Selecting Stars: The k Most Representative Skyline Operator

Xuemin Lin[1] Yidong Yuan[1] Qing Zhang[2] Ying Zhang[1]

[1] School of Computer Science and Engineering
The University of New South Wale & NICTA, Australia
{lxue, yyidong, yingz}@cse.unsw.edu.au

[2] E-Health Research Center
CSIRO ICT Center, Australia
qing.zhang@csiro.au

Abstract

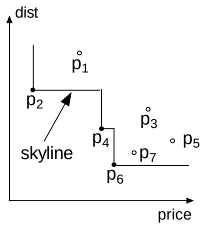
Skyline computation has many applications including multi-criteria decision making. In this paper, we study the problem of selecting k skyline points so that the number of points, which are dominated by at least one of these k skyline points, is maximized. We first present an efficient dynamic programming based exact algorithm in a $2d$-space. Then, we show that the problem is NP-hard when the dimensionality is 3 or more and it can be approximately solved by a polynomial time algorithm with the guaranteed approximation ratio $1 - \frac{1}{e}$. To speed-up the computation, an efficient, scalable, index-based randomized algorithm is developed by applying the FM probabilistic counting technique. A comprehensive performance evaluation demonstrates that our randomized technique is very efficient, highly accurate, and scalable.

1. Introduction

Given a set of d-dimensional points, the skyline consists of the points, called "skyline points", which are not dominated by another point. A point $p = (p[1], p[2], ..., p[d])$ *dominates* another point $q = (q[1], q[2], ..., q[d])$ iff $p[i] \leq q[i]$ (for $1 \leq i \leq d$) and there is at least one dimension j such that $p[j] < q[j]$. The skyline computation (or the skyline operator) is crucial to many multi-criteria decision making applications. A typical example is a list of hotels, each of which contains two numerical attributes *distance* (say, to the beach) and *price*, for on-line booking. Figure 1(a) shows a sample list. In this application, the best choice to a client, who wants to spend holiday in the beach, may be as close as possible to the beach while also cost effective. Consequently, the "best" choices form the skyline (see Figure 1(b)).

id	*dist* (km)	*price* ($)
p_1	4	150
p_2	3	110
p_3	2.5	240
p_4	2	180
p_5	1.7	270
p_6	1	195
p_7	1.2	210

(a) Hotels



(b) Skyline

Figure 1. A Skyline Example

Skyline computation has recently received a great deal of attention in the database community. A number of efficient algorithms for computing all skyline points (i.e. *full skyline*) have been reported in the literature [4, 9, 13, 22, 29, 33]. It has been shown in [3, 13] that the expected number of skyline points is $\Theta(\ln^{d-1} n/(d-1)!)$ for a random dataset. With the presence of a possibly large number of skyline points, the full skyline may be less informative. In the above example, it may be hard for users to make a good, quick selection by referencing the full skyline that consists of too many hotels.

To resolve this, a system may be required to provide k skyline points. Such k skyline points should be most representative. Consider that the full skyline represents a whole dataset. In this paper, we quantify the concept of "representative" by the population. Specifically, we investigate the problem (called "top-k representative skyline points") of computing k skyline points such that the total number of (distinct) data points dominated by one of the k skyline points is maximized. In the above example, p_6 is returned if $k = 1$, while p_2 and p_6 are returned if $k = 2$.

The computation of top-k representative skyline points (top-k RSP) may facilitate many applications including top-k queries when multi-criteria are involved. With respect to the above example, if users only want to see k hotels to make a selection based on the price and the distance, then a solution to the top-k RSP can provide users with the confidence that these k hotels are already representing (better than) the maximum possible number of available options (hotels). Moreover, the top-k RSP also provides a novel ranking mechanism for top-k queries.

Selecting data points with certain designated dominance properties has been recently investigated in [7, 8, 21, 29]. Nevertheless, to the best of our knowledge our top-k RSP problem is novel and it is inherently different than the problems in [7, 8, 21, 29]; consequently these existing techniques are not applicable to the top-k RSP problem. Motivated by this, in this paper we develop efficient, novel algorithms to compute the top-k representative skyline points. This is the first work regarding the problem of top-k RSP. Our contributions can be summarized as follows.

- We propose a novel skyline operator, top-k representative skyline, so that the k skyline points with the maximal number of dominated points can be produced to facilitate on-line user queries.
- In a $2d$-space, we develop an efficient dynamic programming based algorithm to solve the problem.
- We show that the problem is NP-hard in a d-dimensional space when d is 3 or more. Then, we show that by an immediate transformation to the set cover problem, the

problem can be solved by a greedy heuristic with the approximation ratio $1 - \frac{1}{e}$.

- Observe such a greedy heuristic may not be scalable nor efficient. We develop a novel, efficient, scalable, index-based randomized algorithm, with a theoretical accuracy guarantee, by using a probabilistic counting technique – FM algorithm [12].

Besides theoretical analysis, an extensive experimental evaluation demonstrates that our randomized algorithm is both time- and space- efficient, as well as highly accurate.

The rest of the paper is organized as follows. Section 2 gives the problem definition and presents the related work. Section 3 reviews two existing techniques that will be employed in our algorithms. Section 4 presents the exact algorithm in 2-dimensional space and our results for a multi-dimensional space are presented in section 5. Our extensive experimental evaluation is reported in section 6. Section 7 concludes the paper.

2. Background Information

We first state the problem. Then, we present the related work. Below in Table 1 we summarise the math notation used throughout the paper.

Notation	Definition
P	a set of data points
d	dimensionality of a space
n	$\|P\|$
p, q	a data point
S_P	the set of skyline points of P
m	$\|S_P\|$
s (S)	a (set of) skyline point (points)
$D(S)$	the set of points dominated by one $s \in S$
e	an entry in an R-tree
$S.fm$ ($s.fm$)	FM sketch set at S (s)
e^H	the heap top of a heap H

Table 1. Math Notation

2.1. Problem Statement

Suppose that P is a set of d-dimensional data points. For a data point $q \in P$, $D(\{q\})$ denotes the set of points in P that are dominated by q; for a set Q of data points, $D(Q)$ denotes the set of points each of which is dominated by a $q \in Q$. Clearly, $D(Q) = \bigcup_{\forall q \in Q} D(\{q\})$.

Example 1. *Regarding Figure 1(b),* $D(\{p_4\}) = \{p_3\}$, $D(\{p_6\}) = \{p_3, p_5, p_7\}$, *and* $D(\{p_4, p_6\}) = \{p_3, p_5, p_7\}$.

The problem of *top-k representative skyline points* (top-k RSP) is formally defined as follows.

Top-k RSP. *Given a set P of points and an integer k, compute a set S of k skyline points such that $|D(S)|$ is maximized. Note that when $|S_P| \leq k$, S_P is the solution.*

In this paper, we study the problem of efficiently computing top-k RSP.

2.2. Related work

Computing full skyline. Efficiently computing skyline is first investigated by Kung *et al.* in [23]. Bentley *et al.* [3] provide an efficient algorithm with an expected linear running time if the data distribution on each dimension is independent.

Börzsönyi *et al.* [4] first investigate the skyline computation problem in the context of databases and propose an SQL syntax for the skyline query. They also develop the skyline computation techniques based on *block-nested-loop* and *divide-conquer* paradigms, respectively. Chomicki *et al.* [9] propose another block-nested-loop based computation technique, SFS (*sort-filter-skyline*), to take the advantages of a pre-sorting. The SFS paradigm is significantly improved by Godfrey *et al.* in [13]. Tan *et al.* [33] propose the first *progressive* technique that can output skyline points without having to scan the whole dataset. Two auxiliary data structures are proposed, *bitmap* and *search tree*. Kossmann *et al.* [22] present another *progressive* technique based on the nearest neighbour search technique on R-tree [32, 15], which adopts a divide-and-conquer paradigm on the dataset indexed by R-tree. Papadias *et al.* [29] propose a branch and bound search technique (BBS) to progressively output skyline points on datasets indexed by R-tree. One of the most important properties of BBS in [29] is that it guarantees the minimum I/O costs.

Kapoor [20] studies the problem of dynamically maintaining an effective data structure for an incremental skyline computation in a 2-dimensional space. Chan *et al.* [6] investigate the skyline computation problem for partially-ordered value domains.

Data points with designated dominance properties. Observe that the number of skyline points may be large; thus the full skyline is not always very informative. The problem of selecting data points with some designated dominance properties has been recently investigated in [8, 7, 21, 29].

Koltun and Papadimitriou [21] aim to find a minimum set of points to approximately dominate all data points. Specifically, for a given ϵ ($\epsilon \geq 0$), find a subset Q of a given P, with the minimum cardinality, such that every $p \in P$ is dominated by a $(1 - \epsilon)q$ where $q \in Q$. In [21], the maximal vector problem is investigated. Here, we state their problem with our setting. It has been shown that the problem can be solved by a greedy heuristic in a $2d$-space, and it is NP-hard for $d = 3$ or more. Then, it shows the problem can be approximately solved with a poly-logarithmic cardinality (for $\epsilon > 0$) of Q regarding the values' domain. This problem is inherently different than our problem – top-k RSP. First, it does not guarantee that the data points in Q are skyline points unless $\epsilon = 0$; when $\epsilon = 0$, all skyline points are returned. Second, the size of Q is poly-logarithmic regarding the values' domain. Consequently, the results and the techniques in [21] are not applicable to top-k RSP.

Chan *et al.* [8] investigate the problem of computing top-k frequent skyline points based on a new metric, *skyline frequency*. Skyline frequency of a point p is the number of subspaces where p is a skyline point. In [7], Chan *et al.* develop efficient algorithms to compute skyline points each of which is a skyline point in all subspaces with dimensionality k for a given k.

The most related problem to our top-k RSP has been investigated in [29]. Papadias *et al.* [29] propose a k-*dominating query*. It aims to compute a set Q of k points such that $\sum_{p \in Q} |D(\{p\})|$ is maximized. The problem of k-dominating query is also inherently different than our top-

k RSP. First, a k-dominating query does not always return skyline points; for instance, p_6 and p_7 are returned regarding Figure 1 when $k = 2$ (p_7 is not a skyline point). Second, in the problem of k-dominating query we only need to record the number of dominated points for each data point and there is no need to consider the situation that a data point may be dominated by many other points. The algorithms for processing a k-dominating query cannot be used to our top-k RSP.

Other Related Work. There also have been a number of research results in the literature regarding variations of skyline computation. These include computing skyline in a distributed environment [1, 18], continuously processing skyline queries in data streams [26, 34], skyline cube computation [30, 37] and its dynamic maintenance [36], computing skyline efficiently in a subspace [35], effectively materializing dominance relationships [24], and multi-source skyline query processing [10].

3. Preliminaries

We present briefly the skyline computation algorithm, BBS [29], as well as a probabilistic algorithm, FM [12], for counting distinct data elements. They will be employed in our approximate algorithm for top-k RSP.

3.1. BBS Algorithm

Suppose that a dataset P is indexed by an R-tree. To compute skyline, BBS traverses the R-tree in the order, such that it always evaluates and expands the tree node closest to the origin among all un-visited nodes. To do that, a *min-heap* is built against a designated *mindist* (say, the summation of all coordinate values) of the lower-left corner of the minimum bounding box (MBB) of every entry (node).

Initially, BBS inserts all the child entries of the root of the R-tree into the heap. Iteratively, the heap top e of the heap is examined against the already computed skyline points. If e is dominated by an already computed skyline point,[1] then e is just simply discarded from the heap. Otherwise, if e is a data point, then the data point is output as a skyline point; if e is not a data point, then discard e and insert the child entries of e, which are not dominated by any current skyline point, into the heap. BBS terminates when the heap is empty. In order to efficiently examine the dominance relationship, an in-memory R-tree is maintained on the current skyline points.

BBS has the properties that 1) any progressively generated skyline point is guaranteed to be a skyline point against P, 2) it is I/O optimal, 3) a node entry is read by disk I/O only once.

3.2. FM Algorithm

FM algorithm proposed by Flajolet and Martin [12] is a bitmap based algorithm that can efficiently estimate the number of distinct elements (data points). Let B be a bitmap of length L with subindexes $[0, L-1]$, and all bits are initialized as 0 (i.e. $B[j] = 0$ for $0 \le j \le L-1$). Suppose that $h()$ is a randomly generated hash function which hashes each elementID into an integer in $[0, L-1]$ such that for each data point p in a collection $\mathcal{P}$ of data points, $Prob\{h(p) = i\} = \frac{1}{2^{i+1}}$. In our implementation, we use the public code from Massive Data Analysis Lab [27] to randomly generate such hash functions. An FM sketch on $\mathcal{P}$ is a bitmap with length L which is defined as:

$$\mathcal{F}m = \{B : \forall 0 \le j \le L-1, B[j] = 1 \text{ iff } \exists p \in \mathcal{P}, h(p) = j\}.$$

In order to improve the accuracy of FM algorithm, multiple copies (say $\digamma$) of FM sketches are constructed; each is constructed against an independently generated hash function. Let $fm(\mathcal{P})$ represent the set of $\digamma$ FM sketches generated over $\mathcal{P}$. That is, $fm(\mathcal{P}) = \{\mathcal{F}m_1, \mathcal{F}m_2, \dots, \mathcal{F}m_\digamma\}$, where each element $p \in \mathcal{P}$ is hashed into these $\digamma$ FM sketches, respectively, as described above.

Let $min(B)$ denote the least bit (from left) of a bitmap B with value 0; if no such bit exists then $min(B) = L$. The number n of distinct elements in $\mathcal{P}$ is estimated by:

$$A = \frac{1}{\varphi} 2^{\sum_{i=1}^{\digamma} min(\mathcal{F}m_i)/\digamma}, \text{where } \varphi \overset{\text{def}}{=} \frac{2^{E(min(\mathcal{F}m_1))}}{n}. \quad (1)$$

Note that in formula (1), $E(min(\mathcal{F}m_1))$ cannot be explicitly represented and n is not known. In our implementation we approximately choose φ as 0.775351 according to the approximate results in [12]. Each $min(\mathcal{F}m_i)$ related to $\mathcal{F}m_i$ is defined in the same way as $min(B)$ related to B. As shown in [12], $E(min(\mathcal{F}m_i)) = E(min(\mathcal{F}m_j))$ $(1 \le i < j \le \digamma)$; this, together with Theorem 2 in [12] and the *Central Limit Theorem* (pp 229 in [11]), immediately leads the following theorem by the independence assumption.

Theorem 1. *Let n be the number of distinct elements in $\mathcal{P}$ and A be the estimation of FM algorithm as shown in (1). For a given $0 < \delta < 1$ and $\digamma$, if $L = O(\log n + \log \digamma + \log \delta^{-1})$, then $|A - n| < \epsilon n$ holds with probability at least $1 - \delta$, where $\epsilon = O(\sqrt{\frac{\log \delta^{-1}}{\digamma}})$.*

Two sets of fm sketches (say, $fm(\mathcal{P})$ and $fm(\mathcal{Q})$) generated by the same L and the same set of $\digamma$ hashing functions may be merged by the *bitwise-or operator* (denoted by $\bigvee$) as follows. Let $fm(\mathcal{P}) = \{\mathcal{F}m_i : 1 \le i \le \digamma\}$, $fm(\mathcal{Q}) = \{\mathcal{F}m'_i : 1 \le i \le \digamma\}$. We define $fm(\mathcal{P}) \bigvee fm(\mathcal{Q})$ as $\{\mathcal{F}m_i \bigvee \mathcal{F}m'_i : 1 \le i \le \digamma\}$, where each $\mathcal{F}m_i \bigvee \mathcal{F}m'_i$ is also a bitmap with subindexes $[0, L-1]$, such that for $1 \le i \le \digamma$: $\forall 0 \le j \le L-1$,

$$(\mathcal{F}m_i \bigvee \mathcal{F}m'_i)[j] = 1 \text{ iff } \mathcal{F}m_i[j] = 1 \text{ or } \mathcal{F}m'_i[j] = 1.$$

An important feature of FM algorithm is that the bitwise-or operator provides an equivalent way to generate a set of FM sketches over $\mathcal{P} \cup \mathcal{Q}$. The following lemma can be immediately verified.

Lemma 1. *Given a set of $\digamma$ hash functions and two collections, $\mathcal{P}$ and $\mathcal{Q}$, of data points, we have $fm(\mathcal{P} \cup \mathcal{Q}) = fm(\mathcal{P}) \bigvee fm(\mathcal{Q})$.*

4. Two-dimensional Space

We investigate the problem of the top-k RSP in a $2d$-space. We first show the problem can be solved by a dynamic programming algorithm. Then, we develop a sweep-line technique [31] to efficiently compute the parameters needed in the dynamic programming algorithm.

[1] A point s dominates entry e iff s dominates the lower-left corner of e.

4.1. Dynamic Programming Based Algorithm

Suppose that $\{s_1, s_2, \ldots, s_m\}$ is a collection of skyline points in a $2d$-space, which are sorted in the ascending order of x-coordinate values; consequently, they are also sorted in the descending order of y-coordinate values. Each s_i is represented by $(s_i[x], s_i[y])$. As depicted in Figure 2, for each pair $\{s_i, s_j\}$ of skyline points, $\Delta(s_i, s_j)$ denotes the set of data points that are dominated by s_i but not dominated by s_j (see Figure 2 for an example).

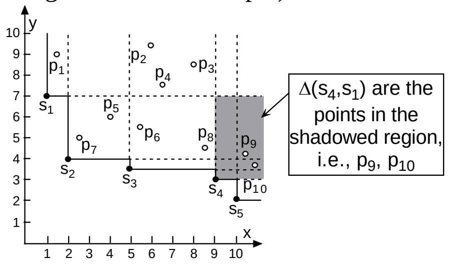


Figure 2. Skyline and $\Delta(s_i, s_j)$

Let $opt(s_i, k)$ denote the number of data points dominated by at least one of the k skyline points in an exact solution of the top-k RSP restricted to $\{s_1, \ldots, s_i\}$, where the exact solution contains s_i. We have

$$opt(s_i, k) = \max_{1 \le j < i} \{|\Delta(s_i, s_j)| + opt(s_j, k-1)\} \qquad (2)$$

Note that $opt(s_i, 1)$ is the number of points dominated by the skyline point s_i. Let $OPT(k)$ denote the the number of points dominated by at least one of the k skyline points in the exact solution of top-k RSP with respect to the whole set of skyline points. Clearly,

$$OPT(k) = \max_{1 \le i \le m} \{opt(s_i, k)\} \qquad (3)$$

Based on formulae (2) and (3), the V-optimal dynamic programming technique [19] can be immediately used to solve our top-k RSP. The algorithm runs in $O(km^2)$ time if each $|\Delta(s_i, s_j)|$ (for $1 \le j < i \le m$) is pre-computed and the skyline points are also pre-computed. In the following subsection, we present an efficient technique to compute the skyline points and $|\Delta|$.

4.2. Computing Skyline and $|\Delta|$

As an immediate approach, the skyline and $|\Delta|$ can be computed separately. First, all skyline points are computed over a given dataset by an existing algorithm. Then, compute the corresponding $|\Delta|$ values (initially, 0) for each data point. This naïve approach is not efficient because a data point may be counted multiple times if it is contained by several $\Delta(s_i, s_j)$. Below, we present a sweep-line [31] based algorithm to efficiently compute $|\Delta|$ values and skyline points simultaneously by sorting data points first.

As depicted in Figures 3, the region dominated by the skyline points can be partitioned into a number of *cells* $\{C_{a,b} : 1 \le a \le b \le m\}$ where the lower-left corner and the upper-right corner of $C_{a,b}$ are $(s_b[x], s_a[y])$ and $(s_{b+1}[x], s_{a-1}[y])$, respectively. When $a = 1$ (or $b = m$), $s_0[y]$ (or $s_{m+1}[x]$) is defined as the maximum value of y-coordinates (or x-coordinates) in the data space.

Let $|C_{a,b}|$ denote the number of data points that are contained by $C_{a,b}$ but not on the *top horizontal line* (i.e., $y = s_{a-1}[y]$) except $a = 1$ nor *on the right vertical line* (i.e., $x = s_{b+1}[x]$) of $C_{a,b}$ except $b = m$. Immediately, when $i > j$,

$$|\Delta(s_i, s_j)| = \sum_{j < a \le i, i \le b \le m} |C_{a,b}| \qquad (4)$$

Consequently, when $i > j$,

$$|\Delta(s_i, s_j)| = |\Delta(s_i, s_{j+1})| + \sum_{i \le b \le m} |C_{j+1,b}| \qquad (5)$$

For example, $|\Delta(s_4, s_1)| = |\Delta(s_4, s_2)| + |C_{2,4}| + |C_{2,5}| = 2$, as depicted in Figure 3.

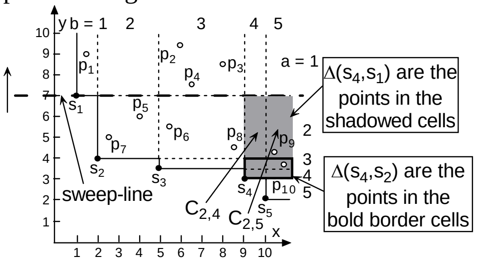

y
x
b = 1
2
3
4
5
a = 1
2
3
4
5
p1
p2
p3
p4
p5
p6
p7
p8
p9
p10
s1
s2
s3
s4
s5
sweep-line
C2,4
C2,5
Δ(s4,s1) are the points in the shadowed cells
Δ(s4,s2) are the points in the bold border cells

Figure 3. Cells and $\Delta(s_i, s_j)$

Our sweep-line algorithm uses a horizontal line to sweep along y-dimension from the bottom to the top to iteratively compute the skyline points and $|C_{a,b}|$s. Specifically, for data points encountered by the sweep-line, we process those data points from left to right as follows. If a data point p is dominated by a current skyline point, then increase the number of points of the cell, containing p, by 1; otherwise, it is a new skyline point. Iteratively, once a new skyline point s_j is found, the computation of every $|C_{j+1,b}|$ (for $b \ge j$) has been completed; consequently, $|\Delta(s_i, s_j)|$ (for $i > j$) now can be calculated by using formula (5). Then, all $|C_{j+1,b}|$ (for $b \ge j$) can be discarded. Clearly, the total space required is $O(m^2)$, which is dominated by maintaining all $|\Delta(s_i, s_j)|$ values ($1 \le j < i \le m$).

To facilitate such a sweep-line technique, we first sort the data points p lexicographically on $(p[y], p[x])$. To speed-up the computation, all $\sum_{i \le b \le m} |C_{j+1,b}|$ can be accumulatively computed from $i = m$ to $i = j+1$. Note that $\{C_{j+1,b} : j+1 \le b \le m\}$ are already sorted according to the x-coordinates of their lower-left corners. Therefore, the computation of the skyline points and all $|\Delta|$ values can be done in $O(m^2 + n \log m)$ where n is the number of total data points. Consequently, the total time of our dynamic programming algorithm runs in $O(km^2 + n \log m)$.

5. Multi-dimensional Space

The dynamic programming algorithm provides an exact solution to top-k RSP in a $2d$-space; nevertheless, there are two issues to be addressed.

Issue 1: The dynamic programming algorithm cannot be extended to computing the top-k RSP in a multi-dimensional space when d is 3 or more. Consequently, the problem of top-k RSP is left open generally.

Issue 2: The space requirement in our dynamic programming algorithm is quadratic $O(m^2)$. Therefore, it is not scalable when m is large since additional I/O costs may be required if the space requirement is too large to fit in memory. Moreover, if the dataset is indexed (say, by an R-tree), this algorithm does not make the use of such index to reduce the I/O costs.

This section is organized as follows. We first address Issue 1 by sections 5.1 and 5.2. Then, we develop an efficient, scalable, index-based (R-tree based) randomized algorithm to address Issue 2.

5.1. Complexity

We show that top-k RSP is NP-hard in a multi-dimensional space when the dimensionality is 3.

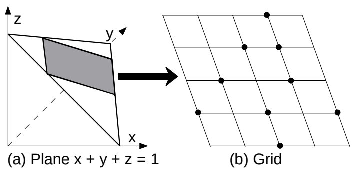


Figure 4. A Transformation

Theorem 2. *Given a set of points in a $3d$-space, the problem of computing top-k RSP is NP-hard.*

Proof. As depicted in Figure 4, we can divide one part (see the shaded area in Figure 4(a)) of the plane $x + y + z = 1$ into grid cells (see Figure 4(b)) such that each grid cell has at most 3 data points. According to the proofs of Theorem 3.3 and Theorem 4.1 in [25], the problem of finding k grid cells to contain the maximum number of points is NP-hard regarding this setting.

For a cell containing at least one data point, we choose 3 grid points including all data points in the cell; in case if the number of data points in the cell is less than 3, then we randomly choose non-data grid points to make it 3. Then, we create a new point by using the minimums, among these 3 grid points, in each coordinate, respectively, as its 3 coordinate values. It can be immediately verified those new created data points are the skyline points which only dominate the data points in its corresponding cells. Therefore, the problem of computing top-k RSP is also NP-hard. $\square$

5.2. Greedy Algorithm

In fact, top-k RSP can be immediately transformed into the *maximum coverage* problem [16]; consequently it can be solved approximately by a greedy heuristic. Below in Algorithm 1, we present the greedy heuristic. Note that $D(S)$ is the set of points each of which is dominated by at least one point in S and S_P is the skyline of dataset P.

Algorithm 1 Greedy (k, P)

```
Input:   $k$: an integer; $P$: a set of data points.
Output:  $k$ skyline points.
Description:
 Step 1: compute $S_P$;
 Step 2: $\forall s \in S_P$: compute $D(\{s\})$;
 Step 3:
 1: $S := \varnothing$;
 2: while $|S| < k$ and $S_P - S \neq \varnothing$ do
 3:     choose $s \in S_P - S$ such that $|D(\{s\} \cup S)|$ is maximized;
 4:     $S := \{s\} \cup S$;
 5: return $S$;
```

Lemma 2. *[16]: Algorithm 1 returns an approximate solution to top-k RSP with the approximate ratio* $1 - \frac{1}{e}$. [2]

In our implementation, we assume that the dataset P is indexed by R-tree [2, 14]. We use BBS to compute S_P in Step 1. Then, for each $p \in P$ we compute $\{D(\{s\}) : \forall s \in S_P\}$ by a window query per data point $p \in P - S_P$ against S_P, where the window uses the origin as the lower-left corner and p as the upper-right corner, as suggested in [29]. We apply the sort-merge paradigm in Step 3.

The space required to compute and store all $D(\{s\})$ is $O(mn)$ and may not fit in the memory. In fact, we found that when $d = 5$, 45% of $D(\{s\})$s have to be written to the disk with 1G memory. This not only requires additional disk I/O in Step 2 but also in Step 3 (I/O costs could be very expensive). We address the issue by our algorithm in the next subsection.

5.3. FM-based Algorithm

We modify Algorithm 1 by removing the requirement of computing and storing every $D(\{s\})$ ($\forall s \in S_P$). As with in $2d$-space we may divide the space dominated by the skyline points into grid cells and compute the number of points contained by each cell. Unlike $2d$-space, in a multidimensional space with $d \geq 3$ an "arbitrary" combination of k skyline points may be probed in Step 3. Consequently, the count of each cell should be stored and the space required is $O(m^d)$ that may be too larger to fit in memory when m is large and $d \geq 3$. Thus, this is not a good alternative.

On the other hand, keeping only $|D(\{s\})|$ ($\forall s \in S_P$) does not provide enough information to accurately estimate $|D(S)|$ for a set S of skyline points since there could be many data points dominated by more than one skyline point in S. For instance, in the example of Figure 2, point p_3 is dominated by s_1, s_2, and s_3.

To overcome the over-counting issue, we apply a duplicate-insensitive counting technique, FM algorithm, to approximately counting the number of points dominated by a skyline point s; that is, we maintain a set $s.fm$ of $\digamma$ FM sketches (bitmaps) at s, each of which has L bits and is generated by hashing the data points in $D(\{s\})$. Then, to compute $|D(S)|$ we only need to apply the bitwise-or operator $\bigvee$, as described in section 3.2, on $\{s.fm : s \in S\}$ to get a FM sketch set (and then use formula (1)). According to Lemma 1, this is the same as if we use FM algorithm to approximately computing $|D(S)|$. This is the basic idea of our algorithm. Below, we outline our FM-based algorithm in Algorithm 2. Note that in our algorithm, all FM sketch sets are created by the same set of $\digamma$ hash functions randomly generated.

Algorithm 2 FMGreedy (k, $\digamma$, L, P)

```
Input:   $k$, $\digamma$, $L$: integers; $P$: a dataset.
Output:  $k$ skyline points.
Description:
 Step 1: compute $S_P$;
 Step 2: $\forall s \in S$, compute FM sketch set $s.fm$;
 Step 3:
 1: $S := \varnothing$; initialize the $\digamma$ bitmaps in $S.fm$;
 2: while $|S| < k$ and $S_P - S \neq \varnothing$ do
 3:     choose $s \in S_P - S$ such that $Est(s.fm \bigvee S.fm)$ is maximized;
 4:     $S := \{s\} \cup S$; $S.fm := s.fm \bigvee S.fm$;
 5: return $S$;
```

Note that $Est(s.fm \bigvee S.fm)$ is calculated by the for-

[2]Here, e is Euler's constant rather than an entry in an R-tree.

mula (1) in section 3.2. The following Theorem immediately follows from Theorem 1 and Lemma 2.

Theorem 3. *Algorithm 2 has the approximate ratio* $(1 - \frac{1}{e})(1 + O(\sqrt{\frac{\log \delta^{-1}}{\digamma}}))$ *with confidence* $1 - \delta$ *if each bitmap has the length* $L = O(\log n + \log \digamma + \log \delta^{-1})$.

Our experiments in section 6 demonstrate that when $\digamma = 32$ and $L = 32$, Algorithm 2 has very similar accuracy to Algorithm 1. The Step 3 of Algorithm 2 can be executed in time $O(k\digamma m)$. Since $\digamma$ is up-to 32 in our implementation, the Step 3 runs in time $O(km)$. The space requirement to store all FM sketch sets is $O(\digamma m)$ that equals $O(m)$ in our execution. Therefore, we reduce the space requirement $O(mn)$ to store $\{D(\{s\}) : s \in S_P\}$ to $O(m)$ to store all FM sketch sets with respect to the skyline points.

Next, we present an efficient algorithm to conduct Step 1 and Step 2 by one-scan of dataset by extending BBS algorithm. For this purpose, we first augment R-tree to include FM sketches.

5.3.1. R^{FM}-tree

At each intermediate entry e of an R^{FM}-tree, besides the required information in a conventional R-tree (e.g., MBB), we maintain a set fm of $\digamma$ FM sketches for estimating the number data points in the subtree of e. Lemma 1 in section 3.2 implies that the sketch set $e.fm$ at entry e can be equivalently obtained by using the bitwise-or operator $\bigvee$ to merge the sets of FM sketches at all e's child entries. Figure 5 shows an R^{FM}-tree when $\digamma = 1$ and $L = 4$.

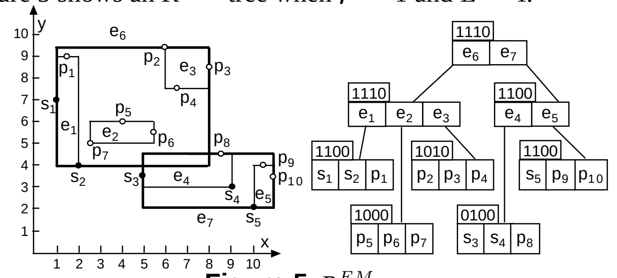


Figure 5. R^{FM}-tree

Note that the FM-based aggregate information practically requires small space only. Our experiments show that if each page has size 4Kbytes, the fanout of an R^{FM}-tree is only 1 less than that of a conventional R-tree when each FM sketch set consists of 8 bitmaps with 32 bits each, and about 3% less when each FM sketch set consists of 32 bitmaps with 32 bits each. It is immediate that such an R^{FM}-tree can be updated in a similar way as a conventional R-tree.

5.3.2. Computing Skyline and FM Sketches

Given that a dataset P is indexed by an R^{FM}-tree, in this subsection we present an efficient algorithm, OnePass, to compute the set of skyline points and their FM sketches (to estimate every $|D(\{s\})|$) simultaneously, so that each entry in R^{FM}-tree is read at most once by disk I/O.

I/O-efficiency. The algorithm BBS computes the skyline points with the minimum I/O costs; nevertheless, it is not immediately applicable to compute FM sketches simultaneously; see the following example.

Example 2. *In the example of Figure 5, BBS outputs the skyline points in the order of* s_2, s_1, s_3, s_4, *and* s_5 *if* mindist *is* $x + y$. *After* s_2 *is returned as the skyline point,* e_7 *is expanded to* e_4 *and* e_5. *Then,* e_2 *is the next to be processed. Since* e_2 *is dominated by* s_2, e_2 *is discarded from the heap* H_1 *in BBS. As* s_3 *is not yet returned as the skyline point, removing* e_2 *means that we have to start from the root of the R*^{FM}*-tree to compute the FM sketches at* s_3.

To resolve the problem in Example 2, in our algorithm OnePass we use another heap H_2 to keep the entries discarded from H_1 by BBS.

Minimizing the size of H_2**.** Clearly, there will be a huge memory requirement if we keep every thing in H_2 until all skyline points are calculated. We need to minimize the size of H_2 so that our algorithm is scalable while pursuing I/O efficiency. It seems difficult to find a necessary and sufficient condition to determine the time to discard an entry $e \in H_2$ so that the progressively returned skyline points afterwards do not dominate any point in e. In our algorithm, we use a sufficient condition below for this purpose.

Suppose that in BBS, the *mindist* is the summation of the coordinate values. For each entry $e \in H_2$, we define $e.maxdist$ as the summation of the coordinate values of e's upper-right corner (e is represented geometrically by MBB); H_2 is a min-heap maintained against $e.maxdist$. Note that BBS has the property that the minimum value of *mindist* among the entries in the current H_1 is not greater than that in the next iteration. Therefore, it can be immediately verified that if $e^{H_2}.maxdist \leq e^{H_1}.mindist$ then any skyline points progressively returned afterwards do not dominate any point in e^{H_2}. Consequently, e^{H_2} should be used to update the FM sketches for the current skyline points; then, e^{H_2} can be discarded from H_2.

Example 3. *Continuing the example in Figure 5,* e_2 *will be popped up (then discarded) from* H_2 *to update the FM sketches at* s_1, s_2, *and* s_3 *when BBS encounters* s_4 *in* H_1.

Algorithm description. Our algorithm to compute the skyline points and their FM sketches is based on the framework of BBS. It is presented below in Algorithm 3.

Algorithm 3 OnePass (P **)**

Input: P is a dataset indexed by an R^{FM}-tree R.
Output: S_P: the skyline point set; FM sketches at s ($\forall s \in S_P$)
Description:

```
1:  $S := \emptyset$; $H_1 := \emptyset$; $H_2 := \emptyset$;
2:  insert all entries of the root of R into $H_1$;
3:  while $H_1 \neq \emptyset$ or $H_2 \neq \emptyset$ do
4:    while $H_2 \neq \emptyset$ do
5:      if $H_1 = \emptyset$ or $e^{H_2}.maxdist \leq e^{H_1}.mindist$ then
6:        UpdateSketch($e^{H_2}$, $S$); remove $e^{H_2}$ from $H_2$;
7:      else break the whileloop;
8:    if $e^{H_1}$ is dominated by an $s \in S$ then
9:      if $e^{H_1}$ is a point then
10:       UpdateSketch($e^{H_1}$, $S$); remove $e^{H_1}$ from $H_1$;
11:     else remove $e^{H_1}$ from $H_1$ to $H_2$;
12:   else
13:     if $e^{H_1}$ is a point then
14:       add $e^{H_1}$ into $S$;
15:     else add child entries dominated by an $s \in S$ to $H_2$ and
          others to $H_1$;
16: return $S$ and FM sketches at each $s \in S$;
```

Note that in Algorithm 3, H_1 is a heap maintained in the same way as that in BBS, while H_2 is a heap discussed above. When the heap top e^{H_1} in H_1 is a point that is dominated by an already obtained skyline point s, instead of putting e^{H_1} into H_2 we can immediately use it to update the FM sketches (then discard e^{H_1}). This is because that e^{H_1} will not be dominated by any skyline point output afterwards due to the following two reasons:

- An entry's $mindist$ equals its $maxdist$ if it is a point.
- Any skyline point output afterwards has its $mindist$ not smaller than the current $mindist$.

In fact, it can be immediately verified that Algorithm 3 is correct, based on our discussions in the part – Minimizing the size of H_2; that is, once an entry e removes from H_2, none of the data points included in e is dominated by the skyline points output afterwards. Algorithm 3 also follows exactly BBS for computing the skyline points S_P; thus, all the skyline points can be obtained correctly.

In Algorithm 3, there is a key operation, UpdateSketch(), to update FM sketch sets at the current skyline points, respectively. Next, we present algorithm for UpdateSketch().

Update FM sketches. Regarding the example in Figure 5, s_2 dominates the entry e_2. Therefore, the update to the FM sketch set at s_2 is already materialized in R^{FM}-tree; that is, we only need to use bitwise-or operator $\bigvee$ on the bitmap $(1, 0, 0, 0)$ and the existing $s_2.fm$, instead of hashing the points p_5, p_6, and p_7. Moreover, if we had a skyline point $s*$ allocated at $(0.5, 0.5)$, then an update to the FM sketch set at $s*$ would be done by only reading the FM sketch set at the "root" in this example. These are the reasons why an R^{FM}-tree is maintained.

As with the algorithm BBS, in Algorithm 3 the progressively generated skyline points are also indexed by an in-memory index. Different than BBS, we use an in-memory R^{FM}-tree instead of R-tree to speed-up the computation (update) of FM sketches as well. Such an in-memory R^{FM}-tree has the same data structure as that to index the dataset except that we also attach FM sketches at each data point in an in-memory R^{FM}-tree. In an in-memory R^{FM}-tree, the FM sketch set at a tree node is used to estimate the number of data points that are "captured" by our algorithm to be commonly dominated by all points in its subtree.

Example 4. *In the example of Figure 5, the points in e_3 are dominated by s_1 and s_2. Consequently, if one entry $e^{1,2}$ of the in-memory R^{FM}-tree consists of s_1 and s_2, then we can immediately use the FM sketch set at e_3 to update the FM sketch set at $e^{1,2}$ by the bitwise-or operator $\bigvee$.*

Based on Lemma 1, we can use the bitwise-or operator iteratively on the FM sketch sets along the path in the in-memory R^{FM}-tree from the root to a skyline point s to obtain the "global" FM sketch set at s to estimate the number of distinct points dominated by s.

We need the following notation in our algorithm description. An entry e (bounding box) *fully dominates* another entry e' if the upper-right corner of e dominates the lower-left corner of e'. An entry e *partially dominates* another entry e' if the lower-left corner of e dominates the upper-right corner of e' but e does not fully dominate e'. An entry e *does not dominate* another entry e' if e does not fully nor partially dominate e'. For example, in Figure 6 e_4 fully dominates e_1, partially dominates e_2, and does not dominate e_3.

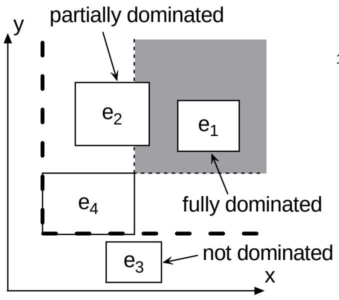


Figure 6. Dominance Relation

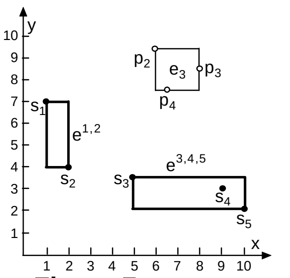


Figure 7. Update fm

We treat UpdateSketch (e^{H_2}, S) as one kind of spatial join [5, 17]. We apply the R-tree traversal paradigms from [5, 17, 28]. To avoid reading a data entry e from disk more than once, we group the entries, at which FM sketch sets may need to be updated by the points or child entries of e, of in-memory R^{FM}-tree for skyline together into a group E_e. We iteratively and *synchronously* traverse e and E_e, while always group the new expanded entries of the in-memory R^{FM}-tree (for skyline points) together for each new expanded data entry for the next iteration. We outline our algorithm in Algorithm 4 and present the details of the procedure Traversal (e, E) in Algorithm 5.

Algorithm 4 UpdateSketch (e^{H_2}, S)

```
Input:  $e^{H_2}$: an entry of R$^{FM}$-tree of a dataset;
        $S$: skyline points indexed by an in-memory R$^{FM}$-tree;
Output: updated FM sketches at the skyline points in $S$;
Description:
1: if $e_S$ fully dominates $e^{H_2}$ then
2:     $e_S.fm := e_S.fm \bigvee e^{H_2}.fm$;
3: else Traversal($e^{H_2}$, $\{e_S\}$);
```

In Algorithm 4, e_S is the root entry (MBB) of the in-memory R^{FM}-tree on S; according to Algorithm 3, e_S either fully or partially dominates e^{H_2}. It reads in the node in R^{FM}-tree corresponding to e^{H_2} to get its FM sketches $e^{H_2}.fm$. If e^{H_2} is a data point, it has to be hashed by FM algorithm to get $e^{H_2}.fm$.

Example 5. *Regarding the example in Figure 5, suppose in the in-memory R^{FM}-tree of skyline points s_1, s_2, $\ldots$, s_5, s_1 and s_2 form one entry $e^{1,2}$ and the entry $e^{3,4,5}$ consists of 3 points s_3, s_4, and s_5 (see Figure 7). When e_3 is pop-up from H_2 to be processed against the in-memory R^{FM}-tree, it enters Algorithm 5. In Algorithm 5, e_3 is subsequently decomposed into p_2, p_3, and p_4, while the root of in-memory R^{FM}-tree is decomposed into $e^{1,2}$ and $e^{3,4,5}$.*

When p_2 is processed against $e^{1,2}$ and $e^{3,4,5}$, $e^{1,2}.fm$ is updated; nevertheless, $(p_2, \{s_3, s_4, s_5\})$ has been sent to Algorithm 5 for a further processing.

We can immediately verify that in Algorithm 4, a data point p dominated by an $s \in S_P$ must be captured by one entry from the root to s in the in-memory R^{FM}-tree. Consequently, as discussed earlier, computing FM sketch set at each skyline point s by performing bitwise-or operator iteratively on the FM sketch sets along the path from the root of in-memory R^{FM}-tree to the skyline point s ensures the correctness of Algorithm 4; that is, it is equivalent to creating

Algorithm 5 Traversal (e, E)

```
Input:   e: an entry of R^{FM}-tree of a dataset;
         E: a set of entries of R^{FM}-tree of skyline points;
Output:  updated FM sketches at the skyline points in E;
Description:
 1: if $e$ is a data point then $E_D := \{e\}$;
 2: else load in the child entries of $e$ to $E_D$;
 3: for each entry $e' \in E_D$ do
 4:     $E_{e'} := \varnothing$;
 5:     for each entry $e_i \in E$ do
 6:         if $e_i$ fully dominates $e'$ then
 7:             GetFM($e'$); $e_i.fm := e_i.fm \bigvee e'.fm$;
 8:         else if $e_i$ partially dominates $e'$ then
 9:             if $e_i$ is a data point entry then
10:                 $E_{e'} := E_{e'} \cup \{e_i\}$;
11:             else
12:                 for each child entry $e_c$ of $e_i$ do
13:                     if $e_c$ fully dominates $e'$ then
14:                         GetFM($e'$); $e_c.fm = e_c.fm \bigvee e'.fm$;
15:                     else if $e_c$ partially dominates $e'$ then
16:                         $E_{e'} := E_{e'} \cup \{e_c\}$;
17:     if $E_{e'} \neq \varnothing$ then Traversal($e'$, $E_{e'}$);
Procedure GetFM ($e$)
if $e$ is a data point, then hash $e$ by FM algorithm to get $e.fm$;
otherwise, read in $e.fm$ by disk I/O if not already in memory.
```

the FM sketch set at s by hashing each data point dominated by s by FM algorithm. Thus, Theorem 3 still holds. Note that if an entry of R^{FM}-tree of the dataset is no longer needed in Algorithm 4, then it immediately removes from memory.

6. Performance Evaluation

We evaluate our algorithms only. Specifically, we focus on evaluating our FM-based algorithm in section 5.3. As there is no existing work, we use the dynamic programming based algorithm in section 4 and the greedy heuristic in section 5.2 as bench-marking algorithms. The following algorithms have been implemented.

1. *EXACT* : the dynamic programming based algorithm proposed in section 4.
2. *GDY* : the greedy algorithm, Algorithm 1, in section 5.2.
3. *FMG* : the FM based greedy algorithm, Algorithm 2. in section 5.3.

Following the common methodology in the literature, two synthetic datasets, Anti-correlated and Independent (random), have been employed in our performance evaluation, which are produced by the data generator in [4]. Their dimensionality varies from 2 to 5 and the number of data points varies from $200K$ to $3M$. A real data set, Stock, from NYSE (New York Stock Exchange) is also used in our performance study. It contains 3M stock transaction records of NT (Nortel Networks Corporation,USA) from Dec 1st 2000 to May 22nd 2001. The average price per volume, volume, and time are recorded for each transaction; consequently it is used as a dataset in a $3d$-space.

All of the datasets are indexed by an R^{FM}-tree with node page size 4Kbytes. In our implementation, each bitmap (FM sketch) has 32 bits (i.e., 4 bytes); a bitmap with 32 bits should be large enough to guarantee the FM accuracy while counting a massive number of distinct elements. As illustrated in our experiment, FMG is highly accurate (with relative errors less than 5%) when 32 bitmaps (FM sketches) are used; in this case, the fanout (number of children entries) of R^{FM}-tree is about 97% of that by a conventional R-tree.

All algorithms are implemented by C++. The experiments are conducted on the PCs with Intel P4 2.8GHz CPU and 1G memory under the operating system – Debian Linux. Table 2 below lists the parameters which may potentially have an impact on our performance study. In our experiments, all parameters use default values unless otherwise specified.

Notation	Definition (Default Values)
n	Number of points in the dataset (1M)
k	Number of representative skyline points (30)
d	Dimensionality of the dataset (3)
$\digamma$	Number of sketches in FMG (32)

Table 2. System Parameters

6.1. Evaluating Accuracy

In this subsection, we experimentally evaluate the accuracy of FMG against different settings; that is, the number of points dominated by one of the k skyline points (comparing with that by EXACT in a $2d$-space and that by GDY when d is between 3 to 5, respectively). Note that we did not do the accuracy evaluation against the exact solution for $d \geq 3$ because the problem is NP-hard and it is too slow to produce exact solution for top-k RSP.

The first set of experiment is conducted against 4 datasets, Anti-correlated datasets (Anti) (in $2d$ and $5d$ spaces, respectively), Independent (Indep) (in a $5d$ space), and Stock. The experiment results are reported in Figure 8. It demonstrates that FMG is quite accurate when $\digamma = 32$, while GDY generates exact solution for Anti in $2d$.

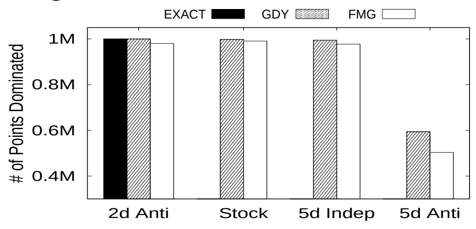


Figure 8. Number of Points Dominated

In the rest of this subsection, we evaluate the accuracy FMG by *relative errors* – $\frac{|N_{FMG}-N'|}{N'}$. N' is the number of data points dominated by at least one of the k skyline points in the exact solution in a $2d$-space, N' is such a number by GDY when $d \geq 3$, N_{FMG} is such a number by FMG.

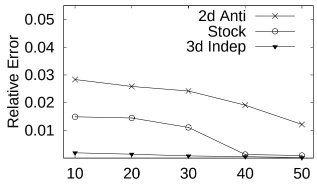


Figure 9. Various k

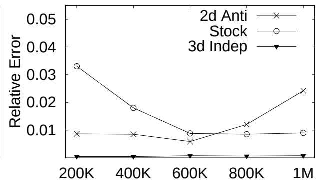
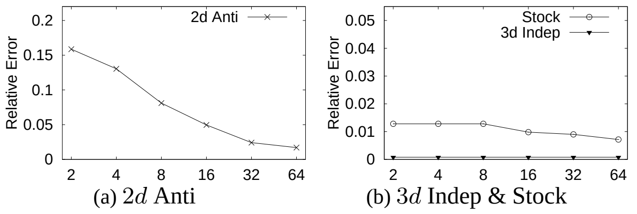


Figure 10. Various n

The second experiment evaluates possible impacts by different k values. It is conducted against three datasets, Anti in $2d$ space, Indep in $3d$ space, and the real datasets, Stock. The experiment results are depicted in Figure 9.

Again, it shows FMG is highly accurate. As anticipated, the accuracy improves when k increases.

The third experiment examines an impact of the cardinality of datasets. The results depicted in Figure 10 indicate that the accuracy is not quite relevant to a dataset size. The last experiment examines an impact of the number $\digamma$ of FM sketches. The results reported in Figure 11 confirm Theorem 3; that is, the accuracy improves when $\digamma$ increases.



(a) $2d$ Anti (b) $3d$ Indep & Stock

Figure 11. Relative Error vs $\digamma$

6.2. Evaluating Efficiency

The first experiment, as depicted in Figure 12 where the numbers on these bar figure tops give the actual running time, is conducted against the 4 datasets and the 3 algorithms EXACT, GDY, and FMG. It shows that GDY is very slow when $d = 5$; this is because $\{D(\{s\}) : s \in S_P\}$ is too large to fit in memory. Thus, additional I/O costs in Step 2 and Step 3 are required as stated in section 5.2. This makes Step 3 very slow. The results in Figure 12 demonstrate that FMG is most efficient.

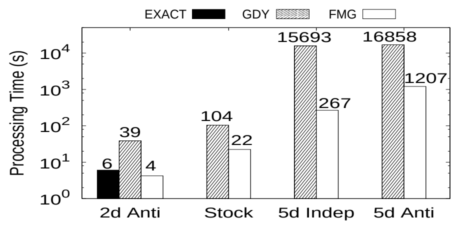
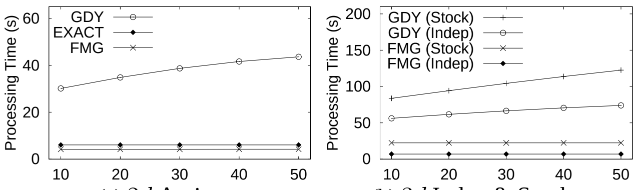


Figure 12. Processing Time

The second experiment tests an effect of k. The experiment results are depicted in Figure 13. They also demonstrate that FMG is most efficient. While FMG and EXACT are not very sensitive to different k values, the time costs of GDY increase when k increases. This is mainly because that Step 3 in GDY involves a sort-merge process.

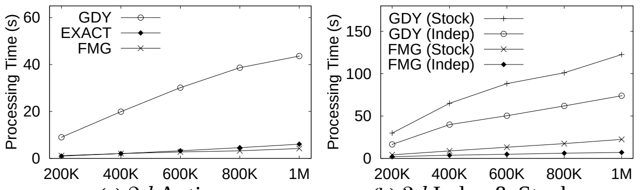


(a) $2d$ Anti (b) $3d$ Indep & Stock

Figure 13. Time Efficiency vs k

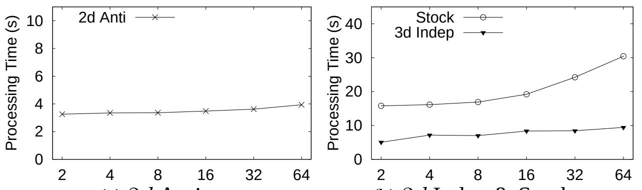
The results of our third experiment are reported in Figure 14. They show that EXACT, GDY, FMG are all sensitive to different data sizes. Note that GDY is very sensitive to data sizes due to the same reason as mentioned in the last experiment – computation in Step 3.

The final experiment in this part is to evaluate an impact of $\digamma$ in FMG. As depicted in Figure 15, the time costs of FMG increase when $\digamma$ increases; this is simply because that a data element needs to be hashed more times and more FM sketches are involved in a bitwise-or operator $\bigvee$ when $\digamma$ gets increased.



(a) $2d$ Anti (b) $3d$ Indep & Stock

Figure 14. Time Efficiency vs Cardinality



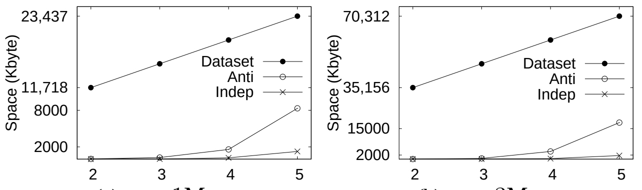
(a) $2d$ Anti (b) $3d$ Indep & Stock

Figure 15. Time Efficiency vs $\digamma$

6.3. Space and I/O Efficiency

We first evaluate the space efficiency. We illustrate the results for FMG only, since EXACT only works for $2d$-space and GDY requires very large memory space – at least (but could be much larger than) the size of whole dataset. In our experiment, we record the maximum space usage in the whole computation, including storing H_1, H_2, the space required to execute Algorithm 4. In H_1 and H_2, we store ID and its MBB (or data point) for each entry.

The first set of experiments, conducted against Anti-correlated and Independent datasets with d in $[2, 5]$, evaluate effects of d. The results are reported in Figure 16 where linear scale is used in y-coordinate and we show some important marks only on y-coordinate. It demonstrates that the memory space required is significantly less than the dataset size. Note that we also examined the space requirements of EXACT in a $2d$-space and storing $\{D(\{s\}) : s \in S_P\}$ in GDY, respectively. Our experiment demonstrates that 1) EXACT requires space about 10 times more than that of FMG even the number m of skyline points is less than 67, and 2) GDY requires space about 3.63 times ($d = 2$) to 82.5 times ($d = 5$) of the corresponding dataset sizes.

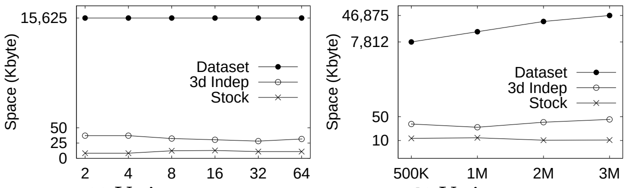


(a) $n = 1$M (b) $n = 3$M

Figure 16. Space vs Dimensionality d

The second experiment investigates impacts from different data sizes and $\digamma$. The results are depicted in Figure 17. They demonstrate that the memory space is not sensitive to $\digamma$ because we do not keep FM sketches in heaps and we get FM sketches only when update them. It is also interesting to note that the memory space requirement is not very sensitive to a data size. This implies that FMG is scalable.

We then evaluate the I/O efficiency of FMG. For a $2d$ independent dataset with size 1M, only 60% of the nodes are



(a) Various F (b) Various n

Figure 17. Space vs F and Cardinality n

accessed by FMG, while one-scan of the whole R^{FM}-tree is required by FMG when $d \geq 3$. In our experiment, we also consider an alternative to implement FMG by "two-scans": first, the skyline points are computed by BBS and maintained by the in-memory R^{FM}-tree; second, the data pages are read in one by one to perform Algorithm 4. Our experiment indicates the I/O cost ratio of FMG to this alternative decreases significantly as d increases. For instance, the ratio is around 99% on Anti-correlated data set with $d = 2$ and $n = 1$M, while it drops to 63% when $d = 5$.

Moreover, our experiment also shows that I/O costs of GDY are much more expensive than FMG regarding 1G fixed memory. The I/O costs of GDY vary from 1.4 ($d = 2$) to 1772 ($d = 5$) times of those of FMG.

6.4. Summary

As a short summary, our performance evaluation indicates that FMG is quite accurate, efficient, and scalable regarding data size. When the number of skyline points is small, EXACT is a good choice in $2d$-space. Although GDY is slightly more accurate than FMG, it requires huge memory space; thus, it is not scalable.

7. Conclusion

In this paper, we investigate the problem of computing the top-k representative skyline points. This is among the first attempts to develop efficient and scalable algorithms to solve the problem. After introducing the novel skyline operator: top-k representative skyline points, we present an efficient dynamic programming based algorithm for a $2d$-space in which an exact solution can be achieved. As shown in the paper, this problem is NP hard for space with dimensionality $d \geq 3$ and the greedy heuristic for set cover problem can be immediately applied to provide the approximation ratio $1 - \frac{1}{e}$. We then develop an efficient, scalable randomised algorithm with a theoretical accuracy guarantee. As our performance study indicated, our randomized algorithm is both time and space efficient, as well as highly accurate.

Acknowledgement. The work was partially supported by ARC-DP(DP0666428) and UNSW FRG(FRGP, PS08709).

References

[1] W.-T. Balke, U. Güntzer, and J. X. Zheng. Efficient distributed skylining for web information systems. In *EDBT 2004*.
[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD 1990*.
[3] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *JACM*, 25(4):536–543, 1978.
[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE 2001*.
[5] T. Brinkhoff, H.-P. Kriegel, and B. Seeger. Efficient processing of spatial joins using r-trees. In *SIGMOD 1993*.
[6] C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD 2005*.
[7] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD 2006*.
[8] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT 2006*.
[9] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE 2003*.
[10] K. Deng, X. Zhou, and H. T. Shen. Multi-source skyline query processing in road networks. In *ICDE 2007*.
[11] W. Feller. *An Introduction to Probability Theory and Its Applications*. John Wiley & Sons, Inc., 1966.
[12] P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of Computer and System Sciences*, 31(2):182–209, 1985.
[13] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB 2005*.
[14] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD 1984*.
[15] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *TODS*, 24(2):265–318, 1999.
[16] D. S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11(3):555–556, 1982.
[17] Y.-W. Huang, N. Jing, and E. A. Rundensteiner. Spatial joins using R-trees: Breadth-first traversal with global optimizations. In *VLDB 1997*.
[18] Z. Huang, C. S. Jensen, H. Li, and B. C. Ooi. Skyline queries against mobile lightweight devices in MANETs. In *ICDE 2006*.
[19] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB 1998*.
[20] S. Kapoor. Dynamic maintenance of maxima of 2-d point sets. *SIAM Journal on Computing*, 29(6):1858–1877, 2000.
[21] V. Koltun and C. H. Papadimitriou. Approximately dominating representatives. In *ICDT 2005*.
[22] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB 2002*.
[23] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4):469–476, 1975.
[24] C. Li, B. C. Ooi, A. K. Tung, and S. Wang. DATA: A data cube for dominant relationship analysis. In *SIGMOD 2006*.
[25] X. Lin, Q. Liu, Y. Yuan, X. Zhou, and H. Lu. Summarizing level-two topological relations in large spatial datasets. *TODS*, 31(2):584–630, 2006.
[26] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE 2005*.
[27] Massive Data Analysis Lab. `http://www.cs.rutgers.edu/~muthu/massdal.html`.
[28] D. Papadias, N. Mamoulis, and Y. Theodoridis. Processing and optimization of multiway spatial joins using R-trees. In *PODS 1999*.
[29] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD 2003*.
[30] J. Pei, W. Jin, M. Ester, and Y. Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *VLDB 2005*.
[31] P. Rigaux, M. Scholl, and A. Voisard. *Introduction to Spatial Databases: Applications to GIS*. 2000.
[32] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *SIGMOD 1995*.
[33] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB 2001*.
[34] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *TKDE*, 18(2):377–391, 2006.
[35] Y. Tao, X. Xiao, and J. Pei. SUBSKY: Efficient computation of skylines in subspaces. In *ICDE 2006*.
[36] T. Xia and D. Zhang. Refreshing the sky: The compressed skycube with efficient support for frequent updates. In *SIGMOD 2006*.
[37] Y. Yuan, X. Lin, Q. Liu, W. Wang, J. X. Yu, and Q. Zhang. Efficient computation of the skyline cube. In *VLDB 2005*.