Categorical Skylines for Streaming Data

Nikos Sarkas
University of Toronto
nsarkas@cs.toronto.edu

Gautam Das
University of Texas at Arlington
gdas@cse.uta.edu

Nick Koudas
University of Toronto
koudas@cs.toronto.edu

Anthony K. H. Tung
National Univ. of Singapore
atung@comp.nus.edu.sg

ABSTRACT

The problem of skyline computation has attracted considerable research attention. In the categorical domain the problem becomes more complicated, primarily due to the partially-ordered nature of the attributes of tuples.

In this paper, we initiate a study of streaming categorical skylines. We identify the limitations of existing work for *offline* categorical skyline computation and realize novel techniques for the problem of maintaining the skyline of categorical data in a *streaming* environment. In particular, we develop a lightweight data structure for indexing the tuples in the streaming buffer, that can gracefully adapt to tuples with many attributes and partially ordered domains of any size and complexity. Additionally, our study of the dominance relation in the dual space allows us to utilize geometric arrangements in order to index the categorical skyline and efficiently evaluate dominance queries. Lastly, a thorough experimental study evaluates the efficiency of the proposed techniques.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications

General Terms

Algorithms, Performance

Keywords

Skyline, data stream, categorical, partial order

1. INTRODUCTION

Abundance of data has been both a boon and a curse, as it has become increasingly difficult to process data in order to isolate useful and relevant information. In order to compensate, the research community has invested considerable effort into developing tools that facilitate the exploration of a data space. One such successful tool is the *skyline* query. The skyline of a data set is the subset of tuples that are not dominated on all of their attributes by any other tuple. Intuitively, the skyline consists of the tuples that have a uniquely interesting combination of attribute values that no other tuple can match.

Given the practicality of skyline queries, which arises from the elegant and parameterless manner in which they capture a notion of interestingness, previous work has concentrated on the efficient evaluation of skyline queries in both offline and online environments. In an offline environment data resides on disk and skyline queries are answered on demand, while in an online data streaming environment the skyline of the most recent stream data is continuously maintained up to date.

Most of this work reasoned about tuples with *numerical* attributes and totally ordered domains, and relied on the cleanness of the dominance relation between tuples, induced by the linearity of the numerical domains, in order to derive efficient solutions. However, in real applications data can either include or be exclusively comprised of attributes that are *categorical* and *partially ordered* in nature. A hierarchical categorical domain is possibly the most familiar example of a partially ordered domain. Complex relationships and hierarchal structure cannot be captured by a simple mapping of categorical values to numbers and this immensely complicates the skyline computation problem.

Recent work [5] considered the on-demand evaluation of skyline queries on tuples with partially ordered categorical attributes, but in an *offline* environment. As we will subsequently argue and experimentally demonstrate, the proposed techniques are inappropriate for a highly dynamic *data streaming environment* where tuples constantly flow into the system. In this setting, we require an efficient solution to continuously maintain the skyline up to date for the most recent tuples that arrived in the stream. This leaves a significant gap in the array of available skyline techniques. The omission becomes more important when one considers the wide applicability of such an online skyline maintenance solution for partially ordered data.

As an example, consider a service that aggregates and displays news articles as they are published by news sources. News posts streaming into the system are associated with categorical attributes like the name of the news source, the subject of the event and the geographical area associated with the event. An expert has defined a partial order over the categorical domains expressing the service's preferences and defining its unique style. The system will then select for display, or more extensive filtering by an expert, the most "interesting" news that comprise the skyline of the most recent articles. For example, a skyline article is of interest because it was published in a high quality news source and is related to a popular geographical area, even though the subject itself might not be that preferable.

In this paper, we identify and study the problem of maintain-

SIGMOD'08, June 9–12, 2008, Vancouver, BC, Canada.

ing the skyline of streaming data with partially ordered, categorical attributes and realize two novel techniques that constitute the building blocks of STARS (Streaming Arrangement Skyline), the proposed efficient solution to the problem.

In particular, we assume a sliding window model of stream computation and introduce a lightweight, grid-based data structure for indexing the tuples in the system buffer. Our initial, basic indexing solution is progressively refined: we first identify and utilize a property that is unique to the grid-oriented structure of the index and subsequently develop techniques that offer flexibility in controlling the granularity of the grid. The resulting indexing structure can gracefully adapt to tuples with many attributes and partially ordered domains of any size and structure in an *optimal* manner.

We subsequently study the dominance relation between two tuples with partially ordered attributes in the *dual space*: tuples are mapped to lines and the interaction of their corresponding lines is used in order to infer their dominance relation. This mapping allows us to utilize powerful tools from computational geometry, known as *geometric arrangements*, in order to organize and query the skyline efficiently. As we discuss, the arrangement-based organization of the skyline allows us to answer dominance queries by considering only a small fraction of the skyline tuples.

The rest of the paper is organized as follows. In Section 2 we survey related work. In Section 3 we introduce the basic definitions and notations that we will use throughout this paper and offer a high level discussion of the skyline maintenance task. The core techniques comprising our solution are presented in Section 4. Section 5 presents our experimental evaluation of the proposed solution, while Section 6 concludes the paper.

2. RELATED WORK

Since the inception of the skyline operator [4], researchers have considered the efficient evaluation of skyline queries over numerical data in the absence [8, 12] or presence [24, 16, 23, 18] of supporting indexing structures. More recent work has concentrated on variations of the original query [7, 6, 20]. In this section we briefly review the relevant literature. A more comprehensive review of the area is available elsewhere [18].

The Block-nested-loops algorithm introduced in [4] reads data sequentially from disk, while maintaining in memory the temporary skyline of the tuples that have been read so far. At the end of the computation, when all data have been read, the temporary skyline is the true skyline of the data set. [8] observed that if the data are first sorted in an order that is compatible with the dominance relation, then the maintenance of the in-memory temporary skyline becomes simpler and more efficient. The LESS algorithm of [12] capitalizes on this idea and integrates skyline computation with the external sorting procedure.

Skyline computation can greatly benefit by the presence of a supporting indexing structure. The BBS algorithm of [23] utilizes an R-tree to progressively construct the skyline. The R-tree is used to prioritize access over nodes that are likely to contain skyline tuples and to prune entire nodes whose tuples are definitely dominated by the partially constructed skyline. The BBS algorithm was found to be superior than the index-based solutions presented in [24, 16]. Recent work [18] introduced a B-tree variant that stores and clusters the tuples based on their Z-order, which is compatible with the dominance relation.

The size of the skyline can increase dramatically with data dimensionality. This is incompatible with the skyline's role as an exploratory tool. In order to compensate, [6] introduced the notion of k-dominance: a tuple is declared dominant as long as it is better than another tuple on a subset of its attributes. [20] adopted a different approach to the same problem and introduced algorithms for computing the k most "representative" skyline tuples.

With respect to skyline maintenance, [26, 18] discuss the problem for non-streaming, numerical data. However, most relevant to our work are existing solutions for performing skyline maintenance for a *stream* of *numerical* data [25, 19, 21]. These techniques make no provision for *partially ordered categorical data*. The work of [5] is designed to handle such data, albeit in an *offline* environment. Because of the limitations of the techniques presented in [5] when applied to streaming data, we initiate a study of the problem of maintaining the skyline of *streaming data with partially ordered attributes* and design a novel solution for performing this task.

3. BACKGROUND

3.1 Definitions

An unbounded stream of tuples arrives at the system at high rates. We maintain a limited-capacity, sliding-window buffer $\mathcal{B}$ in memory that only stores the n most recent tuples from the stream. When a new tuple arrives from the stream, the oldest tuple in the buffer is removed in order to free up space for the incoming tuple.

At any time instance, the contents of the buffer constitute a data set denoted by D. The data set is comprised of n tuples $t_1, \ldots, t_n$ with d *categorical* attributes $X_1, \ldots, X_d$. The domain Dom_i of each of the attributes is *partially ordered* and constitutes a *partially ordered set*, also referred to as a *poset*. Each domain Dom_i is associated with a binary relation $\preceq_i$. Let a, b, c be three elements of Dom_i. The partial order relation $\preceq_i$ is *transitive* ($a \preceq_i b$ and $b \preceq_i c$ implies $a \preceq_i c$) , *reflexive* ($a \preceq_i a$ holds) and *antisymmetric* (if $a \preceq_i b$ and $b \preceq_i a$, then $a = b$). We further denote with $\prec_i$ the strict ordering relation, i.e., $a \prec_i b$ implies that $a \preceq_i b$ and $a \neq b$. We will also refer to the relation $b \prec_i a$ as *a dominates b*. Additionally, we say that a and b are *comparable* if either $a \prec_i b$ or $b \prec_i a$ and *incomparable* otherwise. Lastly, we will also denote the relation $a \preceq_i b$ as $b \succeq_i a$.

Posets are commonly represented as directed acyclic graphs. Each domain value is mapped to a vertex and a directed edge is introduced for each pair of comparable values whose relation *cannot* be inferred by using the transitive property of the partial order relation. The following example clarifies the aforementioned definitions.

EXAMPLE 1. *Consider the poset of Figure 1. Values a and b are comparable and a dominates b. On the contrary, values b and c are incomparable. Furthermore, due to transitivity in the ordering relation, a dominates d, but an edge on the graph between a and d would be redundant, since their relation can be easily inferred. One can notice that given a node, every other node in the poset that is* reachable *by it, is also dominated by it.*

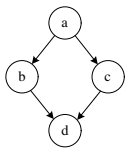


Figure 1: A simple poset.

The definitions can be easily extended to tuples comprised of d categorical attributes. We will say that tuple t_1 *dominates* tuple t_2 and write $t_2 \prec t_1$ or $t_1 \succ t_2$, if for every attribute X_i, $t_2.X_i \preceq_i t_1.X_i$ and there is at least one attribute X_j such that $t_2.X_j \prec t_1.X_j$. When two tuples t_1 and t_2 do not dominate one

another, we will say that they are *tied* and write $t_1 \sim t_2$. Then, the *skyline* of data set D is the subset of all tuples that are not dominated by any other tuple.

Notice that the dominance relation for tuples is transitive and not symmetric. This implies that given a data set D, its skyline S and a tuple $t \notin D$, in order to find out if t belongs to the skyline of $D \cup \{t\}$ we only need to check if t is dominated by tuples in S. If it is dominated by a tuple in S, then of course it cannot be part of the skyline. However, if it is not dominated by any tuple in S, then it is not dominated by any tuple in D and should therefore become part of the skyline of $D \cup \{t\}$

EXAMPLE 2. *Consider three tuples t_1, t_2 and t_3 with two categorical attributes. The domain of both attributes is the poset of Figure 1. Let us assume that $t_1 = (a, b)$, $t_2 = (b, d)$ and $t_3 = (b, c)$. Then, both t_1 and t_3 dominate t_2, but t_1 and t_2 are tied (values b and c are incomparable). Therefore, the skyline of this small data set consists of tuples t_1 and t_3.*

3.2 Skyline maintenance

At the core of existing solutions that maintain the skyline of streaming tuples with *numerical* attributes [19, 21, 25] is a simple framework that can also be utilized for streaming tuples with *categorical* attributes. As a matter of fact, the framework is independent of the definition of tuple dominance. The definition only becomes relevant in the realization of the abstract framework that we describe. At any given time, the buffer $\mathcal{B}$ contains the n most recent tuples from the stream. Let S be the skyline of the tuples in the buffer. A buffer update, i.e., the insertion of a new tuple in the buffer and the expiration of the oldest one, can affect the skyline in a limited number of ways.

In particular, the incoming tuple can either be dominated by at least one tuple in the skyline and therefore fails to affect the skyline, or is not dominated by *any tuple* in the skyline and should therefore become part of the skyline itself (as we argued in Section 3.1). In that case, the incoming tuple might also dominate tuples currently in the skyline which must of course be removed. Respectively, if the outgoing tuple does not belong in the skyline, then its expiration has not effect. However, if the tuple is part of the skyline, then all tuples in the buffer dominated *exclusively* by the outgoing tuple (i.e., dominated by the outgoing tuple, but no other tuple in the skyline) must be inserted in the skyline.

Therefore, a skyline maintenance solution must efficiently support two operations: (i) checking whether a tuple is dominated by the current skyline and (ii) retrieving the tuples in the buffer that are dominated by the outgoing skyline tuple, since only these tuples are candidates for entering the skyline. The ability of any technique to perform this second task efficiently can be augmented by utilizing the following observation.

LEMMA 1. *Let $t_1, t_2 \in \mathcal{B}$ be two tuples so that $t_1 \prec t_2$. Then, if t_2 arrived after t_1 in the stream, t_1 will never be in the skyline of $\mathcal{B}$.*

PROOF. Since t_1 arrived before t_2, it will also leave the buffer before t_2. Therefore, while t_1 is in the buffer, there will be at least one tuple, namely t_2, that dominates it and consequently cannot become part of the skyline. □

The lemma implies that a significant number of tuples in the buffer is irrelevant for the skyline maintenance task, since they can never become part of the skyline. We will refer to the relevant part of the buffer as the *skybuffer*. Thus, when we need to mend the skyline after the expiration of a skyline tuple, we only need to consider tuples in the skybuffer instead of the entire buffer. The incremental maintenance of the skybuffer is simple. When a new tuple arrives from the stream, it is inserted in the skybuffer, while all the skybuffer tuples dominated by it are removed. When a tuple expires, it is simply removed from the skybuffer.

Algorithm 1 summarizes the high level strategy that is employed in order keep the skyline of the buffer up to date. Notice that the skyline S is a subset of the skybuffer SB. After a buffer update, any changes to the composition of the skyline can be optionally reported.

Algorithm 1 Skyline maintenance framework

Input: skybuffer SB, skyline $S \subseteq SB$, incoming tuple in, outgoing tuple out

if in not dominated by S **then**
  Insert in in S and remove any dominated tuples from S;
end if
Insert in in SB and remove any dominated tuples from SB;
if out is in S **then**
  Find tuples in SB dominated by out and use them to mend S;
end if
Remove out from SB;

4. EFFICIENT SKYLINE MAINTENANCE FOR CATEGORICAL TUPLES

In this section we present two novel techniques for realizing the building blocks of the skyline maintenance framework: indexing the skybuffer so that we can efficiently identify the skybuffer tuples dominated by a query tuple and organizing the skyline in order to be able to rapidly answer whether a query tuple is dominated by the skyline.

We initiate the presentation of the proposed techniques by briefly reviewing the notion of the topological sort of a single poset and discussing the extension of this idea to tuples comprised of multiple partially ordered attributes.

Based on these results, we introduce our baseline grid-based solution for indexing the skybuffer and discuss the advantages of this approach over potential alternatives. This basic solution is progressively refined. The first enhancement exploits the unique structure of the grid index to optimize query evaluation by *focusing* on specific, relevant cells instead of an entire region of cells, many of which can be irrelevant. The index is further refined by developing a domain partitioning technique that offers absolute control over the granularity of the grid. Part of the technique is an algorithm for constructing an *optimal* poset partition that minimizes the expected query evaluation cost.

We then focus our attention on designing an efficient skyline organization. This is accomplished by identifying the connection between the dominance relation of two tuples and the interaction of their *dual space* representation as *lines*. The mapping of tuples to lines allows us to utilize powerful tools from computational geometry, known as *geometric arrangements*, in order to organize and query the skyline efficiently. As we discuss, the arrangement-based organization allows us to answer dominance queries by considering only a small fraction of the skyline tuples.

4.1 Topological sorting

A *topological sort* [2] is a numbering of the vertices of a DAG such that every edge from a vertex numbered i to a vertex numbered j satisfies $i < j$. Figure 2 presents a poset and two possible topological sorts. A poset can have a large number of valid topological

sorts, although for our purposes any one of them will be equally appropriate. Finding a topological sort for a poset is a linear cost operation [2].

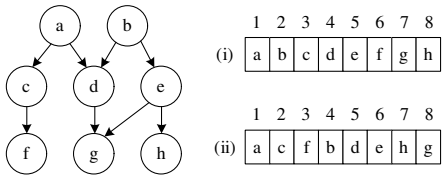


Figure 2: Topological sorting.

Informally, a topological sort of a poset is a linear ordering of its values which is compatible with its partial-ordering relation. In other words, if for two values a, b we have $a \succ b$, then a will appear before b in the ordering. Therefore, the guarantee we obtain is the following: for a given value x, all values dominated by x will appear after it in the linear order, while x can never dominate any value that precedes it in the ordering. Conversely, no value appearing after x can ever dominate it, while all values that dominate x appear before it. This intuition is captured by the following lemma.

DEFINITION 1. *Let v be a value of a partially-ordered domain. We denote by $r(v)$ the integer corresponding to v's position in a certain topological sort of the domain.*

LEMMA 2. *Let $v_1, \ldots, v_m$ be the m values of a partially-ordered domain Dom. Then $v_i \succ v_j$ only if $r(v_i) < r(v_j)$.*

Consider for example value d and the topological sort (ii) in Figure 2. d does not dominate any of the values that appear before it and all values that dominate it appear before it (a, b in that case). Furthermore, all values dominated by d (only g in this case) appear after it in the order.

These observations and reasoning concerning a single domain, can be extended to multiple attribute domains. Consider a set of tuples with d partially-ordered categorical attributes. For a tuple t_1 to dominate another tuple t_2, it needs to dominate t_2 in every attribute[1]. In order for this to be possible, all d attributes of t_1 must be located before t_2's attributes in the corresponding topological sorts. As before, this is not an *if and only if* relation. The guarantee we have is that for t_1 to dominate t_2, its attributes must be located before t_2's attributes in the corresponding linear orders. Formally:

LEMMA 3. *Let t_1, t_2 be two tuples with d partially-ordered categorical attributes $X_1, \ldots, X_d$. Then $t_1 \succ t_2$ only if $r(t_1.X_i) < r(t_2.X_i), 1 \leq i \leq d$.*

This relation has another very useful implication: given two tuples t_1 and t_2, if there is a disagreement in the ordering of the values for two of their attributes, then one cannot dominate the other and are therefore tied.

LEMMA 4. *Let t_1, t_2 be two tuples with d partially-ordered categorical attributes $X_1, \ldots, X_d$. If $\exists i, j$ such that $r(t_1.X_i) < r(t_2.X_i)$ and $r(t_1.X_j) > r(t_2.X_j)$, then $t_1 \sim t_2$.*

4.2 Organizing the skybuffer tuples in a grid

One of our goals in developing an efficient skyline maintenance solution is indexing the skybuffer so that we can efficiently insert and delete tuples, as well as identify the tuples dominated by a query tuple. The linearization of poset values that we introduced will allow us to do so.

A tuple $t(X_1, \ldots, X_d)$ in the skybuffer can be mapped to a point $(r(t.X_1), \ldots, r(t.X_d))$. Then, in order to identify the skybuffer tuples that are dominated by a query tuple $q(X_1, \ldots, X_d)$, we only need to consider tuples/points t with $q(X_1) \leq t(X_1), \ldots, q(X_d) \leq t(X_d)$. All these tuples will be checked for dominance against q in order to identify the ones dominated. Notice however that this is precisely a rectangular range query that can be efficiently supported by a variety of spatial data structures, like a grid or an R-tree.

We chose to build our skybuffer indexing solution around a simple grid. The reason for doing so is twofold. Firstly, previous research argues [9, 17, 22] that in a streaming environment, the potential query performance gains by using a more sophisticated data structure are offset by the heavy maintenance costs induced by the data volatility, which is inherent in a streaming application. Secondly, the unique structure of the grid interacts favorably with the properties of our problem. This interaction will allow us to subsequently introduce optimizations and refinements whose applicability is exclusive to our grid-based index.

Let us discuss a concrete example that will help clarify how a grid can be used to index the skybuffer and identify the skybuffer tuples dominated by a query tuple.

EXAMPLE 3. *Consider a set of tuples with two categorical attributes, the domain of both attributes being the poset of Figure 2. Suppose that the poset values have been mapped to integers according to topological sort (i) of Figure 2. Then, we create the grid by using this topological sort as the grid scales and place the skybuffer tuples in its cells. Figure 3 illustrates. Each grid cell corresponds to a unique combination of attribute values and only contains tuples with these exact values. As an example, the cell corresponding to tuples with attribute values (d, e) has been marked with an "$\times$" in the figure.*

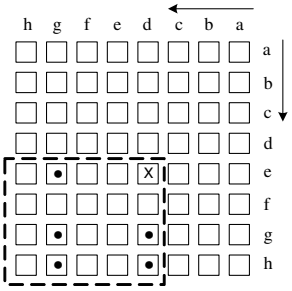


Figure 3: Organizing the skybuffer as a grid.

In our 2-dimensional example, all tuples dominated by a certain tuple t must have values in *both* attributes that appear after the corresponding values of t in the linear order. However, tuples with this property are placed in cells lying in a single rectangular area of the grid. In our running example of Figure 3, all tuples that are dominated by tuple (d, e) are located in the rectangular area whose upper-right corner is cell (d, e). Notice that this does not imply that *all* tuples in that area are dominated. It merely means that the tuples that are dominated by (d, e) must lie in that area. As a matter of fact, only tuples located in the cells marked with $\bullet$ in Figure 3 are dominated.

Summarizing our progress so far, we demonstrated how a grid can be used to index the skybuffer tuples. The scales of the grid

[1]To simplify the discussion, we ignore the case of equal attribute values.

for each dimension is the linearization of the corresponding poset. Then, in order to identify the skybuffer tuples that are dominated by a query tuple, we need to issue a rectangular range query, which the grid can efficiently support, and only consider the tuples lying in the query area. Insertions and deletions can also be carried out extremely efficiently.

4.3 Improving the skybuffer organization

4.3.1 Visiting only relevant cells

An additional advantage of the grid-based index is that instead of issuing a rectangular range query in order to visit the cells that potentially contain dominated tuples, we can directly identify and process precisely the cells that contain dominated tuples. We will refer to this unique capability of the grid-based index as the ability to perform *focused search*.

Let us revisit the example of Figure 3. We claim that we can directly visit the cells marked with • instead of every cell in the rectangular region. The query tuple is (d, e). The domain values dominated by d (and including d) are $\{d, g\}$. Respectively, the domain values dominated by e are $\{e, g, h\}$. Then, due to the definition of dominance (Section 3.1), only tuples with values in $\{d, g\} \times \{e, g, h\}$ are dominated by (d, e). This is captured by the following lemma.

LEMMA 5. (Focused Search) *Let t be a tuple with d partially-ordered categorical attributes $X_1 \in Dom_i, \ldots, X_d \in Dom_d$. Let $dom(t.X_i)$ be the values in Dom_i such that $t.X_i \succeq v$, $v \in Dom_i$. Then, t dominates a tuple s if and only if $s \in dom(t.X_1) \times \cdots \times dom(t.X_d)$.*

The lemma allows us to identify and focus on the cells in the grid containing tuples dominated by the query tuple, which is clearly much more efficient than issuing a rectangular range query and considering tuples that are irrelevant to the query.

4.3.2 Controlling the grid granularity

A problem with the grid-based index - as presented so far - is the lack of control over the granularity of the grid. The scales of the grid for each dimension are directly derived from a topological sort of the corresponding domain and introduce as many buckets per scale as values in the domain. While this fine granularity might be acceptable for tuples with few attributes and domains with a handful of values, it is obvious that the solution does not scale. The number of cells in the grid for tuples with d attributes is $|Dom_1| \times \cdots \times |Dom_d|$. As an example, if the tuples have 4 attributes and each domain size is about 500, then the grid would be comprised of 62.5 billion cells, which is clearly infeasible.

In general, the grid granularity has significant impact on the performance of any grid-based indexing solution. Since pruning at query time occurs at the cell level, coarser granularity and therefore bigger cells result in less effective pruning. On the other hand, setting a finer granularity produces a greater number of smaller cells. Besides increased memory requirements, such an arrangement results in increased query time, since accessing a cell is associated with an overhead. There is always a well-performing granularity range where these competing trends do not dominate query time. This range is certainly application specific, but can also be data dependent, so it is paramount to provide flexibility in controlling grid granularity.

We begin the introduction of our granularity control mechanism by conducting a few simple observations. First, let us formally define the *depth* of a domain value.

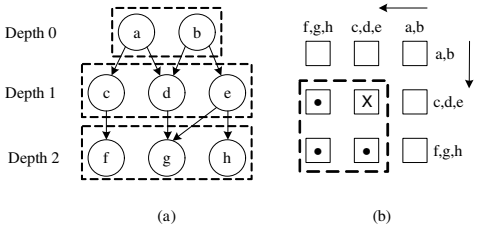


Figure 4: Depth-based value grouping.

DEFINITION 2. *Consider a DAG and its vertices. A vertex is a source if it has no incoming edges. Then, the depth of a vertex in a DAG is the length of the* longest *path from a source to the vertex.*

Figure 4(a) depicts an example poset and the associated depth of its domain values. In the figure we can observe that a value only dominates other values that are located "deeper" in the poset. This implies that sorting the vertices in increasing depth values (and breaking ties arbitrarily) produces a perfectly valid topological sort. Using this insight, we can restate Lemma 2 as follows.

LEMMA 6. *Let $v_1, \ldots, v_m$ be the m values of a partially-ordered domain Dom. Then $v_i \succ v_j$ only if $depth(v_i) < depth(v_j)$.*

The lemma guarantees that given a poset value, all values that it dominates have to be located deeper in the poset, but this doesn't imply that *all* values located deeper are dominated. Lemmata 3 and 4 can also be restated in terms of depth, but we do not do so in the interest of space. The implication of these results is that we can create and use the grid using scales at the granularity of a depth level. Let us discuss a concrete example.

EXAMPLE 4. *Consider a data set consisting of tuples with 2 partially-ordered categorical attributes, the domain of both of them being the poset of Figure 4(a). Then, we can group domain values that lie on the same depth and create the grid of Figure 4(b). In Section 4.2, every cell would contain tuples with the same attribute values. Now, tuples with corresponding attributes that lie at the same depth level are placed in the same cell. For example, tuple (d, e) lies in the cell marked with "×" in Figure 4(b), along with other tuples with values in $\{c, d, e\} \times \{c, d, e\}$.*

This reduction of the grid granularity does not come without a price. When querying the grid to retrieve the tuples dominated by the query tuple, we can no longer perform focused search and only access cells that exclusively contain tuples dominated by the query tuple. Instead, we must access all cells containing tuples with attribute values located deeper in their corresponding posets. This is equivalent to a full rectangular range query (Figure 4(b)). Furthermore, the cells can contain tuples both dominated and not dominated by the query tuple and therefore a dominance check against all the tuples in the cells is required to identify the dominated tuples.

4.3.3 Poset partitioning

The proposed depth-based grouping of poset values improves the initial solution, but does not allow to explicitly set the desired grid granularity and is tied to the structure of the posets - a constraint that can introduce problems. As an example, consider a poset with only two depth levels, but many values spread evenly between the levels. The solution of Section 4.3.2 would produce oversized grid cells and therefore reduced pruning efficiency at query time.

In Section 4.3.2, we grouped all the values of a poset having the same depth. We can further refine this partition of values into groups by creating more than one group per depth level. Let us see the potential advantages of such an approach with the following example.

EXAMPLE 5. *Suppose that our data set contains tuples with two partially-ordered categorical attributes, the domain of both being the poset of Figure 5(a). The values of the domain have been grouped as illustrated in the figure. If we order the groups in ascending order of their depth value, breaking ties arbitrarily, we again come up with a valid topological sort for the poset. We can therefore use the same rationale as before to create and query the grid.*

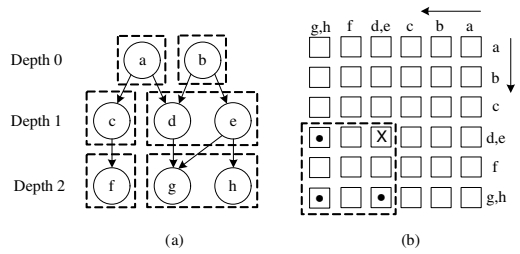


Figure 5: Refined depth-based value grouping.

Figure 5(b) presents the resulting grid by using as scales the grouping of Figure 5(a). As an example, tuple (d, e) lies in the cell marked with "×". As before, all cells that can potentially contain tuples that are dominated by (d, e) are located in the rectangular area highlighted in Figure 5(b).

However, unlike the example we studied in Section 4.3.2, not all cells in the rectangular area contain tuples that can be dominated by the query tuple. The cells that contain candidates are marked with •. This allows us to use focused search in order to directly access relevant cells instead of issuing a range query. Note that cells will contain tuples that are both dominated and not dominated by the query tuple, but we can completely ignore cells that exclusively contain not dominated tuples.

Formally, let Dom be a partially ordered domain with values $v_1, \ldots, v_m$ that has been partitioned into k groups, $g_1, \ldots, g_k$. We say that group g_i dominates another group g_j if there exists one value in $v_i \in g_i$ and another value $v_j \in g_j$ such that v_i dominates v_j. Let $dom(g)$ be the union of the groups dominated by g, including g itself. Furthermore, let $g(v), v \in Dom$ be the group that value v belongs to. Then, in order to locate the tuples dominated by a query tuple $q(X_1, \ldots, X_d)$ we only need to visit the grid cells that contain groups $dom(g(q.X_1)) \times \cdots \times dom(g(q.X_d))$. The intuition is that we only need to check tuples in the groups where the values dominated by v_i lie. To ease notation we will denote the values in $dom(g(v_i))$ as $dom_g(v_i)$ and say that the values are *group-dominated* by v_i.

A natural question that arises is that given a budget of B buckets for a poset, how should the poset be partitioned into groups. Not all groupings will offer equally good pruning opportunities that the focused search procedure can exploit. In the worst case, we can come up with a scenario where even though we have used more buckets than depth levels, we still end up visiting as many cells as we would visit by issuing a range query on the grid. This case would be equivalent in cost to the simpler depth-based grouping.

Let us concentrate on a single attribute with values $v_1, \ldots, v_m$ and a grouping $g_1, \ldots, g_k$. Had we created the grid at the finest granularity possible, then, given a query tuple with value v_i for the attribute, we would only examine (check for dominance) tuples with values in $dom(v_i)$. However, because of the grouping of domain values we also need to examine all tuples with values that are group-dominated by v_i, i.e., tuples with values in $dom_g(v_i)$.

Let us assume that the domain values are uniformly distributed. Then, the number of tuples that must be checked for dominance is $\frac{n}{m}|dom_g(v_i)|$, i.e., $\frac{n}{m}$ tuples, where n is total number of tuples in the skybuffer, for every value in $dom_g(v_i)$. Furthermore, the probability that the query tuple has attribute value v_i is $\frac{1}{m}$. Therefore, the expected number of tuples that must be checked for dominance in every query is $E = \sum_i^m \frac{1}{m}\frac{n}{m}|dom_g(v_i)|$. Since n and m are constants, in order to minimize E we need to come up with a grouping that minimizes $\sum_i |dom_g(v_i)|$, which is the sum of the number of group-dominated values, from every value in the domain. Unfortunately, we can demonstrate that even for simple instances, the problem is NP-complete.

THEOREM 1. *Given a partially ordered domain Dom with values $v_1, \ldots, v_m$, the problem of identifying a partition of the domain into k groups $g_1, \ldots, g_k$, so that $\sum_i^m |dom_g(v_i)|$ is minimized, is NP-complete.*

PROOF. A simple instance, involving a domain with only two levels, can be reduced from the "maximum k-set packing" problem, which is NP-Complete [3]. □

To compensate, we developed a partitioning heuristic that we found to work extremely well in practice, as we experimentally demonstrate in Section 5. The heuristic has two main steps: it first allocates a fraction of the buckets available to each level and then performs a greedy, bottom-up partition of the poset, i.e., it starts from the deepest level and partitions it, then partitions the level above and so on, until the uppermost level is partitioned. In order to partition a level, the heuristic leverages local information coming from the already partitioned level below and the level above.

The bucket allocation strategy that we selected is the following: each level is initially assigned one bucket and the remaining buckets are distributed among the depth levels proportionally to the number of nodes that they contain. The outline of this partition framework is illustrated in Algorithm 2.

Algorithm 2 Partitioning heuristic framework

Input: Poset with values $v_1, \ldots, v_m$, number of buckets B
Output: Disjoint value groups $G_1, \ldots, G_B$
Variables: Maximum depth level of the poset h, set of groups at depth i, $\{G\}_i$

Group together values according to their depth into groups $D_1, \ldots, D_h$;

Allocate to level i, $b_i = 1 + \lceil \frac{|D_i|}{m}(B - h) \rceil$ buckets;

for $i = h$ down to 1 **do**
  Utilize information from domain values in D_{i-1} (level above) and groups in $\{G\}_{i+1}$ (level below);

  Partition values in D_i into b_i groups $G_{i,1}, \ldots, G_{i,b_i}$ and place them in $\{G\}_i$;
end for

Let us now concentrate on how a depth level is partitioned given a budget of k buckets for that level. The partitioning is based on local information, coming from the level above and the level below that has already been partitioned. For each pair of nodes we define a notion of benefit that we can expect by placing these nodes into different groups.

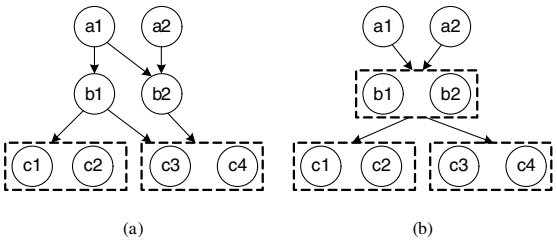


Figure 6: Poset partitioning heuristic.

EXAMPLE 6. *Consider the example of Figure 6. Suppose that we want to measure the benefit of separating nodes $b1$ and $b2$ (Figure 6(a)). Remember that we need to minimize the number of poset values that each value group-dominates. Then, if we place them in the same group (Figure 6(b)):*

- *Node $b1$ will not be affected, since it already group-dominates nodes $c1$ to $c4$ in the lower level. Therefore the benefit by separating $b1$ and $b2$ is 0.*
- *Node $b2$ however, will now group-dominate nodes $c1$ and $c2$. This would increase the cost function by at least 2. Therefore, the benefit for separating $b1$ and $b2$ is +2.*
- *Node $a1$ already group dominates nodes $b1$ and $b2$. Therefore no benefit is derived be separating them.*
- *If $b1$ and $b2$ are grouped together, node $a2$ will group-dominate node $b1$, besides node $b2$. Therefore, by separating $b1$ and $b2$ we derive a benefit of +1 for $a2$.*
- *The total benefit for separating $b1$ and $b2$ is now 0+2+0+1=3.*

Therefore, for each pair of nodes in a level, we can connect them with an undirected edge weighted by the benefit we can derive by separating the nodes. Then, a good partitioning for this level can be done by considering the *maximum k-cut* of the nodes. The maximum k-cut will partition the nodes of the level into k groups so that the total weight of edges spanning two groups is maximized. Effectively, this procedure maximizes the sum of the pairwise benefits for this level.

Generating a maximum k-cut of a graph with n nodes is an NP-complete problem. [11] introduced a semi-definite programming (SDP) relaxation algorithm that provides a $1 - 1/k + 2\ln k/k^2$ approximation to the optimal solution. However, a much simpler greedy heuristic can identify a $1 - 1/k$ approximation to the optimal solution, a guarantee that is only marginally worse than the one offered by the SDP relaxation. The greedy algorithm considers the nodes in arbitrary order and places them in one of the k groups $g_1, \ldots, g_k$. A node v is placed in the group whose nodes v_g minimize the sum $\sum_{v_g \in g} w(v, v_g)$, where $w(v, v_g)$ is the weight (benefit) of the edge between v and v_g, thus maximizing the weight of the "cut" edges . The procedure for partitioning the poset values of a depth level is illustrated in Algorithm 3.

THEOREM 2. *The weight of the k-cut produced by Algorithm 3 is greater than $(1 - 1/k)OPT$.*

PROOF. The weight C of the k-cut is the sum of the edges that span different groups. Let $G(V, E)$ be the graph and $v_1, \ldots, v_j, \ldots, v_l$ the order in which its nodes are processed. We define a disjoint partition of edges E into groups $E_j = \{v_i v_j | i < j\}$. Then, the placement of node v_j in a group contributes by $C_j \geq (1 - 1/k)w(E_j)$ to the weight of the cut, where $w(E_j)$ is the sum of edge weights in E_j. This is a direct consequence of our placement strategy. Then $C = \sum_{j=1}^{l} C_j \geq (1 - 1/k) \sum_{j=1}^{l} w(E_j) = (1 - 1/k)w(E)$, where $w(E)$ is the sum of all edge weights in the graph. However, $w(E) \geq OPT$, therefore, $C \geq (1 - 1/k)OPT$. □

Algorithm 3 Level partitioning algorithm

Input: Poset values $v_1, \ldots, v_l$, number of buckets k
Output: Disjoint value groups $g_1, \ldots, g_k$

For every pair of nodes v_i, v_j, calculate benefit $w(v_i, v_j)$;

Initialize groups to be empty;

for $i = 1$ up to l **do**
  Place v_i in group g_j such that $\sum_{v_j \in g_j} w(v_i, v_j)$ is minimum;
end for

Our partitioning heuristic assumes that attribute values are uniformly distributed. Nevertheless, potential knowledge about the value distribution could be incorporated in the partitioning process by modifying in an appropriate manner the benefit values between the nodes.

Lastly, we have assumed that the number of buckets that are allocated for partitioning a poset is greater than the poset's maximum depth, so that at least one bucket is available per depth level. In the rare case that fewer buckets than levels are available, we can merge into a single group *consecutive* depth levels. A natural strategy that can utilize this observation would be to merge consecutive levels into groups, so that every group contains approximately the same number of poset values. This is identical to the process of producing an equi-depth histogram [15].

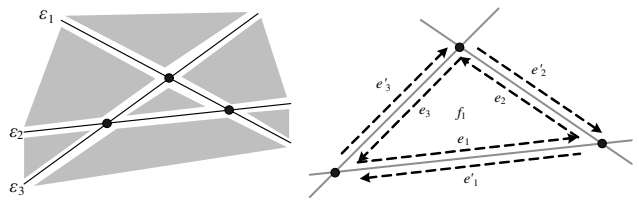
4.4 Arrangement representation of the skyline

The second building block of the maintenance framework is an efficient skyline organization. The employed indexing structure must be able to determine whether a query tuple is dominated by the skyline, checking as few skyline tuples as possible. This operation is essential for good performance as it is conducted for each incoming stream tuple, as well as during the skyline mending procedure that occurs after a skyline tuple expiration. To achieve this goal, we study the dominance relation in the *dual space* and design an efficient solution that utilizes *geometric arrangements* [1, 13].

4.4.1 Arrangements of lines

The *arrangement* $\mathcal{A}(L)$ of a finite collection of lines L, is the partition of the plane induced by the lines in L [1, 13]. The lines decompose the 2-dimensional plane into 0-dimensional *vertices* (intersections of lines), 1-dimensional *edges* (line segments between vertices) and 2-dimensional *faces* (the convex tiles of the plane bounded by the intersecting lines). Figure 7(a) presents an arrangement of three lines, with faces highlighted in grey. Arrangements are well studied structures and there exists a wealth of combinatorial results that reason about their complexity, as well as a fair number of main memory data structures for storing and operating on them.

An arrangement of s lines is composed of $O(s^2)$ vertices, $O(s^2)$ edges and $O(s^2)$ faces. An interesting substructure of an arrangement is the *zone* of a line not present in the arrangement. The zone is comprised of the faces stabbed by the line, and the celebrated Zone Theorem states that these faces are in turn comprised of only $O(s)$ edges [1, 13]. It is partly because of the favorable combinatorial bound promised by the Zone Theorem, that arrangements are



(a) Arrangement example (b) The DCEL data structure

Figure 7: Arrangements of lines.

of great practical, besides theoretical interest: insertions and deletions of lines, and many interesting queries on the arrangement can be performed in linear $O(s)$ time.

Although a variety of data structures is available for representing an arrangement, the one most suitable for our purposes is the *doubly-connected-edge-list* (DCEL) data structure [10] (Figure 7(b)). At its core, the arrangement is a planar graph consisting of vertices and undirected edges connecting these vertices. The DCEL uses a pair of twin directed half-edges, moving in opposite directions, to represent each edge connecting two vertices. Furthermore, the DCEL maintains additional incidence and connectivity information in order to offer great flexibility in traversing the arrangement. For example, all the half-edges comprising the internal boundary of a face are linked in a circular list (edges e_1, e_2 and e_3 in Figure 7(b)) and each half-edge maintains a link to its twin (edges e_i and e'_i in Figure 7(b)). This information suffices to enable the traversal of the zone of an external line and consequently the identification of all lines in the arrangement intersected by the external line.

To summarize, the DCEL can store the arrangement of s lines using $O(s^2)$ space and can perform insertions, deletions in $O(s)$ time. Furthermore, it can identify all lines intersected by a query line in $O(s)$ time. Although the combinatorial bounds associated with the arrangements might not seem attractive, arrangements have been successfully utilized in a demanding streaming environment in order to perform operations similar to the ones required in the current problem [9]. Given that in our scenario s will be equal to the size of the skyline, which should be normally small, we can expect the arrangement based skyline organization to perform well, in addition to the advantages that we subsequently present.

4.4.2 Dominance checking in the dual space

In the context of computational geometry and related disciplines, the *dual* space is a symmetrical version of the original (or *primal*) problem space, where each point in the primal is mapped to a line in the dual and vice versa. Primal/dual transformations are used widely as they offer fresh insight into the problem and point to solutions that are not easy to conceive in the primal space.

We have already discussed in Sections 4.1 and 4.2 how tuples with d partially-ordered categorical attributes can be mapped to d-dimensional points by utilizing a topological sort of the attributes and representing each attribute value with its position in the corresponding linear order. Lemmata 3 and 4 utilized this representation to reason about the possible dominance relation between two tuples, given their point representation.

Let us initially concentrate on tuples with two attributes. Remember that we denote with $r(v)$ the position of value v in a topological sort of its corresponding domain. We define the following mapping: each tuple $t(a, b)$ is mapped to a line $y = r(a) \cdot x - r(b)$ in the dual cartesian plane (x, y). Then, we can prove that two tuples are comparable (one dominates the other) if the intersection of their corresponding lines lies in the positive half of the x axis in the dual plane. As with all our results, this is not an iff relation. However, we can be sure that if the intersection point lies on the negative half of the axis, then the tuples are definitely tied.

LEMMA 7. *Consider two tuples t_1, t_2 with two categorical attributes X_1, X_2. Then, t_1, t_2 are comparable only if the intersection point (x_I, y_I) of lines $y = r(t_1.X_1) \cdot x - r(t_1.X_2)$ and $y = r(t_1.X_1) \cdot x - r(t_1.X_2)$ has $x_I > 0$.*

PROOF. Using simple algebra, we can find that the x_I coordinate of the intersection point is $x_I = \dfrac{r(t_1.X_2) - r(t_2.X_2)}{r(t_1.X_1) - r(t_2.X_1)}$. Notice that if $x_I < 0$, then $r(t_1.X_2) < r(t_2.X_2)$ and $r(t_1.X_1) > r(t_2.X_1)$ or $r(t_1.X_2) > r(t_2.X_2)$ and $r(t_1.X_1) < (t_2.X_1)$. In either case, Lemma 4 states that tuple t_1 and t_2 are tied.

If $x_I > 0$, then $r(t_1.X_2) < r(t_2.X_2)$ and $r(t_1.X_1) < r(t_2.X_1)$ or $r(t_1.X_2) > r(t_2.X_2)$ and $r(t_1.X_1) > (t_2.X_1)$. Again, in either case, Lemma 3 states that the tuples can be comparable. □

Lemma 7 points to a technique that allows us to prune a significant fraction of the skyline tuples when checking a query tuple t for dominance: we can map the skyline tuples to lines and store in the arrangement only the part of the lines that lies on the positive half of the x-axis. Then, in order to answer whether t is dominated by the skyline, we map t to a line and query the arrangement to retrieve the lines/tuples intersected by t. Their intersection point is guaranteed to have $x_I > 0$, since only the positive part of the lines is stored in the arrangement. Consequently, tuples not intersected by t in the arrangement, and which are definitely tied with t, are pruned. An additional advantage is that the query on the arrangement returns the intersected tuples progressively, therefore the computation can stop as soon as a tuple that dominated t is found.

EXAMPLE 7. *Consider a skyline consisting of three tuples t_1, t_2, t_3. The tuples are mapped to lines in the dual plane and their positive half is stored in an arrangement (Figure 8). The vertices of the arrangement are emphasized using solid bullets. In order to determine whether a query tuple q is dominated by a skyline tuple, q is also mapped to a line and the arrangement is queried. Using the connectivity information provided by the DCEL structure to traverse the arrangement, we progressively retrieve the tuples intersected by q, i.e., t_2 and t_3 in our example. The intersection of q with t_3 is encountered first, therefore q will initially be checked for dominance against t_3. If q is dominated, the query is answered and the traversal stops of the arrangement stops. Otherwise, the traversal continues and intersected tuple t_2 is recovered and checked for dominance. Notice that q does not intersect t_1 and therefore we do not need to check for dominance against it.*

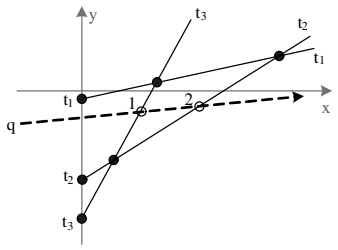


Figure 8: Utilizing an arrangement for pruning.

The technique can be directly applied to tuples with d attributes $X_1, \ldots, X_d$. In that case, we need to arbitrarily select two attributes X_i and X_j and use them consistently to map tuples t to

lines $y = r(t.X_i) \cdot x - r(t.X_j)$ and store them in an arrangement. The query tuple is also mapped to a line using the same two attributes X_i and X_j and the aforementioned transformation. Then, as in the 2-dimensional case, the query tuple will be tied with the skyline tuples that it does not intersect in the arrangement and therefore we only need to progressively check for dominance against intersected tuples. This holds since if there is an ordering mismatch in the corresponding topological sorts for any two of their attributes, then the tuples are tied (Lemma 4). Formally:

LEMMA 8. *Consider two tuples t_1, t_2 with d categorical attributes $X_1 \ldots, X_d$. Let X_i, X_j be any two of the attributes. Then, t_1, t_2 are comparable only if the intersection point (x_I, y_I) of lines $y = r(t_1.X_i) \cdot x - r(t_1.X_j)$ and $y = r(t_1.X_i) \cdot x - r(t_1.X_j)$ has $x_I > 0$.*

For tuples with more than two attributes, we also considered more complex skyline organizations that utilized multiple low dimensional arrangements, but the improved pruning efficiency and query performance failed to compensate for the additional maintenance overhead.

4.5 Numerical attributes

So far in our discussion, we have only considered tuples comprised exclusively of partially-ordered categorical attributes. Nevertheless, the proposed solutions can handle tuples with mixed categorical and numerical attributes without requiring any additional modification. This is possible since partial-ordered domains are a generalization of fully-ordered numerical domains. The DAG representing a numerical domain is simply a linear chain of values and, as it is evident, the proposed techniques can handle posets of any size and shape.

More specifically, for the arrangement organization of the skyline, when a numerical attribute is used in the tuple to line mapping, we can directly use the numerical values instead of their position in the (unique) topological sort and our analysis remains perfectly valid. On the other hand, when constructing and using the grid-based skybuffer index, the scales of the numerical attributes will consist of buckets corresponding to disjoint ranges of numerical values, i.e., as in normal numerical grids. Then, each bucket group-dominates all the other buckets that contain smaller numerical values.

5. EXPERIMENTAL EVALUATION

5.1 Adapting existing work

In lack of techniques dealing directly with the problem of maintaining the skyline of streaming categorical data, we adapted the offline, categorical skyline evaluation technique of [5] for use in a streaming environment.

Previous work on maintaining the skyline of numerical tuples [25] has utilized R-trees to index the skybuffer and the skyline. In the numerical domain, identifying the tuples of a data set that either dominate or are dominated by a query tuple is equivalent to a rectangular range search operation that can be efficiently supported by an R-tree, or any other data structure offering similar query capabilities.

Chan et al. [5] study the problem of evaluating *external* (i.e., disk-based) skyline queries against tuples with categorical attributes. In their solution, every categorical attribute is mapped to two numerical attributes, $Dom_i \mapsto \mathbb{R}^2_+$. Therefore, a tuple with d categorical attributes is mapped to a tuple with $2d$ numerical attributes. In the suggested solution, the tuples of the data set are indexed in the numerical domain.

The mapping that [5] employs has the following property. Consider two categorical values $a, b \in Dom_i$ that are mapped to two 2-dimensional points $p(a)$ and $p(b)$. If $p(a)$ dominates $p(b)$, then conclusively $a \succ b$. However, if $p(a)$ and $p(b)$ are tied, we can make no inference about the relation of a and b: we could have $a \succ b$, $a \prec b$ or $a \sim b$. This is illustrated in Figure 9. Categorical value a is mapped to point $p(a)$ in the numerical domain and partitions the plane into four quadrants. Any value that is mapped to area (I) dominates a, values mapped to area (II) are dominated by a, while we cannot draw any conclusions if the value is mapped to quadrants (III) or (IV).

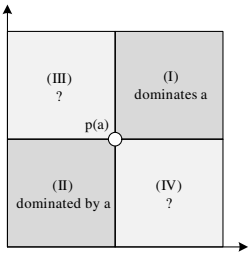


Figure 9: Categorical to numerical domain mapping in [5].

Due to the properties of the mapping, if the skyline itself is organized in the transformed numerical domain, *false positive* tuples will creep into the skyline. In order to compensate, the authors suggest a solution that organizes the skyline in the original categorical domain. The tuples are partitioned into four disjoint groups that have the following property: some tuple groups can never dominate some of the other tuple groups. This reduces the number of tuples that need to be considered when checking if a tuple is dominated by the skyline or not.

The main components of the solution presented in [5] can be adapted to the skyline maintenance framework. The skyline can be organized as suggested by the authors. The skybuffer can be indexed with an R-tree or a grid. When a tuple needs to be inserted in the skybuffer, a range query identifies and removes tuples lying in the high-dimensional equivalent of area (II). However, this implies that tuple in areas (III) and (IV) that are dominated by the inserted tuple will remain in the skybuffer, increasing its size. When a tuple is removed from the skyline, we must use all tuples that lie in areas (II), (III) and (IV) in order to retrieve all the tuples in the skybuffer that are dominated by the expiring tuple. However, in a high dimensional space, these three areas comprise almost the entire search space.

Although the techniques of [5] are efficient for evaluating skyline queries on disk-based categorical data, they do not provide an attractive streaming solution. As we subsequently demonstrate in Section 5.4, the increase in the data dimensionality incurred by the mapping, as well as the complexities of maintaining and querying the skybuffer, result in reduced performance.

5.2 Data generation

We performed our experimental evaluation utilizing both real and synthetic data. Our real data set will be described later in the section. For synthetic data, we generated partially-ordered domains with different structures and shape to cover a wide range of possible settings. Figure 10 illustrates two classes we used. The poset of Figure 10(a) has a "tree" structure. Every depth level has twice the number of values from the level above. On the other hand, the poset of Figure 10(b) has a "wall" structure and all depth levels have the same number of values. A poset is defined by its structure, i.e., tree

or wall, and a triplet (m, h, c), where m is the number of domain values, h the height (number of depth levels) of the poset and c is the internal connectivity of the poset. This parameter lies in interval $(0, 1]$ and is the fraction of values in the immediate deeper level that a value dominates. Having fixed the number of attributes and their domains for the tuples, values were generated uniformly and independently.

We restrict our synthetic data experiments to tuples whose attribute values were generated independently. To compensate, we experiment with real data that exhibit a high degree of non-uniformity and negative attribute correlation.

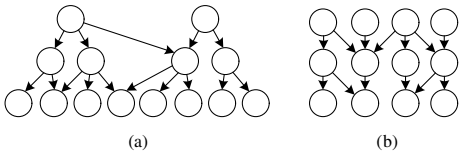


Figure 10: Poset structure.

5.3 Experimental setting

We will refer to the adapted solution of [5] as SDC (Stratification by Dominance Classification). For fairness, we used a lightweight grid to index the skybuffer instead of heavyweight R-tree, as was initially suggested in the numerical skyline maintenance solution of [25]. This is supported by recent work that argues that a grid is a more appropriate index than an R-tree in the context of streaming applications [9, 17, 22].

A stream of tuples arrives continuously at the system which maintains a sliding window buffer. The buffer stores the n most recent tuples that have arrived from the stream. When a new tuple arrives, the oldest tuple in the buffer is removed to free up space for the incoming tuple. After each buffer update, the skyline as well as all supporting indices are brought up to date.

The experimental setting parameters that can potentially affect performance are the size of the buffer n, the dimensionality of the data d and the structure of the categorical domains. As was described in Section 5.2, domains are characterized by their type (tree, wall) and a triplet (m, h, c). Therefore, we designed a set of experiments to evaluate the impact of these parameters on the performance of the two techniques. We also designed experiments that offer us insight into the inner workings of the techniques and help us understand how different experimental and method parameters affect performance. A detailed analysis of memory requirements is omitted due to space constraints. We comment that the memory overhead of our techniques is reasonable and within the capabilities of modern commodity hardware.

The techniques were implemented in Java and all experiments were carried out in 2.4GHz Opteron 850 processor with 4GB of memory, using the 32-bit Java 1.6 Server VM.

5.4 Experimental results

5.4.1 Performance evaluation of STARS and SDC

The goal of our first set of experiments is to identify the impact of the buffer size, data dimensionality and domain structure on performance, as measured by the average time required to process a buffer update, i.e., a combined tuple arrival and expiration. This includes the time required to update the skyline (if necessary) and all supporting indexing structures.

Figures 11(a)-(c) present the average time required per buffer update in milliseconds, for buffer sizes ranging from 10 thousand to 1 million tuples and tuples with 2, 3 and 4 attributes. For all these experiments, the domain of the attributes were trees with parameters $(500, 8, 0.3)$.

Both SDC and STARS employ a grid to index the skybuffer. Therefore, the performance of both techniques is affected by our choice of the skybuffer grid granularity. This is especially true for the case of SDC. As we elaborated in Section 5.1, SDC maps d-dimensional categorical tuples to $2d$-dimensional numerical tuples. Given the high dimensionality of the SDC grid, small changes in granularity can have huge impact on performance, as the number and size of cells is affected in an exponential manner. Nevertheless, for each individual experiment we used for SDC the granularity values that resulted in the best performance. Instead, for the STARS technique we kept the grid granularity at 10 buckets per dimension. Later in the section, we vary the grid granularity for STARS and observe performance trends.

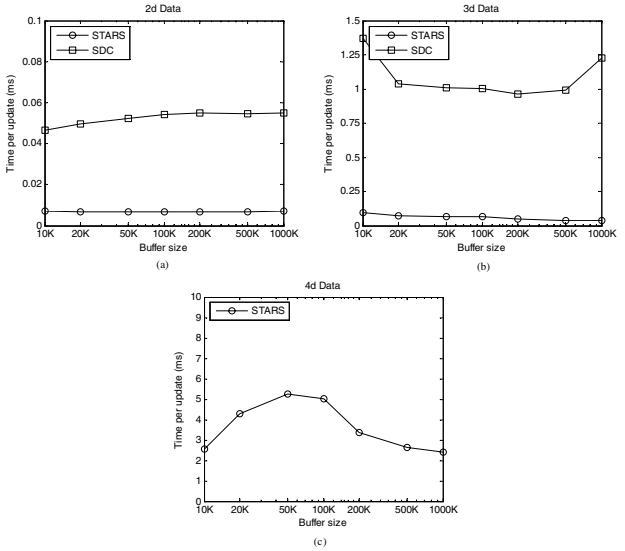


Figure 11: Effect of buffer size and dimensionality on performance.

As it is obvious in Figures 11(a)-(c), STARS outperforms SDC by an order of magnitude. In Figure 11(c) we completely omitted the SDC technique, since its performance for tuples with four attributes deteriorated. Furthermore, the time required by STARS in order to handle a buffer update is in the order of a millisecond or less, thus rendering its use in real life applications entirely realistic. We will provide further evidence that support this claim when we subsequently present our real data experiments.

Note that the performance trends in Figures 11(a)-(c) can be non-monotone with respect to the size of the buffer. This is to be expected, as there exist two competing trends that depend on the buffer size and affect performance. For example, as the buffer size increases, we can expect both the size of the skybuffer and the size of the skyline to increase. On the other hand, as the buffer size increases, the probability that the expiration of a tuple will affect the skyline decreases and so does the probability that an expensive skyline mending operation will have to be triggered. Remember that when a tuple belonging to the skyline expires, we need to identify all the tuples that it exclusively dominated and insert them in the skyline. The converse is true when the buffer size decreases: the skyline size decreases while the invocations of reconstruction operations increase.

We performed additional experiments involving tree and wall

domains with a wide range of parameters values (m, h, c). However, we omit these experimental results as we observed similar trends and performance differences between the two techniques. The shape and size of the attribute domains influence performance mostly indirectly, by affecting the average size of the skyline: domains that produce larger skylines are associated with higher buffer update cost. Keeping two of the poset parameters (m, h, c) fixed, decreasing c (internal poset connectivity) increases the skyline size, and so does increasing m (number of poset values) and decreasing h (the number of depth levels).

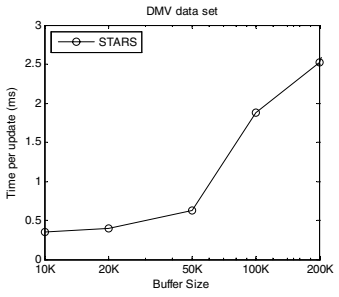


Figure 12: Performance on skewed real data.

Besides synthetic data, we also employed a stream of real, skewed and correlated data for our performance experiments. We used the DMV data set [14] and three categorical attributes of the "cars" table: Maker/Model (38 possible values), Color (504 possible values) and Year (74 possible values). The resulting 3-dimensional tuples are skewed and correlated, e.g., few of the 38 car models are popular, while Ferraris are almost exclusively red. The degree of skew and correlation in this real data set is fixed. Also, since the categorical attributes are not associated with a partial-order, we manually organized their values in tree-structured posets (Figure 10(a)).

Figure 12 depicts the results of the experiment. For buffer sizes between 10K and 200K, the time required by STARS to process a buffer update is in the order of a millisecond - an entirely realistic figure. SDC's results are omitted from the figure as it fared poorly: the update time ranged from 6ms for a 10K buffer to more than 100ms for a large, 200K buffer. The figure demonstrates a clear upward trend in the update time for larger buffer sizes. This can be attributed to the presence of skew and correlation in the data that leads to considerably larger skylines as the buffer size increases.

Our next experiment utilizes synthetic data in order to offer insight on the performance differential between SDC and STARS. Figure 13 depicts the size of the skybuffer maintained by the techniques , for tuples with two (Figure 13(a)) and three (Figure 13(b)) attributes. The domains were trees with parameters $(500, 8, 0.3)$.

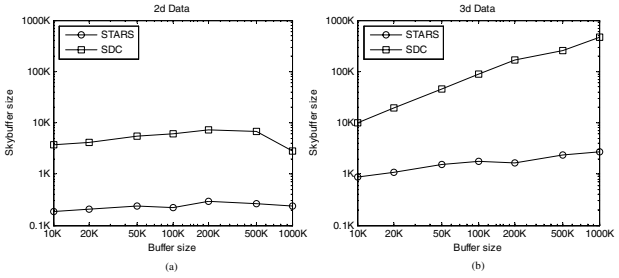


Figure 13: Size of the skybuffer as a function of the buffer size.

SDC employs a mapping of d-dimensional categorical tuples to $2d$-dimensional numerical tuples. However, the mapping is not exact in the sense that dominance in the categorical space does not imply dominance in the numerical space (Figure 9). The implication of this relation is that when a tuple is inserted in the skybuffer, a rectangular range search fails to identify all the skybuffer tuples that are dominated by the inserted tuple. Therefore, the skybuffer of SDC can contain many more tuples that the skybuffer of STARS, since tuples that could have been removed, still reside in the skybuffer. This has a big impact when SDC attempts to repair the skyline after a tuple expiration. Then, the additional tuples in the skybuffer that need to be examined become a huge burden.

Notice that both axes are in logarithmic scale. In the case of tuples with two attributes, the skybuffer size for SDC is greater than STARS's, yet it is still manageable. However, for tuples with three attributes, SDC's skybuffer size explodes: as the dimensionality increases, the number of categorical dominance relations that the mapping to the numerical domain fails to capture, increases.

5.4.2 Further evaluation of STARS

We also designed and performed experiments to further study the STARS technique and its two components. In particular, we performed experiments to quantify the pruning efficiency of the arrangement-based skyline organization, as well as the gains that can be achieved by utilizing the proposed granularity setting mechanism in conjunction with the poset partitioning technique.

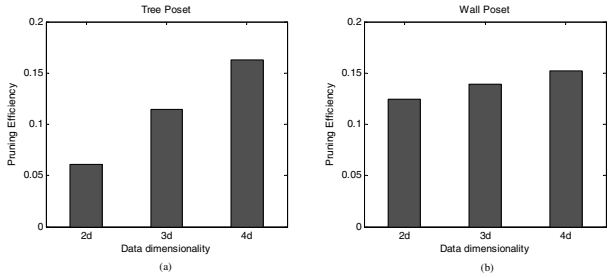


Figure 14: Pruning efficiency of arrangement skyline organization.

Figures 14(a) and 14(b) present results demonstrating the pruning efficiency of the arrangement-based skyline organization. A dominance query against the skyline determines whether a query tuple is dominated by the skyline. The objective is to do so by checking as few skyline tuples as possible. Therefore, pruning efficiency is measured as the fraction of skyline tuples that need to be examined on average in order to answer a dominance query.

As it is evident in Figures 14(a) and 14(b), the arrangement organization is able to answer a dominance query by considering on average about 10% of the skyline tuples. This is true for both tree and wall structured domains. For this experiment, we materialized the tree structured domains with parameters $(500, 8, 0.3)$ and the wall domains with parameters $(250, 10, 0.3)$. Notice, that the pruning efficiency decreases as the dimensionality increases, although not considerably. This is reasonable to expect, since the pruning technique utilizes information from only two of the tuple's attributes, therefore failing to exploit some pruning opportunities as the dimensionality increases.

Our next experiment demonstrates the potential performance benefits by utilizing the techniques of Section 4.3, that allow us to set the skybuffer grid granularity. This involves partitioning the attribute domains in disjoint value groups so that the desired grid granularity is matched and the expected query time is minimized. For this experiment, we measured the average time in milliseconds

required to perform a skybuffer update, i.e., identify all the tuples in the skybuffer dominated by an incoming tuple, remove them and insert the incoming tuple in the skybuffer.

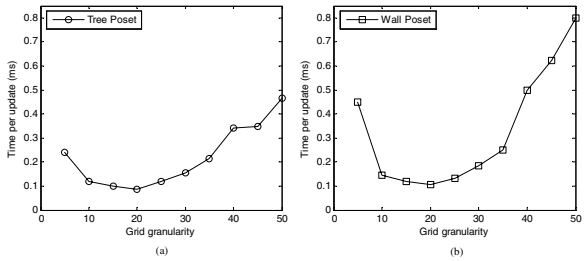


Figure 15: Effect of grid granularity on performance.

Figure 15 depicts the time required to perform a skybuffer update versus the grid granularity, for tuples with three attributes. The grid granularity is measured as the number of allocated buckets per dimension. More specifically, Figure 15(a) presents the results when the domains are tree structured posets with parameters $(200, 4, 0.1)$, while Figure 15(b) the results when the domains are wall structure posets with parameters $(200, 4, 0.1)$.

The leftmost values in the plots correspond to a partitioning that allocates a single bucket per depth level. By increasing the granularity we can achieve better performance. This performance increase would not be possible without our poset partitioning heuristic: a bad partitioning strategy would result in performance inferior to the bucket-per-depth-level, partitioning scheme. However, the poset partitioning technique allows us to translate an increase in the grid granularity to increase in performance. The benefit of increasing the grid granularity is eventually offset by the overhead of visiting many sparse cells. This additional overhead explains the knee in the curves of Figure 15. Notice that even though performance can be poor for extreme granularity values, there is a wide range of values that offer near optimal performance.

5.4.3 Summary

To summarize, our experimental evaluation demonstrated the applicability of the proposed solution to a wide range of buffer sizes and data dimensionality, for both synthetic (Figures 11(a)-(c)) and real (Figure 12) data. We also verified our claim that the skybuffer indexing technique can adapt to posets of any shape and size by offering flexibility in controlling the granularity of the grid-based indexing structure (Figure 15). The second claim that we verified was the pruning efficiency of the skyline organization, which was also found to be resilient to increases in data dimensionality (Figure 14). Lastly, we demonstrated the inapplicability of the existing offline skyline evaluation techniques of [5] in a streaming environment (Figures 11(a)-(c)) and identified the inherent reasons behind this poor performance (Figure 13).

6. CONCLUSIONS

In this paper, we identified and motivated the problem of maintaining the skyline of streaming data with partially ordered, categorical attributes and realized two novel techniques that constitute the building blocks of an efficient solution to the problem.

We introduced a lightweight data structure for indexing the tuples in the streaming buffer, that can gracefully adapt to tuples with many attributes and partially ordered domains of any size and complexity. We subsequently studied the dominance relation in the dual space and utilized geometric arrangements in order to index the categorical skyline and efficiently evaluate dominance queries. Lastly, we performed a thorough experimental study to evaluate the efficiency of the proposed techniques.

7. REFERENCES

[1] P. K. Agarwal and M. Sharir. Arrangements and their applications. In *Handbook of Computational Geometry*, chapter 2, pages 49–119. Elsevier, 2000.

[2] M. J. Atallah and S. Fox, editors. *Algorithms and Theory of Computation Handbook*. CRC Press, Inc., 1998.

[3] G. Ausiello, M. Protasi, A. Marchetti-Spaccamela, G. Gambosi, P. Crescenzi, and V. Kann. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-Verlag New York, Inc., 1999.

[4] S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[5] C. Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD*, pages 203–214, 2005.

[6] 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.

[7] C. Y. Chan, H. V. Jagadish, K.-L. Tan, A. K. H. Tung, and Z. Zhang. On high dimensional skylines. In *EDBT*, pages 478–495, 2006.

[8] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–816, 2003.

[9] G. Das, D. Gunopulos, N. Koudas, and N. Sarkas. Ad-hoc top-k query answering for data streams. In *VLDB*, pages 183–194, 2007.

[10] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications (2nd Edition)*. Springer-Verlag, 2000.

[11] A. M. Frieze and M. Jerrum. Improved approximation algorithms for max k-cut and max bisection. In *IPCO*, pages 1–13, 1995.

[12] P. Godfrey, R. Shipley, and J. Gryz. Maximal vector computation in large data sets. In *VLDB*, pages 229–240, 2005.

[13] D. Halperin. Arrangements. In *Handbook of Discrete and Computational Geometry*, chapter 24, pages 529–562. CRC Press, 2004.

[14] I. F. Ilyas, V. Markl, P. J. Haas, P. Brown, and A. Aboulnaga. Cords: Automatic discovery of correlations and soft functional dependencies. In *SIGMOD*, pages 647–658, 2004.

[15] H. V. Jagadish, N. Koudas, S. Muthukrishnan, V. Poosala, K. C. Sevcik, and T. Suel. Optimal histograms with quality guarantees. In *VLDB*, pages 275–286, 1998.

[16] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, pages 275–286, 2002.

[17] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *VLDB*, pages 804–815, 2004.

[18] K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the skyline in z order. In *VLDB*, pages 279–290, 2007.

[19] X. Lin, Y. Yuan, W. Wang, and H. Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *ICDE*, pages 502–513, 2005.

[20] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95, 2007.

[21] M. D. Morse, J. M. Patel, and W. I. Grosky. Efficient continuous skyline computation. In *ICDE*, page 108, 2006.

[22] K. Mouratidis, S. Bakiras, and D. Papadias. Continuous monitoring of top-k queries over sliding windows. In *SIGMOD*, pages 635–646, 2006.

[23] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD*, pages 467–478, 2003.

[24] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *VLDB*, pages 301–310, 2001.

[25] Y. Tao and D. Papadias. Maintaining sliding window skylines on data streams. *IEEE TKDE.*, 18(2):377–391, 2006.

[26] P. Wu, D. Agrawal, Ö. Egecioglu, and A. E. Abbadi. Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In *ICDE*, pages 486–495, 2007.