Efficient Skyline Maintenance for Streaming Data with Partially-Ordered Domains

Yuan Fang[1] and Chee-Yong Chan[2]

[1] Institute for Infocomm Research, Singapore*. yfang@i2r.a-star.edu.sg
[2] National University of Singapore. chancy@comp.nus.edu.sg

Abstract. We address the problem of skyline query processing for a count-based window of continuous streaming data that involves both totally- and partially-ordered attribute domains. In this problem, a fixed-size buffer of the N most recent tuples is dynamically maintained and the key challenge is how to efficiently maintain the skyline of the sliding window of N tuples as new tuples arrive and old tuples expire. We identify the limitations of the state-of-the-art approach STARS, and propose two new approaches, STARS$^+$ and SkyGrid, to address its drawbacks. STARS$^+$ is an enhancement of STARS with three new optimization techniques, while SkyGrid is a simplification STARS that eliminates a key data structure used in STARS. While both new approaches outperform STARS significantly, the surprising result is that the best approach turns out to be the simplest approach, SkyGrid.

1 Introduction

Due to the usefulness of skyline queries in identifying interesting data points and its conceptual simplicity, there is a lot of research attention on how to efficiently process skyline queries. Given a set of tuples S, a skyline query (with respect to a collection A of attributes of interest) returns the subset of S (the so called "*skyline*") that are dominating with respect to A. Specifically, a tuple t_x **dominates** another tuple t_y iff t_x is better than or equal to t_y in every attribute in A, and is strictly better in at least one such attribute. Thus, the **skyline** of S, which consists of all tuples in S that are not dominated by any tuple in S, represents the subset of the most interesting points (with respect to A).

Using the popular example of a tourist who is looking for a hotel that is both cheap as well as close to the city, the "skyline" hotels that are of interest to the tourist are the hotels not dominated by any other hotel, where a hotel h_x dominates another hotel h_y if it satisfies the following conditions: (1) $h_x.price \leq h_y.price$, (2) $h_x.distance \leq h_y.distance$, and (3) at least one of the inequalities in (1) and (2) is strict.

Much of the early work on skyline queries are in the context of attributes with *totally-ordered* domains (as illustrated by the skyline hotel example), and

* Part of this work was done when the author was a student at National University of Singapore.

focuses on query processing in *offline* environment where a skyline result is computed in response to a query on a disk-resident dataset (e.g., [5, 8, 6]). Recent research effort has shifted towards query processing in *online* environment, where a skyline result is dynamically maintained for a long-standing skyline query over continuous streaming data [10, 12].

The key challenge for the streaming data environment is how to efficiently update the skyline for a *sliding window* of tuples. There are two models for the sliding window length N in streaming data applications. In the *time-based window* model, N represents the lifespan of each tuple in some number of time units. Each arriving tuple t_i has an arrival time-stamp s_i and expires after $s_i + N$ time units. Thus, the skyline is computed over all non-expired tuples and is updated whenever a new tuple arrives or an existing tuple expires [12]. In the *count-based window* model, the skyline is maintained for the most recent N tuples [10]. Thus, the skyline is updated whenever a new tuple arrives, and the arrival of the new tuple may also cause the oldest existing tuple to expire if there are already N tuples before the new arrival.

Several recent work on skyline queries have broadened the scope to include categorical attributes with *partially-ordered* domains in both offline [2, 9] as well as online [10] environment. Categorical attributes are more general than numerical attributes as the dominance relationships among the domain values for a categorical attribute are based on a partial ordering instead of a total ordering.

In this paper, we address the problem of skyline query processing for a count-based window of continuous streaming data that involves both totally- and partially-ordered attribute domains. In this problem, a fixed-size buffer of the N most recent tuples is dynamically maintained and the key challenge is how to efficiently maintain the skyline of the sliding window of N tuples as new tuples arrive and old tuples expire. The state-of-the-art approach for this skyline problem is the STARS method [10], which is based on two key data structures: a multi-dimensional grid to organize the tuples in the buffer, and a geometric arrangement structure to organize the skyline tuples.

There are many interesting applications that require dynamic skyline maintenance of streaming objects in our setting. Consider an Internet search webservice that continuously accepts search requests from users, where each search request is associated with various categorical attributes of interest such as the search language, geographical region of the request, and the browser software and operating system used. The webservice can define a partial order over the attributes indicating its preferences of the search requests it wants to track. The system will then filter out and maintain a subset of recent interesting search requests, which can be exploited to study trends for better search results. Another application in news services is illustrated in [10].

In this paper, we make the following contributions. We identify the limitations of the STARS method and propose two new approaches, STARS$^+$ and SkyGrid, to address its drawbacks. STARS$^+$ is an enhancement of STARS with three new optimization techniques, while SkyGrid is a simplification STARS that completely eliminates a key data structure used in STARS. While both new ap-

proaches outperform STARS significantly, the surprising result is that the best approach turns out to be the simplest approach, SkyGrid, which outperforms both STARS and STARS$^+$ by up to a factor of 3 and 2.1, respectively.

The rest of this paper is organized as follows. In Section 2 we review the STARS algorithm. We present two new approaches, STARS$^+$ and SkyGrid, in Sections 3 and 4, respectively. In Section 5, we present an experimental evaluation of the proposed algorithms. Finally, Section 6 concludes the paper. Due to space constraint, all proofs are omitted.

2 Overview of STARS Approach

In this section, we give an overview of the STARS approach [10], which is the state-of-the-art algorithm for the skyline problem that we are addressing in this paper and the basis of our STARS$^+$ approach.

The STARS approach, which is based on the count-based sliding window model, maintains a fixed-size buffer of N tuples and updates the skyline of the N most recent tuples as new tuples arrive and old tuples expire. Whenever a new tuple t_{in} arrives and the buffer is already full with N tuples, the oldest tuple t_{out} in the buffer is expired and t_{in} becomes part of the skyline if it is not dominated by any other tuples in the buffer. Moreover, if t_{out} was a skyline tuple, then it is possible for some of the non-skyline tuples in the buffer to be *promoted* to become skyline tuples. Specifically, for each non-skyline tuple t in the buffer, if t is exclusively dominated by t_{out} (i.e., t_{out} is the only skyline tuple that dominates t), then t is promoted to a skyline tuple.

To avoid unnecessary dominance comparisons, STARS minimizes the set of tuples in the buffer by discarding *irrelevant tuples* from the buffer. A tuple t in the buffer is classified as irrelevant if it is dominated by a younger tuple t' (i.e., t' arrives later than t). The reason is that since t' will only expire after t, t is guaranteed to be dominated by at least one tuple throughout its remaining lifespan in the buffer which means that t can never be promoted to a skyline tuple. Thus, any tuple t that is dominated by a newly arrived tuple t_{in} is irrelevant and can be immediately discarded from the buffer. STARS refers to the buffer containing only relevant tuples as the **skybuffer**.

The skyline maintenance algorithm in STARS, which is invoked whenever a new tuple arrives, is shown in Fig. 1. The algorithm has the following inputs: SB is the buffer of relevant tuples (skybuffer), $S \subseteq SB$ is the set of skyline tuples, t_{in} is the newly arrived tuple, and t_{out} is the oldest tuple to be expired.

The key operations in the maintenance algorithm can be classified into the following three types of queries. (1) **D-query**: Given a tuple t, return the set of buffer tuples that are dominated by t; (2) **S-query**: Given a tuple t, determine whether t is dominated by any skyline tuple; and (3) **P-query**: Given a skyline tuple t that expires, return the set of buffer tuples that are promoted to skyline tuples due to the expiry of t. In Fig. 1, a D-query is used in steps 2 and 4, an S-query is used in step 1, and a P-query is used in step 8. Note that a P-query can be evaluated in terms of a D-query and multiple S-queries: given an

Algorithm: `SkylineMaintenance` (SB, S, t_{in}, t_{out})

Input: SB is the skybuffer.
$S \subseteq SB$ is the skyline.
t_{in} is the newest (arriving) tuple.
t_{out} is the oldest (expiring) tuple.

1) **if** t_{in} not dominated by any tuple in S **then**
2) Remove tuples dominated by t_{in} from S;
3) Insert t_{in} into S;
endif
4) Remove tuples dominated by t_{in} from SB;
5) Insert t_{in} into SB;
6) **if** t_{out} is in S **then**
7) Remove t_{out} from S;
8) $P = \{t \in SB :$
t is exclusively dominated by $t_{out}\}$;
9) Insert tuples in P into S;
endif
10) Remove t_{out} from SB.

Fig. 1: Skyline maintenance framework of STARS

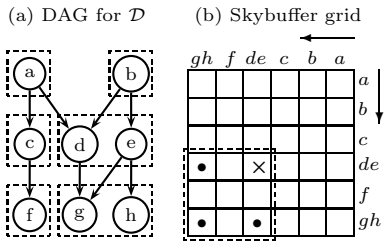

Fig. 2: Skybuffer organization

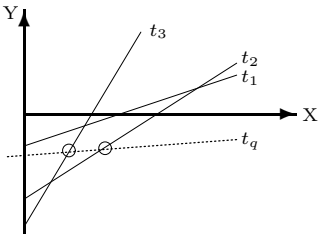

Fig. 3: Skyline organization

expiring skyline tuple t, a D-query is first used to find the set of tuples T that are dominated by t, and then for each tuple t' in T, an S-query is used to check if t' is exclusively dominated by t.

To efficiently support the core operations (i.e. D-queries and S-queries) for skyline maintenance, the STARS approach organizes the buffer tuples and skyline tuples using two key data structures.

Multi-dimensional Grid. Suppose the skyline is computed wrt d attributes, $A_1, \cdots A_d$. STARS organizes the tuples in the buffer using a d-dimensional grid, where the i^{th} dimension corresponds to attribute A_i. The objective is to map and store each tuple into a grid cell to support efficient D-queries.

To enable this mapping, the partially-ordered domain of each categorical attribute is linearized into a total ordering by a topological sort of the attribute domain's partial order. More specifically, the partially-ordered domain of a categorical attribute is represented by a directed acyclic graph (DAG), where each vertex in the DAG represents a domain value, and each edge represents the dominance relationship between two attribute values that cannot be be inferred by transitivity such that a value v is better than v' iff there exists a directed path from v's vertex to v''s vertex in the DAG. Let $r(v)$ denote the rank of the vertex corresponding to value v in a topological sort of the DAG. It follows that if $r(v) > r(v')$, then v cannot dominate v'. However, if $r(v) < r(v')$, then either v dominates v' or the two values are incomparable.

In this way, the scales of the grid on each dimension is bucketized into as many buckets as the number of domain values. Thus, each tuple $t = (a_1, \cdots, a_d)$ is mapped into the cell given by $\langle r(a_1), \cdots, r(a_d)\rangle$. To find the set of buffer tuples that are dominated by t (i.e., evaluate a D-query), STARS only needs to consider tuples $t' = (a'_1, \cdots, a'_d)$ that are located in the cells satisfying the

following range query wrt t: $r(a_1) \leq r(a'_1)$,$\cdots$, and $r(a_d) \leq r(a'_d)$. Additionally, many cells satisfying the range query are false positives and can be pruned as well. Thus, the grid organization enables STARS to eliminate many unnecessary dominance comparisons against tuples that cannot be dominated by t. To make the method scale to a large number of attributes or large attribute domains, STARS introduces techniques to control grid granularity by grouping multiple values into the same bucket. We use Fig. 2 (from [10]) to illustrate this.

Example 1. Consider the domain $\mathcal{D}$ of a categorical attribute consisting of the values $\{a, \cdots, h\}$ that are organized into the partial order depicted in Fig. 2(a). A possible topological sort is $a, \cdots, h$, which can be grouped into six buckets, each indicated by a dotted box in Fig. 2(a). A grid to organize a 2D dataset on $\mathcal{D} \times \mathcal{D}$ is depicted in Fig. 2(b). Consider a tuple t that is mapped to the cell marked $\times$ in Fig. 2(b). The dotted region in Fig. 2(b), which corresponds to the range query wrt t, represents the set of cells that could contain tuples dominated by t. Note that among the nine cells in the region, only the three cells marked $\bullet$ are candidate cells; the remaining six cells are false positives that cannot contain tuples dominated by t, which can be eliminated as well. □

Geometric Arrangements. To efficiently support S-queries, STARS organizes the skyline tuples using a geometric arrangement of lines that maps skyline tuples onto a 2D plane. For this mapping, STARS needs to first choose two of the attributes (say A_i and A_j) among the attributes of interest for the skyline computation. Then each skyline tuple $t = (a_1, \cdots, a_d)$ is represented by a line $y = r(a_i) \cdot x - r(a_j)$ in the 2D plane, where a_i and a_j are t's values for attributes A_i and A_j, respectively. Based on this geometric line arrangement, two tuples t and t' are incomparable if the intersection point (x_I, y_I) of their line representations has $x_I < 0$. STARS uses the doubly-connected-edge-list (DCEL) data structure [4] to represent the positive half (wrt the x-axis) of the line representation of each skyline tuple. Using DCEL, STARS is able to efficiently evaluate an S-query wrt a tuple t by retrieving the lines that intersect with t's line in the positive half of the x-axis. In this way, many skyline tuples incomparable to t are pruned. Moreover, evaluating an S-query is progressive as it can terminate once an intersecting skyline tuple is found. We use the example (from [10]) shown in Fig. 3 to illustrate this concept.

Example 2. Suppose there are three skyline tuples t_1, t_2, and t_3. Their line representations are shown as labelled in Fig. 3. Consider an S-query wrt a tuple t_q, which is represented by the line as labelled in Fig. 3. Using DCEL, t_3 is the first line found to intersect with t_q, and a dominance comparison is performed to check if t_q is dominated by t_3. If so, then the the evaluation of the S-query completes; otherwise, the next line that intersects t_q is t_2 and a dominance comparison between t_2 and t_q is performed and so on. Observe that t_1 is pruned as it intersects with t_q at the point (x_I, y_I) where $x_I < 0$. □

3 STARS$^+$ Approach

In this section, we present three optimization techniques to improve the performance of STARS. We refer to this optimized variant as STARS$^+$. The first technique reduces the number of S-queries required for evaluating P-queries. The second technique introduces auxiliary structures to improve the evaluation of D-queries. The third technique optimizes the line arrangement technique to improve the evaluation of S-queries. Our experimental results show that STARS$^+$ significantly outperforms the unoptimized STARS.

3.1 Dominating Tuple (DT) Optimization

In the STARS approach, each P-query (step 6 in `SkylineMaintenance` algorithm) to find the tuples that are exclusively dominated by an expiring skyline tuple is evaluated in terms of one D-query and multiple S-queries, which incurs a rather high computation overhead. One way to speed up a P-query evaluation is to reduce the number of S-query evaluations.

One approach to reduce the the number of S-query evaluations is to keep track of the number of tuples that dominate each tuple. This idea is referred to as the "eager approach" [12] in contrast to the non-optimized "lazy approach". Specifically, each tuple t is associated with a counter, denoted by $t.counter$, which represents the number of skyline tuples that dominate t. When t first arrives, $t.counter$ is initialized to the number of skyline tuples that dominate t. Subsequently, whenever a skyline tuple t_{out} expires, for each tuple t dominated by t_{out}, $t.counter$ is decremented by one to indicate that t is dominated by one fewer tuple due to the expiry of t_{out}. Clearly, if $t.counter > 0$, we can conclude that t is not exclusively dominated by t_{out} without requiring a S-query evaluation. While the advantage of the eager approach is that a P-query can be evaluated with significantly fewer S-queries, the drawback is that the initialization of $t.counter$ requires the entire skyline to be scanned when t arrives. In fact, the performance of the eager approach was shown to be worse than the lazy approach [12].

To avoid the overhead of the eager approach, we adopt a "semi-eager" approach for STARS$^+$ that simply associates each tuple t with a single skyline tuple, denoted by $t.dt$, that dominates t. The knowledge about this dominating tuple $t.dt$ is available virtually "for free" as part of the S-query issued to check if t is a skyline tuple when t arrives; thus, only a minor modification to the S-query evaluation procedure is needed to return a skyline tuple that dominates t when t is not a skyline tuple. Subsequently, whenever a skyline tuple t_{out} expires, for each tuple t dominated by t_{out}, if $t.dt$ is not equal to t_{out}, then t is not exclusively dominated by t_{out} and we save the cost of an S-query to determine this. However, if $t.dt$ is equal to t_{out}, an S-query is invoked to check if there is another skyline tuple (besides t_{out}) that dominates t. If there is indeed another tuple t' that dominates t, then we update $t.dt$ to be t' and conclude that t is not exclusively dominated by t_{out}.

The following result shows that our proposed DT optimization can significantly reduce the number of S-queries to be evaluated for a P-query.

Theorem 1. *Suppose that a newly arrived tuple t in the skybuffer is dominated by all the existing skyline tuples. Let s denote the number of the skyline tuples. If no new skyline tuple is encountered, then with the* DT *optimization, the total expected number of S-queries that are executed (to check if t should be promoted) due to the expiry of the s skyline tuples is bounded by $\Theta(\ln s)$. In contrast, the total number of such S-query evaluations for the lazy approach is s.*

3.2 Empty Cell (EC) Optimization

Recall that in STARS, a D-query wrt a tuple t is evaluated by examining all tuples in each candidate cell within the region specified by a range query wrt t. We observe that many candidate cells are empty, particularly for high-dimensional data as the number of grid cells grows exponentially with data dimensionality. Consequently, a large overhead is incurred in examining empty cells.

To get an idea of the sparsity of the grid cells, let us consider a d-dimensional dataset and a buffer size of N tuples. Assuming the attribute values are independent, the average number of tuples in the skybuffer is given by $O(\ln^d N)$ [7]. Let the granularity of each grid dimension be g (i.e., each dimension scale has g buckets). Then, the number of tuples per grid cell is given by $\rho = \frac{\ln^d N}{g^d}$. For high-dimensional data, ρ is often very small (e.g., $\rho = 0.022$ when $N = 10^5, d = 4$ and $g = 30$).

To reduce the overhead of examining empty grid cells when evaluating D-queries in a d-dimensional grid, the Empty Cell (EC) optimization technique maintains $d-1$ additional structures, termed *index grids*, to keep track of the number of tuples in the grid. Each index grid C_i ($1 \leq i \leq d-1$) is i-dimensional, having the same scales as the first i dimensions of the original grid. All the cells in C_i have an initial value of 0. When a tuple is added to or removed from the buffer, `EC-Indexing` in Fig. 4(a) is invoked to update the index grids. During the evaluation of a D-query, the candidate cells are examined by enumerating the cell coordinates in a systematic manner: for each prefix of the d-length enumeration, STARS$^+$ invokes `EC-Checking` in Fig. 4(b) to check if the enumeration for the current prefix can be terminated due to an empty region. If a *true* value is returned, STARS$^+$ terminates further enumeration for the current prefix and backtracks.

Example 3. For a 3D grid, two index grids C_1 and C_2 are maintained. All of their cells are initialized to 0. Suppose a tuple is added to the buffer at $\langle 2, 5, 3\rangle$. Then $C_1\langle 2\rangle$ and $C_2\langle 2, 5\rangle$ are updated to 1. A D-query evaluation starts enumerating the candidate cells to be examined with the enumeration prefix $\langle 1\rangle$. Since $C_1\langle 1\rangle = 0$, STARS$^+$ terminates further enumeration with this prefix, and backtracks to the next prefix $\langle 2\rangle$. Since $C_1\langle 2\rangle \neq 0$, STARS$^+$ continues the enumeration with the next dimension to consider $\langle 2, 1\rangle$. Since $C_2\langle 2, 1\rangle = 0$, STARS$^+$ terminates further enumeration with $\langle 2, 1\rangle$ and backtracks to $\langle 2, 2\rangle$. Since $C_2\langle 2, 2\rangle = 0$, STARS$^+$ continues backtracking until $\langle 2, 5\rangle$. □

When a D-query evaluation is enumerating a prefix with i dimensions, there is a probability of $p_i = k^{g^{d-i}}$ that the enumeration will backtrack, where k is

Alg. (a): `EC-Indexing` $(\langle k_1, k_2 \dots k_d \rangle, e)$

Input: $\langle k_1, k_2 \dots k_d \rangle$ are the coordinates in the skybuffer grid, where a tuple is added or removed.
e indicates an add or remove event.

```
1) for i = 1 to d − 1 do
2)     if e is "add" then
3)         Increase C_i⟨k_1, k_2 ... k_i⟩ by 1;
4)     else
           /* e is "remove" */
5)         Decrease C_i⟨k_1, k_2 ... k_i⟩ by 1;
       endif
   endfor
```

Alg. (b): `EC-Checking` $(\langle k_1, k_2 \dots k_m \rangle)$

Input: $\langle k_1, k_2 \dots k_m \rangle$ $(1 \leq m < d)$ is the coordinates prefix in the skybuffer grid, enumerated in a D-query.

Output: a boolean indicating if all cells with coordinates prefix $\langle k_1, k_2 \dots k_m \rangle$ are empty.

```
1) for i = 1 to m do
2)     if C_i⟨k_1, k_2 ... k_i⟩ is 0 then
3)         return true
       endif
   endfor
4) return false
```

Fig. 4: Empty Cell (EC) optimization

the average probability that a cell is empty. Therefore, a D-query evaluation *with* EC is expected to examine a fraction λ of the candidate cells, where $\lambda = \prod_{i=1}^{d-1}(1-p_i) = \prod_{i=1}^{d-1}(1-k^{g^{d-i}}) \leq 1-k^g$. Suppose $k = 0.99$ and $g = 30$, then $\lambda < 0.26$. As d or g increases, k approaches 1, and so EC becomes more effective.

The overhead incurred by EC is low. The cost to update the index grids when a tuple is added or removed is $O(d)$ which is negligible since d is usually small. The space overhead for each index grid C_i is $O(g^i)$; thus, the total space requirement of $O(g^{d-1})$ is insignificant relative to the $O(g^d)$ space requirement of the original grid.

3.3 Geometric Arrangement (Minmax) Optimization

Our third optimization concerns the geometric arrangement technique for evaluating S-queries. In STARS, the two attributes used for line mapping are selected arbitrarily when data dimensionality is higher than two. To assess the performance impact of the choice of the attribute pair, we conducted an experiment to compare the performance of skyline maintenance for every possible attribute pair and found that the performance gap between the best and worst pair can exceed 20%. Thus, the choice of the attribute pair for the mapping is important but there is no clear heuristic that can be used to optimize this selection. Another drawback of STARS is that it utilizes only two attributes for the mapping. Intuitively, using more attributes is likely to provide better pruning power as more information about the data is being exploited.

We present an enhanced variant of the line mapping, termed Minmax, that utilizes all attributes. Consider a d-tuple $t = (a_1, \cdots, a_d)$. Minmax maps t to the line $y = C \cdot x - D$, where $C = \max(r(t.a_1), \cdots, r(t.a_d))$ and $D = \min(r(t.a_1), \cdots, r(t.a_d))$. The following result establishes its correctness.

Theorem 2. *Let l_1 and l_2 represent the two lines mapped from two d-tuples t_1 and t_2 based on Minmax, respectively. If l_1 and l_2 intersect at the point (x_I, y_I) where $x_I < 0$, then t_1 and t_2 are incomparable.*

4 SkyGrid Approach

In this section, we present a more extreme approach to optimize STARS by actually eliminating the use of the geometric arrangement technique for S-queries. Instead, all skyline maintenance operations are performed using only the grid data structure. To distinguish between the skyline and non-skyline tuples in the buffer, each tuple is associated with a single bit that is set to *true* iff the tuple is a skyline tuple. We refer to this new approach as SkyGrid.

The simplified skyline maintenance framework for SkyGrid is shown in Fig. 5. Clearly, using only a single data structure in SkyGrid simplfies the skyline maintanance operations. Recall that for both STARS and STARS$^+$ (refer to Fig. 1), if t_{in} is a skyline tuple, we need to update two structures with the following operations: (1) remove the line representations of any skyline tuples that are dominated by t_{in} (step 2); (2) insert the line representation of t_{in} (step 3); (3) remove the tuples in the buffer that are dominated by t_{in} (step 4); and (4) insert the t_{in} into the buffer (step 5). In contrast, for SkyGrid, if t_{in} is a skyline tuple, only the grid structure needs to be updated with the following operations (refer to Fig. 5): (1) insert t_{in} into the buffer (step 3); and (2) remove the tuples in the buffer that are dominated by t_{in} (step 4).

The simpler skyline maintenance operations in SkyGrid results in better performance. In STARS$^+$, the cost of inserting or removing the line representation of a skyline tuple in the geometric arrangement is $O(s)$ using DCEL, where s is the number of skyline tuples [4]. In contrast, for SkyGrid, the cost for promoting a tuple into the skyline is only $O(1)$ (by marking a skyline status bit).

Algorithm: `SkylineMaintenance+` (SB, t_{in}, t_{out})
Input: SB is the skybuffer.
t_{in} is the newest (arriving) tuple.
t_{out} is the oldest (expiring) tuple.

1) **if** t_{in} not dominated by skyline tuples in SB **then**
2) Mark t_{in} as "skyline";
endif
3) Insert t_{in} into SB;
4) Remove tuples dominated by t_{in} from SB;
5) Remove t_{out} from SB;
6) **if** t_{out} was marked as "skyline" **then**
7) $P = \{t \in SB :$
t is exclusively dominated by $t_{out}\}$;
8) Mark tuples in P as "skyline";
endif

Fig. 5: Simplified skyline maintenance framework

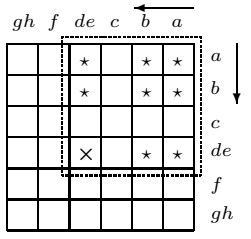

Fig. 6: S-query in SkyGrid

To support S-queries, SkyGrid can simply find the candidate cells in a similar way as in D-queries, but in the opposite direction. Specifically, SkyGrid only needs to consider d-tuples $t' = (a'_1, \cdots, a'_d)$ that are located in the cells satisfying the following range query wrt $t = (a_1, \cdots, a_d)$: $r(a_1) \geq r(a'_1), \cdots$, and $r(a_d) \geq r(a'_d)$. The following example illustrates this idea.

Example 4. Consider the domain $\mathcal{D}$ depicted in Fig. 2(a). A grid to organize a 2D dataset on $\mathcal{D} \times \mathcal{D}$ is depicted in Fig. 6. Consider a tuple t that is mapped to the cell marked $\times$ in Fig. 6. The dotted region in Fig. 6, which corresponds to the range query wrt t, represents the set of cells that could contain tuples dominating t. The actual candidate cells for the S-query are marked by $\star$. □

Next, we compare the pruning potential of STARS$^+$ and SkyGrid. As S-query is progressive in both, we ignore the progressiveness. Assuming independent attribute values, the following result states the expected pruning ratio of STARS$^+$.

Theorem 3. *STARS$^+$ (utilizing the Minmax mapping) is expected to prune fewer than half of the number of skyline tuples in an S-query evaluation.*

On the other hand, we expect SkyGrid to be able to prune more skyline tuples. While a formal computation is difficult as it depends on data domains, we can obtain an estimation. The evaluation of an S-query wrt to a tuple t examines a fraction $\prod_{i=1}^{d} k_i$ of all the cells as candidates, where $k_i \in [0, 1]$ is the fraction of buckets dominating t on each dimension. Hence $(1 - \prod_{i=1}^{d} k_i)$ of the cells are pruned. For high dimensional data with a reasonable value of k_i (e.g., $d > 2$ and $k_i < 0.7$), the estimated number of cells (and hence tuples) that are pruned is more than half.

To further improve performance, SkyGrid also incorporates both the DT and EC optimizations of STARS$^+$. Note that we can use two sets of EC index grids for the buffer and skyline, respectively. In this way, when evaluating an S-query, SkyGrid identifies candidate cells by utilizing only the index grids for the skyline. This avoids the need to examine most candidate cells that contain no skyline tuples, thereby reducing the number of skyline status bits that have to be checked.

In terms of space requirement, the cost for STARS$^+$ is $O(s^2)$ using DCEL to organize s skyline tuples as a geometric arrangement [4]. In contrast, since SkyGrid organizes the skyline tuples as part of the skybuffer, no additional space is required. However, each tuple in SkyGrid requires a skyline status bit, so an extra $O(s_b)$ space is needed, where s_b is the size of the skybuffer. When s is reasonably large, STARS$^+$ incurs a higher space overhead than SkyGrid.

5 Experimental Evaluation

5.1 Experiment Settings

In our experiments, we generated synthetic partially-ordered domains following the approach in [10]. Each domain is modeled as a DAG and is characterized by the parameters (m, h, c, f), where m is the number of vertices, h is the height of the DAG, $c \in (0, 1]$ is the fraction of the vertices at the next level that are connected to a vertex, and f refers to the type of DAG which is either "t" for tree-like or "w" for wall-like DAG. We refer to an attribute domain by these parameters; for instance, (500, 8, 0.3, t).

We generated four 4-dimensional datasets shown in Table 1, where each column corresponds to one dataset and the i^{th} row corresponds to the domain for

the i^{th} attribute; d-dimensional datasets, where $d \in \{2, 3\}$, are generated from Table 1 by simply considering only the first d rows of the table. For each algorithm being evaluated, we ran it on each of the four datasets, and report the average performance over the four datasets (unless stated otherwise).

For each data domain, we also considered three different distributions: (1) *independent*, where the attribute values of the tuples follow a uniform distribution; (2) *correlated*, where a tuple that is good in one attribute also tends to be good in other attributes; (3) *anti-correlated*, where a tuple that is good in one attribute tends to be bad in at least one other attribute [1, 5, 11].

Table 1: Synthesized sets of data domains

Dataset I	Dataset II	Dataset III	Dataset IV
(250, 7, 0.3, t)	(120, 7, 0.2, t)	(100, 10, 0.1, w)	(500, 8, 0.3, t)
(180, 6, 0.6, t)	(120, 7, 0.2, t)	(100, 10, 0.2, w)	(500, 8, 0.3, t)
(180, 20, 0.3, w)	(120, 5, 0.2, t)	(100, 10, 0.4, w)	(500, 8, 0.3, t)
(90, 4, 0.2, t)	(120, 5, 0.2, t)	(100, 10, 0.8, w)	(500, 8, 0.3, t)

Table 2: Skyline sizes

Dim	Corr	Indep	Anti
$d = 2$	240	25	45
$d = 3$	395	480	418
$d = 4$	636	3779	4444
$d = 5$	1298	12363	15875

The number of data dimensions, denoted by d, was varied from 2 to 4. Table 2 shows the skyline sizes for datasets with domain (500, 8, 0.3, t) on each attribute, using a 100K buffer with different data distributions. Note that datasets with partially-ordered domains have much more skylines than totally-ordered datasets since two tuples are more likely to be incomparable [1, 11]. For independent or anti-correlated datasets, the size of the skylines becomes very large once $d \geq 5$; therefore, finding conventional skylines for $d \geq 5$ becomes less interesting [3].

Furthermore, we varied buffer sizes from 10K to 1M. Lastly, we chose $g = 20$ as the default grid granularity if it is not stated.

All the algorithms were implemented using Java. The experiments were conducted on a 3.0GHz PC with 3GB of main memory running Windows OS.

5.2 Evaluating STARS$^+$ Optimizations

In this subsection, we evaluate the effectiveness of each of the three optimizations DT, EC and Minmax that are introduced for STARS$^+$.

Dominating Tuple. STARS$^+$ utilizes DT to improve the performance of P-queries. Figure 7 shows the average time per P-query evaluation *without* DT (normalized wrt *with* DT)[3]. DT clearly improves the evaluation of P-queries, up to 2.3 times faster. The speed-up is greater when the skyline is larger, which is often caused by a larger buffer, particularly when $d = 4$. For $d \in \{2, 3\}$, we notice a non-monotonous speed-up wrt buffer size. On lower dimensional datasets, the skyline sizes are much smaller. Their skylines soon become "saturated" (i.e., no longer growing and maybe shrinking due to randomness in data) when the buffers become larger. Hence, when the buffer increases beyond the saturation point, their skyline sizes become non-monotonous, resulting in the non-monotonicity of the performance speed-up by DT. When $d = 4$, the saturation point is well beyond 1M, so we only observe an improving speed-up.

[3] To be fair, both used the same attribute pair for line mapping.

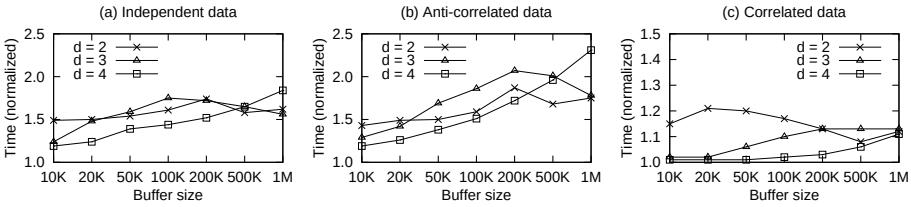

Fig. 7: Effectiveness of DT on P-query evaluation

Empty Cell. STARS$^+$ utilizes EC to improve the performance of D-query evaluation. The average time per D-query *without* EC (normalized wrt *with* EC)^3 is shown in Fig. 8. There is negligible improvement when $d = 2$, as the number of cells is small. However, when $d > 2$, EC becomes very effective. This is especially so for correlated data, where the tuples distribute unevenly in the grid resulting in more empty cells.

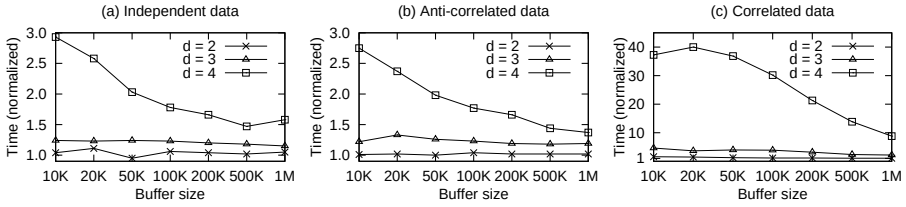

Fig. 8: Effectiveness of EC on D-query evaluation

Figure 9 compares the average evaluation time per D-query under varying grid granularity. When the granularity is initially increased from a small value, the performance of D-query evaluation both *with* and *without* EC improve due to a finer grid. However, as the granularity increases beyond 20, the performance without EC quickly deteriorates due to the rapid growth of the number of empty cells. In contrast, with EC the performance degradation is less pronounced, as most of the empty cells are pruned.

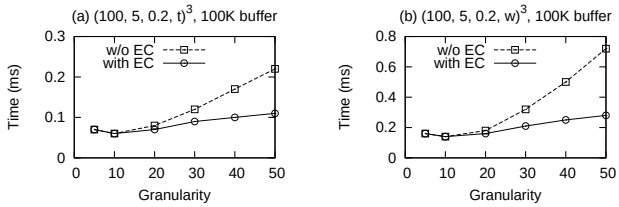

Fig. 9: Effect of grid granularity on D-query evaluation

Pruning Efficiency of Minmax. STARS$^+$ utilizes Minmax to improve the skyline organization by pruning more skyline tuples in an S-query evaluation. Following [10], we define the pruning efficiency (PE) of an S-query as the fraction of skyline tuples that require dominance comparison (i.e., that are not pruned). Thus, smaller PE values are better. Figure 10 compares the PE of S-queries for STARS, STARS$^+$ and SkyGrid, where $d = 4$. For the performance results of STARS, instead of arbitrarily choosing two attributes for line mapping, we evaluated STARS with *all* possible attribute pairs, and present the performance results corresponding to the best pair (STARS-Best) as well as the worst pair (STARS-Worst) for comparison.

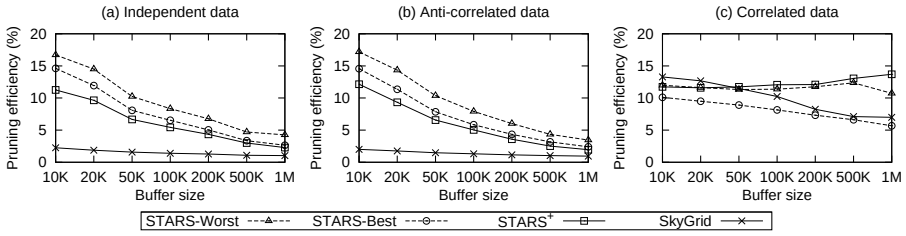

Fig. 10: Pruning efficiency of S-query

The results reveal that there could be a performance gap between STARS-Best and STARS-Worst. In Figs. 10(a) and (b), STARS$^+$ not only closes the gap, but also maintains a lead over STARS-Best. In addition, SkyGrid is much better than STARS$^+$ in terms of PE.

However, in Fig. 10(c) on correlated data, the PE of STARS$^+$ is generally on par with STARS-Worst. The reason is that the Minmax optimization in STARS$^+$ is not effective on correlated data. Consider two tuples with correlated attribute values that map to the lines $l_1 : y = C_1 \cdot x - D_1$ and $l_2 : y = C_2 \cdot x - D_2$, respectively. If $C_1 > C_2$, it is likely that $D_1 > D_2$; therefore, it is also likely that $\frac{D_1 - D_2}{C_1 - C_2} > 0$, the x-coordinate where l_1 and l_2 intersect. By Theorem 2, the two tuples are unlikely to be pruned. On the other hand, SkyGrid outperforms both Minmax and STARS-Worst, but loses marginally to STARS-Best on buffers larger than 50K. The reason is that tuples with correlated attribute values distribute unevenly in the grid, resulting in less efficient S-queries. However, SkyGrid still achieves the best overall performance despite this (see Section 5.3).

5.3 Evaluating Overall Performance

In this subsection, we compare the overall performance of STARS (both STARS-Best and STARS-Worst), STARS$^+$ and SkyGrid. To be fair to STARS and SkyGrid, we also implemented in them the two optimizations of STARS$^+$, DT and EC. Due to space constraints, we only present the results for $d = 4$; similar trends are observed for $d \in \{2, 3\}$.

Tuple update time. We measure the average time per tuple update, which corresponds to the time for one invocation of the `SkylineMaintenance` algorithm. The results are presented in Fig. 11.

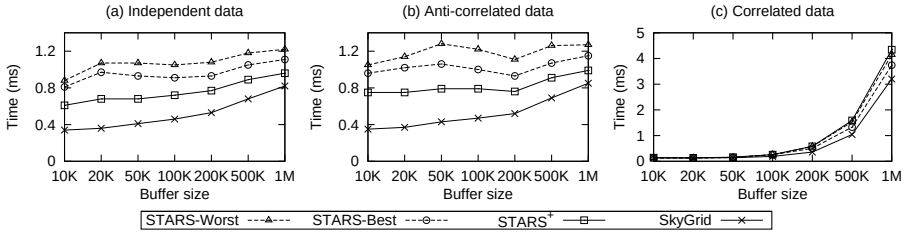

Fig. 11: Comparison of tuple update time with DT and EC

The results for independent and anti-correlated data in Figs. 11(a) and (b) reveal that SkyGrid achieves the best overall performance, followed by STARS$^+$, and lastly STARS. However, with large buffers, the performance gap between SkyGrid and STARS$^+$ narrows. The reason is that although the performance of SkyGrid for P-queries is much better than STARS$^+$, P-queries occur less frequently with large buffers due to the decreased probability for an expiring tuple to be a skyline tuple [10]. However, despite this, SkyGrid still performs better than STARS$^+$ by a clear margin with buffers as large as 1M. Note that SkyGrid is still preferable in time-critical applications, where the cost of each individual update is more important than the amortized cost. SkyGrid greatly improves the otherwise very expensive tuple updates that involve a P-query.

On correlated data, as shown in Fig. 11(c), STARS$^+$ is only marginally outperformed by STARS, although the former performs poorly in PE. Tuples with correlated attributes tend to distribute unevenly in the grid, resulting in less efficient D-queries. Thus, the performance of P-queries becomes a less dominating factor in the overall performance. Also, skylines on correlated data are generally smaller, resulting in a lower frequency of P-queries. So the PE of S-queries matters less to overall performance. This also explains why SkyGrid has the best overall performance even though it is not so in terms of PE.

Figure 12 studies the effect of grid granularity on the average time per tuple update of the three approaches with a 100K buffer. Observe that SkyGrid remains the best approach under different grid granularities.

Space requirement. The memory usage of the three approaches is shown in Table 3, with a 100K buffer on Dataset IV ($d = 4$). Clearly, SkyGrid uses the least memory, as it requires little extra space for the skyline representation. On the other hand, STARS$^+$ and STARS require comparable amount of memory, since they both use a geometric arrangement to organize the skyline. Also note that the differences are insignificant on correlated data because of a smaller skyline.

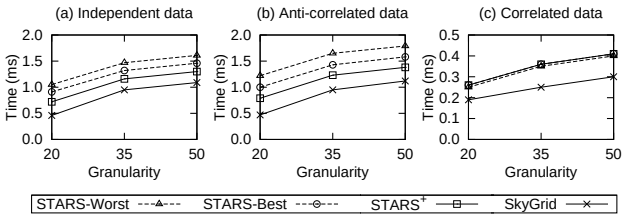

Fig. 12: Effects of granularity on tuple update time

Table 3: Memory usage

	Corr	Indep	Anti
Skyline	636	3779	4444
	Memory (MB)		
STARS -Worst	55	574	731
STARS -Best	56	505	693
STARS$^+$	55	442	589
SkyGrid	54	55	55

6 Conclusion

In this paper, we have presented two new approaches, STARS$^+$ and SkyGrid, to compute skylines for streaming data that involves partially-ordered attribute domains. Our experimental results show that both STARS$^+$ and SkyGrid outperform the state-of-the-art STARS approach, with the surprisingly result that the simplest approach, SkyGrid is the best approach.

References

1. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
2. C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, pages 203–214, 2005.
3. C.-Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. Finding k-dominant skylines in high dimensional space. In *SIGMOD*, pages 503–514, 2006.
4. M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, second edition, 2000.
5. D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: an online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.
6. K. C. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in Z order. In *VLDB*, pages 279–290, 2007.
7. X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005.
8. D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.
9. D. Sacharidis, S. Papadopoulos, and D. Papadias. Topologically-sorted skyline for partially-ordered domains. In *ICDE*, 2009.
10. N. Sarkas, G. Das, N. Koudas, and A. K. Tung. Categorical skylines for streaming data. In *SIGMOD*, pages 239–250, 2008.
11. K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.
12. Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE TKDE*, 18(3):377–391, 2006.