B.Comp. Dissertation

Finding Skyline Objects in Streaming Data

By

Fang Yuan

Department of Computer Science

School of Computing

National University of Singapore

2008/2009

B.Comp. Dissertation

Finding Skyline Objects in Streaming Data

By

Fang Yuan

Department of Computer Science

School of Computing

National University of Singapore

2008/2009

Project No. : H024310

Advisor : Associate Professor Chan Chee Yong

Deliverables :

Report: 1 Volume

Program and test cases: 1 CD

Abstract

Recently there is much research interest in skyline computation. In this project, we focus on skyline query over partially-ordered attribute domains in an online streaming context. We study an existing work, identify problems and limitations of it, and realize techniques to address them. In particular, we introduce an extension algorithm to adapt the existing work for a more general dominance definition that can work correctly without making assumptions about the attributes. In addition, we develop a new scheme to map tuples into lines in the Cartesian plane, which considers all attributes instead of the arbitrarily chosen two. The new mapping scheme improves pruning efficiency of the geometric arrangement. Finally, we conduct experiments to analyze the existing work and evaluate our proposed techniques.

Subject Descriptors:

E.1 Data Structures

H.2.8 Database Applications

H.3.3 Information Search and Retrieval

I.1.2 Algorithms

Keywords:

Skyline, Dominance, Query, Data Stream, Partial Order

Implementation Software and Hardware:

Java Standard Edition 1.5

JGraphT 0.7.3

Acknowledgement

I would like to give special thanks to my project advisor, Associate Professor Chan Chee Yong from the Department of Computer Science.

Prof. Chan has been patiently guiding me throughout this project, offering me insights into the project and feedback on my work from time to time.

I would also like to thank my family who have always shown me their unwavering support.

List of Figures

* Figures reproduced from [9] *Categorical Skylines for Streaming Data* (N. Sarkas, G. Das, N. Koudas, and Anthony K.H. Tung, 2008).

List of Tables

Table of Contents

1 Introduction

Recently, skyline queries have been under much research interest. A skyline query returns a set of tuples (the so called “skyline”) that are considered dominant among all available data tuples. Let us first review some definitions [5] for ease of further discussion.

Definition 1. A tuple *X dominates* a tuple *Y* iff *X* is better than or equal to *Y* in every attribute, and is better in at least one attribute. Two tuples are *tied* if neither of them dominates the other.

Definition 2. The *skyline* of a set of data tuples consists of all tuples that are not dominated by any other tuples in the set. A skyline tuple is said to be dominant.

Clearly, the skyline is highly sought by users because of its dominant nature [5]. It is especially valuable in the presence of a large amount of data, in extracting relevant information that might interest users.

Using the common example in the literature, consider the scenario where a tourist is looking for a hotel. Suppose the tourist prefers to stay at a cheap hotel that is close to the city. A hotel *X* is considered to dominate another hotel *Y* if and only if:

(1) *X*.price ≤ *Y*.price; and

(2) *X*.distance ≤ *Y*.distance; and

(3) at least one of the two relations in (1) and (2) is strict.

The skyline of hotels consists of all hotels that are not dominated by any other, which are desired by the tourist.

While much work focuses on skyline queries with totally-ordered attribute domains

[5][6][8], some deals with partially-ordered domains instead [2][9]. Partially-ordered domains have wider applicability as in the representations of hierarchies, preferences and interval data [2]. Attribute values on a partially-ordered domain may be comparable or incomparable, and their relations are commonly represented as directed acyclic graphs (DAG). Each attribute value is mapped to a vertex, and a directed edge is used to indicate the relation between two comparable values whose relation cannot be inferred by transitivity [9].

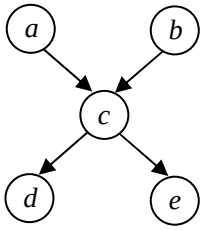


Figure 1: A sample DAG representing a partially-ordered domain

For example, in *Figure 1*, there are five possible values in a partially-ordered domain (*a*, *b*, *c*, *d* and *e*). Among those which are comparable, for example, *a* is better than *c*, and *c* is better than *d*, as indicated by the directed edges connecting them. *a* is also better than *d*, which can be inferred by the transitive property and therefore a directed edge from *a* to *d* is not required. If two values are incomparable, neither is better than the other (*a* and *b*, *d* and *e*). Additionally, if two values are said to be equal, they are meant to be represented by the same vertex. Using this notion, dominance and skyline are well defined in *Definition 1* and *Definition 2* for tuples with partially-ordered attribute domains.

Skyline queries may also take place in an online or offline environment. In an online environment, there is a stream of incoming data tuples to the system. The skyline is continuously updated upon the arrival of new tuples. In an offline environment, however, the data is less dynamic and skylines are answered on demand instead of continuously computed [9].

In this project, we investigate efficient algorithms to compute skylines on partially-ordered domains in an online streaming data context. We examine the state-of-the-art algorithm Streaming Arrangement Skyline (STARS) [9], identify its problems and limitations, and design solutions to address them. We have made the following major contribution:

(1) Extended STARS to work with the standard definition of dominance as introduced in *Definition 1*;
(2) Identified the limitation of STARS in using a geometric arrangement for the skyline, and introduced a novel scheme “minmax” to utilize it more efficiently;
(3) Conducted extensive experiments to analyze and compare STARS and our proposed techniques.

The rest of this report is organized as follows. In *Section 2* we review related work, with an emphasis on STARS. In *Section 3* we analyze the design adopted by STARS, identify its problems and limitations, and devise possible solutions to address them. In *Section 4* we address any implementation issues while in *Section 5*, we showcase experimental evaluations. Finally, *Section 6* concludes the report.

2 Related Work

A number of algorithms have been proposed to answer skyline queries. While earlier algorithms do not utilize any index structure, most of the recent ones use some index structure (e.g. R-tree is utilized in [5] and [8], and ZBtree in [6]). It is generally agreed that non-index-based algorithms are inferior to the index-based ones [2], due to the capability of effective pruning by the index structures.

The above works of NN [5], ZBtree [6] and BBS [8] deal exclusively with totally ordered domains. There are also some algorithms that are able to work on partially-ordered domains (e.g. SDC [2] and STARS [9]). As expected, the latter is more complex than the former, as the latter must transform partially-ordered domains in a suitable way in order to utilize some index structure for effective pruning.

In SDC [2], each value v in a partially-ordered domain is mapped to an interval $f_i(v) \in \boldsymbol{N} \times \boldsymbol{N}$, where $\boldsymbol{N}$ denotes the set of natural numbers, such that if $f_i(v)$ contains $f_i(v')$, then v dominates v'. But the inverse is not true, thus there may be false positives in the skylines computed based on the transformed domain, which must be checked. R-tree [2] is then used as an index structure on the transformed domains to exploit the pruning potential.

In STARS [9], each value v in a partially-ordered domain is mapped to its order $r(v)$ in a specific topological sort [7] of the DAG that represents the domain.

Definition 3. A *topological sort* of a DAG is a linear ordering of all the vertices in the DAG such that for any directed edge, the vertex where it starts is always listed before the vertex where it ends. We denote the integer indicating vertex v's position in a specific topological sort by $r(v)$.

According to *Definition 3*, there could be more than one valid topological sort for a DAG. However, for the purpose of STARS, any specific topological sort suffices [9]. Additionally, by the definition of a topological sort, if $r(v) \geq r(v')$, then v cannot be better than v' (or equivalently, v cannot dominate v'). Similarly, its inverse is not true. In this case, the actual and more expensive dominance comparison must be invoked to determine the dominance relation. The geometric arrangement [9] is then used in STARS on the transformed domains.

As our project focuses on skyline computation on partially-ordered domains in an online streaming data context, the approach by STARS is more appropriate. Although the SDC approach is efficient in an offline environment on partially-ordered domains, it suffers from the increase in data dimensionality (each attribute value is mapped to an interval represented by two integers, effectively doubling each dimension), in addition to the reduced performance in maintaining and querying the buffer in a streaming context [9]. Therefore, for the rest of this section, we review the STARS algorithm which this project is based on.

2.1 Overview of Skyline Computation in STARS

In STARS [9], a buffer of fixed size is maintained. A sliding window model is assumed, which means a new incoming tuple is inserted into the buffer, while the oldest tuple is removed or expires from the buffer if it was already full. The skyline of the streaming data is computed based only on the current buffer. Following each incoming tuple, the skyline is updated to reflect the changes in the buffer accordingly.

Furthermore, older tuples that are dominated by newer ones can never be promoted to the skyline because newer ones expire only after older ones. Therefore, these older tuples are irrelevant for the skyline computation. Only the relevant part of the buffer (the so called “skybuffer”) needs to be considered.

Algorithm Skyline computation framework

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

1. **if** *in* not dominated by *S* **then**
2. Insert *in* in *S* and remove any dominated tuples from *S*;

   endif
3. Insert *in* in *SB* and remove any dominated tuples from *SB*;
4. **if** *out* is in *S* **then**
5. Remove *out* from *S*;
6. Retrieve tuples in *SB* dominated by *out*;
7. Insert retrieved tuples that are not dominated by *S* into *S*;

   endif
8. Remove *out* from *SB*;

Figure 2: Algorithm of skyline computation framework

An overview of the skyline computation framework from STARS [9] is reproduced in *Figure 2*. The framework is invoked for every incoming tuple in a stream. It is abstract and independent of the underlying index structures. However, it reveals that retrieving tuples in the skybuffer that are dominated by a query tuple[1] (line 3 and 6), and answering if a query tuple is dominated by the skyline (line 1 and 7) are two major operations that are performed every time the computation framework is invoked. To address them, STARS introduces a SkyGrid structure for the skybuffer and geometric arrangement structure for the skyline, which are capable of pruning irrelevant tuples during queries, and thus allowing the two otherwise expensive operations to be executed more efficiently.

[1] A query tuple *Q* is a tuple involved in a query issued to a data structure, such as the skybuffer or skyline. A set of tuples satisfying certain relations to *Q*, or a result related to *Q*, is expected to be returned.

This project is based on the same framework in *Figure 2*, with modifications mainly in the sub-operations of the framework.

2.2 Skybuffer Organization in STARS

Skybuffer uses a SkyGrid as its underlying index structure. As discussed in *Section 2.1*, the main query it needs to support is to return the set of tuples in skybuffer that are dominated by a query tuple.

Each dimension of a data tuple is mapped to a dimension of the SkyGrid, forming a multi-dimensional grid. Values in a dimension are grouped, with each group (which may consist of one or more values) mapped to a bucket in the corresponding dimension of the grid. Grouping controls grid granularity without which the solution does not scale because the number of grid cells increases rapidly with the size of domains.

We will use the figure from [9] to illustrate the SkyGrid, which is reproduced in *Figure 3*. In this example, the data is 2-dimensional, and its domain on each dimension is represented by the DAG in *Figure 3(a)*. Given a desired grid granularity, grouping of values is done using a partitioning heuristic as in *Figure 3(a)*. The SkyGrid based on such a grouping is shown in *Figure 3(b)*.

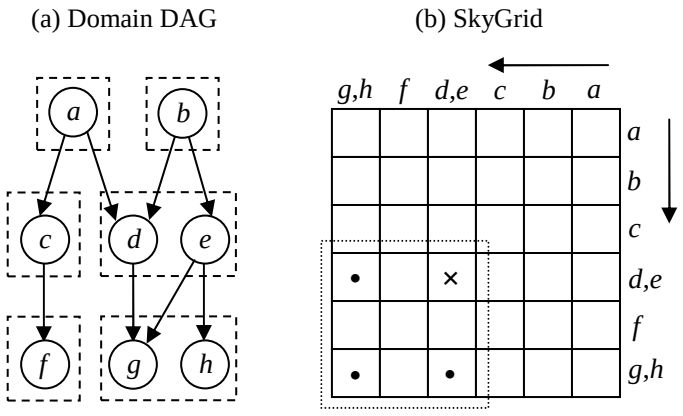


Figure 3: Value grouping and focused search in SkyGrid

The partition heuristic aims to reduce the number of SkyGrid cells returned by focused search. Focused search is the pruning capability provided by the SkyGrid, which finds relevant cells that can possibly be dominated by the query cell[2] where the query tuple would have belonged to. In *Figure 3(b)*, the query cell is marked by a cross (×), and the candidate cells returned by focused search are marked by dots (•). The actual and more expensive dominance comparison needs to be invoked only for tuples in candidate cells found by focused search. All other SkyGrid cells are ignored, as they contain no tuple which can be dominated by the query tuple.

2.3 Skyline Organization in STARS

For the skyline, each tuple is mapped to a line $y = r(a) \cdot x - r(b)$ in the Cartesian plane, where a, b are two attributes of the tuple. Recall that the notation $r(v)$ refers to the topological sorting order of v as introduced in *Definition 3*. The two attributes a and b are selected arbitrarily but statically before skyline computation starts. This means once a selection is determined, it remains bound for all tuples.

The skyline is then organized as a geometric arrangement of the mapped lines in the Cartesian plane [9]. As discussed in *Section 2.1*, the main query it needs to support is to answer whether the any skyline tuple dominates a query tuple.

STARS claims a query tuple T_Q can be dominated by a skyline tuple T_S only if the lines mapped from them intersect on the positive half of the x-axis[3]. It follows from the reasoning that if the x-coordinate of the intersecting point is negative, the two tuples are tied. Using basic algebra, the x-coordinate of the intersecting point is

$$x = \frac{r(T_Q.b) - r(T_S.b)}{r(T_Q.a) - r(T_S.a)}.$$

[2] By saying a cell X can possibly dominate another cell Y, we mean it is possible for some tuples in X to dominate some tuples in Y. We call X the query cell, and Y a candidate cell for X.

[3] As we will see in *Section 3.1*, this claim is only true given the assumption in [9], which ignores the case of equal attribute values in dominance comparison.

If $x < 0$, then either

$$r(T_Q.b) > r(T_S.b) \text{ and } r(T_Q.a) < r(T_S.a),$$

or $$r(T_Q.b) < r(T_S.b) \text{ and } r(T_Q.a) > r(T_S.a),$$

which implies either

T_Q does not dominate T_S and T_S does not dominate T_Q

or T_S does not dominate T_Q and T_Q does not dominate T_S

In either case, the two tuples are tied and can be pruned.

Therefore, the geometric arrangement only needs to store the parts of lines on the positive half of the *x*-axis, and only the lines that intersected by the query line needs to be further evaluated. Additionally, the query is progressive, returning immediately if a positive result is encountered (i.e. a skyline tuple dominates the query tuple). This progressive process can be demonstrated by the flowchart in *Figure 4*.

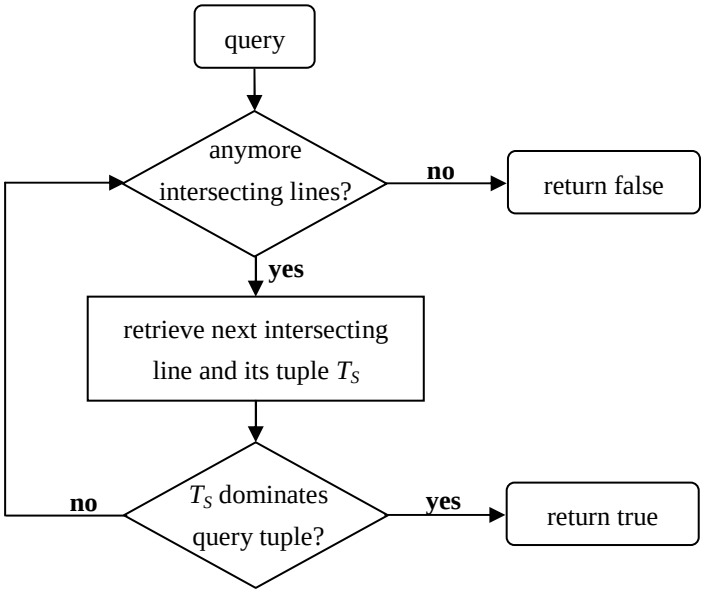


Figure 4: Flowchart for skyline arrangement query

Again, we will use the figure from [9] to illustrate the geometric arrangement, which is reproduced in *Figure 5*. In this example, the skyline consists of three tuples T_1, T_2 and T_3 which are mapped to l_1, l_2 and l_3 respectively. The query tuple T_Q is mapped to l_Q and is represented by the dotted line in *Figure 5*.

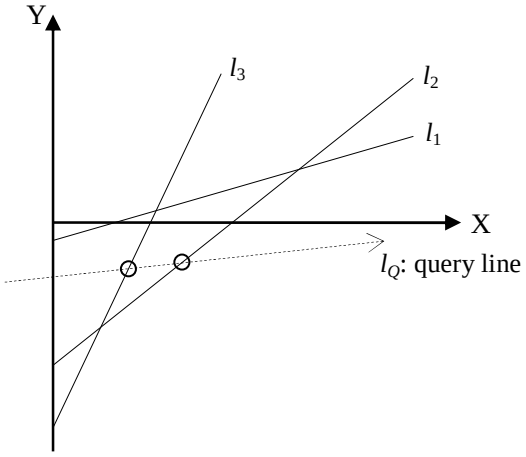


Figure 5: Geometric arrangement for the skyline

Starting from the *y*-axis, the query progressively encounters lines intersected by the query line l_Q, namely l_3 and l_2, in that order. Since l_Q encounters l_3 first, STARS first checks whether T_3 dominates T_Q by invoking the actual dominance comparison. If not, it continues and checks whether T_2 dominates T_Q, and so on. l_1 is pruned and no actual dominance comparison is needed, because it does not intersect with l_Q on positive half of the *x*-axis.

To ensure efficient operations of the geometric arrangement, STARS makes use of the data structure *doubly-connected-edge-list* (DCEL) [1], which allows the retrieval of lines intersected by a query line in $O(s)$ time, where s is the size of the skyline.

3 Design

Our design is based on the STARS [9] approach introduced in *Section 2*. We have examined this approach, and identified some problems and limitations with it. In the following subsections, we discuss our proposed solutions:

(1) An extension algorithm which allows STARS to work correctly with the standard definition of dominance introduced in *Definition 1*;

(2) A novel "minmax" mapping scheme for the geometric arrangement used in the skyline, which has better performance;

(3) Other minor optimizations.

3.1 Extension Algorithm to Query the Skyline

Sarkas et al [9] ignores the case of equal attribute values in order to simplify the discussion. With this assumption, their definition of dominance can be simplified as:

"A tuple *X* dominates a tuple *Y* iff *X* is better than *Y* in every attribute."

Comparing this with the standard definition in the literature as stated in *Definition 1*, the case of equal attribute values are eliminated.

Although this assumption seems trivial, direct application of STARS to the standard definition of dominance invalidates the pruning of the skyline by the geometric arrangement. Recall that in STARS, the geometric arrangement only stores the parts of lines mapped from skyline tuples on the positive half of the *x*-axis. A query tuple T_Q can be dominated by a skyline tuple T_S, only if the lines mapped from them intersect on the positive half of the *x*-axis. Unfortunately, this is only valid given the assumption in [9], where the case of equal attribute values is ignored.

The STARS approach reasons the validity of only checking lines intersecting on the positive half of the *x*-axis by proving lines intersecting on the negative half are

irrelevant (see *Section 2.3*). However, the relation of two lines that do not intersect on the negative half of the *x*-axis can have three disjoint scenarios:

(1) They intersect on the positive half of the *x*-axis;

(2) They intersect exactly on the *y*-axis (i.e. same *y*-intercept);

(3) They are parallel (i.e. same gradient).

The STARS approach only addresses *Scenario (1)*, which is sufficient under their assumption, as the other two scenarios can be discarded if the case of equal attribute values is not considered.

Let us now consider the case of equal attribute values. Each tuple is mapped to a line $y = r(a) \cdot x - r(b)$, where a, b are two selected attributes of the tuple. For tuples with equal value in attribute a, they map to parallel lines; for tuples with equal value in attribute b, they map to lines with same y-intercept. By *Definition 1*, they are still possible to dominate each other. Therefore, we have to consider the other two scenarios if we want to apply STARS to the standard definition.

Falsely pruned tuples in *Scenario (2)* is trivial to recover. Since the lines intersect on the *y*-axis, we just need to extend the geometric arrangement to store the parts of the lines on the non-negative half of the *x*-axis, as opposed to only the positive half in the STARS approach.

However, *Scenario (3)* needs some modifications to the original query algorithm. The original query is unable to retrieve any parallel lines. Therefore, a second auxiliary query is required to retrieve all lines parallel to the original query line l_Q. Note that doing two queries instead of one does not change the efficiency class of the algorithm.

The problem now lies in choosing a suitable auxiliary query. Apparently the auxiliary query line l_A must have a different gradient as l_Q in order to intersect lines parallel to l_Q. Furthermore, its gradient has to be larger, not only different. A query tuple T_Q can be dominated by a skyline tuple T_S when their line representations are parallel, only if

$r(T_S.b) \leq r(T_Q.b)$, i.e. only by the parallel lines above the query line l_Q[4]. In order for l_A to intersect with all lines parallel to and above l_Q, l_A must have a gradient larger than l_Q, and have the same y-intercept as l_Q.

As illustrated in *Figure 6*, the skyline consists of four tuples T_i, which map to four parallel lines l_i respectively, where i = 1, 2, 3, 4. The query tuple T_Q maps to the line l_Q. Only T_3 and T_4 can possibly dominate T_Q, because their lines l_3 and l_4 are above l_Q. T_1 and T_2 can be pruned immediately because of their neither better nor equal values in attribute b. An auxiliary query line l_A with a larger gradient and the same y-intercept as the original query line l_Q would suffice.

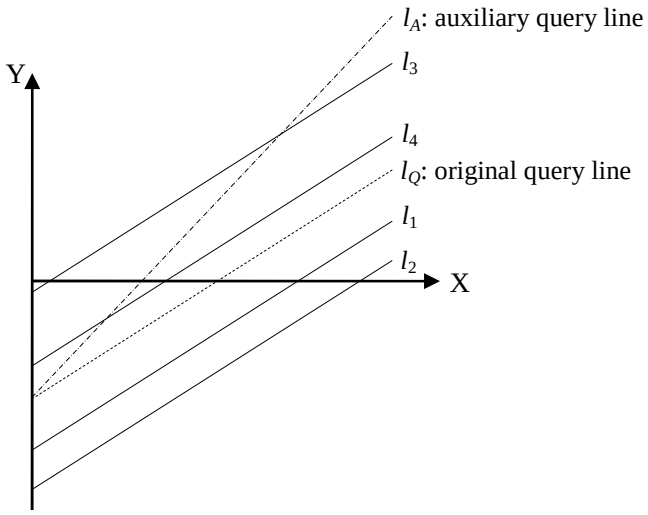


Figure 6: Auxiliary query line

Furthermore, in the process of an auxiliary query, any non-parallel lines are irrelevant and must be discarded. Therefore, we want l_A to intersect as few non-parallel lines as possible, which implies its gradient should be as small as possible. If the gradient is represented by integer, we could set l_A's gradient to $r(T_Q.a) + 1$, where $r(T_Q.a)$ is the gradient of l_Q.

[4] From *Definition 3*, a smaller topological sorting order implies a possible better value. In addition, y-intercept of a line is $-r(b)$, and thus a better value (with a smaller $r(b)$) corresponds to a higher line.

Based on our analysis, we now suggest an extension algorithm for the skyline arrangement query, shown in *Figure 7*. In line 1 and 4, the *Query* calls invoke the query function of the skyline arrangement in the original STARS approach. In essence, the first *Query* call (line 1) is the original query, making use of the original query line; while the second *Query* call (line 4) is the auxiliary query, making use of the auxiliary query line discussed earlier. If the original query returns false, it could be a mistake. In this case, the auxiliary query is issued, trying to identify and check any incorrectly pruned skyline tuple that may still be able to dominate the query tuple.

Algorithm Extension to skyline arrangement query

Input: skyline *s*, query line *l* mapped from the query tuple

Output: whether *s* dominates the query tuple (true or false)

```
1.  if Query(s, l) then                 /* original query */
2.      return true;
    else
3.      Add 1 to the gradient of l;
4.      return Query(s, l);             /* auxiliary query */
    endif
```

Figure 7: Extension algorithm of skyline arrangement query

3.2 The Minmax Mapping Scheme

Another observation of the skyline computation framework in *Figure 2*, is that the so-called “skyline mending” operation (line 7) is very expensive. The mending operation identifies all tuples in the skybuffer that are exclusively dominated by the expiring tuple from the skyline (i.e. not dominated by any other tuple in the skyline), which are then promoted to the skyline. For each tuple in the skybuffer that is dominated by the expiring tuple from the skyline, it needs a query to the skyline to

determine the exclusiveness. Therefore, the operation that checks whether a query tuple is dominated by the skyline is invoked repeatedly in a single skyline mending operation, which makes it very expensive[5].

Therefore, improve the pruning efficiency of the geometric arrangement is crucial in improving the overall performance. Currently, the pruning capability of the geometric arrangement is provided by ignoring lines that do not intersect with the query line. However, how the mapping from tuples to lines affects pruning efficiency is not discussed in the STARS approach [9].

In the STARS approach, a tuple is mapped to a line $y = r(a) \cdot x - r(b)$, where a, b are two arbitrarily and statically selected attributes of the tuple (see *Section 2.3*). There are two problems associated with this mapping scheme on data with more than two dimensions.

Firstly, it fails to specify how to select attributes. There could be a significant performance gap between the best and worst pair of selected attributes[6]. The arbitrary and static selection scheme ignores other information available, such as the characteristics of the attribute domains. If we make use of this information to make a more informed selection, we may overcome such a performance gap between the best and worst pair. Unfortunately, while there are some trends associated with certain parameters of the attribute domains, we fail to find a clear cut between best and worst pairs in other cases.

Secondly, this approach prunes tuples based on only two of their attributes. In higher dimensional data tuples, there exists other attributes whose values are simply disregarded during the pruning process. This hurts the pruning efficiency, as values of the other non-selected attributes can possibly prune off more tuples, if they were also

[5] See the performance analysis of major STARS operations in *Section 5.1*, *Figure 11*.
[6] There is indeed a significant performance gap especially on higher dimensional uniform or anti-correlated data, as we will see in *Section 5.3*, *Figure 16(a)(b)*.

considered during the prune. A mapping scheme that considers all of the attributes is conceivably better because it utilizes more available information. We expect such a scheme to perform even better as the data dimensionality increases, as in higher dimensional data, more information is ignored in the STARS approach. Also, in such a scheme, we do not have to devise an attributes selection algorithm, as all attributes are considered.

For reasons stated above, we define a new mapping scheme "minmax" that makes use of all attributes. An *n*-dimensional tuple *X* with attributes *attr*-1, *attr*-2, …, *attr*-*n*, is mapped to a line $y = A \cdot x - B$, where

$$A = \max(r(X.attr\text{-}1), r(X.attr\text{-}2), \ldots, r(X.attr\text{-}n))$$

and

$$B = \min(r(X.attr\text{-}1), r(X.attr\text{-}2), \ldots, r(X.attr\text{-}n)).$$

The maximal and minimal topological sorting orders of attribute values for each tuple are computed only once on tuple creation, as show in *Figure 8*.

Algorithm Minmax mapping scheme

Input: tuple *t*

Output: the maximal and minimal topological sorting order pair (*max_t*, *min_t*).

1. Set *max_t* to -∞, and *min_t* to +∞;
2. **foreach** attribute *attr* **in** *t*
4. **if** *r*(*attr*) > *max_t* **then** Set *max_t* to *r*(*attr*) **endif**;
5. **if** *r*(*attr*) < *min_t* **then** Set *min_t* to *r*(*attr*) **endif**;

endforeach

6. **return** (*max_t*, *min_t*)

Figure 8: Algorithm of minmax mapping scheme

This scheme considers all of the attributes, but yet it does not invalidate the pruning

capability of the geometric arrangement. Before we prove its correctness[7], we introduce two lemmas on which our proof is based.

Let l_X and l_Y be the lines mapped from n-dimensional tuples X and Y respectively, where each tuple has n attributes *attr*-1, *attr*-2, …, *attr-n*. Also let $l.A$ be the gradient and $l.B$ be the negative of the y-intercept of the line l.

Lemma 1. If $l_X.A$ is greater (or smaller) than $l_Y.A$, then there exists at least one pair of corresponding attributes in X and Y, say $X.attr\text{-}k$ and $Y.attr\text{-}k$, that satisfies the relation $r(X.attr\text{-}k)$ is greater (or smaller) than $r(Y.attr\text{-}k)$.

Proof. From the mapping scheme, we have

$l_X.A = r(X.attr\text{-}i)$ so that $r(X.attr\text{-}i) > r(X.attr\text{-}k)$, where $k \neq i$,

and $l_Y.A = r(Y.attr\text{-}j)$ so that $r(Y.attr\text{-}j) > r(Y.attr\text{-}k)$, where $k \neq j$.

Given that $l_X.A$ is greater (or smaller) than $l_Y.A$, so $r(X.attr\text{-}i)$ is also greater (or smaller) than $r(Y.attr\text{-}j)$. Now we have two cases.

Case 1: $i = j$. There is a corresponding pair $X.attr\text{-}k$ and $Y.attr\text{-}k$, where $k = i = j$, that follows the relation $r(X.attr\text{-}k)$ is greater (or smaller) than $r(Y.attr\text{-}k)$.

Case 2: $i \neq j$. We have to separate the discussion of the greater and smaller than relations. In the greater than relation, there is a corresponding pair $X.attr\text{-}i$ and $Y.attr\text{-}i$, that follows $r(X.attr\text{-}i) > r(Y.attr\text{-}j) > r(Y.attr\text{-}i)$. In the smaller than relation, there is a corresponding pair $X.attr\text{-}j$ and $Y.attr\text{-}j$, that follows $r(X.attr\text{-}j) < r(X.attr\text{-}i) < r(Y.attr\text{-}j)$.

In either case, there exists at least one pair of corresponding attributes that satisfy the

[7] We first ignore the case of equal attribute values as in the original STARS approach. Later, we show that the extension algorithm introduced in *Section 3.1* is still applicable to the minmax scheme.

relation. *Q.E.D.*

Lemma 2. If $l_X.B$ is greater (or smaller) than $l_Y.B$, then there is at least one pair of corresponding attributes from X and Y, say $X.attr\text{-}k$ and $Y.attr\text{-}k$, that satisfies the relation $r(X.attr\text{-}k)$ is greater (or smaller) than $r(Y.attr\text{-}k)$.

We do not include its proof here, because it is similar to the proof of Lemma 1. B is the minimum which is symmetric to A the maximum in *Lemma 1*.

Now, we claim that if two lines l_X and l_Y intersect on the negative half of the x-axis, they can be pruned, i.e. there is no need to invoke the actual but more expensive dominance comparison operation.

Proof. Using basic algebra, the x-coordinate of the intersecting point of two lines l_X and l_Y is

$$x = \frac{l_X.B - l_Y.B}{l_X.A - l_Y.A}.$$

If they intersect on the negative half of the x-axis, then we have $x < 0$, which implies either

$$l_X.B > l_Y.B \text{ and } l_X.A < l_Y.A,$$

or $$l_X.B < l_Y.B \text{ and } l_X.A > l_Y.A.$$

By *Lemma 1* and *Lemma 2*, it is equivalent to either

$$\exists\, i, j: \quad r(X.attr\text{-}i) > r(Y.attr\text{-}i) \quad \text{and} \quad r(X.attr\text{-}j) < r(Y.attr\text{-}j),$$

or $$\exists\, i, j: \quad r(X.attr\text{-}i) < r(Y.attr\text{-}i) \quad \text{and} \quad r(X.attr\text{-}j) > r(Y.attr\text{-}j).$$

In either case, there exist at least two pairs of corresponding attributes from X and Y that make X and Y tied. They can never dominate each other; therefore, the actual dominance comparison can be avoided. *Q.E.D.*

Let us also consider the effect of equal attribute values on the minmax mapping scheme. As discussed in *Section 3.1*, *Scenario (2)* is trivial and we only need to validate *Scenario (3)*. Fortunately, the extension algorithm suggested in *Section 3.1* can still be applied to the minmax mapping scheme.

Likewise, to retrieve parallel lines, we require an auxiliary query in the extension algorithm. The auxiliary query line l_A in *Figure 6* only intersects with lines above the original query line l_Q, for example l_3 and l_4. Only tuples mapped to these lines are possible to dominate the query tuple. Tuples mapped to lines below l_Q, for example l_1 and l_2, are impossible to dominate the query tuple. Say l_1 is mapped from the skyline tuple T_1, and l_Q mapped from the query tuple T_Q. We then have $l_1.B > l_Q.B$, because B is the negative of y-intercept. By *Lemma 2*, there exists a k such that $r(T_1.attr\text{-}k) > r(T_Q.attr\text{-}k)$, which implies T_1 can never dominate T_Q. Therefore, the extension algorithm is still applicable.

3.3 Other Optimizations

3.3.1 SkyGrid for the Skyline

Referring to *Figure 2* in *Section 2.1*, the skyline includes an operation that retrieves tuples which are dominated by a query tuple (line 2), in a similar fashion as the skybuffer does[8] (line 3 and 6). Generally, as the buffer size increases, so does the size of the skyline. The skyline may also become larger in the presence of anti-correlated data than in the case of uniform or correlated data. Without a proper index structure (for example, the SkyGrid), the skyline retrieval operation is expected to be more expensive than the skybuffer retrieval operation. Given that the SkyGrid is an effective index structure for the skybuffer[9], it is natural to speculate whether it is possible to port it for the skyline as well.

[8] We name the two operations "skyline retrieval" and "skybuffer retrieval" respectively. See *Table 2* in *Section 5*.
[9] See the performance analysis of major STARS operations in *Section 5.1*, *Figure 11*.

However, performance analysis in *Section 5.1* also reveals that the skyline retrieval operation occurs with very low frequency when compared with the skybuffer retrieval operation (see *Figure 13*). Therefore, we expect any improvement on skyline retrieval operation would only bring a marginal benefit to the overall performance. The improvement would be outweighed by the overhead required to maintain a SkyGrid structure for the skyline.

Therefore, we do not see using a SkyGrid structure for the skyline is crucial, and we do not support using it.

3.3.2 Focused Search Pre-computation

Focused search retrieves possible candidates from the skybuffer that may be dominated by a query tuple. Only the retrieved candidates are subjected to the more expensive dominance comparison operation, while other tuples are pruned by the method. We notice that for a given mapping from data dimensionality to a SkyGrid, candidate cells found by focused search in the grid can be determined independent of any data tuple present in the grid. Thus in the preprocessing stage we can do a focused search for every cell (or “query cell”, where a query tuple would have belonged to), pre-computing and storing their candidate cells.

To store the pre-computed candidate cells, we need a separate grid. It would have the same structure as the SkyGrid, except that instead of storing tuples, indices of candidate cells are stored. When a query is issued, instead of doing an on-the-fly focused search for the query cell, the candidates are directly retrieved from the new grid.

For each incoming tuple, we can expect the computation framework in *Figure 2* from *Section 2.1* to save no better than a constant time. The saving would be more significant in higher dimensional data, as the time needed to do a focused search

increases with data dimensionality. However, this saving in time does not come without a price. An $O(n^{2d})$ space overhead and preprocessing time is needed for the new grid, where n is the grid granularity and d is the dimensionality, because each of the n^d cells stores $O(n^d)$ candidates. As dimensionality increases, the storage overhead increases exponentially, making the pre-computation of candidate cells for all query cells infeasible. For example, in 4-dimensional data with n = 20, the space overhead and preprocessing time is on the scale of $20^{2\times4} = 2.56 \times 10^{10}$.

Given the large space overhead and preprocessing time needed, and the relative small portion of time spent by focused search[10], we do not support the pre-computation of focused search.

[10] Focused search is a part of the skybuffer retrieval operation as defined in *Table 2* from *Section 5*. Skybuffer retrieval itself is not an expensive operation as evident from *Section 5.1*, *Figure 11*.

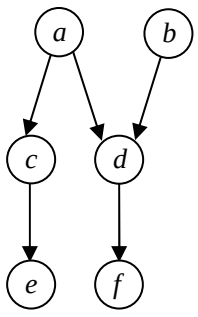
4 Implementation

Our project is implemented based on the original STARS technique [9], with modifications suggested in *Section 3*. The implementation is done in Java, with the help of a free external library JGraphT 0.7.3 [4] (which is used for implementing DAGs). The STARS technique in [9] only sketches a general outline on the algorithm. To build an actual efficient framework, many implementation details cannot be overlooked. In the following subsections, we discuss the challenges encountered in the implementation of the STARS and our proposed techniques.

4.1 Transitive Closure of DAG

The transitive closure of a directed graph with n vertices is given by an $n \times n$ matrix, which stores a Boolean value in its (i, j) entry indicating whether a path from i-th vertex to j-th vertex exists. The existence of such a path also implies that the value represented by the i-th vertex is better than that of the j-th. For example, the transitive closure matrix of the DAG in *Figure 9(a)* is shown in *Figure 9(b)*. This matrix can be computed by Warshall's algorithm [7] in $O(n^3)$ time, where n is the number of vertices in the DAG.

(a) DAG



(b) Transitive closure matrix

f	*e*	*d*	*c*	*b*	*a*	
1	1	1	1	0	0	*a*
1	0	1	0	0	0	*b*
0	1	0	0	0	0	*c*
1	0	0	0	0	0	*d*
0	0	0	0	0	0	*e*
0	0	0	0	0	0	*f*

Figure 9: Transitive closure of a DAG

Without a pre-computed transitive closure, dominance comparison between two tuples is extremely expensive because of the complexity of checking graph connectivity on-the-fly. The benefit of the transitive closure outweighs the one-time overhead of cubic time requirement. The transitive closure must be pre-computed in order to answer skyline queries with any reasonably sized domain.

4.2 SkyGrid Implementation

The SkyGrid is a multi-dimensional grid structure with variable dimensionality and granularity. Although most modern programming languages do support multi-dimensional arrays, it remains difficult to construct such a grid directly. Instead, in our implementation, the SkyGrid is mapped to a one-dimensional auxiliary array. Each row in the grid is mapped to the array successively, as illustrated in *Figure 10*. Accessing a cell in the grid thus requires a method to index into the auxiliary array, which is computed based on the indices, dimensionality and granularity of the grid. An Abstract Data Type is used to hide array access details and to create a virtual SkyGrid with variable dimensionality and granularity.

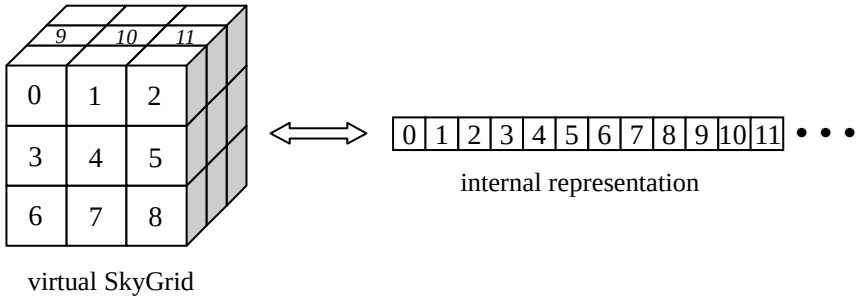


Figure 10: Mapping of SkyGrid to 1D array

4.3 Skybuffer as a FIFO Queue

Recall that skybuffer uses the SkyGrid index structure to allow efficient pruning when

answering queries. But we also have to model the skybuffer as a first-in-first-out (FIFO) queue in order to keep track of the order of the tuples coming into the skybuffer. When a new tuple comes in, and the buffer was already full, the oldest tuple expires and will be removed from both the buffer and skybuffer (if it resides in the latter as well). A FIFO queue allows retrieval of the oldest tuple at the head of the queue from the skybuffer in constant time, which will be compared against the oldest tuple in the full buffer. If they match, they are the same tuple which is expiring. Removing the expiring tuple at the head of the queue can also be done in constant time.

In our implementation, a FIFO queue is used in addition to the SkyGrid to model the skybuffer. While the former allows efficient enqueue and dequeue operations, the latter allows efficient pruning for a given query. A memory overhead of $O(s)$ is required, where s is the size of the skybuffer. But given the relative smaller size of the skybuffer as compared to the actual buffer, we expect the additional linear memory requirement is reasonable with any modern hardware.

4.4 Skyline Arrangement as a Hash Table

A hash table is used in the geometric arrangement data structure to keep track of all lines present in the structure. Each class of identical lines (i.e. lines with same gradient and y-intercept, mapped from different tuples) are stored in a list, which is in turn stored in the hash table. For each list, only one line is actually involved in the arrangement. This reduces duplication of identical lines, simplifies the arrangement, and improves query time. Using a hash table facilitates an amortized constant time access to a specific list as required by tuple addition and removal operations.

5 Experiments

This section showcases our experimental evaluations to support our design proposed in *Section 3*. While *Section 5.1* analyzes and identifies potential room for improvements in the STARS technique [9], subsequent subsections compare our suggested design with the original STARS design.

Before going into the experiments, we would like to introduce the notations used in [9] to specify input forms. Recall that an attribute domain can be modeled by a DAG. A DAG can be characterized by parameters (*m*, *h*, *c*, *f*), which are defined in *Definition 4* and *Table 1* [9]. We will refer to a DAG and the domain it represents by its parameters, for example, (500, 8, 0.3, tree).

Definition 4. In a directed acyclic graph, a vertex without any incoming edge is a *source*. The *depth* or *depth level* of a vertex is the length of the longest path from any source to this vertex.

Parameter	**Description**
m	Number of vertices in the DAG.
h	Height of the DAG. It is the number of depth levels of all vertices in the DAG.
c	Inter-connectivity ratio. A vertex has outgoing edges directed to *c* of the vertices on the next depth level, where $0 < c \leq 1$.
f	Tree or wall structure. In a tree structure, each depth level has twice as many vertices as the previous depth level has; while in a wall structure, each depth level has the same number of vertices.

Table 1: Parameters of a DAG

We also name four major operations of the STARS technique in *Table 2* to ease further discussion.

Operation	**Description**
Tuple update	A complete run of the skyline computation framework as outlined in *Section 2.1* and *Figure 2*.
Skyline mending	Identify tuples in the skybuffer that are exclusively dominated by the expiring tuple, and promote them into the skyline.
Skyline retrieval	Retrieve all tuples from the skyline that are dominated by a query tuple.
Skybuffer retrieval	Retrieve all tuples from the skybuffer that are dominated by a query tuple.

Table 2: Major operations in STARS

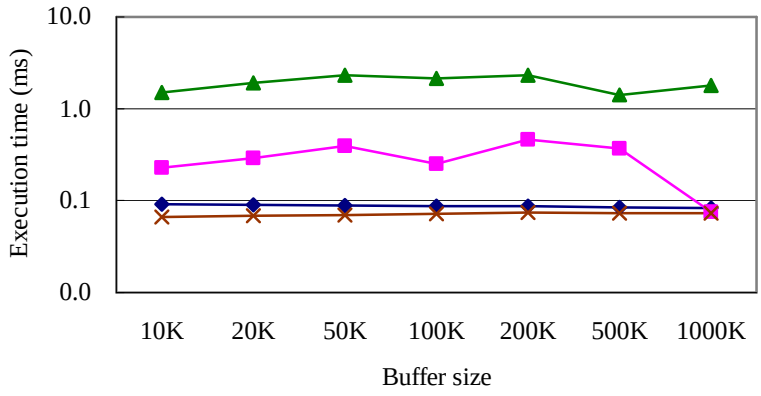
All experiments are conducted on Solaris 10, in Java Server Virtual Machine (VM) version 1.5. A heap size of 3.5GB is allocated for the Java VM.

5.1 Analysis of STARS

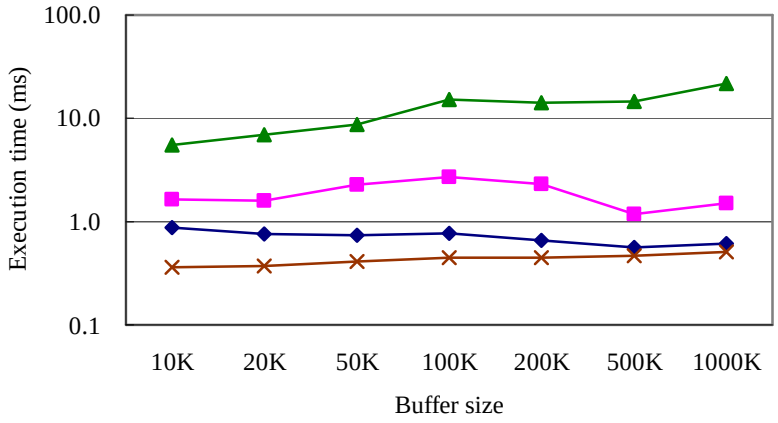
Experiments on 2D, 3D and 4D data are conducted. Synthetic data on domains (500, 8, 0.3, tree) is used, and each tuple in the stream is generated uniformly and independently. The average execution time for major STARS operations is shown in *Figure 11*. Note that the scales on the *y*-axis are logarithmic.

By comparing the experiment results on 2D data in *Figure 11(a)*, 3D data in *Figure 11(b)* and 4D data in *Figure 11(c)*, we observe the sharp increase in execution time for all operations with respect to data dimensionality. In addition, skyline mending is the most expensive operation across all dimensions of data.

(a) 2D data



(b) 3D data



(c) 4D data

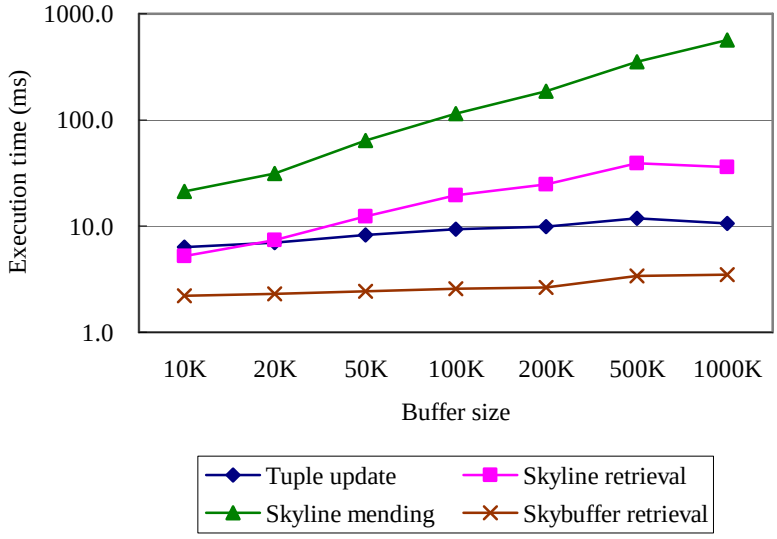


Figure 11: Performance analysis of major STARS operations

It also reveals that the SkyGrid is an efficient index structure to retrieve the set of tuples dominated by a query tuple, as evident in the small skybuffer retrieval time. In addition, as the buffer increases, the skybuffer retrieval time is fairly stable and remains small.

The skyline retrieval operation does the same job on the skyline, as the skybuffer retrieval operation does on the skybuffer. However, the skyline lacks an efficient index structure such as the SkyGrid; therefore, the skyline retrieval operation is expected to be more expensive than the skybuffer retrieval operation. Fortunately, due to the smaller size of the skyline with respect to the skybuffer, the skyline retrieval operation still gives acceptable performance, especially in comparison with the skyline mending operation.

On the other hand, the skyline mending operation is very expensive, and grows rapidly particularly in higher dimensional data. In a skyline mending operation, there are repeated queries of the skyline in order to answer if tuples in the skybuffer are exclusively dominated by the expiring tuple from the skyline. The repeated invocations to query the skyline attribute to the expensive nature of the skyline mending operation.

Now we have identified two operations that have potential room for improvement: the skyline mending and skyline retrieval operations. In order to see if they are relevant to the overall performance, we must examine the frequency of such operations as well. *Figure 12* and *Figure 13* shows that as buffer size increases, the frequencies of skyline mending and retrieval operations decrease. As the buffer size increases, the probability of an expiring tuple to affect the skyline decreases, resulting in a decreased frequency of skyline mending operation. In addition, when the buffer size increases, the skyline becomes more saturated, resulting in a decreased probability for an incoming tuple to affect the skyline. This implies a decreased frequency of skyline

retrieval operation.

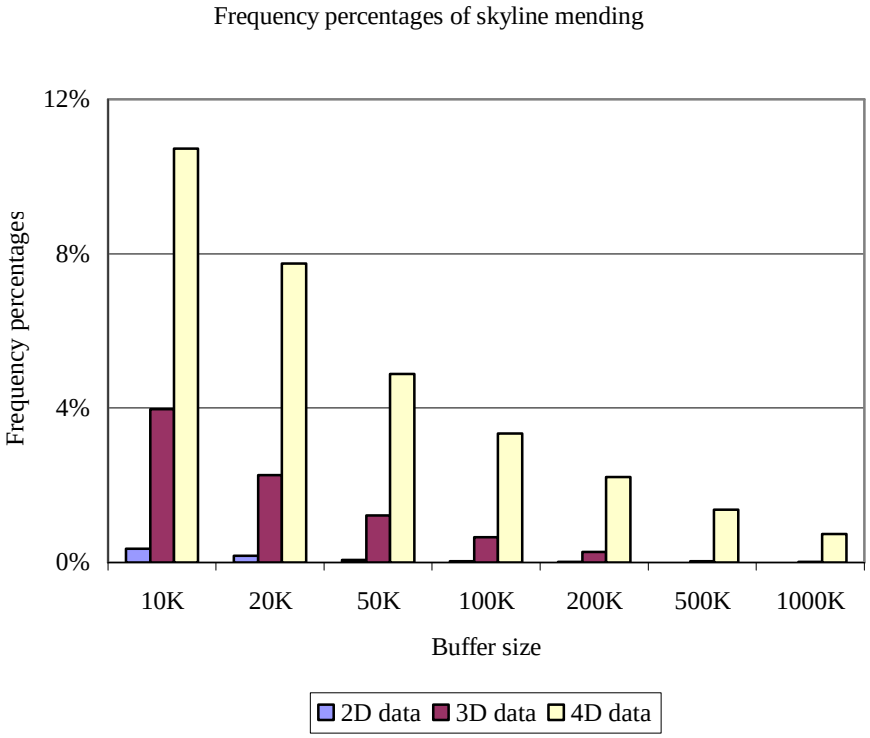


Figure 12: Frequency percentages of skyline mending operation

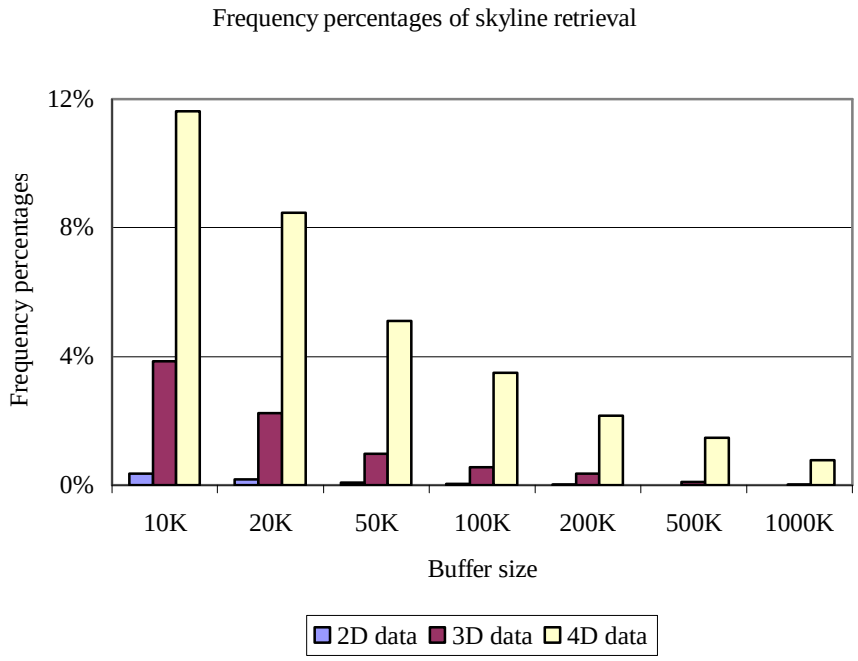


Figure 13: Frequency percentages of skyline retrieval operation

Another observation is that the two identified operations are more frequent on higher dimensional data. This is also expected, as the size of the skyline is generally larger on higher dimensional data, and thus there is a higher probability that an incoming or expiring tuple affects the skyline.

Given the expensiveness of the skyline mending operation, and its non-negligible frequencies especially on higher dimensional data, it is justified to improve it in order to improve the overall performance[11]. Note that even though the frequency is low in certain cases, and the overall performance would be less affected, it is still worth to improve the skyline mending operation, especially in time-critical systems where every tuple update must be accomplished in a specified period of time.

On the other hand, the skyline retrieval operation is much cheaper than the skyline mending operation. Although their frequencies of occurrences are at a similar level, any improvement to the skyline retrieval operation would only be marginal to the overall performance. If we were using the SkyGrid for the skyline, its improvement would be outweighed by the overhead required to maintain such a grid structure.

5.2 Correctness of the Extension Algorithm

While we have explained the correctness of the extension algorithm for the original STARS mapping scheme in *Section 3.1*, and for the minmax mapping scheme in *Section 3.2*, our experimental results have also shown support for its correctness. For the extension algorithm on the original STARS mapping scheme, we compare it with a brute force approach on small buffers. Results show that the skylines generated by both methods are identical. For the minmax mapping scheme, we compare it with the STARS mapping scheme. Again, results confirm that the generated skylines are identical.

[11] We have improved it by using the minmax scheme as introduced in *Section 3.2*.

5.3 Effects of the Minmax Mapping Scheme

We have conducted experiments comparing the minmax and the STARS arbitrary selection scheme on 3D and 4D data[12]. For the STARS scheme, we have run experiments on all possible selections of the attribute pairs, and recorded performances of the worst and best pairs.

We have synthesized domains of various parameters in *Table 3*. We have also fed each set of experiments streams of different statistical distribution, namely uniform, correlated and anti-correlated tuples [3][5]. The definition of an anti-correlated tuple is not clear when there are more than two dimensions. In our experiments, we randomly choose two dimensions and make them anti-correlated [5]. Additionally, within each set of experiments, we vary the buffer size from 10K to 1000K.

Dimension		**Set I**	**Set II**	**Set III**	**Set IV**
3D		(254, 7, 0.3, tree)	(127, 7, 0.2, tree)	(100, 10, 0.1, wall)	(510, 8, 0.3, tree)
		(189, 6, 0.6, tree)	(127, 7, 0.2, tree)	(100, 10, 0.2, wall)	(510, 8, 0.3, tree)
		(180, 20, 0.3, wall)	(124, 5, 0.2, tree)	(100, 10, 0.4, wall)	(510, 8, 0.3, tree)
	4D	(90, 4, 0.2, tree)	(124, 5, 0.2, tree)	(100, 10, 0.8, wall)	(510, 8, 0.3, tree)
Rationale		randomized parameters	different heights	different inter-connectivity ratios	same parameters

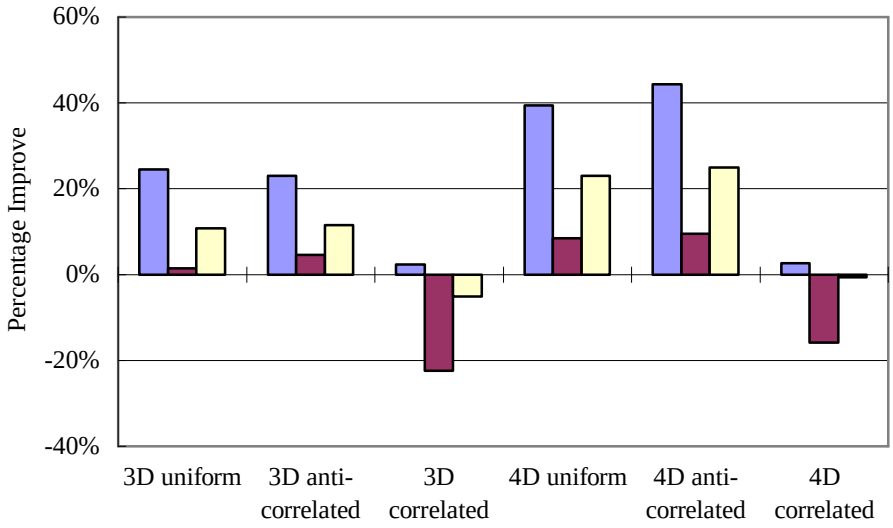
Table 3: Domains used in experiments to evaluate the minmax scheme

For each experiment, average performance[13] per tuple update using minmax scheme is compared with the best and worst performing pairs of attributes using the STARS approach. The performance of the minmax scheme and the best pair are normalized against that of the worst pair. All worst pairs have their performance normalized to 1 or 100%, serving as one standard unit. A summary[14] of all experiments comparing the

[12] For 2D data, all mappings are consistent, because there are only two attributes in a tuple.
[13] Performance refers to the measure of the execution time required for a certain operation.
[14] The detailed results for each experiment can be found in *Appendix B: Results of Experiment Sets I, II, III, IV.*

performance of minmax scheme to the worst and best performing pairs is presented in *Figure 14(a)* and *Figure 14(b)* respectively.

(a) Summary of minmax versus worst performing pairs



(b) Summary of minmax versus best performing pairs

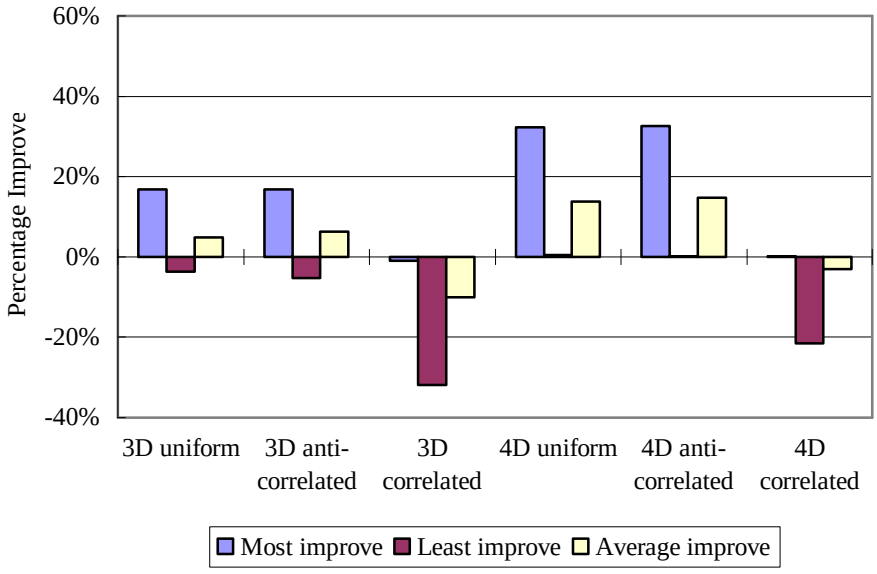
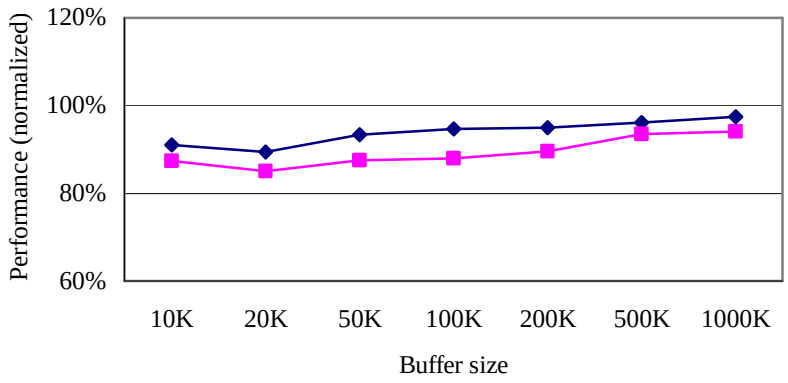


Figure 14: Summary of minmax versus worst and best performing pairs

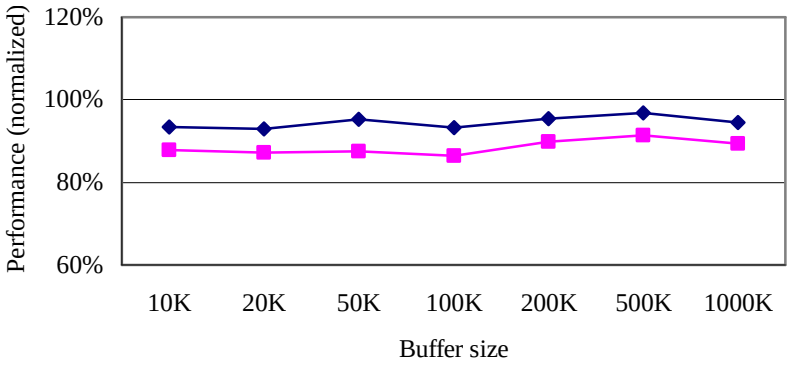
Results are also averaged across all sets of experiments, grouped by data dimensionality, statistical distribution and buffer size, as shown in *Figure 15* for 3D

data and in *Figure 16* for 4D data.

(a) 3D uniform data average



(b) 3D anti-correlated data average



(c) 3D correlated data average

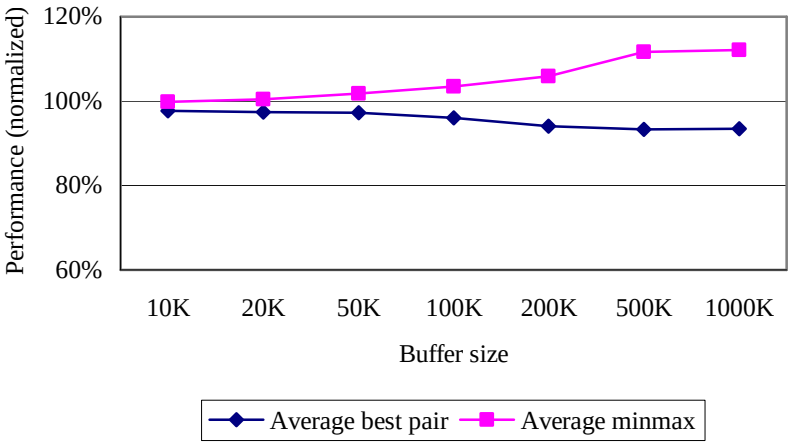
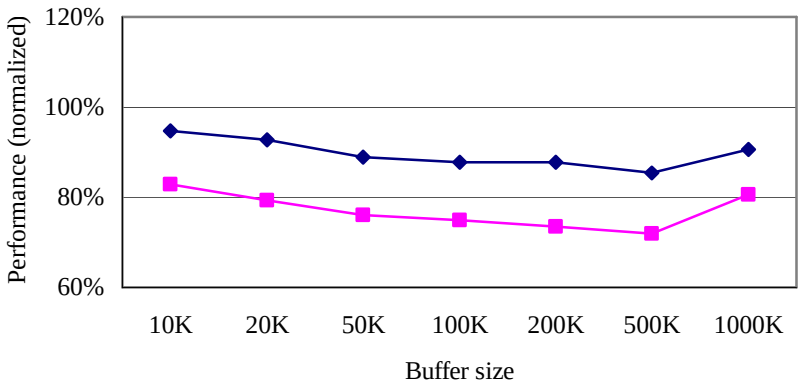
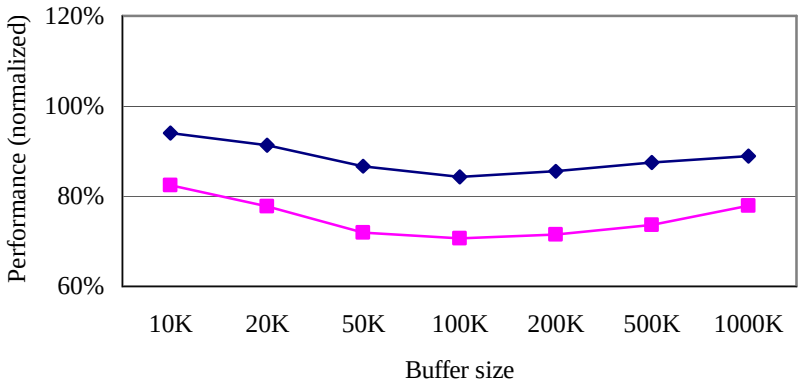


Figure 15: Comparison of average performance on 3D data

(a) 4D uniform data average



(b) 4D anti-correlated data average



(c) 4D correlated data average

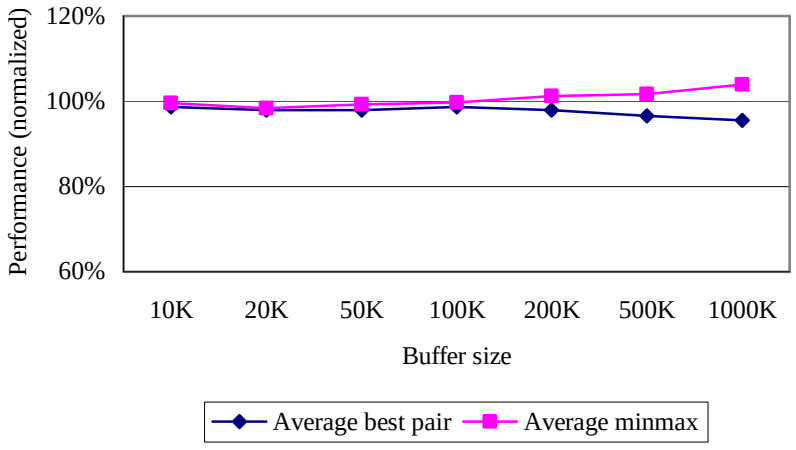


Figure 16: Comparison of average performance on 4D data

One immediate observation from the results is the minmax scheme performs better in higher dimensional data for all statistical distributions (see *Figure 14*, and compare *Figure 15* with *Figure 16*). This is expected, as in higher dimensional data, the original STARS scheme ignores more attributes, which could be otherwise useful in pruning irrelevant tuples. For example, on 3D data 1 out of 3 (or 33%) attributes are disregarded, while on 4D data 2 out of 4 (or 50%) attributes are disregarded. Also, the skyline mending operation, which utilizes the minmax scheme, occurs more frequently on higher dimensional data, as evident in *Figure 12*.

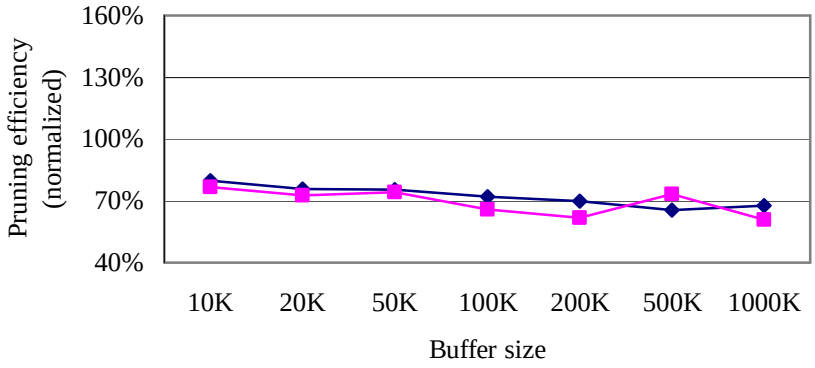
In addition to performance, we have also recorded the pruning efficiency[15] of each experiment. Results are aggregated, normalized and averaged in a similar fashion as performance. The average pruning efficiency[16], grouped by data dimensionality, statistical distribution and buffer size, is presented in *Figure 17* for 3D data and in *Figure 18* for 4D data.

From the results in *Figure 17(a)(b)* and *Figure 18(a)(b)*, we observe that on uniform and anti-correlated data, the minmax scheme generally gives better pruning efficiency than the original STARS technique. Consequently, the minmax scheme not only overcomes the performance gap between the best and worst pairs, but also gives a lead over the best pair, as evident in *Figure 15(a)(b)* and *Figure 16(a)(b)*. Moreover, on anti-correlated data, the minmax scheme achieves a slightly better performance (see *Figure 14*), as the skyline tends to be larger on anti-correlated data, which increases the frequency of the skyline mending operation.

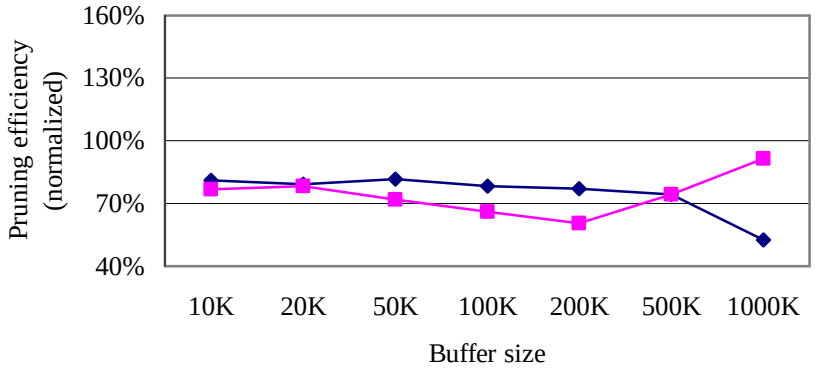
Also note that in *Figure 17(a)(b)*, there are some irregularities in the pruning efficiency with 3D data when the buffer is large (500K and 1000K). This irregularity could be caused by insufficient sample, as the frequency of skyline mending operations is low on 3D data when the buffer is large (see *Figure 12*).

[15] It is the portion of skyline tuples that require actual dominance comparison (i.e. portion of the skyline tuples that are not pruned). A smaller value indicates better pruning efficiency.
[16] The detailed results for each experiment can be found in *Appendix B: Results of Experiment Sets I, II, III, IV*.

(a) 3D uniform data average



(b) 3D anti-correlated data average



(c) 3D correlated data average

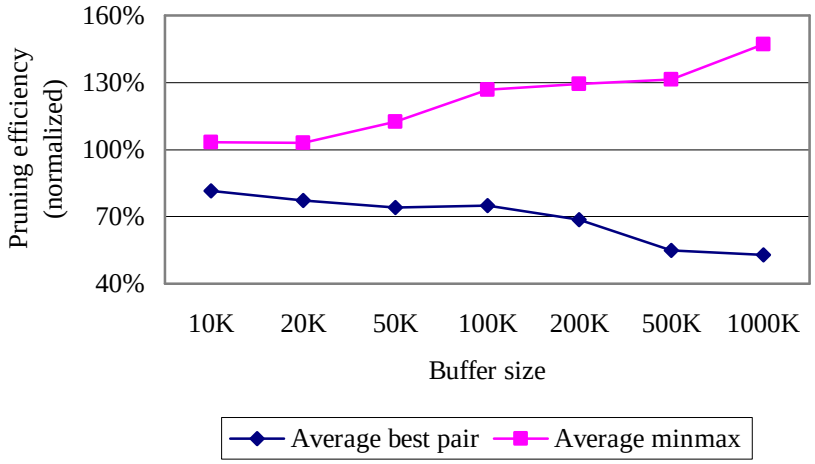
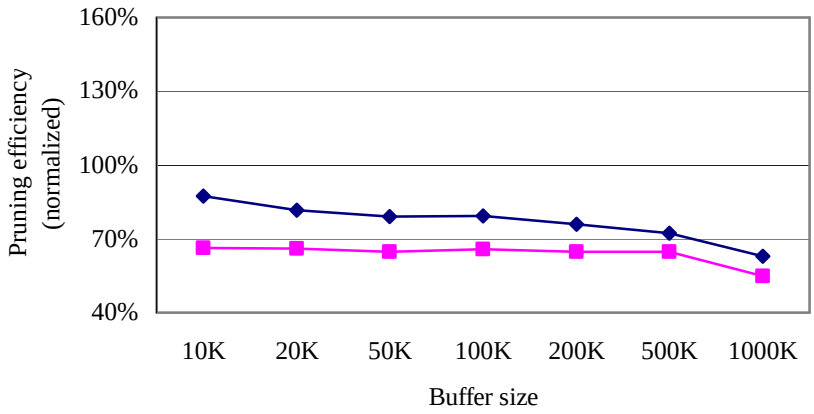
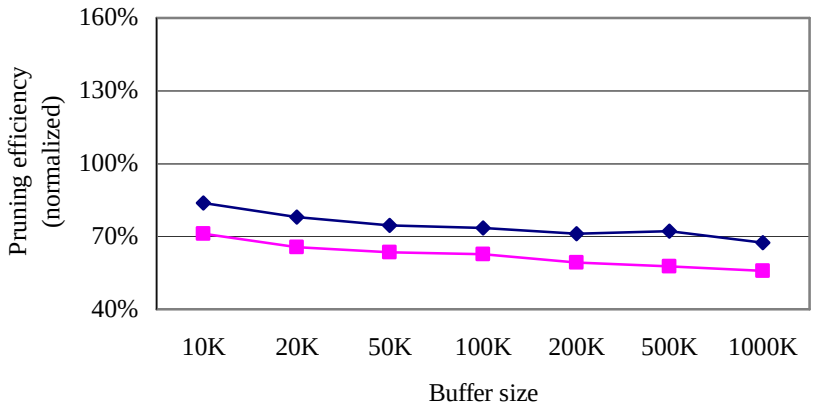


Figure 17: Comparison of average pruning efficiency on 3D data

(a) 4D uniform data average



(b) 4D anti-correlated data average



(c) 4D correlated data average

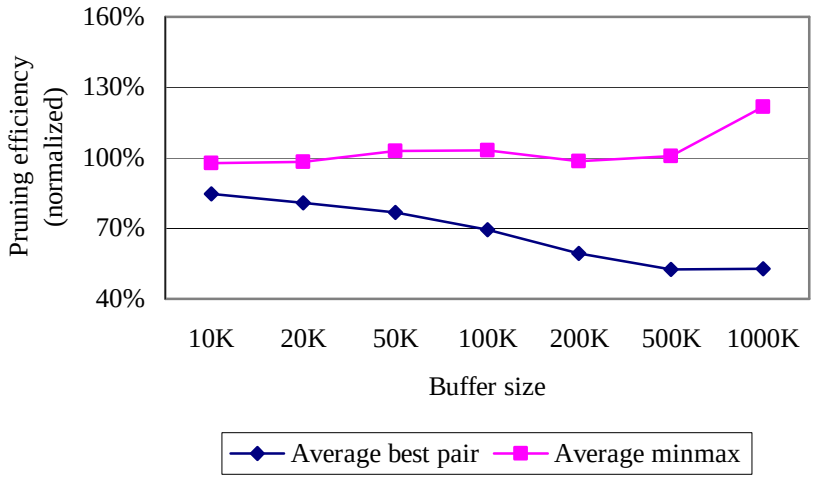


Figure 18: Comparison of average pruning efficiency on 4D data

On the other hand, on correlated data, the minmax scheme fails in terms of pruning efficiency (see *Figure 17(c)* and *Figure 18(c)*). This failure is expected due to the nature of correlated tuples.

Say we have two independently generated tuples X and Y, each with n correlated attributes *attr*-1, *attr*-2, …, *attr*-*n*.

If $\max(r(X.attr\text{-}1), \ldots, r(X.attr\text{-}n)) > \max(r(Y.attr\text{-}1), \ldots, r(Y.attr\text{-}n))$,

then the following tends to hold as well:

$$\min(r(X.attr\text{-}1), \ldots, r(X.attr\text{-}n)) > \min(r(Y.attr\text{-}1), \ldots, r(Y.attr\text{-}n)).$$

This tendency hurts the pruning efficiency, which relies on conflicting attribute pairs to conclude that X and Y are tied and can be pruned.

However, in terms of overall performance, the minmax scheme is only marginally inferior to the original STARS scheme when used with correlated data, especially on higher dimensional data (see *Figure 14*, *Figure 15(c)* and *Figure 16(c)*). There are two reasons for this result. Firstly, with correlated data, the skyline generated is generally smaller when compared to that with uniform or anti-correlated data. This leads to a lower frequency of skyline mending operations. Secondly, with correlated data, tuples in the skybuffer tends to concentrate in a few cells of the SkyGrid, which is used to model the skybuffer. This tendency reduces the effectiveness of the SkyGrid, resulting in more expensive skybuffer retrieval operations. As a result, the skyline mending operation takes up a smaller portion of the overall tuple update time, and any difference in the pruning efficiency of the arrangement structure would contribute less to the overall performance.

Based on the results of these experiments, we conclude that the minmax scheme is successful, particularly when used on higher dimensional, uniform or anti-correlated data. On correlated data, it is marginally inferior to the original STARS scheme in terms of overall performance, especially on higher dimensional data.

6 Conclusion

6.1 Summary

In this project, we studied techniques for skyline computation on partially-ordered domains in an online streaming context. We used the STARS [9] approach as a starting point, identified its limitations and problems, and devised improvements and solutions to address them.

To summarize this project, we have:

(1) Introduced an extension algorithm to apply STARS technique to the standard definition of dominance in *Definition 1*, which works correctly with equal attribute values;

(2) Designed the novel minmax mapping scheme, which considers all attributes instead of the arbitrarily chosen two. The minmax mapping scheme improves performance significantly, especially when used on high-dimensional uniform or anti-correlated data;

(3) Discussed the possibility of pre-computing focused search as well as using the SkyGrid structure for the skyline;

(4) Presented the challenges encountered during implementation, and our solutions for them;

(5) Conducted extensive experiments to analyze STARS and evaluate our proposed techniques.

6.2 Limitations and Future Work

One limitation in our work is the inability to choose a scheme based on the statistical distribution of the data stream. Although the minmax scheme is generally better than

the STARS approach, it is outperformed sometimes as we see in *Section 5.3* when correlated data is used. If the distribution of the incoming stream is known beforehand, it might be easier to address this limitation. Otherwise, it is possible to analyze a sample of the data stream before the actual skyline computation begins.

Another limitation lies in the implementation of this project. In current implementation, all data resides in the main memory. One notable data structure is the geometric arrangement for the skyline. It requires an $O(s^2)$ space, which can grow out of bound when the size s of the skyline increases. Additionally, the buffer may become very large in certain real-life applications. Future work may consider the storage of some less frequently used data in secondary memory.

Finally, the SkyGrid may deserve our attention in future work. It becomes less efficient when used on correlated data.

References

[1] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. (2000). Computational Geometry: Algorithms and Applications. Second edition. Springer-Verlag, 2000.

[2] C.Y. Chan, P.-K. Eng, and K.-L. Tan. (2005). Stratified Computation of Skylines with Partially-Ordered Domains. In *SIGMOD*, pages 203-214, 2005.

[3] P.-K. Eng and Y. Li (2008). Skyline Data Generator.

[4] JGraphT team. (2008). JGraphT 0.7.3. At http://jgrapht.sourceforge.net/.

[5] D. Kossmann, F. Ramsak, and S. Rost. (2002). Shooting Stars in the Sky: an Online Algorithm for Skyline Queries. In *VLDB*, pages 275-286, 2002.

[6] Ken C.K. Lee, B. Zheng, H. Li, and W.-C. Lee. (2007). Approaching the Skyline in Z Order. In *VLDB*, pages 279-290, 2007.

[7] A. Levitina. (2007). Introduction to the Design and Analysis of Algorithms. Second edition. Pearson Education, 2007.

[8] D. Papadias, Y. Tao, G. Fu, and B. Seeger. (2003). An Optimal and Progressive Algorithm for Skyline Queries. In *SIGMOD*, pages 467-478, 2003.

[9] N. Sarkas, G. Das, N. Koudas, and Anthony K.H. Tung. (2008). Categorical Skylines for Streaming Data. In *SIGMOD*, pages 239-250, 2008.

Appendix A: Notations Used in the Report

We have tabulated in *Table 4* a list of notations that are frequently used in this report. Some standard or widely used notations are not included.

Notation	**Meaning**
DAG	Directed acyclic graph, a directed graph without cycles.
STARS	Streaming Arrangement Skyline algorithm.
$r(v)$, or topological sorting order of v	The integer indicating vertex v's position in a specific topological sort of the DAG containing v. Sometimes, we also use v as an attribute value in a partially-ordered domain which corresponds to a vertex in the DAG that represents the domain.
$T.v$	The value of attribute v in tuple T.
$T.a$	The value of the first arbitrarily selected attribute in tuple T, used in the STARS technique.
$T.b$	The value of the second arbitrarily selected attribute in tuple T, used in the STARS technique.
$l.A$	Gradient of the line l in the Cartesian plane.
$l.B$	Negative of the y-intercept of the line l in the Cartesian plane.
$\max(a_1, \ldots, a_n)$	The maximum value among a_i, where $i = 1, 2, \ldots, n$.
$\min(a_1, \ldots, a_n)$	The minimum value among a_i, where $i = 1, 2, \ldots, n$.
nD	n-dimensional, where $n \in \boldsymbol{N}$, the set of natural numbers.

Table 4: Notations used in this report

Appendix B: Results of Experiment Sets I, II, III, IV

Detailed results of experiment sets (as designed in *Table 3*) are tabulated in this appendix.

Table 5, *Table 6* and *Table 7* contain raw results for 3D uniform, anti-correlated and correlated data respectively. 3D data tuples have three attributes numbered 0 through 2, and an attribute selection is a pair of integers indicating the attribute combination selected. There are a total of three possible attribute selections, namely (0, 1), (0, 2) and (1, 2).

Similarly *Table 8*, *Table 9* and *Table 10* contain raw results for 4D uniform, anti-correlated and correlated data respectively. 4D data tuples have four attributes numbered 0 through 3, and there are a total of six possible attribute selections, namely (0, 1), (0, 2), (0, 3), (1, 2), (1, 3) and (2, 3).

Finally, results derived from comparing the performance of the minmax scheme and the best and worst pairs are presented in *Table 11*. Statistical indicators are given on the performance improvement.

(The rest of this page is intentionally left blank.)

Buffer	Experiment	Set I		Set II		Set III		Set IV	
size	Attribute selection	Perf.^	P.E.+	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.
	(0, 1)	0.92	8.06%	1.36	14.08%	2.00	10.45%	0.87	8.68%
10K	(0, 2)	0.82	5.62%	1.21	10.29%	1.93	10.11%	0.93	10.25%
	(1, 2)	0.84	5.56%	1.22	10.05%	1.85	9.80%	0.92	10.18%
	Minmax	0.85	6.27%	1.18	11.58%	1.77	9.60%	0.76	5.56%
	(0, 1)	0.83	6.70%	0.94	10.70%	1.87	9.42%	0.76	6.37%
20K	(0, 2)	0.76	4.21%	0.85	8.50%	1.77	8.56%	0.86	8.61%
	(1, 2)	0.76	4.34%	0.84	8.00%	1.73	8.63%	0.90	8.44%
	Minmax	0.76	5.01%	0.81	8.44%	1.62	8.25%	0.68	4.23%
	(0, 1)	0.68	5.28%	0.70	8.80%	1.71	7.56%	0.74	5.39%
50K	(0, 2)	0.64	3.52%	0.68	7.29%	1.65	7.57%	0.83	7.72%
	(1, 2)	0.64	3.47%	0.66	7.38%	1.64	9.06%	0.83	7.23%
	Minmax	0.65	4.69%	0.62	7.27%	1.50	7.08%	0.65	3.60%
	(0, 1)	0.66	5.80%	0.57	9.29%	1.64	6.39%	0.77	6.50%
100K	(0, 2)	0.62	2.75%	0.56	8.47%	1.58	6.12%	0.82	5.16%
	(1, 2)	0.62	2.68%	0.54	8.49%	1.58	7.84%	0.80	4.71%
	Minmax	0.62	4.46%	0.52	6.71%	1.45	5.67%	0.64	2.74%
	(0, 1)	0.68	6.13%	0.62	8.33%	1.61	7.47%	0.66	4.83%
200K	(0, 2)	0.64	3.14%	0.63	11.21%	1.52	6.30%	0.70	3.70%
	(1, 2)	0.66	3.07%	0.61	9.15%	1.53	7.56%	0.70	3.49%
	Minmax	0.64	4.28%	0.58	6.62%	1.41	4.90%	0.59	2.60%
	(0, 1)	0.65	4.15%	0.61	6.26%	1.60	8.09%	0.56	4.54%
500K	(0, 2)	0.63	2.82%	0.59	9.12%	1.54	7.26%	0.58	2.98%
	(1, 2)	0.64	2.72%	0.58	5.78%	1.61	10.79%	0.58	3.00%
	Minmax	0.64	3.36%	0.56	7.46%	1.46	4.52%	0.54	4.00%
	(0, 1)	0.68	3.92%	0.55	5.46%	1.68	9.53%	0.61	2.73%
1000K	(0, 2)	0.67	3.01%	0.55	7.32%	1.64	8.89%	0.62	1.75%
	(1, 2)	0.67	2.78%	0.55	5.09%	1.77	13.36%	0.62	1.99%
	Minmax	0.66	3.05%	0.54	3.68%	1.55	4.10%	0.58	2.32%

Table 5: Raw results for 3D uniform data

^ Perf. = Performance (ms).
+ P.E. = Pruning efficiency (%).

Buffer	Experiment	Set I		Set II		Set III		Set IV	
size	Attribute selection	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.
	(0, 1)	0.87	9.30%	1.61	10.69%	2.08	10.86%	0.91	9.03%
10K	(0, 2)	0.80	5.25%	1.64	11.75%	2.02	9.95%	0.97	10.34%
	(1, 2)	0.76	5.42%	1.61	11.64%	1.96	9.84%	0.95	10.51%
	Minmax	0.80	6.39%	1.49	11.21%	1.83	9.71%	0.78	5.65%
	(0, 1)	0.78	7.63%	1.26	9.14%	1.92	9.62%	0.87	6.87%
20K	(0, 2)	0.73	4.40%	1.22	8.71%	1.85	8.72%	0.96	8.98%
	(1, 2)	0.70	4.43%	1.22	8.66%	1.81	8.40%	0.94	8.89%
	Minmax	0.71	5.36%	1.17	9.48%	1.69	8.78%	0.74	4.30%
	(0, 1)	0.82	5.50%	0.92	7.53%	1.69	7.71%	0.64	5.51%
50K	(0, 2)	0.79	3.29%	0.90	7.02%	1.62	7.50%	0.67	5.81%
	(1, 2)	0.78	3.48%	0.89	6.44%	1.58	7.03%	0.66	6.12%
	Minmax	0.75	3.72%	0.83	6.69%	1.48	6.81%	0.54	2.61%
	(0, 1)	0.79	5.21%	0.91	6.91%	1.63	6.26%	0.65	5.54%
100K	(0, 2)	0.70	2.71%	0.88	5.88%	1.56	5.91%	0.70	6.45%
	(1, 2)	0.72	3.01%	0.87	5.63%	1.56	6.35%	0.69	6.32%
	Minmax	0.72	3.55%	0.82	5.71%	1.42	5.12%	0.54	2.14%
	(0, 1)	0.77	4.57%	0.79	5.18%	1.60	5.79%	0.62	3.54%
200K	(0, 2)	0.71	2.14%	0.79	4.96%	1.55	5.85%	0.66	3.74%
	(1, 2)	0.74	2.28%	0.78	4.69%	1.58	6.98%	0.66	4.05%
	Minmax	0.72	2.99%	0.73	4.04%	1.42	4.29%	0.56	1.51%
	(0, 1)	0.76	3.25%	0.65	5.85%	1.56	7.32%	0.71	2.51%
500K	(0, 2)	0.74	2.33%	0.65	4.42%	1.53	7.51%	0.74	3.11%
	(1, 2)	0.72	2.40%	0.65	4.54%	1.56	9.19%	0.75	3.55%
	Minmax	0.71	3.80%	0.62	5.20%	1.41	4.54%	0.65	1.50%
	(0, 1)	0.70	4.52%	0.61	7.44%	1.65	10.05%	0.65	2.63%
1000K	(0, 2)	0.68	1.70%	0.60	3.09%	1.61	8.36%	0.71	3.84%
	(1, 2)	0.66	1.24%	0.59	4.89%	1.69	10.41%	0.71	4.29%
	Minmax	0.64	6.59%	0.57	11.19%	1.47	3.85%	0.61	1.37%

Table 6: Raw results for 3D anti-correlated data

Buffer	Experiment	Set I		Set II		Set III		Set IV	
size	Attribute selection	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.
10K	(0, 1)	1.67	6.91%	1.23	32.48%	3.26	12.38%	1.06	12.98%
	(0, 2)	1.60	6.85%	1.23	27.58%	3.23	12.24%	1.07	11.89%
	(1, 2)	1.60	6.75%	1.21	21.87%	3.19	9.47%	1.07	11.04%
	Minmax	1.63	8.86%	1.25	31.08%	3.23	12.36%	1.08	11.59%
20K	(0, 1)	1.67	5.86%	1.18	22.22%	3.72	11.53%	1.07	18.97%
	(0, 2)	1.65	6.66%	1.15	24.79%	3.84	11.03%	1.07	15.86%
	(1, 2)	1.61	6.26%	1.16	23.62%	3.75	7.66%	1.06	12.31%
	Minmax	1.65	8.54%	1.21	25.58%	3.83	11.52%	1.08	15.34%
50K	(0, 1)	1.85	5.37%	1.10	21.74%	5.84	9.75%	1.08	17.96%
	(0, 2)	1.88	5.15%	1.13	26.45%	5.84	8.83%	1.07	14.68%
	(1, 2)	1.85	5.28%	1.14	24.21%	5.55	6.12%	1.07	9.96%
	Minmax	1.94	8.94%	1.15	19.71%	6.03	11.14%	1.08	16.93%
100K	(0, 1)	2.14	4.96%	1.20	22.61%	9.14	9.71%	1.04	12.92%
	(0, 2)	2.18	4.97%	1.24	27.99%	8.92	8.39%	1.03	10.24%
	(1, 2)	2.14	4.96%	1.27	26.03%	8.47	5.55%	1.03	8.06%
	Minmax	2.29	9.00%	1.31	22.07%	9.64	13.43%	1.04	14.09%
200K	(0, 1)	2.44	3.61%	1.32	19.03%	15.38	10.49%	1.12	11.52%
	(0, 2)	2.50	3.81%	1.39	28.58%	14.86	8.45%	1.12	10.45%
	(1, 2)	2.52	3.41%	1.44	26.97%	13.62	4.51%	1.13	8.72%
	Minmax	2.77	8.28%	1.50	16.99%	16.69	13.69%	1.14	12.67%
500K	(0, 1)	3.22	4.71%	1.59	17.90%	32.98	6.88%	1.17	20.74%
	(0, 2)	3.43	7.08%	1.67	17.94%	32.58	6.01%	1.15	11.08%
	(1, 2)	3.37	5.86%	1.81	11.11%	30.97	4.33%	1.14	5.94%
	Minmax	3.90	10.30%	2.01	16.70%	39.33	12.42%	1.20	22.13%
1000K	(0, 1)	4.79	5.64%	1.48	9.47%	62.27	6.71%	1.17	1.50%
	(0, 2)	5.01	7.51%	1.59	6.64%	60.74	6.39%	1.17	5.06%
	(1, 2)	4.98	6.35%	1.71	4.28%	57.81	4.65%	1.16	6.84%
	Minmax	6.08	12.30%	1.73	4.93%	76.23	13.32%	1.21	11.92%

Table 7: Raw results for 3D correlated data

Buffer size	Experiment	Set I		Set II		Set III		Set IV	
	Attribute selection	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.
10K	(0, 1)	8.18	12.75%	19.55	22.27%	19.41	12.20%	6.33	14.85%
	(0, 2)	8.65	16.15%	18.70	21.88%	19.36	12.60%	6.39	14.85%
	(0, 3)	7.88	10.65%	18.34	21.36%	19.32	12.58%	6.36	15.40%
	(1, 2)	8.69	16.52%	18.38	21.63%	19.23	11.92%	6.29	14.55%
	(1, 3)	7.76	10.75%	18.69	21.58%	19.14	11.88%	6.31	15.02%
	(2, 3)	7.83	10.70%	18.30	21.90%	19.07	11.91%	6.25	14.74%
	minmax	7.66	10.70%	13.42	18.37%	17.75	9.71%	5.30	6.25%
20K	(0, 1)	8.51	10.71%	23.86	18.32%	18.56	10.44%	6.98	14.92%
	(0, 2)	9.23	13.48%	22.92	17.88%	18.56	10.19%	6.83	12.68%
	(0, 3)	8.14	8.49%	22.99	18.50%	18.50	10.49%	6.71	12.74%
	(1, 2)	9.03	13.94%	22.64	17.59%	17.93	9.62%	6.73	12.35%
	(1, 3)	7.85	8.84%	22.65	18.39%	18.41	9.75%	6.61	12.21%
	(2, 3)	8.14	8.83%	22.69	18.63%	18.05	9.43%	6.61	12.46%
	minmax	7.77	9.15%	15.33	15.62%	16.72	7.85%	5.50	5.94%
50K	(0, 1)	8.50	6.92%	19.11	13.41%	17.14	6.87%	8.21	10.77%
	(0, 2)	9.79	9.41%	17.81	11.98%	17.08	7.00%	7.97	9.24%
	(0, 3)	8.07	5.22%	17.51	12.14%	17.05	6.89%	8.05	10.23%
	(1, 2)	9.24	9.76%	18.26	12.06%	17.04	6.78%	7.52	9.24%
	(1, 3)	7.67	5.87%	17.56	11.85%	16.72	6.54%	7.72	9.79%
	(2, 3)	7.79	5.38%	17.10	11.47%	16.52	6.41%	7.74	9.93%
	minmax	7.64	6.21%	12.18	10.76%	15.18	5.02%	6.08	4.69%
100K	(0, 1)	8.60	5.24%	13.88	8.92%	17.39	6.39%	9.32	9.03%
	(0, 2)	10.35	8.10%	13.02	8.41%	17.01	6.50%	9.06	8.19%
	(0, 3)	8.33	4.65%	12.69	8.26%	17.04	6.56%	8.93	7.91%
	(1, 2)	10.00	8.83%	12.93	8.17%	16.72	6.43%	8.85	8.03%
	(1, 3)	7.87	5.22%	12.69	8.09%	16.70	6.29%	8.81	7.55%
	(2, 3)	8.14	4.62%	12.11	7.77%	16.45	6.18%	8.71	7.95%
	minmax	7.73	5.70%	9.42	7.04%	14.87	4.84%	6.66	4.18%
200K	(0, 1)	8.29	4.74%	12.63	6.45%	16.35	5.42%	9.81	6.71%
	(0, 2)	10.08	7.19%	11.41	5.16%	15.73	5.68%	9.41	6.00%
	(0, 3)	7.78	3.95%	11.36	5.93%	15.30	5.01%	9.38	6.05%
	(1, 2)	9.72	8.09%	11.18	5.29%	15.58	5.84%	9.59	6.09%
	(1, 3)	7.46	4.71%	11.50	5.90%	15.49	5.37%	9.62	6.09%
	(2, 3)	7.76	4.01%	11.39	5.76%	15.22	5.17%	9.49	6.19%
	minmax	7.26	4.84%	8.51	5.24%	14.32	4.01%	6.58	3.30%
500K	(0, 1)	8.04	2.74%	10.97	4.00%	15.85	3.90%	11.83	4.49%
	(0, 2)	9.97	4.38%	9.35	3.05%	15.75	4.38%	12.50	4.63%
	(0, 3)	7.92	2.19%	9.46	3.62%	15.09	4.13%	12.60	4.52%
	(1, 2)	9.56	5.29%	9.31	3.01%	15.43	4.61%	11.66	4.60%
	(1, 3)	7.39	2.91%	9.38	3.55%	15.22	4.77%	11.73	4.32%
	(2, 3)	7.81	2.31%	9.34	3.57%	14.95	4.87%	13.23	4.67%
	minmax	6.98	2.98%	7.55	3.49%	14.01	3.02%	8.02	2.50%
1000K	(0, 1)	8.14	2.23%	8.07	3.60%	15.36	3.14%	10.63	4.40%
	(0, 2)	9.49	3.46%	7.45	2.42%	14.77	3.44%	10.05	3.10%
	(0, 3)	8.18	1.93%	7.42	2.91%	14.95	3.14%	10.56	3.52%
	(1, 2)	9.24	4.75%	7.50	2.50%	15.06	3.79%	10.25	3.12%
	(1, 3)	7.87	3.10%	7.23	2.77%	14.87	3.70%	10.44	3.42%
	(2, 3)	7.83	1.99%	7.31	2.93%	14.85	4.27%	10.04	3.58%
	minmax	7.13	2.41%	6.66	2.66%	13.88	2.11%	7.91	1.98%

Table 8: Raw results for 4D uniform data

Buffer size	Experiment	Set I		Set II		Set III		Set IV	
	Attribute selection	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.	Perf.	P.E.
10K	(0, 1)	10.66	11.67%	20.97	23.09%	19.64	12.75%	9.10	17.94%
	(0, 2)	11.22	14.26%	20.17	21.54%	19.47	12.35%	9.03	17.45%
	(0, 3)	10.18	9.45%	20.16	21.32%	19.33	12.36%	9.19	18.05%
	(1, 2)	11.14	14.85%	20.25	22.55%	19.02	11.60%	8.81	16.33%
	(1, 3)	9.97	9.87%	19.95	22.03%	19.08	11.74%	9.02	17.43%
	(2, 3)	10.16	10.27%	20.00	20.98%	18.97	11.46%	9.11	17.68%
	minmax	9.95	11.38%	14.42	18.10%	17.75	10.10%	7.54	9.02%
20K	(0, 1)	9.61	7.99%	21.20	18.77%	18.55	10.12%	11.28	15.42%
	(0, 2)	10.32	10.64%	19.95	17.15%	18.31	9.82%	11.95	17.07%
	(0, 3)	9.21	6.40%	19.68	16.39%	18.25	9.56%	11.38	14.73%
	(1, 2)	10.17	11.34%	19.79	18.17%	18.03	9.40%	11.33	14.97%
	(1, 3)	8.89	6.85%	19.07	16.76%	18.11	9.10%	11.29	14.55%
	(2, 3)	9.30	6.82%	19.12	16.06%	17.83	8.95%	11.16	13.97%
	minmax	8.72	7.33%	13.29	14.39%	16.52	7.49%	8.93	8.11%
50K	(0, 1)	9.27	5.22%	23.18	13.67%	16.98	7.00%	11.12	12.35%
	(0, 2)	10.56	7.87%	20.82	11.28%	16.74	6.86%	11.37	12.57%
	(0, 3)	8.98	4.28%	20.85	11.41%	16.57	6.26%	10.45	10.07%
	(1, 2)	10.15	8.22%	20.27	12.15%	16.54	6.57%	10.66	10.98%
	(1, 3)	8.32	4.83%	19.88	11.68%	16.24	6.10%	10.55	10.22%
	(2, 3)	8.62	4.66%	19.00	11.12%	16.12	5.97%	10.37	10.01%
	minmax	7.87	4.75%	13.37	10.11%	14.05	5.06%	8.25	6.30%
100K	(0, 1)	8.46	3.84%	19.61	10.72%	16.39	6.32%	11.82	8.12%
	(0, 2)	10.23	5.95%	15.64	7.73%	16.03	5.93%	11.93	7.91%
	(0, 3)	8.10	3.20%	16.61	8.12%	15.94	5.92%	11.42	7.17%
	(1, 2)	9.50	6.53%	15.86	8.27%	15.72	5.76%	11.50	7.41%
	(1, 3)	7.75	3.75%	15.57	8.57%	15.68	5.65%	11.22	6.77%
	(2, 3)	8.18	3.64%	14.33	8.18%	15.47	5.82%	11.21	7.00%
	minmax	7.25	3.72%	10.92	8.11%	14.31	4.51%	8.20	3.80%
200K	(0, 1)	8.10	2.78%	13.80	7.25%	16.05	5.53%	9.67	6.37%
	(0, 2)	10.07	4.80%	11.74	5.10%	15.68	5.58%	9.86	6.40%
	(0, 3)	7.91	2.14%	13.76	5.03%	15.43	5.02%	9.47	5.60%
	(1, 2)	9.30	4.85%	11.53	5.46%	15.46	5.15%	9.83	6.11%
	(1, 3)	7.69	2.58%	11.56	5.03%	15.23	4.79%	9.38	5.58%
	(2, 3)	7.72	2.41%	10.81	4.91%	14.96	4.88%	9.31	5.53%
	minmax	7.06	2.79%	8.85	5.41%	14.09	3.71%	6.28	2.46%
500K	(0, 1)	8.42	2.26%	12.74	4.79%	15.52	4.27%	8.73	4.07%
	(0, 2)	9.71	3.49%	10.82	3.12%	15.29	4.34%	8.71	4.31%
	(0, 3)	7.83	1.62%	11.53	3.37%	15.13	4.08%	8.78	4.03%
	(1, 2)	9.05	3.65%	10.58	3.30%	15.08	4.26%	8.67	4.24%
	(1, 3)	7.24	2.05%	11.03	3.45%	14.94	3.96%	8.69	4.03%
	(2, 3)	7.56	1.81%	10.44	3.48%	14.70	4.60%	8.68	4.03%
	minmax	6.81	2.31%	8.77	3.68%	13.78	2.81%	5.87	1.29%
1000K	(0, 1)	7.65	1.60%	10.07	3.71%	16.37	4.00%	9.02	3.17%
	(0, 2)	8.61	2.64%	8.88	2.43%	15.18	4.02%	8.87	2.99%
	(0, 3)	7.62	1.04%	9.03	2.32%	15.18	3.88%	8.80	2.60%
	(1, 2)	8.59	2.88%	8.69	2.62%	15.11	3.67%	8.90	2.87%
	(1, 3)	7.24	1.38%	8.89	2.42%	14.87	3.62%	8.81	2.60%
	(2, 3)	7.40	1.20%	8.64	2.40%	14.71	4.08%	8.67	2.66%
	minmax	6.67	1.72%	7.98	2.70%	13.85	2.38%	6.31	1.02%

Table 9: Raw results for 4D anti-correlated data

Buffer size	Experiment Attribute selection	Set I Perf.	Set I P.E.	Set II Perf.	Set II P.E.	Set III Perf.	Set III P.E.	Set IV Perf.	Set IV P.E.
10K	(0, 1)	29.65	7.60%	24.68	16.47%	48.14	11.09%	18.66	9.95%
	(0, 2)	29.81	6.98%	24.47	14.86%	47.47	11.47%	18.60	10.08%
	(0, 3)	29.67	6.55%	24.34	17.01%	48.06	11.11%	18.68	10.41%
	(1, 2)	29.60	7.56%	24.41	14.70%	47.18	10.15%	18.62	10.57%
	(1, 3)	29.53	7.96%	24.44	17.01%	47.99	9.56%	18.63	10.57%
	(2, 3)	29.74	6.67%	24.34	17.76%	48.23	9.24%	18.62	10.74%
	minmax	29.62	7.65%	24.55	17.58%	48.06	11.43%	18.66	10.26%
20K	(0, 1)	29.28	7.63%	23.55	15.31%	46.10	10.75%	18.53	10.33%
	(0, 2)	30.15	6.65%	23.52	14.14%	47.22	11.15%	18.45	9.55%
	(0, 3)	29.37	6.24%	23.18	15.74%	46.22	10.90%	18.35	10.57%
	(1, 2)	29.27	7.35%	23.59	14.07%	47.33	8.85%	18.42	9.72%
	(1, 3)	29.67	8.01%	23.60	15.41%	46.25	8.94%	18.35	10.55%
	(2, 3)	29.33	6.31%	23.51	16.20%	46.86	8.11%	18.38	11.13%
	minmax	29.33	8.08%	23.46	18.45%	46.26	10.00%	18.34	9.83%
50K	(0, 1)	28.57	7.92%	22.27	14.60%	47.70	9.41%	17.93	11.72%
	(0, 2)	28.66	6.27%	22.36	12.91%	47.81	10.46%	17.86	10.86%
	(0, 3)	28.39	5.59%	22.30	13.95%	48.62	10.37%	17.92	11.58%
	(1, 2)	28.41	6.43%	22.16	12.46%	48.67	7.05%	17.86	11.05%
	(1, 3)	28.82	7.01%	21.66	12.54%	48.58	7.94%	17.95	11.53%
	(2, 3)	28.46	5.26%	22.23	14.37%	47.19	7.22%	17.92	12.31%
	minmax	28.60	8.61%	22.44	17.52%	47.90	9.20%	17.85	11.62%
100K	(0, 1)	28.71	7.36%	22.78	15.01%	48.61	8.63%	17.89	10.73%
	(0, 2)	28.75	6.21%	22.59	13.29%	48.86	9.00%	17.91	10.08%
	(0, 3)	28.62	5.67%	22.67	14.41%	49.02	9.68%	17.95	11.24%
	(1, 2)	29.03	7.17%	22.54	11.84%	49.18	6.05%	18.00	10.85%
	(1, 3)	28.82	8.34%	22.59	12.70%	49.70	7.50%	18.03	11.87%
	(2, 3)	28.65	4.82%	22.63	14.51%	48.98	5.84%	18.01	12.48%
	minmax	28.77	8.37%	22.93	19.29%	49.50	7.98%	17.98	12.63%
200K	(0, 1)	28.45	5.90%	21.63	15.51%	54.35	7.23%	17.09	10.87%
	(0, 2)	28.79	5.80%	21.58	13.24%	55.09	8.53%	17.10	10.10%
	(0, 3)	28.68	4.48%	21.64	14.73%	55.90	9.64%	17.16	11.55%
	(1, 2)	29.04	9.33%	21.54	11.40%	54.66	5.43%	17.10	9.48%
	(1, 3)	28.79	7.39%	21.54	12.31%	53.96	6.51%	17.15	10.22%
	(2, 3)	28.42	3.83%	21.78	14.52%	53.33	4.60%	17.10	12.55%
	minmax	29.13	7.07%	23.01	19.35%	55.56	7.80%	17.13	14.18%
500K	(0, 1)	29.51	5.74%	22.01	16.53%	71.09	8.24%	17.01	12.43%
	(0, 2)	29.32	6.43%	21.71	11.35%	69.64	7.58%	17.04	9.18%
	(0, 3)	28.88	5.31%	21.84	15.41%	71.28	10.53%	17.31	11.04%
	(1, 2)	30.01	9.94%	21.69	9.38%	67.22	4.46%	17.05	7.59%
	(1, 3)	30.06	8.82%	21.77	13.08%	69.73	6.80%	17.07	10.34%
	(2, 3)	28.74	5.01%	21.64	15.55%	68.07	5.73%	17.05	12.28%
	minmax	29.54	7.57%	23.81	22.10%	72.58	8.37%	17.02	14.14%
1000K	(0, 1)	31.23	7.74%	23.77	14.28%	99.10	6.00%	16.75	8.08%
	(0, 2)	31.07	6.24%	22.74	8.08%	99.47	6.58%	16.74	7.84%
	(0, 3)	30.92	5.50%	23.31	11.52%	101.61	8.53%	16.84	9.67%
	(1, 2)	32.66	9.50%	23.50	6.74%	97.76	3.34%	16.79	8.26%
	(1, 3)	32.68	9.47%	23.86	11.45%	97.43	4.57%	16.74	10.07%
	(2, 3)	31.07	4.80%	23.72	13.64%	95.05	4.06%	16.97	10.63%
	minmax	32.60	10.61%	27.63	23.83%	103.04	7.19%	16.80	13.11%

Table 10: Raw results for 4D correlated data

(a) Performance improvement of minmax against worst pair

Distribution	Uniform		Anti-correlated		Correlated	
Dimension / Improve	3D	4D	3D	4D	3D	4D
Average	10.73%	22.97%	11.48%	24.91%	-5.02%	-0.57%
Maximum	24.44%	39.38%	22.92%	44.31%	2.40%	2.72%
3rd quartile	13.27%	31.23%	13.10%	31.25%	-0.88%	0.92%
Median	10.38%	25.09%	9.83%	26.46%	-2.86%	0.47%
1st quartile	6.79%	13.97%	8.42%	15.47%	-6.23%	-0.32%
Minimum	1.54%	8.55%	4.62%	9.62%	-22.42%	-15.80%

(b) Performance improvement of minmax against best pair

Distribution	Uniform		Anti-correlated		Correlated	
Dimension / Improve	3D	4D	3D	4D	3D	4D
Average	4.91%	13.89%	6.26%	14.82%	-10.08%	-3.06%
Maximum	16.88%	32.29%	16.92%	32.55%	-0.93%	0.06%
3rd quartile	7.49%	22.54%	8.51%	24.56%	-2.33%	-0.34%
Median	4.62%	9.27%	6.52%	10.52%	-5.24%	-1.36%
1st quartile	1.74%	6.00%	3.74%	6.39%	-14.58%	-3.75%
Minimum	-3.66%	0.39%	-5.26%	0.20%	-31.86%	-21.50%

Table 11: Performance improvement of minmax against best and worst pairs

Appendix C: Program Listing

The program source code is organized by Java packages. We list here a directory of source code files, together with their descriptions.

(default package)

- Stars.java — contains main class, an entry point to STARS

(datastruct)

- Dag.java — implements directed acyclic graph
- Dimensionality.java — handles data dimensionality
- IntList.java — a list structure for holding intergers
- SkyBuffer.java — implements skybuffer
- SkyGrid.java — implements SkyGrid
- Skyline.java — implements skyline
- SkylineMaintenance.java — skyline computation framework
- Tuple.java — data structure for tuple
- TupleList.java — a list structure for holding tuples
- ValueList.java — a list structure for holding attribute values

(geometry)

- Arrangement.java — implements geometric arrangment
- Face.java — data structure for face in arrangment
- HalfEdge.java — data structure for half edge in arrangment
- Line.java — data structure for line in Cartesian plane
- LineList.java — a list structure for holding lines
- Vertex.java — data structure for vertex in arrangment

(generator)

- DagGenerator.java — a generator of attribute domain
- StreamGenerator.java — a generator of tuple stream