrate our solution is both efficient and effective.

Future work will focus on using some more efficient index structures to improve the performance of our algorithms, extending our algorithms to stream environments, and on more experimentation.

Acknowledgments This work is supported by the Shanghai Rising-Star Program (No. 15QA1403900), the National Natural Science Foundation of China (Nos. 61272268, 61072138, 61379005), the Program for New Century Excellent Talents in University (NCET-12-0413), the Fok Ying-Tong Education Foundation (142002), the Southwest University of Science and Technology (12zx7127).

References

1. Lin, X.; Xu, J.; Hu, H.; Lee, W.: Authenticating location-based skyline queries in arbitrary subspaces. IEEE Trans. Knowl. Data Eng. **26**(6), 1479–1493 (2014)
2. Jiang, T.; Zhang, B.; Lin, D. et al.: Incremental evaluation of top-*k* combinatorial metric skyline query. Knowl. Based Syst. **74**, 89–105 (2015)
3. Godfrey, P.: Skyline cardinality for relational processing. In: International Conference on Foundations of Information and Knowledge Systems, pp. 78–97 (2004)
4. Borzsonyi, S.; Kossmann, D.; Stocker, K.: The skyline operator. In: International Conference on Data Engineering, pp. 421–430 (2001)
5. Kossmann, D.; Ramsak, F.; Rost, S.: Shooting stars in the sky: an online algorithm for skyline queries. In: Proceedings of the 28th International Conference on Very Large Data Bases, pp. 311–322 (2002)
6. Papadias, D.; Tao, Y.; Fu, G.; Seeger, B.: An optimal and progressive algorithm for skyline queries. In: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 467–478 (2003)
7. Papadias, D.; Tao, Y.; Fu, G.; Seeger, B.: Progressive skyline computation in data systems. ACM Trans. Database Syst. **30**(1), 41–82 (2005)
8. Chomicki, J.; Godfrey, P.; Gryz, J.; Liang, D.: Skyline with Presorting:Theory and Optimization. In: International Conference on Intelligent Information Systems, pp. 595–604 (2005)
9. Chan, C.; Jagadish, H.; Tan, K.; Tung, A.; Zhang Z.: On High Dimensional Skylines. In: International Conference on Extending Database Technology, pp. 478–495 (2006)
10. Huang, Z.; Guo, J.; Sun, S.; Wei, W.: Efficient optimization of multiple subspace skyline queries. J. Comput. Sci. Technol. **23**(1), 103–111 (2008)
11. Li, Y.; Li, Z.; Dong, M. et al.: Efficient subspace skyline query based on user preference using MapReduce. Ad Hoc Netw. **35**, 105–115 (2015)
12. Huang, Z.; Sun, S.; Wang, W.: Efficient mining of skyline objects in subspaces over data streams. Knowl. Inf. Syst. **22**(2), 159–183 (2010)
13. Sander, J.; Ester, M.; Kriegel, H.P. et al.: Density-based clustering in spatial databases: The algorithm gdbscan and its applications. Data Min. Knowl. Discov. **2**(2), 169–194 (1998)
14. Gan, J.; Tao, Y.: DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 519–530. ACM (2015)
15. Jin, W.; Han, J.; Ester, M.: Mining thick skylines over large databases. In: Proceedings of PKDD, pp. 255–266 (2004)
16. Chan, C.; Jagadish, H.; Tan, K.; Tung, A.; Zhang, Z.: Finding K-dominant skylines in high dimensional space. In: Proceedings of ACM SIGMOD, pp. 503–514 (2006)
17. Huang, Z.; Xiang, Y.; Lin, Z.: *l*-Skydiv query: effectively improve the usefulness of skylines. Sci. China Inf. Sci. **53**(9), 1785–1799 (2010)
18. Lin, X.; Yuan, Y.; Zhang, Q.; Zhang, Y.: Selecting stars: the *k* most representative skyline operator. In: International Conference on Data Engineering, pp. 86–95 (2007)
19. Tao, Y.; Ding, L.; Lin, X.; Pei, J.: Distance-based representative skyline. In: International Conference on Data Engineering, pp. 892-903 (2009)
20. Lee, J.; You, G.; Hwang, S. et al.: Personalized top-*k* skyline queries in high-dimensional space. Inf. Syst. **34**(1), 45–61 (2009)
21. Huang, Z.; Xiang, Y.; Zhang, B. et al.: A clustering based approach for skyline diversity. Expert Syst. Appl. **38**(7), 7984–7993 (2011)
22. Berchtold, S.; Böhm, C.; Keim, D.A.; et al.: A cost model for nearest neighbor search in high-dimensional data space. In: Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, pp. 78–86. ACM (1997)
23. Lee, J.; You, G.; Hwang, S.: Personalized top-k skyline queries in high-dimensional space. Inf. Syst. **34**(1), 45–61 (2009)