Dominant and K Nearest Probabilistic Skylines*

Gabriel Pui Cheong Fung[1], Wei Lu[2,3], and Xiaoyong Du[2,3]

[1] School of ITEE, The University of Queensland, Australia
g.fung@uq.edu.au
[2] Key Labs of Data Engineering and Knowledge Engineering, Ministry of Education, China
[3] School of Information, Renmin University of China, China
{uqwlu,duyong}@ruc.edu.cn

Abstract. By definition, objects that are skyline points cannot be compared with each other. Yet, thanks to the probabilistic skyline model, skyline points with repeated observations can now be compared. In this model, each object will be assigned a value to denote for its probability of being a skyline point. When we are using this model, some questions will naturally be asked: (1) Which of the objects have skyline probabilities larger than a given object? (2) Which of the objects are the K nearest neighbors to a given object according to their skyline probabilities? (3) What is the ranking of these objects based on their skyline probabilities? Up to our knowledge, no existing work answers any of these questions. Yet, answering them is not trivial. For just a medium-size dataset, it may take more than an hour to obtain the skyline probabilities of all the objects in there. In this paper, we propose a tree called SPTree that answers all these queries efficiently. SPTree is based on the idea of space partition. We partition the dataspace into several subspaces so that we do not need to compute the skyline probabilities of all objects. Extensive experiments are conducted. The encouraging results show that our work is highly feasible.

1 Introduction

Given a set of objects, X, let U and V be two different objects in X. U is said to dominate V if U performs better than V in at least one dimension and not worse than V in all other dimensions. An object that cannot be dominated by any other object in X is called a skyline point. A collection of skyline points formulates a skyline. Traditionally, skyline points cannot be compared with each other because each skyline point performs superior than the other skyline points in some, but not all, dimensions, and different dimension is incomparable.

Yet, thanks to the idea of probabilistic skyline [1], skyline points can now be compared with each others if each skyline point contains repeated observations. For example, if we want to evaluate a basketball team, we will first collect the games (observations) that this team is involved, then identify the number of games that this team performed at skyline point, and finally obtain a probability to denote how likely this team can be a skyline point (i.e. skyline probability). Accordingly, we can compare the performances of all teams, as they should have different skyline probabilities.

* Wei Lu's and Xiaoyong Du's research is partially supported by National Natural Science Foundation of China under Grant 60573092 and Grant 60873017.

X. Zhou et al. (Eds.): DASFAA 2009, LNCS 5463, pp. 263–277, 2009.

When we are using the probabilistic skyline model, three questions will naturally be asked: (1) Which of the objects have skyline probabilities larger than a given object (Extract dominant probabilistic skyline points)? (2) Which of the objects are the K nearest neighbors to a given object according to their skyline probabilities (Identify K nearest probabilistic skyline points)? (3) What is the ranking of these objects based on their skyline probabilities (Rank the extracted probabilistic skyline points)? Up to our knowledge, no existing work can effectively answer any of the above questions. Yet, answering them is not trivial. It can take more than an hour to compute the skyline probabilities of all the objects in a 3 dimensional dataset with just 100 objects and each object contains 2,000 observations

In this paper, we propose a novel tree structure, called Space-Partitioning Tree (SP-Tree), that can answer all of the above questions efficiently. Our motivations and contributions are both solid. The idea of SPTree is came from space partitioning. Extensive experiments are conducted to evaluate the efficiency of our work. The results are highly encouraging. The rest of the paper is organized as follows – Section 2 defines our problem; Section 3 presents SPTree in details; Section 4 reports the experimental results; Section 5 discusses some important related works; Section 6 concludes this paper.

2 Problem Definition

Table 1 shows the symbols that would be used in this paper. Let $D = \{d_1, d_2, \ldots, d_N\}$ be a N-dimensional space, where d_i is a dimension. Given d_i, the smaller the value it is, the more preferable it will be. Let X be a set of objects in D.

By definition, given two objects, $U \in X$ and $V \in X$, and their observations, $u \in U$ and $v \in V$, u is said to dominate v, $u \prec v$, iff the following two conditions hold: (1) $\forall d_i \in D, u.d_i \leq v.d_i$; (2) $\exists d_i \in D, u.d_i < v.d_i$. The skyline probability of U, $P(U)$ is [1]:

$$P(U) = \frac{1}{|U|} \sum_{u \in U} P(u), \quad P(u) = \prod_{V} \left(1 - \frac{\|v \in V | v \prec u\|}{|V|}\right). \tag{1}$$

In this paper, we define the following new concepts:

Definition 1. *p*-dominant and *p*-dominant object *An object, V, is said to p-dominant another object, U, iff $P(V) > p$ and $P(U) = p$. Here, V is a p-dominant object.*

Table 1. Symbols and their meanings

Symbols	Meanings
D, d_i	D is an N dimensional space; d_i is a dimension of D, $d_i \in D$
X	A set of objects in D
U, u	U is a target object, $U \in X$; u is an observation, $u \in U$
V, v	V is any object other than U, $V \in X, V \neq U$; v is an observation of V, $v \in V$
$P(U)$ $(P(V))$	The skyline probability of U (V)
$P(u)$ $(P(v))$	The skyline probability of u (v)
$P_u(U)$ $(P_u(V))$	The upper bound skyline probability of U (V)
$P_l(U)$ $(P_l(V))$	The lower bound skyline probability of U (V)
S, V^s	S is a subspace after space partition; V^s is a subset of V drops in S, $V^s \subseteq V$
V^s_{min} (V^s_{max})	The min (max) point of the minimum bounding rectangle of V^s

Definition 2. *K*-nearest *p*-skyline, S_K *Given U with $P(U) = p$, K-nearest p-skyline is a set of K objects that is most nearest to U according to the skyline probabilities.*

Given an object, U, we try to solve the following three problems: (1) Extract objects that p-dominant U; (2) Rank p-dominant objects; and (3) Identify K-nearest p-skyline with respect to U. Note that computing $P(U)$ is trivial if we obtained all $P(u)$. Unfortunately, computing $P(u)$ is very time consuming because $\forall u \in U$ we have to identify how many $v \in V$ cannot dominate it. This is the major bottleneck.

3 Proposed Work

We propose a novel tree, Space-Partitioning Tree (SPTree), to answer the three questions stated in the previous section. In the followings, we first present the motivation of SPTree, then give an overview of it, and finally describe its implementation detail.

3.1 Motivation

Since computing the skyline probabilities of all the objects in a dataset is very expensive, we therefore try to think whether it is possible for not to compute their exact values, but to identify their range only, for solving our problems. Given U and $P(U)$, We observe that our problems have some interesting properties:

First, we only need to return the objects that satisfy the queries, but not the skyline probabilities. Second, let $P_l(V)$ and $P_u(V)$ be two probabilities, such that $P_l(V) \leq P(V) \leq P_u(V)$. If $P_l(V) \geq P(U)$, then V must be a p-dominant object. Similarly, if $P_u(V) < P(U)$, then V will never be a p-dominant object. Third, if we can obtain all $P_l(V)$ and all $P_u(V)$, without computing any $P(V)$, then we can immediately conclude some of their relationships with U. We only need to compute $P(V)$ for those V that their relationships with U are ambiguous. Forth, let $V' \neq V$ be another object in X. For ranking the p-dominant objects, if $\exists V', P_l(V) \geq P_u(V')$, then V must be ranked higher than V'. Furthermore, if $P_l(V) \geq P_u(V'), \forall V'$, then V must have the highest skyline probability. Similar for the case of $P_l(V) < P_u(V')$. Fifth, let $D_{min}(V,U)$ and $D_{max}(V,U)$ be the min difference and max difference between $P(V)$ and $P(U)$, respectively. For identifying K-nearest p-skyline, if there are K or more objects with max differences to $P(U)$ smaller than $D_{min}(V,U)$, then V can be pruned away. Similarly, $\forall V' \in X, V' \neq V$, if $D_{max}(V,U) \leq D_{min}(V',U)$, then V must belong to the K-nearest p-skyline of U.

To conclude, we use "range values" ($P_l(V)$ and $P_u(V)$) instead of "exact value" ($P(V)$) to solve our problem. Now our question is: how to compute $P_l(V)$ and $P_u(V)$?

3.2 Space Partitioning Approach

Given U, without any prior information, if we want to extract the objects which are p-dominant to U or are K-nearest to U, the first step is certainly to compute $P(U)$. Once we have obtained $P(U)$, we can utilize it to facilitate other processes. Thus, we build a tree (SPTree) according to U to index all the other objects.

Fig. 1 respectively show a sample dataset and the corresponding SPTree. In Fig. 1, there are three objects, A, B and U. To construct the SPTree, we first need to partition

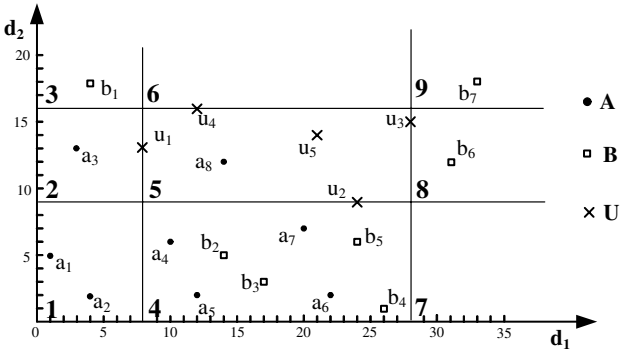


Fig. 1. An example dataset

the dataspace into subspaces based on the min and max values of U in each dimension. E.g. There are 9 subspaces in Fig. 1. Let S be a subspace. In Fig. 1, $S \in \{1,2,\ldots,9\}$. Only subspace 5 contains u. Observations of other objects may drop into more than one subspaces. Let $V^S \subset V$ be a set of observations drops in S. E.g. $A^1 = \{a_1, a_2\}$. We conduct the space partition because of the following reasons:

Recall that $P(u)$ is already computed. According to Theorem 2 in Appendix, if $\exists u \prec v, \forall v \in V^s$, then we can identify $P_u(V^s)$ by u, where

$$P_u(V^s) \leq p\left(\frac{|V^s| \times (|U| - |\{u \in U | u \prec v\}|)}{|U| \times (|V^s| - |\{v \in V^s | v \prec u\}|)}\right),\ p = \min_{u \in U}\{P(u) \mid u \prec v, \forall v \in V^s\}. \quad (2)$$

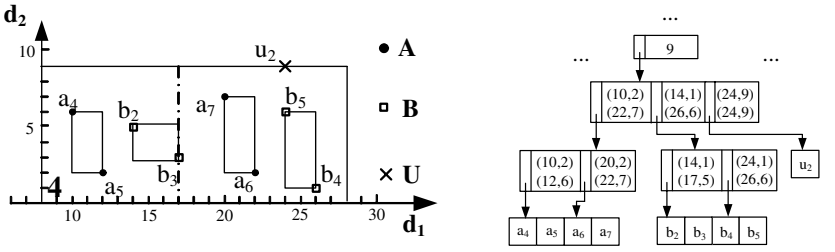
This process is very efficient because identifying the dominant relationship is simple [2] and we only need to consider U and a particular V. Similarly, we can obtain:

$$P_l(V^s) \geq p\left(\frac{|V^s| \times (|U| - |u \in U | u \prec v|)}{|U| \times (|V^s| - |\{v \in V^s | v \prec u\}|)}\right),\ p = \max_{u \in U}\{P(u) \mid v \prec u, \forall v \in V^s\}. \quad (3)$$

Hence, partitioning the dataspace can compute $P_u(V^S)$ and $P_l(V^S)$ efficiently.

There are two cases where we cannot use the above equations to compute $P_l(V^s)$ and $P_u(V^s)$: (1) We cannot find any pair of (u, v) that satisfies Eq. (2) or Eq. (3); and (2) We cannot further refine $P_l(V^s)$ and $P_u(V^s)$ by *recursively partitioning* the subspaces. We will explain case (2) shortly. If we cannot use Eq. (2) or Eq. (3) to compute $P_l(V^s)$ and $P_u(V^s)$, we have to compute them with the help of other objects. Yet, we do not need to consider all observations in the dataset but just need to consider the observations in the subspaces that may dominate U. E.g. in Fig. 1, if we compute $P_l(A^8)$ and $P_u(A^8)$, we only need to consider the observations of A and B in the subspaces 2, 5, 7 and 8 because objects in the subspace 1 and 4 will always dominate objects in subspace 8, and no objects in subspaces 3, 6 and 9 can dominate objects in subspace 8.

Let V^s_{min} and V^s_{max} be two points represent the min point and the max point of the minimum bounding box of V^s. Let $V'^s \neq V^s$ be another subspace. If either $V'^s_{max} \prec V^s_{min}$ or $V'^s_{min} \not\prec V^s_{max}$, then we do not need to compare the observations in V'^s with V^s. Hence, with space partition, the number of comparisons among objects is reduced dramatically,



(a) Objects split in Subspace 4 (b) The corresponding nodes split in SPTree

Fig. 2. Nodes split in Subspace 4

whereas identifying V^s_{max} and V^s_{min} are trivial. Mathematically,

$$P_u(V^s) = \prod_{O \in X, O \neq V} \frac{\sum_{\forall O^{s'} \in O} |O^{s'}, O^{s'}_{max} \prec V^s_{min}|}{|O|},\ P_l(V^s) = \prod_{O \in X, O \neq V} \frac{\sum_{\forall O^{s'} \in O} |O^{s'}_{min} \prec V^s_{max}|}{|O|}. \quad (4)$$

Once we obtained $P_l(V^s)$ and $P_u(V^s)$, we can update $P_l(V)$ and $P_u(V)$ incrementally:

$$P_l(V) = \sum_{V^s \in V} P_l(V^s) \times \frac{|V^s|}{|V|}, \quad P_u(V) = \sum_{V^s \in V} P_u(V^s) \times \frac{|V^s|}{|V|}. \quad (5)$$

Note that computing $P_l(V^s)$ and $P_u(V^s)$ is far more efficient than computing $P_l(V)$ and $P_u(V)$ because the search space in a subspace is far less than the whole space.

Let us refer to Fig 1 again. Assume that $P(U) = 0.4$. Suppose we obtained $0.6 \leq P(A) \leq 0.8$ and $0.45 \leq P(B) \leq 0.7$ after the above computation. Since $P_l(B) > P(U)$, B p-dominant U. For A, its relationship with U is undefined as its range overlap with $P(U)$. We therefore need to refine $P_l(A)$ and $P_u(A)$. The refinement is based on *recursively partitioning* each subspace further into two disjoint subspaces and compute $P_l(V^s)$ and $P_u(V^s)$ for each resulting subspace until we can distinguish the relationship between U and V. Two rules governed how we the partition the subspace. Rule 1: the total number of observations for all objects whose relationship with U are unknown have to be the same in both subspaces. By maintaining the subspaces with equal number of observations, we can usually converge to a solution much faster. Rule 2: partition a subspace depends solely on one particular dimension, d_i, where d_i varies for each iteration. Specifically, given two consecutive subspaces, S_1 and S_2 where $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$, if $v.d_i < \delta, v \in V^s$, then v will be classified into S_1, otherwise it will be classified into S_2. Finally, let us use an example to illustrate how d_i is chosen. Assume $D = \{d_1, d_2\}$. The first and second iteration of the recursive partition will respectively depend on d_1 and d_2. The third iteration will depend back to d_1 again. For identifying δ, it is trivial once we constructed the SPTree as we will discuss it shortly. With these two rules, we can guarantee the size of the subspaces will not easily be biased by some dimensions, meanwhile the computational cost for each iteration will be less as we only need to partition the data according to one dimension at a time.

Let us refer to Fig. 1 again. Suppose the relationship of both A and B with U are ambiguous, then we need to partition the subspaces. Let us take Subspace 4 as an example. Fig. 2 shows the SPTree after nodes split in Subspace 4. According to Rule 1,

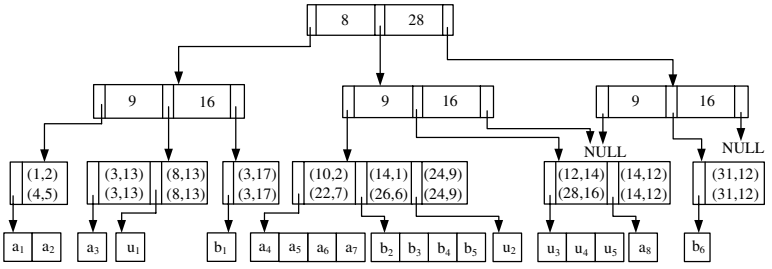


Fig. 3. An SPTree

we should partition Subspace 4 into two and each partition should have 4 observations. According to Rule 2, we partition the subspace based on d_1.

Let $\mathcal{V} \subseteq X$ be a set of objects p-dominant U. For the problem of extracting $\mathcal{V}$, it can be solved by applying the aforementioned framework. For ranking $\mathcal{V}$, we can identify the relationships among all $V \in \mathcal{V}$ by applying the same framework for each V. Finally, for identifying the K-nearest p-skyline, we can apply this framework to identify the relationship among all $V \in X$, and changing the ranking criterion from skyline probabilities to distance between $P(U)$ and $P(V)$. For implementing this framework, unfortunately, it is not simple. Deriving an appropriate data structure for efficient processing and identifying some special rules to further enhance the computational time are two non-trivial problems. In this paper, a Space Partitioning Tree (SPTree) is proposed.

3.3 SPTree Construction

Fig. 3 shows the SPTree for the dataset in Figure 1. Each node in the first N levels (N is the number of dimensions, i.e. $N = 2$ in this case) of a SPTree contains 2 keys and 3 pointers. Each pointer points to a node in the next level. For the nodes at the same level, they store the same pair of keys which denote the minimum and the maximum values of U at a particular dimension, i.e. at level i, the left key is $\min_{u \in U} u.d_i$ and the right key is $\max_{u \in U} u.d_i$. Each node at the $N+1$ level of SPTree corresponds to a subspace, S. The number of entries in a leave node depends on the number of objects that has observation in the corresponding S. Each entry corresponds to a V^s. It contains a pair of coordinations which denote V^s_{min} and V^s_{max} (i.e. the minimum bounding rectangle of V^s) and a pointer pointing to the first observation, v, in a leave node that stores all $v \in V^s$.

The SPTree may grow dynamically according to the case if a subspace requires recursively partition. Nodes would be added between the $N+1$ level of SPTree and the leave nodes. Fig. 2 (b) shows what would happen to the SPTree if we need to partition Subspace 4 in Fig. 1 (a). Fig. 2 (a) magnifies this subspace to include more details. In Subspace 4, there are 8 observations: a_4, a_5, a_6, a_7, b_2, b_3, b_4 and b_5. According to Rule 1, we should partition the subspace into two such that each of them contains 4 observations. According to Rule 2, we should partition the subspace based on only one of the dimensions. Suppose we based our decision on dimension d_1. Then, a_4, a_5, b_2 and b_3 will be grouped into one partition, and a_6, a_7, b_4 and b_5 will be grouped into another one. Hence, we will attach two nodes into the forth node (from left to right) at the 3rd level of the SPTree in Fig. 1. The result is shown in Fig. 2 (b). For the two attached nodes, each

Algorithm 1. `createSPTree(`U, X`)`

input : A target (U) and a set of objects ($X, U \notin X$)
output: SPTree, T

```
1  create first N level of T; // N is the number of dimensions
2  foreach V ∈ X do
3      foreach v ∈ V do
4          S ← T.root; // T.root is the root node of T. S is a node
5          for i ← 1 to N do
6              if v.d_i < S.key_1 then S ← S.pointer_1;
7              else if S.key_1 ≤ v.d_i < S.key_2 then S ← S.pointer_2;
8              else S ← S.pointer_3;
9          end
10         insert v into S group by (V, d_1);
11         incremental update V^s_min and V^s_max according to v ∈ V;
12     end
13 end
14 return T;
```

of them contains two entries. The meanings of these entries are the same as the entries of their immediate parent nodes. Note that the observations in the leave nodes are group by S, such that the pointers in their parents node can point to the correct entries.

Algorithm 1 outlines how SPTree is constructed. Line 1 creates the first N levels of the SPTree according to the minimum and the maximum values of U in each dimension. This process is trivial, so we will not elaborate it further. Line 2 – 13 show how each observation, $v \in V$, is being inserted into the correct leave node in the SPTree, which is self-explained. Algorithm 2 outlines the major steps for identifying the p-dominant objects, $\mathcal{V}$. First, we compute $P(U)$ and $P(u)$ (line 1). All u are indexed by R-tree for efficient computation. All $P_u(V^s)$ and $P_l(V^s)$ are initialized to 1 and 0, respectively (line 2). After some parameters initialize (line 3), the main loop begins from line 4 to 23. In line 6 – 17, for each $v \in V^s, V \in X$, we try to update the values of $P_u(V)$ and $P_l(V)$ with the help of $P_u(V^s)$ and $P_l(V^s)$ according to equations Eq. (2) to Eq. (5). Once we identify the relationship between V and U, V will be removed from X (line 11 – 16). Finally, if the size of V^s is larger than 1 (i.e. the number of observations of an object that drops in S) and the relationship between V and U is not yet identified (i.e. $V \in X$), then we will partition S into two subspaces (line 20). Algorithm 3 outlines the partition process. It tries to get a value, δ (line 2), that guide us how the observations in S should be assigned to the two new nodes (line 3). δ is obtained by first sorting all the observations in S by a given dimension, d_i, (line 1) and then extract the observation, v, which is at the median position of S after sorting, and δ is simply the value of $v.d_i$ (line 2). Line 4 – 7 try to assign the observations to the correct node, which are same explained. Finally, we link all of them together (line 8 – 10).

Up to now, although we have only shown how we can extract the p-dominant objects, $\mathcal{V}$, from X, we can still use this similar framework to rank $\mathcal{V}$ or to identify the K-nearest p-skyline, which has already been discussed in the last paragraph of the last section. As

Algorithm 2. `overview(`U`,` X`)`

input : A target (U) and a set of objects ($X, U \notin X$)
output: p-dominant skyline points ($\mathcal{V}$)

```
1  compute P(U) and P(u), ∀u ∈ U; // Base on Eq. (1)
2  P_u(V^S) ← 1, ∀V^s;   P_l(V^S) ← 0, ∀V^s;
3  V ← ∅;   T ← createSPTree(U, X);   i ← 1;
4  repeat
5      foreach P ∈ T (P is a parent node of a leave node) do
6          foreach S ∈ T (S is an entry) do
7              foreach v ∈ V^S, V ∈ X do
8                  update P_u(V) and P_l(V) according to Eq. (2) and Eq. (3);
9                  if P_u(V) is not changed then update P_u(V) according to Eq. (4);
10                 if P_l(V) is not changed then update P_l(V) according to Eq. (4);
11                 update P_l(V) and P_u(V) according to Eq. (5);
12                 if V is a p-dominant object then
13                     remove V from X;
14                     add V into V;
15                 else if V can be pruned (see Section 3.4) then
16                     remove V from X;
17                 end
18             end
19         end
20     end
21     foreach P ∈ T (P is a parent node of a leave node) do
22         foreach S ∈ L (S is an entry) do
23             if ∃V ∈ X and |V^s| > 1 then partition(S, d_i);
24         end
25     end
26     i ← (i = N) ? 1 : i + 1; // N is the number of dimensions
27 until X = ∅ ;
28 return V;
```

the modification is rather simple, we do not attempt to further elaborate it due to the space limitation.

3.4 Pruning Strategies

Given a SPTree, T, after we compute $P_l(V^s)$ and $P_u(V^s)$, if $P_u(V^s) = 0$, then all the observations in V^s must be dominated by some other objects, we can then prune away V^s from T. If $P_l(V^s)$ equals to $P_u(V^s)$, then all $P(v)$ for $v \in V^s$ will be the same. So the offspring of V^s do not need to compute. For the problem of ranking p-dominant objects, if $P_u(V) < P(U)$, then V can be pruned away. However, the reverse is not true. If $P_u(V) > P(U)$, then even V can dominate U, we cannot prune it because we have to rank it. Finally, if $P_l(V) \geq P_u(V')$ for all $V' \in X, V' \neq V, V' \neq U$, then V would be the object with the highest skyline probability. Once we have identified this object, then

Algorithm 3. `partition`(S, d_i)

input : An entry, S, and a dimension, d_i
1 sort all $v \in V^s$ at the leaf-node by d_i;
2 $\delta \leftarrow v.d_i$ where v is the observation at the median position;
3 create a new node, L, with two entries, S_1 and S_2;
4 **foreach** $v \in V^s$ **do**
5 | **if** $v.d_i \leq \delta$ **then** incremental update $V^{s_1}_{min}$ and $V^{s_1}_{max}$ by v;
6 | **else** incremental update $V^{s_2}_{min}$ and $V^{s_2}_{max}$ by v;
7 **end**
8 $S.pointer \leftarrow L$;
9 $S_1.pointer \leftarrow$ first position of v in V^s at the leaf-node;
10 $S_2.pointer \leftarrow$ first v with $v.d_i = \delta$ in V^s at the leaf-node;

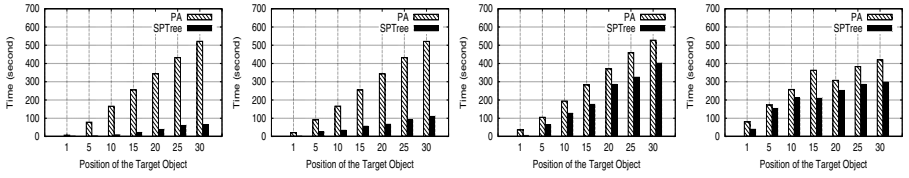
we can recursively use this rule to identify the object with the second highest skyline probability, and so on so forth.

3.5 Complexity

Let us call the nodes in the first N level of a SPTree as space nodes and all the other nodes as data nodes. For an N dimensional space, there will be at most 3^N space nodes. Each node contains one pointer and one value. For the space complexity, the best case is that all observations drop within one subspace. The worst case is that each subspace has at least one observation. Hence, the space complexities for space nodes are $O(1)$ and $O(3^N)$ for the best case and the worse case, respectively. Since V^s will be partitioned if we cannot identify the relationship between V and U, the best case is that no object is split and the worst case is that each object will be split until each data node has only one observation. Let n be the total number of observations in the dataset. The space complexities for data nodes are $O(n)$ and $O(4n-1)$ for the best case and the worse case, respectively. Combining all together, the space complexity for the average case would be $(3^N + n)$.

4 Experimental Study

All experiments are conducted with Intel 2.20GHz dual core CPU and 1GB memory using Java. We employed four datasets: (1) Uniformly Distributed (*UD*) – We generate the center of 200 objects randomly, and randomly generate 2,000 observations for each object. None of the objects are overlapped. (2) Slightly Overlapped (*SO*) – We generate this dataset in the way similar to that of *UD* except that the objects can be overlapped. On average, an object will be overlapped with 9 objects. (3) Highly Overlapped (*HO*) – We generate this dataset in the way similar to that of *SO* except that an object will be overlapped with 34 objects on average. (4) NBA – It is downloaded from `http://www.databasebasketball.com`. It contains 339,721 records of 1,313 players (from 1991 to 2005). Each player is treated as an object, and the records of the same player represent his observations. Each record has three attributes: points, assistants and rebounds.



(a) UD (b) SO (c) HO (d) NBA

Fig. 4. Retrieval and ranking performances of the dominant probabilistic skylines

4.1 Dominant Skylines Identification

For each dataset, we conduct seven experiments. The first experiment takes the object with the highest skyline probability as U, and then extract the objects, V, with $P(V) > P(U)$. Since U has the highest skyline probability, no object will be extracted. The rest of the experiments are conducted in the way similar to the first experiment, except that we respectively take the object with the 5^{th}, 10^{th}, 15^{th}, 20^{th}, 25^{th} and 30^{th} highest skyline probability as U in each different experiment.

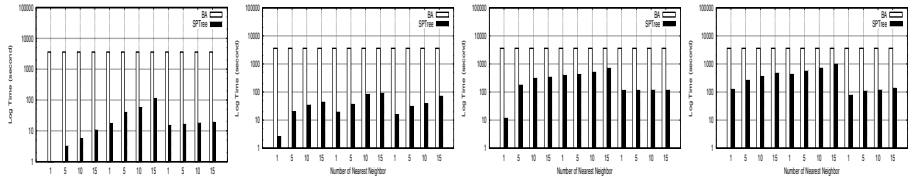
For comparison, we implement this approach as described in [1][1]. We call it as baseline approach (BA). This approach identifies all V with $P(V)$ larger than a *user-defined value* without computing the exact $P(V)$. Note that this approach cannot rank V. We slightly modify this algorithm for comparison as follows: (1) Given U, we first compute $P(U)$ to denote the user-defined value; (2) Use BA to extract all V with $P(V) > P(U)$; (3) We compute the exact $P(V), \forall V$ in order to rank them.

Figure 4 (a), 4 (b), 4 (c) and 4 (d) show the performances of extracting and ranking the dominant probabilistic skylines. In the figures, x-axes denote the ranking of U and y-axes denote the computational time (in sec). The black charts denote the performances of our SPTree and the shaded charts denote the baseline approach (BA).

At the first glance, SPTree excels BA for all datasets and all situations, especially for the datasets *UD* and *SO*. SPTree takes 0 – 65 sec for *UD*, 1 – 110 sec for *SO* and 6 – 400 sec for *HO*. For a reference, computing the skyline probabilities of all objects requires more than 3,600 sec. The computational time of SPTree on the synthetic datasets is roughly: $(UD > SO > HO)$.

For the NBA dataset, SPTree seems does not have a clear pattern, i.e. even the given U is at a lower ranking, the computational time may not necessary be longer. For example, the time for extracting and ranking the dominant probabilistic skylines when U is at the 15^{th} rank is faster than that at 10^{th}. This is interesting because, intuitively, a higher rank object should take less computational time, as there are fewer objects dominating it, so that we only need to extract and rank fewer objects. But this is not the case. As such, we analysis the time spent on the most time consuming part of the algorithm – split and partition. As expected, more number of split and partition is needed if the computational time is higher. In order to understand why we have to perform more number of split and partition, we take a closer look at the distribution of the objects. We found out the number of split and partition we need depends on the overlapping between U

[1] In their paper, they have proposed two approaches, namely: (1) Top-Down; and (2) Bottom-Up. We have implemented the former one, because its excels the later one in most situations.



(a) UD (b) SO (c) HO (d) NBA

Fig. 5. Performances of the K nearest neighbor p-skylines

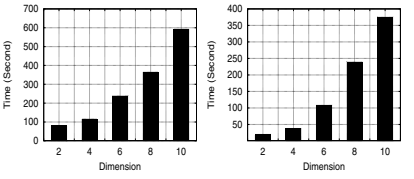
and the other objects. As such, we draw the following conclusion: (1) It is merely the distribution of the observations that affect the algorithm performance; and (2) There is no direct relationship between the skyline probabilities and the computational time.

4.2 K Nearest Skylines Identification

For each dataset, we first identify the objects with: (1) Highest skyline probability; (2) Lowest skyline probability; and (3) Median skyline probability, so as to serve as our target object, U. Then, we try to extract K objects that are nearest to each of these three objects. Here, $K = \{1, 5, 10, 15\}$. As a result, for each dataset, we have to conduct $3 \times 4 = 12$ experiments. Since no existing work tries to address this problem, we have implemented the following approach: Given a set of objects, we compute all of their skyline probabilities, and then identify the K nearest neighbor of U accoriding to the skyline probabilities. We call it as Baseline Approach (BA).

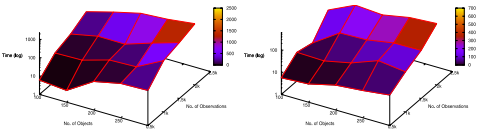
Figure 5 (a), 5 (b), 5 (c) and 5 (d) show the retrieval performances of obtaining the K nearest probabilistic skylines. In the figures, x-axes denote different K, and y-axes is the computational time (*in log scale*) measured in seconds. The black charts denote our proposed work (SPTree) and the white charts denote the Baseline Approach (BA). The first four pairs, middle fours pairs, and last fours pairs of charts represent the time required for identifying the K nearest probabilistic skylines of U, where U is the object at the top, middle and bottom ranking according to the skyline probability.

From the figures, it is obvious that SPTree outperforms BA significantly (note that the y-axes are all in log scale). This is not surprise as we have several efficiency strategies to improve our identification process. The computational efficiency also follows the same conclusion as in the previous section: (UD $>$ SO $>$ HO). In general, the computational time increases linearly according to the number of nearest probabilistic skylines, K, which we have to extract. In addition, for SPTree, the middle four charts usually perform inferior than the first four charts and the last four charts. This finding suggests that the position of the target object, U, may affect the computational time. The reason is that overlapping among objects with skyline probabilities dropped in the middle range is usually highest. Moreover, the skyline probabilities of these objects are usually very similar. This will increase the computational cost significantly. For the NBA dataset, the best case is the last four charts, which is the objects with the lowest skyline probabilities. This is not surprise if we understand the data distribution in there. In NBA dataset, the lowest overlapping is the area where those players have



(a) (b)

Fig. 6. Effect of the dimensionality. (a) Ranking dominant p-skyline. (b) Identifying K nearest p-skyline.



(a) (b)

Fig. 7. Relationship between no. of instances and no. of objects. (a) Ranking dominant p-skyline (b) Identifying K nearest p-skyline.

the lowest skyline probabilities, whereas both the top and middle range are heavily overlapped.

4.3 Further Discussion and Analysis

In this section, we analyze the following two issues: (1) The effect of dimensionality, and (2) The relationship between the number of objects and the number of instances. We conduct the following analysis: (1) Case 1 (Dimensionality) – We conduct two set of experiments, one for ranking the dominant probabilistic skylines, and the other one for identifying $K = 10$ nearest probabilistic skylines. The dataset that we use is similar to that of *SO* except that the dimensionality varies from $\{2,4,6,8,10\}$. For both experiments, U is the one with the 10th highest skyline probability. (2) Case 2 (Number of Objects and Instances) – We conduct two sets of experiments with the parameters and settings similar to Case 1, except that $N = \{100,150,200,250,300\}$ and $M = \{500,1000,1500,2,000,2,500\}$.

Figure 6 (a) and 6 (b) show the evaluation results of Case 1. The x-axes represent the dimensions of the dataspace, and the y-axes represent the time in seconds. When the dimensionality increases linearly, the computational time increases exponentially. This is expected because when the number of dimension increases linearly, the number of skyline point will be increased exponentially. As the number of skyline points increase dramatically, the number of comparison among points must also increase dramatically. Furthermore, if the dimensionality increases linearly, the number of subspaces will increase exponentially. This further contributions to the increase of computational cost.

Figure 7 (a) and 7 (b) show the evaluation results about the relationship between objects and observations (Case 2). In the figures, x-axes denote the total number of instances, y-axes represent the total number of observations, and z-axes are the computational time measured in second. From the figures, we have two important findings. The first finding is: the computational cost increases slowly if the number of objects increases while keeping the number of observations belong to an object is the same. Yet, the computational cost increases quickly if the number of observations increases while keeping the number of objects are the same. The reason is that the time required for splitting the nodes in the SPTree will be fewer if the number of observations is fewer. The second finding in this experiment is: the computational time required for identifying the K nearest p-skyline is much more stable and much less than that of ranking the dominant p-skyline. This is because if we rank the dominant p-skylines, we need to

distinguish the partial relationship between every two skyline points, which is not the case for the K nearest p-skyline identification.

5 Related Work

Skyline query has been proposed for several years. Borzsonyi et.al [3] are the first to introduce skyline query into database community and extend the SQL statement to support the skyline query. In their work, four algorithms, BNL (Block-nested-loop) based nested-loop, DC (divide and conquer), B-tree based and R-tree based index, are proposed to compute the skyline. DC algorithm recursively partitions the data set into several parts and identifies skyline in each part. Then the separated parts will be recursively merged into one part. Sort-filter-skyline (SFS) algorithm, based on nested-loop as well, is proposed by Chomicki et al. [4]. SFS presorts the dataset in non-decreasing order with the key value of each point such that each point only needs to be compared with the points ranking before itself. Tan et al. [5] propose progressive skyline computation methods based on Bitmap and B-tree index. Kossmann et al. [6] propose a new online algorithm based on the nearest neighbor (NN) search to improve their methods. However, the NN algorithm needs to traverse the R-tree for several times and eliminate duplicates if dimension is larger than 2. Papadias et al. [7] propose a branch-and-bound algorithm (BBS) based on nearest neighbor search as well and indexed by R-tree. This algorithm is IO optimal and traverses the R-tree for only once.

Recently, researchers mainly focus on the following three aspects. Firstly, decrease the number of skyline points [8,9] when the dimensionality is high or the dataset is anti-correlated. Secondly, some variations of skyline query are proposed, including reverse skyline [10], multi-source skyline query [11], subspace skyline [12,13,14,15]. Thirdly, combine skyline query with other techniques, including continuously computing skyline in data streams [16,17,18], computing skyline in a distributed environment [19,20] and computing probabilistic skyline in uncertain data [1].

Most previous work focuses on computing skyline on certain data except the probabilistic skyline on uncertain data. Here, our work focuses on uncertain data as well, but we have different motivations with probabilistic skyline as explained before. We extend the probabilistic skyline to dominant and K nearest probabilistic skyline.

6 Conclusion and Future Work

In this paper, we proposed a novel algorithm, called SPTree, to answer the following questions: (1) Which of the objects have the skyline probabilities that dominate (larger than) the given object? (2) What is the skyline probability ranking of the objects that dominate the given object? (3) Which of the objects are the K nearest neighbors to the given object according to the skyline probabilities? Extensive experiments are conducted to evaluate it. The encouraging results indicated that SPTree is highly practical. However, SPTree is far from perfect. It may not be very efficiency if the dimensionality is high meanwhile the overlapping among object is huge. So, our future work will investigate this issue.

References

1. Pei, J., Jiang, B., Lin, X., Yuan, Y.: Probabilistic skylines on uncertain data. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007) (2007)
2. Papadias, D., Tao, Y., Fu, G., Seeger, B.: Progressive skyline computation in database systems. ACM Trans. Database Syst (TODS) 30(1), 41–82 (2005)
3. Borzsonyi, S., Kossmann, D., Stocker, K.: The skyline operator. In: Proceedings of the 17th International Conference on Data Engineering (ICDE 2001) (2001)
4. Chomicki, J., Godfrey, P., Gryz, J., Liang, D.: Skyline with presorting. In: Proceedings of the 19th International Conference on Data Engineering (ICDE 2003) (2003)
5. Tan, K.L., Eng, P.K., Ooi, B.C.: Efficient progressive skyline computation. In: VLDB 2001, Proceedings of 27th International Conference on Very Large Data Bases (VLDB 2001) (2001)
6. 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 (VLDB 2002) (2002)
7. 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 (SIGMOD 2003) (2003)
8. Chan, C.Y., Jagadish, H.V., Tan, K.L., Tung, A.K.H., Zhang, Z.: Finding k-dominant skylines in high dimensional space. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006) (2006)
9. Lin, X., Yuan, Y., Zhang, Q., Zhang, Y.: Selecting stars: The k most representative skyline operator. In: Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007) (2007)
10. Dellis, E., Seeger, B.: Efficient computation of reverse skyline queries. In: Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB 2007) (2007)
11. Deng, K., Zhou, X., Shen, H.T.: Multi-source skyline query processing in road networks. In: Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007) (2007)
12. Yuan, Y., Lin, X., Liu, Q., Wang, W., Yu, J.X., Zhang, Q.: Efficient computation of the skyline cube. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005) (2005)
13. Pei, J., Jin, W., Ester, M., Tao, Y.: Catching the best views of skyline: A semantic approach based on decisive subspaces. In: Proceedings of the 31st International Conference on Very Large Data Bases (VLDB 2005) (2005)
14. Tao, Y., Xiaokui Xiao, J.P.: Subsky: Efficient computation of skylines in subspaces. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006) (2006)
15. Tian Xia, D.Z.: Refreshing the sky: the compressed skycube with efficient support for frequent updates. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006) (2006)
16. Mouratidis, K., Bakiras, S., Papadias, D.: Continuous monitoring of top-k queries over sliding windows. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 2006) (2006)
17. Lin, X., Yuan, Y., Wang, W., Lu, H.: Stabbing the sky: Efficient skyline computation over sliding windows. In: Proceedings of the 21st International Conference on Data Engineering (ICDE 2005) (2005)
18. Tao, Y., Papadias, D.: Maintaining sliding window skylines on data streams. IEEE Trans. Knowl. Data Eng. (TKDE) 18(2), 377–391 (2006)
19. Cui, B., Lu, H., Xu, Q., Chen, L., Dai, Y., Zhou, Y.: Stabbing the sky: Efficient skyline computation over sliding windows. In: Proceedings of the 24th International Conference on Data Engineering (ICDE 2008) (2008)

20. Huang, Z., Jensen, C.S., Lu, H., Ooi, B.C.: Skyline queries against mobile lightweight devices in manets. In: Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006) (2006)

Appendix

Theorem 1 ([1]). *Let $V = \{v_1, ..., v_n\}$ be the object where $v_1, ..., v_n$ are the observations of V. Then $P(V_{min}) \geq P(V) \geq P(V_{max})$.*

According to Theorem 1, we can use $P(V_{min})$ and $P(V_{max})$ to compute $P_u(V)$ and $P_l(V)$ respectively.

However, when computing the values of $P(V_{min})$ and $P(V_{max})$, we still need to traverse all the observations of the other objects. Practically, based on the observations of U and any other object $V \in X$, we deduce the following two important theorems, by which we can compute $P_u(V)$ and $P_l(V)$ only using the two objects V and U.

Theorem 2. *Given $v \in V$, if $\exists u \in U$ s.t. $u \prec v$, then*

$$P(v) \leq P(u) \times \left(1 - \frac{|\{u \in U | u \prec v\}|}{|U|}\right)\left(\frac{|V|}{|V| - |\{v \in V | v \prec u\}|}\right)$$

Proof. *$\forall O \in X (O \neq V, O \neq U)$, $\forall o \in O$ satisfying $o \prec u$, then by the definition of skyline, $o \prec v$. Thus, $|\{o \in O | o \prec v\}| \geq |\{o \in O | o \prec u\}|$. Let $R = X - \{U, V\}$, then:*

$$\begin{aligned} P(v) &= \prod_{O \in R} (1 - \frac{|\{o \in O | o \prec v\}|}{|O|}) \times (1 - \frac{|\{u \in U, u \prec v\}|}{|U|}) \\ &\leq \prod_{O \in R} (1 - \frac{|\{o \in O | o \prec u\}|}{|O|}) \times (1 - \frac{|\{u \in U, u \prec v\}|}{|U|}) \\ &= P(u) \times (\frac{|V|}{|V| - |v \in V | v \prec u|}) \times (1 - \frac{|\{u \in U | u \prec v\}|}{|U|}). \end{aligned}$$

The result follows. □

When identifying the value of $P(V_{min})$, if there exists an observation $u \in U$ and $u \prec V_{min}$, according to Theorem 2, only U and V will be searched.

Theorem 3. *Given $v \in V$, if $\exists u \in U$ s.t. $v \prec u$, then*

$$P(v) \geq P(u) \times \left(1 - \frac{|u \in U | u \prec v|}{|U|}\right)\left(\frac{|V|}{|V| - |v \in V | v \prec u|}\right)$$

Proof. *Trivial. Similar to the proof of Theorem 2* □

Similarly, when identifying the value of $P(V_{max})$, according to the Theorem 3, if there exist an observation $u \in U$ and $V_{max} \prec u$, only U and V will be considered.