Efficient Skyline Computation in High-Dimensionality Domains

Rui Liu
University of Tours, France
rui.liu@univ-tours.fr

Dominique Li
LIFAT Laboratory, University of Tours, France
dominique.li@univ-tours.fr

ABSTRACT
We present a dimension indexing based algorithm for skyline computation. We first show that the dominance tests required to determine a skyline tuple can be efficiently bounded to a subset of the current skyline, and then propose the algorithm SDI, of which the time complexity is better than the best known algorithm in high-dimensionality domains with reasonably low cardinality. Our performance evaluation on synthetic and real datasets shows that SDI outperforms the state-of-the-art skyline algorithm in both low-dimensionality and high-dimensionality domains.

1 INTRODUCTION
The skyline computation problem aims at retrieving the complete set of dominating tuples from multidimensional data, with respect to a monotonic preference order on all dimensions. Over several past decades, many algorithms have been developed, which can be categorized into sorting based [1, 4, 6, 7, 14] and partitioning based [2–4, 8–13, 15]. Most existing skyline algorithms have been designed for low-dimensionality domains because of the quadratic issue raised by the worst case time complexity.

In this paper, we present a dimension indexing based skyline algorithm SDI (Scalable Dimension Indexing) that is efficient in high-dimensionality domains as well as in low-dimensionality domains. We show that by indexing all dimensions, it is sufficient to test a tuple only with the existing skyline tuples on an arbitrary dimension instead of with the complete set of skyline tuples. We also show that any skyline tuple can be used as a stop line that traverses the indexed dimensions to stop the computation, which is much performant than the calculation of stop point mentioned in SaLSa [1]. Furthermore, SDI adopts the weak incomparability checking to take the incomparability between tuples into account, which is the most important feature of the state-of-the-art skyline algorithm BSkyTree [10]. Our analysis shows that the worst time complexity of SDI is better than the best known one [13] in high-dimensionality domains with reasonably low cardinality, and our performance evaluation shows that SDI outperforms BSkyTree on both low and high dimensional data, but less efficient than BSkyTree on medium dimensional data.

The rest of this paper is organized as follows. Section 2 presents the SDI algorithm with preliminary definitions. We show our theoretical analysis of the computational complexity of SDI in Section 3. Section 4 reports the performance evaluation of SDI on both synthetic and real datasets. We conclude in Section 5.

2 THE SDI APPROACH
Let \( t \) be a \( d \)-dimensional tuple, we denote \( t[i] \) the dimension value of \( t \) on the dimension \( i \), where \( 1 \leq i \leq d \). We define the preference order, denoted by \( \prec \), as a total order that covers each dimension such that given two tuples \( t_1 \) and \( t_2 \), \( t_1[i] \) is better than \( t_2[i] \) if \( t_1[i] < t_2[i] \); \( t_1[i] \) is equal to \( t_2[i] \) if \( t_1[i] = t_2[i] \); \( t_1[i] \) is not worse than \( t_2[i] \) if \( t_1[i] < t_2[i] \) or \( t_1[i] = t_2[i] \) holds, denoted by \( t_1[i] \leq t_2[i] \).

We say that a tuple \( t \) dominates a tuple \( u \), denoted by \( t \prec u \), if and only if for each dimension \( i \), we have \( t[i] \leq u[i] \), and for at least one dimension \( k \) we have \( t[k] < u[k] \). We denote \( t \not\prec u \) that \( t \) does not dominate \( u \), and \( t \sim u \) that \( t \) and \( u \) are incomparable, that is, \( t \not\prec u \) and \( u \not\prec t \). Considering a \( d \)-dimensional database \( D \) and a preference order \( \prec \) on \( D \), a tuple \( t \in D \) is a skyline tuple if only if \( \nexists u \in D \) such that \( u \prec t \). The skyline of \( D \) is the complete set \( S \) of skyline tuples such that \( S = \{ t \in D : \nexists u \in D, u \prec t \} \). We have \( s \sim t \) for any two skyline tuples \( s \) and \( t \).

Given a database, the dimension index \( I \), is the set of \( d \) ordered lists on a preference order \( \prec \), in which each list \( I_i \) is a dimension subindex that contains all dimension values sorted with respect to \( \prec \).

![Figure 1: A dimension index example.](image)

Let us consider the 5 tuples \( \{a, e, g, h, i\} \) presented in Figure 1 that shows a dimension index example consisting of 10 tuples and 6 subindexes, where the dashed lines link the tuple among the subindexes. Obviously, \( a \) is a skyline tuple, which can be independently concluded from \( I_1 \), \( I_2 \), and \( I_6 \) because no tuple can dominate \( a \); if we regard only \( I_6 \), \( h \) is immediately a skyline tuple and in order to determine whether \( i \), the second tuple in this subindex, is a skyline tuple, it is enough to compare \( i \) with \( h \) because any tuple \( x \) located after the position of \( i \) in \( I_6 \) cannot dominate \( i \) since we have \( i[4] > x[4] \) in \( I_6 \). Nevertheless, if we focus on \( I_5 \), we see that \( h \prec i \) and \( i \prec h \) must be first tested in order to decide whether \( h \) or/and \( i \) shall be skyline tuple(s) since \( h[3] = i[3] \) (indeed we have \( h \prec i \) and \( i \not\prec h \)). That is also the case in \( I_6 \), where 3 tuples contain the same dimension value 0.15, so all these 3 tuples must be first locally compared in order to filter the potential skyline tuples (in our example, \( e \)). We call such tuples as \( h \) in \( I_5 \) and \( i \) in \( I_6 \) the local skyline tuple. It is easy to see that any tuple is a local skyline tuple on a given dimension if there are no identical dimension values.

**Theorem 1.** Let \( I \) be the dimension index of a database \( D \) and \( I_i \in I \) be an arbitrary subindex of the dimension \( i \). Then, a local
skyline tuple $t$ is a skyline tuple if and only if (1) there is no tuple $u$ such that $u[i] \prec t[i]$, or (2) for any skyline tuple $s$ such that $s[i] \prec t[i]$, we have $s \not\prec t$.

**Proof.** Let $t$ be a local skyline tuple. If $t$ is the first tuple in $L_i$, then $t$ is a skyline tuple because no tuple is better than $t$. Otherwise, let $s$ be a skyline tuple such that $s[i] \prec t[i]$, then if $s \not\prec t$, $t$ cannot be a skyline tuple since $t$ dominates $s$; if $s \prec t$, then there exists at least on dimension $k \neq i$ such that $t[k] < s[k]$ so no tuple dominated by $s$ dominates $t$. Hence, for each skyline tuple $s$ such that $s[i] \prec t[i]$, if we have $s \not\prec t$, then no tuple in the database dominates $t$, thus, $t$ is a skyline tuple. If $t[i] < s[i]$, then $s \not\prec t$ so it is meaningless to compare $t$ with $s$.

Theorem 1 allows to determine whether a tuple is a skyline tuple only with a subset of the existing skyline. Furthermore, once the dimension index has been constructed, Theorem 1 allows to switch among subindexes so that the best one containing the least of known skyline tuples can always be selected. Indeed, let $p$ be a skyline tuple, then any tuple $t$ such that $p \prec t$ can be pruned from the database in order to reduce the computation time. In this paper, we propose the notion of stop line based on the dimension index of which the effectiveness can be guaranteed.

**Theorem 2.** Let $I$ be a dimension index of a $d$-dimensional database $D$ and $p \in D$ be an arbitrary skyline tuple. Let $o_i(p)$ denote the largest offset of any tuple $x$ such that $x[i] = p[i]$ in the dimensional subindex $I_i \in I$, if all offsets $o_i(p)$, $1 \leq i \leq d$, have been reached by following a top-down traversal on all dimensions, then the complete set of skyline tuples has been identified and the computation can be terminated.

**Proof.** Let $p$ and $t \in D \setminus p$ be two skyline tuples in $D$, we have: (1) $t \not\prec p$; or (2) $t$ and $p$ have identical values on all dimensions. We denote $L_p = \bigcup_{1 \leq i \leq d} o_i(p)$ the set of all offsets $o_i(p)$. In the first case, $t \not\prec p \Rightarrow \exists k$ such that $p[k] < t[k]$, i.e. $o_k(p) < o_k(t)$, hence, if the index traversal reaches all offsets in $L_p$, $t$ must have been identified at least in the dimension $k$. In the second case, we have $p[i] = t[i]$ on any dimension $i$. In both cases, if all offsets in $L_p$ have been reached, all skyline tuples have been identified.

We call the set $L_p = \bigcup_{1 \leq i \leq d} o_i(p)$ a stop line that can safely terminate the skyline computation of SDI. In theoretical, any skyline tuple can be selected as a stop line, however, different stop lines behave differently in pruning irrelevant tuples. We propose therefore a function $\min_{\text{stop}}$ to find the best stop line $L_p^*$ defined as:

$$L_p^* = \min_{\text{stop}}(p) = \arg \min_{p} \{\max(o_i(p)), \sum_{i=1}^{d} o_i(p)\},$$

where $d$ is the dimensionality of the data. The function $\min_{\text{stop}}$ sorts first by the maximum offset, then by the sum of offsets in all dimensions, so the skyline tuple having the minimized value is the best stop line. The best stop line $L_p^*$ can be dynamically maintained by keeping $\min_{\text{stop}}(p) < \min_{\text{stop}}(t)$ for any two skyline tuples $p$ and $t$.

The incomparability checking is taken into account while a dominance test is proceeding. In our approach, we consider that a $d$-dimensional dominance test runs in $O(d)$ time, so any tuple comparison better than $O(d)$ time shall improve the efficiency of SDI. Indeed, to efficiently determine $s \not\prec t$ in steady of testing $s < t$ in the case of $s \not\prec t$ is an essential time-costly task while comparing $t$ with all existing skyline tuples in a dimensional subindex. In this paper, we propose a weak checking mechanism of the incomparability between a skyline tuple $s$ and a testing tuple $t$ in a dimensional subindex $I_i$ as following.

**Theorem 3.** Let $I$ be a dimension index and $s$ be a skyline tuple present in a dimensional subindex $I_i \in I$. Given a tuple $t$ such that $s[i] < t[i]$, we sufficiently have $s \not\prec t$ if $\max(s) > \max(t)$, or $\min(s) < \min(t)$, or $\sum(s) > \sum(t)$.

**Proof.** The sets $L_s$ and $L_p$ (see the proof of Theorem 2 for the definition) can be considered as the coordinates of two curves in a two-dimensional space. In Euclidean geometry, $\max(s) > \max(t)$, or $\min(s) < \min(t)$, or $\sum(s) > \sum(t)$ are sufficient conditions for the existence of at least one intersection of the curves formed by $s$ and $t$ since we have $s[i] < t[i]$, i.e. $o_i(s) < o_i(t)$, that is, according to the definition of dominance, $s \not\prec t$.

Note that the maximal value, the minimal value, and the sum of a tuple can be pre-calculated while constructing the dimension index, so Theorem 3 can efficiently determine $s \not\prec t$. However, Theorem 3 shows in fact 3 sufficient conditions for $s \not\prec t$, hence a dominance test is necessary to determine $s \not\prec t$ in the cases that are not covered by Theorem 3, for which we call Theorem 3 a weak incomparability checking. The sketch of SDI is listed in Algorithm 1.

**Algorithm 1:** SDI

```
Input: Dimension index $I$
Output: Skyline $S$
1 while true do
2    $I_{\text{best}} \leftarrow \text{BestSubindex}(I)$
3    while $T \leftarrow \text{NextLocalSk}(I_{\text{best}})$ do
4        if $T = \text{null}$ then
5            return $S$
6        foreach $t \in T$ and $t \notin S$ do
7            if $S_{\text{best}} \not\prec t$ then
8                $S_{\text{best}} \leftarrow S_{\text{best}} \cup t$
9                $S \leftarrow S \cup t$
10            if Found new skyline tuples then
11                Update StopLine
12                break
13        if StopLine is reached then
14            return $S$
```

Extensions. (1) SDI computes the skyline in the categorical domains as long as the preference order $\prec$ can be defined; (2) SDI can be immediately adapted to subspace skyline computation by skipping unrelated dimensions; (3) SDI can be extended to the skyline maintenance by dynamically constructing the dimension index; (4) SDI can handle the top-$k$ skyline query by finding the skyline tuples having the best positions in the dimension index.

### 3 THEORETICAL ANALYSIS

We denote $d$ the dimensionality and $N$ the cardinality of the data, and $M$ the size of the skyline. We discuss without duplicate values on any dimension, but if $K$ duplicate values are present in a dimensional subindex, $O(dK^2)$ shall be considered in assuming that BNL is applied to compute local skylines. Note that we consider $O(d)$ time for a $d$-dimensional dominance test, which implies the tests of $s < t$ and $t < s$, hence, the dominance test...
within SDI is considered in $O(d/2)$ time since $s < t$ is sufficient. We also note that the construction of dimension index requires $O(dN \log N)$ time with respect to general $O(N \log N)$ sorting algorithms on $d$ dimensions, and we do not consider the two heuristics of stop line and incomparable checking.

The average time complexity of SDI is measured on $M$ skyline tuples uniformly distributed on $d$ dimensional subindexes.

**Theorem 4.** Considering $M$ skyline tuples, SDI computes the skyline of a $d$-dimensional database with the cardinality $N$ in $O(dN \log N + \frac{M(2N - M - d)}{4})$.

**Proof.** With $M$ skyline tuples uniformly distributed on each dimension, $(N - M)/d$ non skyline tuples must be compared with $M/d$ skyline tuples. For each dimension, in the worst case, $(M/d)(M/d - 1)/2$ dominance tests are required by skyline tuples, $(N - M)/d(M/d)$ dominance tests are required by non skyline tuples, and each dominance test cost $O(d/2)$ time. Therefore, the average time complexity of SDI is $O((M/d)(M/d - 1)/2 + ((N - M)/d)(M/d)(d/2)d)$, that is the result shown in Theorem 4. □

**Theorem 5.** In the worst case, all $N$ tuples in a $d$-dimensional database are skyline tuples. SDI computes the skyline in $O(dN \log N + N^2 - dN)$. □

**Proof.** The proof is immediate if we replace $M$ in Theorem 4 by $N$.

Comparing with the best known worst-case time complexity $O(N \log^\text{max}(1,d^{-2}) N)$ proposed by Sheng and Tao [13], given $d > 2$, the following equation must be resolved:

$$N \log^{d-2} N > dN \log N + \frac{N^2 - dN}{4}.$$ 

The above equation belongs to transcendental equations that have no closed-form solutions. Our numerical simulation results presented in Figure 2 shows that while $d = 6$, SDI is better than the approach of Sheng and Tao for $N = 5 \times 10^3$; and while $d = 8$, the compared approach beats SDI only if $N > 4 \times 10^6$. SDI performs better in high-dimensionality domains with respect to a reasonable data cardinality.

**4 PERFORMANCE EVALUATION**

We evaluate the performance of SDI in comparison with BSkyTree implemented in SkyBench implemented in SkyBench [5] on both synthetic datasets and real world datasets. The synthetic datasets are generated by Skyline Benchmark Data Generator, including uniform independent (UI), correlated (CO), and anti-correlate (AC) data; the real world datasets include NBA, HOUSE, and WEATHER [5]. We implemented SDI in C++ standard and all executables are compiled by LLVM Clang with -O3 option. All experiments have been conducted on an Intel Core i5 2.8 GHz processor with 16GB 1600 MHz DDR3 RAM, running macOS 10.15.1 operating system. We note that all results reported are the average performance over 5 iterations.

First, the effects of (1) dimensionality and (2) cardinality of data have been evaluated. For (1), the cardinality is fixed to 100K and for (2), the dimensionality of data is fixed to 24. Due to the space limitation, only overall elapsed time, that is, the sum of data loading time, data structure construction time, and query time, has been reported in this paper. Indeed, we consider that for single-round skyline queries, to focus on total processing time is much important than to focus only on the query time without looking at data structure building time.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$d = 2$</th>
<th>$d = 4$</th>
<th>$d = 6$</th>
<th>$d = 8$</th>
<th>$d = 10$</th>
<th>$d = 12$</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI</td>
<td>7</td>
<td>259</td>
<td>9,960</td>
<td>25,737</td>
<td>46,301</td>
<td></td>
</tr>
<tr>
<td>CO</td>
<td>1</td>
<td>5</td>
<td>31</td>
<td>120</td>
<td>449</td>
<td>790</td>
</tr>
<tr>
<td>AC</td>
<td>50</td>
<td>4,100</td>
<td>26,713</td>
<td>56,118</td>
<td>75,668</td>
<td>87,857</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$d = 14$</th>
<th>$d = 16$</th>
<th>$d = 18$</th>
<th>$d = 20$</th>
<th>$d = 22$</th>
<th>$d = 24$</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI</td>
<td>67,676</td>
<td>82,286</td>
<td>92,011</td>
<td>96,832</td>
<td>99,059</td>
<td>99,662</td>
</tr>
<tr>
<td>CO</td>
<td>1,439</td>
<td>2,941</td>
<td>5,471</td>
<td>8,936</td>
<td>14,498</td>
<td>16,948</td>
</tr>
<tr>
<td>AC</td>
<td>94,053</td>
<td>96,956</td>
<td>98,413</td>
<td>99,205</td>
<td>99,570</td>
<td>99,760</td>
</tr>
</tbody>
</table>

**Table 1: Skyline size of synthetic datasets ($N = 100K$).**

Table 1 lists the skyline size of synthetic datasets with the cardinality of 100K. We see that the higher the dimensionality of data, the closer to the worst case is likely to be. For instance, the skyline consists of 92% of tuples in UI data while $d = 18$, however while $d = 14$ of AC data, the skyline rate reaches already 94%. Figure 3 shows the effect of dimensionality on SDI where the cardinality $N = 100K$ is reasonable in the most of use cases. SDI outperforms BSkyTree in both low-dimensionality and high-dimensionality domains on UI ($d \leq 10$ or $d > 20$) and AC data ($d \leq 4$ or $d > 20$), but is less efficient than BSkyTree in other dimensionalities. In fact, we note that the stop line takes no advantage in AC data because of the anti-correlated characteristics of data, however SDI systematically outperforms BSkyTree on the CO data because the stop line can efficiently determined with respect to the strong correlation in data. Figure 4 shows the effect of cardinality on SDI with the highest dimensionality in our experiments. In high-dimensionality domains, SDI outperforms BSkyTree in most cases except in AC data. Again, we confirm that the stop line does not show any advantage in high-dimensionality.

1https://github.com/Sean-Chester/SkyBench
2http://philfoundry.org/projects/randdataset
3https://github.com/skyline-sdi/sdi-bench
and high-cardinality AC data. Table 2 lists the pruned tuples by the stop line in different synthetic datasets while \( N = 100K \), that are relevant to our experimental results.

| Dataset | \( d \) | \( N \) | \(|S|\) | Elapsed time (s) |
|---------|-------|-------|-------|-----------------|
| UI      | 2     | 99,849| 88,011| 63,260          |
|         | 4     | 99,997| 97,773| 98,832          |
| AC      | 2     | 36,731| 5,022 | 2,227           |
|         | 4     | 97,656| 96,526| 92,066          |

Table 2: Pruned tuples by the stop line (\( N = 100K \)).

Table 3 shows the performance comparison between SDI and BSkyTree on real datasets. SDI outperforms BSkyTree on NBA and HOUSE datasets but is much slower than BSkyTree on the WEATHER dataset because the huge number of duplicate dimension values in WEATHER makes \( O(dK^2) \) (discussed in Section 3) an important factor.

| Dataset | \( d \) | \( N \) | \(|S|\) | SDI  | BSkyTree |
|---------|-------|-------|-------|------|----------|
| HOUSE   | 2     | 127,931| 5,774 | 306 ms | 839 ms |
| NBA     | 2     | 8      | 17,264| 1,796 | 45 ms    |
| WEATHER | 2     | 566,268| 26,713| 18,680 ms | 11,641 ms|

Table 3: Performance evaluation on real datasets.

5 CONCLUSION

In this paper, we presented an efficient Skyline computation algorithm. We proved that in multidimensional databases, skyline computation can be conducted on an arbitrary dimensional index which is constructed with respect to a predefined total order that determines the skyline. We further showed that any skyline tuple can be used to stop the computation process by outputting the complete skyline. Our experimental evaluation shows that SDI outperforms the state-of-the-art skyline algorithm in both low-dimensionality and high-dimensionality domains.

REFERENCES