# Causal Search for Skylines (CSS): Causally-Informed Selective Data De-Correlation

PRATANU MANDAL, Arizona State University, USA
ABHINAV GORANTLA, Arizona State University, USA
K. SELÇUK CANDAN, Arizona State University, USA
MARIA LUISA SAPINO, University of Turin, Italy

Skyline queries are popular and effective tools in multi-criteria decision support as they extract *interesting* (pareto-optimal) points that help summarize the available data with respect to a given set of preference attributes. Unfortunately, the efficiency of the skyline algorithms depends heavily on the underlying data statistics. In this paper, we argue that the efficiency of the skyline algorithms could be significantly boosted if one could erase any attribute correlations that do not agree with the preference criteria, while preserving (or even boosting) correlations that agree with the user provided criteria. Therefore, we propose a *causally-informed selective de-correlation* mechanism to enable skyline algorithms to better leverage the pruning opportunities provided by the positively-aligned data distributions, without having to suffer from the mis-alignments. In particular, we show that, given a causal graph that describes the underlying causal structure of the data, one can identify a subset of the attributes that can be used to selectively de-correlate the preference attributes. Importantly, the proposed *causal search for skylines (CSS)* approach is agnostic to the underlying candidate enumeration and pruning strategies and, therefore, can be leveraged to improve any popular skyline discovery algorithm. Experiments on multiple real and synthetic data sets and for different skyline discovery algorithms show that the proposed causally-informed selective de-correlation technique significantly reduces both the number of dominance checks as well as the overall time needed to locate skyline points.

CCS Concepts: • **Information systems** → **Database query processing**; • **Computing methodologies** → **Causal reasoning and diagnostics**.

Additional Key Words and Phrases: Skyline queries, causal graphs, selective de-correlation, multi-criteria optimization

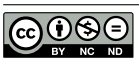
**ACM Reference Format:**
Pratanu Mandal, Abhinav Gorantla, K. Selçuk Candan, and Maria Luisa Sapino. 2026. Causal Search for Skylines (CSS): Causally-Informed Selective Data De-Correlation. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 149 (June 2026), 27 pages. https://doi.org/10.1145/3802026

## 1 Introduction

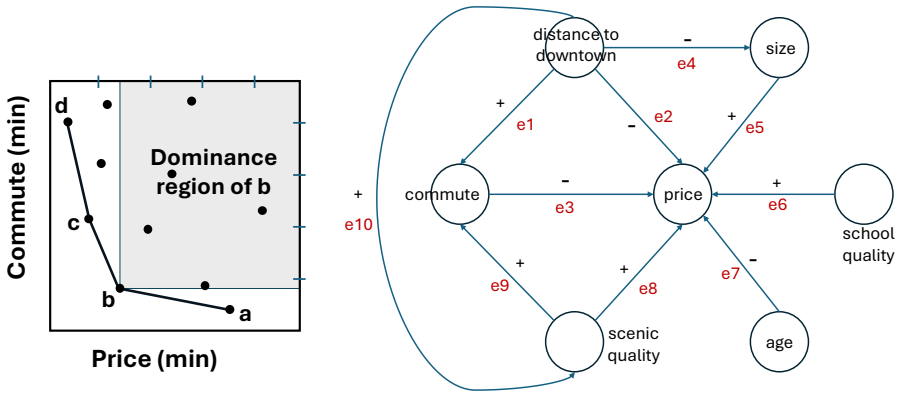
Skyline queries are popular and effective tools in decision support. Intuitively, a skyline query extracts *interesting* (pareto-optimal) points that help summarize the available data question with respect to a given set of preference attributes, providing insights into the diversity of the data across these attributes. Given a set, $D$, of data points in an attribute/feature[1] space, $A$, and a set, $P \subseteq A$ of preference attributes, the *skyline* of $D$ consists of the subset of points that are not dominated[2]

[1]We use the terms "attribute", "feature", and "variable" interchangeably.
[2]A point dominates another point if it is as good or better in all preference attributes, and better in at least one attribute.

Authors' Contact Information: Pratanu Mandal, pmandal5@asu.edu, Arizona State University, Tempe, Arizona, USA; Abhinav Gorantla, agorant2@asu.edu, Arizona State University, Tempe, Arizona, USA; K. Selçuk Candan, candan@asu.edu, Arizona State University, Tempe, Arizona, USA; Maria Luisa Sapino, mlsapino@di.unito.it, University of Turin, Turin, Italy.




ACM 2836-6573/2026/6-ART149
https://doi.org/10.1145/3802026




(a) skyline (b) causal graph

Fig. 1. Running example: skylines for house hunting

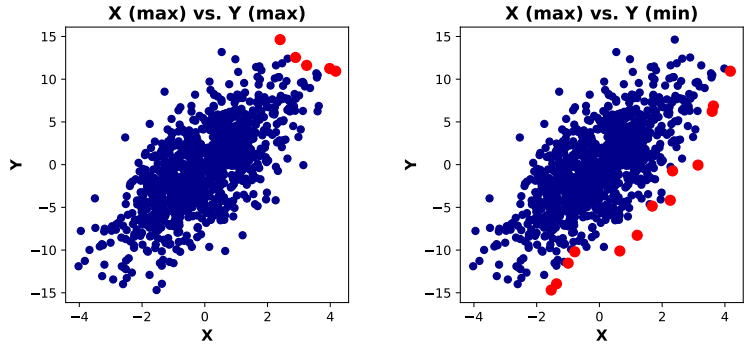
by any other data point in $D$ [13]. Searching for such non-dominated data is valuable in many applications that involve multi-criteria decision making [53].

Example 1 (House hunting). *Let us consider a user who aims to identify a subset of homes in the market with low* `Price` *and low* `Commute`*. This task can be formulated as searching a non-dominated subset of houses in the market with respect to the preference attributes,* $P = \{\texttt{Price}, \texttt{Commute}\}$ *and the corresponding preference criteria,* $\Theta = \{\theta_{\texttt{Price}}, \theta_{\texttt{Commute}}\}$*, where* $\theta_{\texttt{Price}} = min$ *and* $\theta_{\texttt{Commute}} = min$*.* ∘

Based on these specifications, any house that is in the skyline subset will either have a lower `Price` or lower `Commute` than all the other houses in the data set; in other words, the user cannot find any houses that are not in the skyline but have both lower `Price` and lower `Commute` than the houses in the skyline. In Figure 1(a), the shaded area indicates the dominance region of the house $b$: for any house in this preference range, the house $b$ is either cheaper or has lesser commute. Therefore, $b$ can be said to be more *useful* (with respect to the preference criteria) than all houses that it dominates.

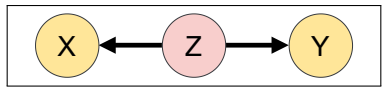
### 1.1 Challenge: Impact of the Data Distribution on Skyline Discovery Performance

As we will further discuss in Section 2, in the literature there are several algorithms developed for identifying skyline data sets. These include nested-loop skylines [13], sort-first skylines [18], and index-supported skylines [56]. Intuitively, all of these algorithms rely on an enumerate-and-prune strategy: they enumerate candidate skyline objects in the data set and they use dominance-checks to prune-away those objects that should not be included in the skyline set. The primary goal of these algorithms is to enumerate the skyline candidates in such a way that, the data set can be pruned away in the most efficient manner (i.e., using the fewest dominance checks and with the least amount of memory to store the candidate objects). Unfortunately, the efficiency of the skyline algorithms depends heavily on the underlying data statistics: as illustrated in Figure 2, for data sets where the attribute correlations mirror the corresponding preference criteria (positively-correlated attributes for `max-max` and `min-min` preference criteria and negatively-correlated for `max-min` and `min-max` criteria), the number of skyline objects are fewer; whereas for data where the attribute correlations are mis-aligned with the preference criteria, the skyline set may contain large numbers of non-dominated data objects.

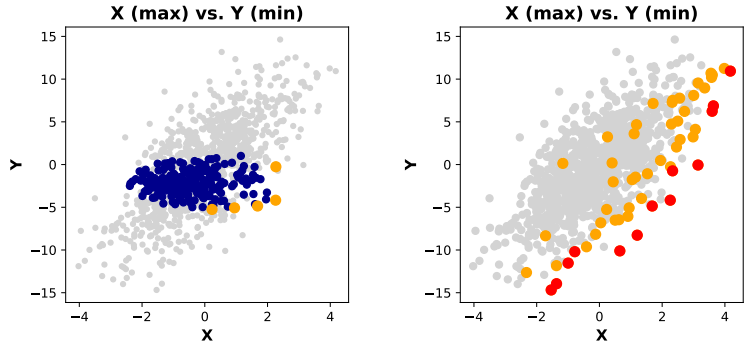



(a) max-max with '+' corr (b) max-min with '+' corr

Fig. 2. Impact of the alignment between data distribution and preference criteria: in (a) the dominance region is determined by a few extreme points, whereas in (b) there are a large number of skyline points on the Pareto front




(a) $Z$ is a common cause of $X$ and $Y$




(b) skyline of one cluster of Z (c) skyline of skylines

Fig. 3. Impact of clustering on the (a) confounder Z: (b) within each cluster, the correlation between $X$ and $Y$ is close to zero, and (c) seeking the skyline among the merged Pareto fronts (orange dots) across clusters is much less expensive, as many tuples have already been pruned

More critically though, when data correlation is mis-aligned with the preference criteria, it becomes especially difficult to enumerate the data objects in a way that helps prune non-skyline objects without having to rely on large numbers of explicit dominance checks. As a result, the cost of enumerating skylines explodes in situations where there is a mis-alignment between the preference criteria and the underlying data distribution. In this paper, we point out that we can potentially exploit underlying causal structures in the data to optimize skyline computation. In Figure 3(a), attribute $Z$ causally influences both $X$ and $Y$, creating an apparent correlation between them and, since the preference criterion is max-min, the resulting misalignment would render the

skyline search expensive as we see in Figure 2(b). As we discuss in Section 3.2, clustering on $Z$ would reduce the correlation between $X$ and $Y$ within each cluster to approximately zero, making the skyline computation for each cluster much less expensive (Figure 3(b)). If we then merge the resulting per-cluster pareto fronts and seek the skyline of the merged set, since the number of considered tuples in the merged set is far smaller than in the original data set, the negative impact of misalignment on the cost of the skyline search can be significantly reduced (Figure 3(c)).

### 1.2 Our Contributions: Causally-Informed Selective Data De-Correlation

As outlined above, in this paper, we argue that the efficiency of the skyline algorithms could be significantly boosted if one could erase any attribute correlations that do not agree with the preference criteria (i.e., *negatively-aligned* data distributions), while preserving (or even boosting) correlations that agree with the user provided criteria (i.e., *positively-aligned* data distributions). If such **selective de-correlation** were possible, the skyline algorithms would benefit from the effective pruning opportunities provided by the positively-aligned data distributions, without having to suffer from the mis-alignments. In the rest of the paper, we show that this challenge can be tackled effectively in situations where we have *a priori* causal-knowledge[3] that explains the underlying data distributions. In particular,

- we show that, given a causal graph that describes the underlying causal structure of the data, one can identify a subset, $\mathcal{Z}$, of the attributes that can be used to selectively de-correlate the preference attributes and
- since positively-aligned attribute pairs are desirable for efficient skyline discovery, whereas negatively-aligned attribute pairs need to be avoided, we present an algorithm to select a subset, $\mathcal{Z}^-$, of attributes that can be used to de-correlate the negatively-aligned preference attributes.

Importantly, the proposed *causal search for skylines (CSS)* approach is agnostic to the underlying candidate enumeration and pruning strategies and, therefore, can be leveraged to improve any popular skyline discovery algorithm, including (but not limited to) nested-loop skylines [13], sort-first skylines [18], index-supported skylines [56], as well as more advanced skyline algorithms, such as $k$-dominant skylines [15]. In Section 7, we experimentally evaluate the proposed *causal-skyline* approach on multiple real and synthetic data sets and for different skyline discovery algorithms. Experiment results show that the proposed causally-informed selective de-correlation technique significantly reduces both the number of dominance checks as well as the time needed to obtain the skyline.

## 2 Related Works

Skyline queries are commonly used in multi-criteria decision making [4, 24, 48, 60, 65]. In [48], authors use skylines to find the best clustering of data. [4] uses skyline queries to generate recommendations based on user preferences. Due to their ability to reduce the volume of data within the context of a user provided preference criterion, the task of processing skyline queries in an efficient manner has attracted considerable attention. The task of finding the non-dominated set of data points was attempted by Kung *et al.* in 1975 under the name of the maximum vector problem [29]. The algorithm proposed in [29] is based on the divide and conquer principle. Kung's algorithm lead to the development of various skyline algorithms designed for specific situations, including those devised for static environments [13, 59], high dimensional data sets [38] and parallel environments [25, 54, 58, 62]. It also inspired novel skyline algorithms for progressive skyline queries [56], online processing [28, 31, 42], skyline-join queries [40, 55, 59] and data streams [34, 57, 64].

[3]When such causal knowledge is not available, we can rely on causal discovery algorithms [17, 51, 52] to learn causal graphs, as shown in Section 7.2.5.

Borzsonyi *et al.* [13] were the first to coin and investigate the *skyline* computation problem in the context of databases. The authors extended Kung's divide and conquer algorithm so that it works well on large databases. They proposed a *Block-Nested-Loops (BNL)* algorithm, which keeps an in-memory window of incomparable points and reports a point as a skyline result only if it is not dominated by any other point in the database. They also proposed a *divide and conquer* based algorithm, which divides the data space into several regions, calculates the skyline in each region, and produces the final skyline from the points in the regional skylines.

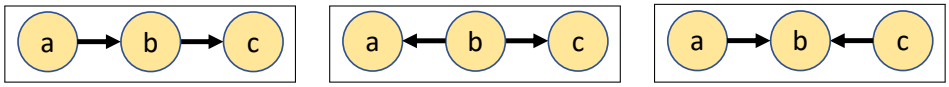
The *Sort-Filter-Skyline (SFS)* algorithm [18], which is based on the same principle as the BNL algorithm, improves on performance by first sorting the data according to a monotone function. Bartolini *et al.* also developed a sort-based skyline technique called the *Sort and Limit Skyline algorithm (SaLSa)* that uses the idea of presorting input tuples to limit the number of tuples read and compared during skyline query processing [8]. Tan *et al.* proposed progressive skyline algorithms called *bitmap* and *index* [56]. The *bitmap* method is completely non-blocking and exploits a bitmap structure to quickly identify whether a point is a skyline result or not. The *index* method, on the other hand, exploits a transformation mechanism and a $B^+$-tree index to return skyline points in batches. Other contributions to skyline query processing include online algorithms, such as [28] and [42], based on nearest-neighbor search. Bentley *et al.* developed the *Fast Linear Expected Time (FLET)* algorithm [11] which improved upon Kung's divide and conquer based algorithm by probabilistically eliminating points which cannot be part of the skyline. Godfrey *et al.* [21] demonstrated that the performance of Kung's divide and conquer based approach is poor with respect to the dimensionality of the problem. They also introduced the *Linear Elimination Sort for Skyline (LESS)* algorithm which combines aspects of BNL, SFS, and FLET, without any aspects of divide and conquer approach.

Park *et al.* proposed a mechanism for parallel processing of skyline queries to exploit multicore processors [43]; the authors also proposed a parallel version of the BBS algorithm. Lee et al. [30] proposed an index structure called *ZBtree* to index and store data points based on the Z-order curve. They also developed several skyline algorithms that utilize the *ZBtree* index to efficiently process skyline queries. Huang *et al.* [25] also proposed a parallel skyline algorithm for multi-processor clusters that utilizes Z-order clustering to reduce dominance checks. Shang *et al.* examined the skyline operator in the context of anti-correlated distributions [49]. The authors proposed a probabilistic cardinality model for anti-correlated distributions and analyzed the upper and lower bounds of the expected value of skyline cardinality. They also developed an algorithm called *SOAD (Skyline Operator on Anti-correlated Distributions)* that effectively eliminates non-promising points based on a *determination* and *elimination* framework.

There has also been significant work on estimating the cardinality of skyline sets. [7] analyzes the distribution of admissible pareto-optimal points and [12] investigates the expected number of maximal vectors in a set, providing insights into the average-case of skyline cardinality. [16, 66] proposed skyline cardinality estimation methods using log-sampling (LS). [66] introduced a non-parametric kernel-based (KB) skyline cardinality estimation technique which improves upon the log-sampling (LS) method and does not depend on the assumption that the preference attributes have to be independent of each other. [36] introduces a so-called purely-sampling based approach (PS), that does not involve complex mathematical computations and, hence is more efficient than KB. One of the more recent works in skyline cardinality estimation is [39], which outperforms the previous state-of-the-art including [16, 36, 66].

## 3 Preliminaries

This section introduces the key concepts and provides preliminary formalisms we will utilize in the rest of the paper. We assume that the reader is familiar with *basic* concepts in skylines (such




(a) chain (b) fork (c) collider

Fig. 4. Three basic causal structures

as tuple dominance) and causal graphs (such as a causal path and a confounder). The extended version of this paper [37] provides a quick refresher.

### 3.1 Causal Graphs and Skylines

Causality is the relationship between the effect (outcome) and cause (treatment) that gives rise to it [46] and is a topic of increasing interest in big data analysis [5, 20], machine learning [26, 35], and reinforcement learning [6]. Among others [32, 33, 50], Pearl has shown that a priori knowledge, in the form of *causal graphs*, is critical in analyzing and managing data [45].

Definition 1 (Causal Graph). *Let $D$ be a data set defined in an attribute space, $A$. A causal graph, $G_A = (V_A, E_A, \lambda)$, corresponding to the attribute set $A$ is an edge-labeled directed acylic graph (DAG), where for each attribute $a_i \in A$, there is a vertex $v_i \in V_A$ and each edge $e_h = \langle a_i, a_j \rangle$ (or $a_i \rightarrow a_j$) indicates a known direct causal relationship between the attributes $a_i$ and $a_j$ and the edge label $\lambda(e_h)$ indicates the nature of the causal relationship between the two attributes.*

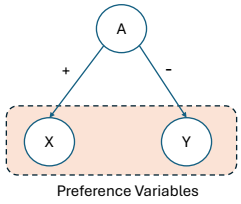
Figure 4 presents three basic causal structures introduced in [45]: (a) a *chain* structure where $a$ causally affects $c$ through its influence on $b$ (here $b$ is referred to as a mediator); (b) a *fork* structure where $b$ is the common cause of both $a$ and $c$ (note that, in this case, $a$ and $c$ are likely dependent but there is no causation between them); (c) a *collider* structure where both $a$ and $c$ independently cause $b$. In this paper, we note that, in the context of skylines, these causal structures underlying the data may introduce statistical dependence among preference attributes:

Example 2 (Weakening or Strengthening of Relationships among Preference Attributes). *Let us reconsider our house hunting running example, with preference attributes* `Commute` *and* `Price`. *Let us further assume that the causal graph underlying the data is as depicted in Figure 1(b). As we see in this example, there is a direct (negative) causal relationship between the preference attributes,* `Commute` *and* `Price` *– more commute generally translates into lower house prices. In addition, there are also several indirect contributors to the relationship between these two preference attributes:*

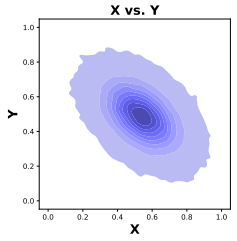
- *The* `Distance_to_city_center` *attribute has a direct causal impact on both* `Price` *and* `Commute` *attributes (and therefore is a confounder): if we consider a city (like London), where house prices in the city center are higher due to housing density and that most of the commutable jobs are also located in the city center, the* `Distance_to_city_center` *attribute would contribute additional negative correlation, strengthening the existing negative correlation between the* `Price` *and* `Commute` *attributes.*
- *In contrast, the* `Scenic_quality` *attribute impacts both* `Commute` *and* `Price` *positively as houses with scenic views generally have longer commutes and they also tend to be more expensive than the other houses. Note that this attribute weakens the negative correlation between the two preference attributes.* ○

### 3.2 Conditioning and De-Correlation

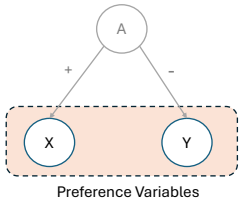
*Conditioning* is the act of clustering the data entries that share the same value for a given (set of) attribute(s).



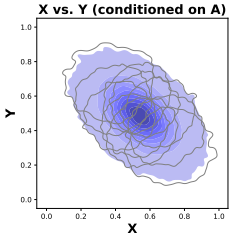

(a) a fork



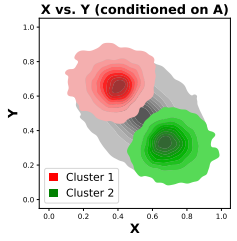

(b) X vs. Y




(c) cond. on the fork




(d) cluster outlines




(e) two sample cond. subsets

Fig. 5. (a,b) The fork attribute $A$ imposes negative correlation on attributes $X$ and $Y$; (c,d) conditioning of the attribute $A$ creates multiple clusters of data points, each cluster lacking any negative correlation; in (e), we highlight two of these resulting clusters in red and green

Definition 2 (Conditioning). *Let $D$ be a data set and let $a \in A$ be an attribute. Let $Dom(a)$ be the domain of this attribute.*

$$cond(D, a) = \{(v, D_v) \quad | \quad (v \in Dom(a)) \wedge (t \in D) \wedge (D_v = \{t.a = v\})\}. \qquad \diamond$$

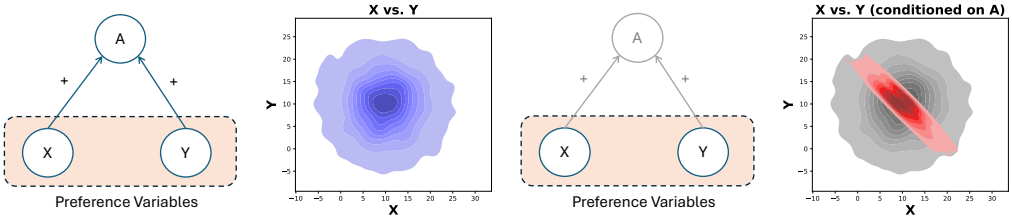
In SQL, this corresponds to a `GROUP BY` operation.

In this paper, we note that the process of conditioning can have impact on the statistical distribution of the values for the remaining attributes. For instance in the *chain* and *fork* structures conditioning on $A$ can eliminate any statistical dependence between $X$ and $Y$ (this is visualized in Figure 5). In contrast, in the *collider* structure conditioning on $A$ can introduce a (negative) correlation between unrelated variables, $X$ and $Y$ (this is visualized in Figure 6).

Key Observation 1 (Selective De-Correlation). *As a corollary to the above, we note that negative correlations can be eliminated by conditioning on the mediator/fork attributes with originally negative impact; similarly, positive correlations can be boosted by conditioning on the colliders which have originally negative impact.*

Consider the following example:

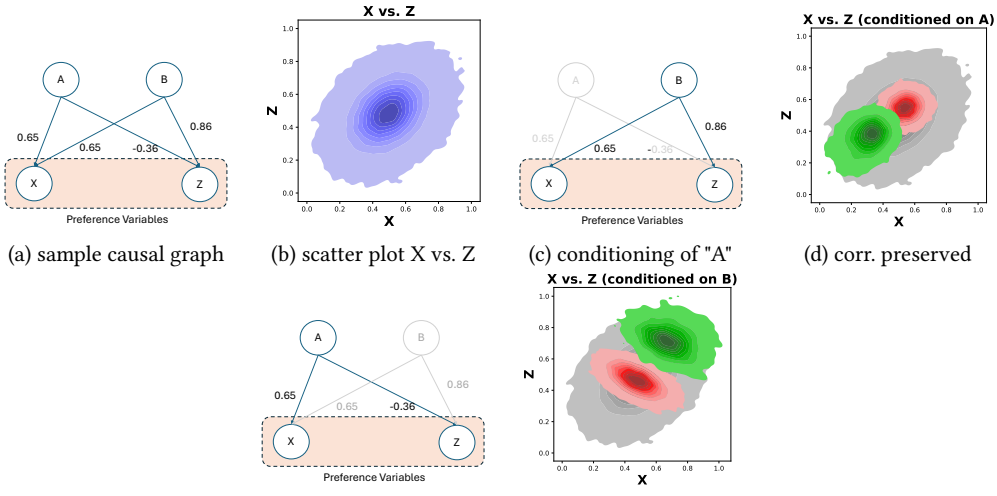
Example 3 (Conditioning to Weaken Negative Correlations among the Preference Attributes). *In Example 1, if we condition on the* `Distance_to_city_center` *by clustering all houses based on their distance to the city center and on whether they have scenic views or not, then within each such cluster, the only remaining causal relationship between* `Price` *and* `Commute` *attributes would be the direct causal relationship of* `Commute` *on* `Price`. ◦




(a) pos. collider (b) X vs. Y (c) cond. of A (d) sample cluster

(e) neg. collider (f) X vs. Y (g) cond. of A (h) sample cluster

Fig. 6. The effect of the conditioning of a collider attribute on the distribution of the data: (a,b,e,f) $X$ and $Y$ are originally independent from each other; (c,d,g,h) conditioning of the collider attribute $A$, however, creates clusters of data points, each cluster having correlation with a sign opposite of the overall sign of the collision path




(a) sample causal graph (b) scatter plot X vs. Z (c) conditioning of "A" (d) corr. preserved

(e) conditioning of "B" (f) corr. reversed

Fig. 7. (a,b) A sample causal graph and the corresponding preference data, (c,d) conditioning of "A" preserves the correlations among the preference variables, while (e,f) conditioning of "B" reverses the correlations among the preference variables

## 4 Causal Search for Skylines (CSS) with Selective De-Correlation

Let us reconsider our house hunting example and the corresponding causal graph visualized in Figure 1. In this example, our preference attributes are `Commute` and `Price`, and our preference

criteria are `min-min`. As we have discussed in the previous section, in this graph, we have two confounders `Distance_to_city_center` and `Scenic_view`, the former introducing a negative correlation between the preference attributes, whereas the latter introducing positive correlation between them. While the negative correlation between `Commute` and `Price` due to the "−" labeled direct causal edge between them cannot be eliminated, the additional correlations due to the existence of the confounders can potentially be eliminated by conditioning them:

- The negative correlation imposed on the preference attributes by the `Distance_to_city_center` confounder can be eliminated by conditioning this attribute. Our intuition is that, since negative correlations are mis-aligned with the `min-min` preference criteria for the `Commute` and `Price` skyline attributes, this should help reduce the number of dominance checks and improve the efficiency of the skyline search process.
- In contrast, however, conditioning of the `Scenic_view` attribute would eliminate a degree of positive correlation between our preference attributes. This clearly is not desirable when our preference criteria is `min-min` as the elimination of this positive correlation would effectively emphasize the negative correlation between the preference attributes due to the "−" labeled direct causal edge between them.

Therefore, while conditioning of the `Distance_to_city_center` confounder is desirable, conditioning of the `Scenic_view` attribute is not. Note that this indicates a potential approach to improve the efficiency of skyline discovery, when there are only two preference attributes: (1) identify the confounders; (2) identify if the statistical correlation implied by the confounder is positively or negatively aligned with the preference criteria; and (3) condition those confounders that have negative alignment with the preferences to selectively de-correlate the preference attributes. While this approach promises an effective way to reduce the number of dominance checks, there are several challenges:

- Firstly, we do not always have only two preference attributes. We therefore need an efficient and effective way to account for attributes that are negatively aligned with the preference attributes.
- Secondly, data attributes may have partial causal impact on the set of preference attributes and they may be positively-aligned with some of the criteria, while being negatively aligned with the rest. In Figure 7(e,f), we see a scenario where conditioning on one of the attributes reverses the positive correlation between the preference variables, potentially increasing the cost of the skyline search. We therefore need an effective way to account for such mis-alignments to promote the efficiency of the skyline discovery process, without negatively effecting the pruning of non-promising skyline candidates.

Therefore, we present a *causal search for skylines* (CSS) which, given a causal graph that describes the underlying causal structure of the data, identifies a subset, $\mathcal{Z}^-$, of the data attributes that can be used to de-correlate the negatively-aligned preference attributes (while preserving the correlations among the positively-aligned ones).

### 4.1 Problem Formulation

Extended version of this paper [37] includes a table with the key notations. Let $R(A)$ be a relation with the attribute set $A$ and let $P \subseteq A$ denote the set of preference/skyline attributes. Let us consider a skyline setting where all preferences are *max* or all of them are *min* (other scenarios can easily be converted to these with simple data transformations). Let $G_R$ (or $G$ for short) be an $\pm$-edge-labeled causal graph for the attributes in R. Let further $C_P$ denote a correlation matrix for the attributes in $P$.

Our goal is to find a subset of conditioning attributes, $\mathcal{Z} \subseteq A$, such that for all $a_i, a_j \in P$, $C_P[i,j] \leq C_P^*[i,j]$, where $C_P^*$ is the correlation matrix obtained after the conditioning on attributes in $\mathcal{Z}$. It is, however, especially important that

- if $C_P[i,j] \ll 0$, then we have $C_P^*[i,j] \gtrsim 0$ and
- if $C_P[i,j] > 0$, then we do not have $C_P^*[i,j] \ll C_P[i,j]$.

More specifically, when the correlation between two preference attributes, $i, j$, is significantly negatively aligned with the preference criteria ($C_P[i,j] << 0$), we would like the correlation between preference variates $i, j$ to be positive, but close to zero ($C_P[i,j] \gtrsim 0$). If, on the other hand, the correlation between a pair of preference attributes, $i, j$, is positively aligned with the preference criteria ($C_P[i,j] > 0$), then we do not want the correlation to have a significant drop (i.e., $C_P^*[i,j] \ll C_P[i,j]$) after we perform conditioning.

### 4.2 Baseline Algorithm - Algorithm #0: Data Driven Conditioning Set Selection

Before discussing algorithms that rely on causal graphs for identifying the conditioning set, we first present a purely data-driven approach to conditioning set selection.

(1) **Conditioning set enumeration.** Let $\mathcal{Z} \subseteq A$ be a subset of the attribute set of the given relation.
   (a) *Data conditioning.* Using a `GROUP BY` operation, the input data set, $D$, is grouped into multiple subsets, $D_1(\mathcal{Z}), \ldots, D_m(\mathcal{Z})$ (where $m$ is the number of unique combinations for the conditioning set of attributes).
   (b) *Pairwise correlation computation.* We consider all pairs of $a_i, a_j \in P$ for group $D_k(\mathcal{Z})$ to compute the corresponding correlation $C_{\mathcal{Z},P}[i,j,k]$ under the given conditioning set.
   (c) *Pairwise correlation gain computation.* We then compute the corresponding correlation gain for the given conditioning set, where $C_P[i,j]$ is the correlation without any conditioning.

$$C_{gain}(\mathcal{Z}) = C_{avg}(\mathcal{Z}) - \sum_{i \neq j} C_P[i,j],$$

   where

$$C_{avg}(\mathcal{Z}) = \frac{1}{k} \sum_{k} \sum_{i \neq j} C_{\mathcal{Z},P}[i,j,k].$$

(2) **Conditioning set selection.** Given all subsets of the attribute set, we select the subset, $\mathcal{Z}$, that provides the largest value of $C_{gain}(\mathcal{Z})$ as the conditioning set.

Once the above process (*Step 0*) is completed, we proceed with identifying the skylines for the input data set $D$ as follows:

*Step 1* using a `GROUP BY` operation, the input data set, $D$, is grouped into multiple subsets, $D_1, \ldots, D_m$ (where $m$ is the number of unique combinations for the conditioning set of attributes),
*Step 2* candidates skylines are identified for each subset separately,
*Step 3* these candidate skylines are merged into $M$, and
*Step 4* the final skyline set is identified by running a skyline algorithm on the merged candidate skyline data set, $M$.

The intuition behind the above process is that the conditioning set we seek should ideally eliminate the negative correlations among the preference variables, while preserving (or even boosting) positive correlations among them. As we empirically establish in Section 7.2.1, the above (baseline) algorithm may be effective in identifying conditioning subsets; yet, this baseline algorithm is also quite costly as in Step 0 we need to compute pairwise correlations among $|P|$ attributes for $2^{|A|}$ conditioning subsets, which can be achieved at $O(|P|^2 \cdot 2^{|A|} \cdot N)$ time, where $N$ is the size of the

input data. In the rest of this paper, we present algorithms to compute conditioning sets relying on causal graphs (when they are available), avoiding the cost of computing pairwise correlations for each potential conditioning subset.

## 5 Algorithm #1: Gain-based Negative Path Blocking

In this section, we describe an algorithm to address this issue by blocking the negative paths, while preserving the positive ones in the underlying causal graph. Below, we provide the outline of the algorithm and discuss each step in detail:

(1) **Path counting.** In the first step, we consider all pairs of $a_i, a_j \in P$ and for each such pair, we enumerate the set of paths from $a_i$ to $a_j$ using exhaustive BFS or DFS. Let
- $Paths_{i,j}$ denote the set of all paths between $a_i$ and $a_j$;
- $Open^{+}_{i,j}$ denote the set of non-blocked (i.e., collider free) paths on which the count of "-" labeled edges is an even number (this implies that the overall correlation implied by the path is positive), and
- $Open^{-}_{i,j}$ denote the set of non-blocked (i.e., collider free) paths on which the count of "-" labeled edges is an odd number (this implies that the correlation implied by the path is negative).

(2) **Conditioning set enumeration.** Let $\mathcal{Z} \subseteq A$ be a subset of the attribute set of the given relation.

(a) *Preference pair assessment.* For this subset, consider all pairs of $a_i, a_j \in P$

(i) *Identify new paths that provisionally open* between $a_i$ and $a_j$. This can happen if a path that was originally blocked through one or more colliders now gets un-blocked, because each collider on the path (or one of its descendants) gets conditioned. Note that, these paths are only provisionally open, because they may in fact be blocked by other conditioned mediator or fork nodes - we will not know this until they are further confirmed.
- $Prov^{+}_{i,j}(\mathcal{Z}) \subseteq Paths_{i,j}$ denotes the set of new paths that (provisionally) open between $a_i$ and $a_j$ by $\mathcal{Z}$, such that the overall correlation implied by the path is positive and
- $Prov^{-}_{i,j}(\mathcal{Z}) \subseteq Paths_{i,j}$ denotes the set of new paths that (provisionally) open between $a_i$ and $a_j$ by $\mathcal{Z}$, such that the overall correlation implied by the path is negative.

Note, however, that each un-blocked collider reverses the positive/negative relationship between the end-points. Therefore, whether a provisionally opened path is positive or negative depends on whether the sum of the number of "-" labeled edges and the number of colliders on the path is even or odd.

(ii) *Revise the sets of open paths with the provisional set.* Given the above, we can revise the sets of open paths between $a_i$ and $a_j$ considering the paths that provisionally open thanks to the un-blocked conditioners in $\mathcal{Z}$:
- $Open^{+}_{i,j}(\mathcal{Z}) = Open^{+}_{i,j} \cup Prov^{+}_{i,j}(\mathcal{Z})$, and
- $Open^{-}_{i,j}(\mathcal{Z}) = Open^{-}_{i,j} \cup Prov^{-}_{i,j}(\mathcal{Z})$,

(iii) *Identify the paths that get blocked by $\mathcal{Z}$.* Next, we consider the open paths between $a_i$ and $a_j$ and see if any of these get blocked by the conditioning of any mediators or forks:
- $Blocked^{+}_{i,j}(\mathcal{Z}) \subseteq Open^{+}_{i,j}(\mathcal{Z})$ denotes the set of positive paths that were (provisionally) open between $a_i$ and $a_j$ that are blocked by conditioning of $\mathcal{Z}$; and
- $Blocked^{-}_{i,j}(\mathcal{Z}) \subseteq Open^{-}_{i,j}(\mathcal{Z})$ denotes the set of negative paths that were (provisionally) open between $a_i$ and $a_j$ that are blocked by conditioning of $\mathcal{Z}$.

(iv) *Evaluate the positive and negative path impact of $\mathcal{Z}$,* in terms of the number of positive and negative paths between $a_i$ and $a_j$ that were open at the beginning versus the number of such paths open after the conditioning of $\mathcal{Z}$:

- $imp^{+}_{i,j}(\mathcal{Z}) = |Open^{+}_{i,j}(\mathcal{Z}) \backslash Blocked^{+}_{i,j}(\mathcal{Z})| - |Open^{+}_{i,j}|$,
- $imp^{-}_{i,j}(\mathcal{Z}) = |Open^{-}_{i,j}(\mathcal{Z}) \backslash Blocked^{-}_{i,j}(\mathcal{Z})| - |Open^{-}_{i,j}|$.

An impact value $< 0$ indicates a drop in the corresponding open paths, whereas a value $> 0$ indicates an increase.

(b) *Compute the overall gain provided by $\mathcal{Z}$.* The overall impact of the conditioning of $\mathcal{Z}$ is a function of the changes in the numbers of positive and negative open paths in the causal graph.

$$gain(\mathcal{Z}) = \sum_{a_i, a_j \in P} [imp^{+}_{i,j}(\mathcal{Z}) - imp^{-}_{i,j}(\mathcal{Z})]$$

(3) **Conditioning set selection.** For conditioning, we select $\mathcal{Z} \subseteq A$ that provides the largest value of $gain(\mathcal{Z})$.

Note that if we are working with graphs where edges are not labeled with $\pm$, we assume that all open paths are negative paths. Note also that, if we have more precise edge impact values associated with the edges, we can also accommodate them. More specifically, we revise the computation of the impact values, $imp^{+}_{i,j}(\mathcal{Z})$ and $imp^{-}_{i,j}(\mathcal{Z})$, in Step 2(a)iv of the above algorithm to take into account the causal weights ($cw$) of the paths:

- $\mathfrak{O}^{+}_{i,j} = \sum_{p \in Open^{+}_{i,j}} cw(p)$,
- $\mathfrak{O}^{-}_{i,j} = \sum_{p \in Open^{-}_{i,j}} cw(p)$,
- $\mathfrak{O}B^{+}_{i,j}(\mathcal{Z}) = \sum_{p \in Open^{+}_{i,j}(\mathcal{Z}) \backslash Blocked^{+}_{i,j}(\mathcal{Z})} cw(p)$,
- $\mathfrak{O}B^{-}_{i,j}(\mathcal{Z}) = \sum_{p \in Open^{-}_{i,j}(\mathcal{Z}) \backslash Blocked^{-}_{i,j}(\mathcal{Z})} cw(p)$,
- $imp^{+}_{i,j}(\mathcal{Z}) = \mathfrak{O}B^{+}_{i,j}(\mathcal{Z}) - \mathfrak{O}^{+}_{i,j}$, and
- $imp^{-}_{i,j}(\mathcal{Z}) = -(\mathfrak{O}B^{-}_{i,j}(\mathcal{Z}) - \mathfrak{O}^{-}_{i,j})$,

where, given a path $p$, the causal weight $cw(p)$ of the path is determined as the multiplication of the causal edges on the causal path; i.e, $cw(p) = \prod_{e \in p} cw(e)$.

This gives us the following *gain-based negative path blocking causal skylines (gnSky)* algorithm: Once the conditioning set is selected based on gain as described above (let us refer to the above process as *Step 0*), we proceed to identify the skylines for the input data set $D$ using the same 4 steps (Steps 1 through 4) in the baseline algorithm (Section 4.2). The outline of the overall algorithm is presented in Algorithm 1. Note that the effectiveness of the algorithm depends on, among other things, the cost of identifying candidate skylines for the subsets in *Step 2* (which is a function of the number and size of the groups and the correlation within each group) and the size of the merged candidate skyline set obtained in *Step 3*. Therefore, as we discuss in the next section, the grouping operation in *Step 1* is critical in the overall performance of causal skyline search.

## 6 Algorithm #2: Leaky Negative Path Blocking

While the algorithm presented in the previous section helps eliminate negative correlations by blocking paths on the causal graph that lead to negative correlations among the preference attributes, this algorithm suffers from several shortcomings as outlined below.

### 6.1 Issue #1: Leaky Blocking

As we have discussed in Section 3.2, the `GROUP BY` operation is analogous to conditioning and the algorithm presented in the previous subsection leverages this to block (for mediator and fork structures) or open (for colliders) paths on a causal graph to eliminate negative correlations among the preference attributes, while maintaining the positive correlations. The problem, of course, is that in practice grouping the input tuples by a single value combination of the conditioning attributes may be overly selective – it is possible that, the `GROUP BY` operation returns a single

**Algorithm 1** Skylines with Gain-based Negative Path Blocking (*gnSky*)

**Input** $R(A)$, $P \subseteq A$, $D$
**Output** Skyline set $S$, on preference attributes, $P$

**procedure** GNSKY($R(A)$; $P$, $D$)

▷ *Step 0:* Gain-based Negative Path Blocking

$C_P \leftarrow correlation\_matrix(P, D)$
**for** $a_i, a_j \in P$ **do**
$(Paths_{i,j}; Open^{+}_{i,j}; Open^{-}_{i,j}) \leftarrow paths(G, a_i, a_j)$
**end for**
**for** $\mathcal{Z} \subseteq A \backslash P$ **do**
$I(\mathcal{Z}) \leftarrow \emptyset$
**for** $a_i, a_j \in P$ **do**
$(Prov^{+}_{i,j}; Prov^{-}_{i,j}) \leftarrow prov\_open(G, \mathcal{Z}, a_i, a_j)$

▷ Below, $*$ stands for + or -

$Open^{*}_{i,j}(\mathcal{Z}) \leftarrow Open_{i,j} \cup Prov^{*}_{i,j}(\mathcal{Z})$
$Blocked^{*}_{i,j}(\mathcal{Z}) \leftarrow get\_blocked(Open^{*}_{i,j}(\mathcal{Z}), \mathcal{Z})$
$imp^{*}_{i,j}(\mathcal{Z}) \leftarrow compute\_impact(Open^{*}_{i,j}(\mathcal{Z}),$
$Blocked^{*}_{i,j}(\mathcal{Z}), Open^{+}_{i,j})$
$I(\mathcal{Z}) \leftarrow I(\mathcal{Z}) \cup imp^{*}_{i,j}(\mathcal{Z})$
**end for**
$gain(\mathcal{Z}) \leftarrow compute\_gain(I(\mathcal{Z}), C_P)$
**end for**
$\mathcal{Z}^{-} \leftarrow argmax_{\mathcal{Z}}\ gain(\mathcal{Z})$

▷ *Step 1:* Conditioning step

$D_1, \ldots, D_m \leftarrow \texttt{GROUP\_BY}_{\mathcal{Z}^-}(D)$

▷ *Step 2:* Candidate skylines per conditioning value

$\forall_i S_i \leftarrow skyline(P, D_i)$

▷ *Step 3:* Merging of candidate skylines

$C \leftarrow \bigcup_i S_i$

▷ *Step 4:* Computing of the final skylines

$S \leftarrow skyline(P, C)$
**end procedure**

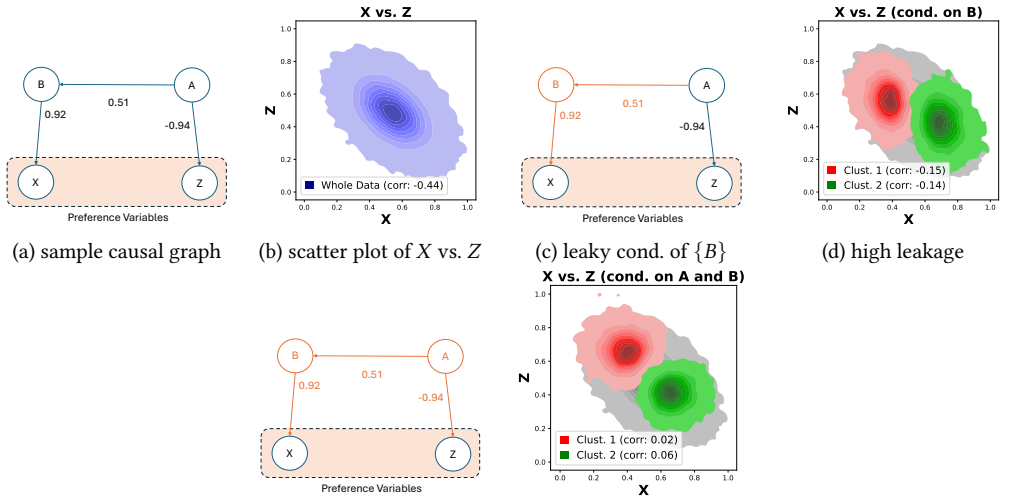
tuple per each unique value combination of the conditioning set of attributes; in such a case, we would have $M = D$ and the cost of the *Step 4* of the algorithm above would be equal to the cost of running the skyline on the original data, rendering the total cost of *Steps 1 through 3* a redundant overhead. In order to avoid this outcome, we can revise the *Step 1* of the algorithm above as follows:

*Step 1** given the target number, $m$, of groups, use a clustering algorithm to group the input data set, $D$, into multiple subsets, $D_1, \ldots, D_m$.

On the positive side, this revision enables us to control the number of groups that are created in *Step 1*, enabling us to deal with the situations where the combined selectivity of the conditioning set of attributes is high. On the negative side, however, the replacement of *single-value grouping* in *Step 1* with *similar value grouping* in *Step 1** potentially reduces the effect of conditioning and path blocking. We next illustrate this problem, which we refer to as *leaky conditioning* or *leaky blocking*, with an example:

Example 4 (Leaky Conditioning/Blocking). *This is visualized in Figure 8, here we see a sample causal graph, which has two alternative conditioning sets, $\{B\}$ and $\{A, B\}$ with the same gain. In this example, the data has been grouped into 10 subsets using K-means clustering and in Figures 8(d) and (f), we are showing two of the resulting clusters for each of the conditioning set scenarios.*

*As we see in the figure, when we use the groups formed by the conditioning set, $\{B\}$, the correlation between preference attributes $X$ and $Z$ is brought down from $-0.44$ for the original data set to $\sim -0.15$ for the resulting clusters; while this conditioning set blocks the negative path between $X$ and $Z$, the blockage is imperfect or* leaky.




(a) sample causal graph (b) scatter plot of $X$ vs. $Z$ (c) leaky cond. of $\{B\}$ (d) high leakage

(e) leaky cond. of $\{A, B\}$ (f) low leakage

Fig. 8. (a) A sample causal graph, (b) scatter plot of two attributes $X$ and $Z$, (c,d) two sample clusters obtained through high leakage conditioning set $\{B\}$, and (e,f) two sample clusters obtained through low leakage conditioning set $\{A, B\}$ – in the causal graphs, orange colored nodes and edges indicate leaky conditioning/blocking; as we see in this figure, the larger the number of conditioned attributes on a path, the lesser the information leak on that path

*In contrast, when we use the conditioning set, $\{A, B\}$, the correlation between preference attributes $X$ and $Z$ is brought down from $-0.44$ to $\sim 0.04$ for the resulting clusters; in other words, this conditioning set blocks the negative path between $X$ and $Z$ very effectively, almost completely eliminating the negative correlation between the preference attributes in the resulting clusters.*

Therefore, when we are relying on cluster-based *similar-value grouping* in *Step 1* of the algorithm, rather than *single-value grouping*, the value of *gain* defined in Section 5 is not sufficient to differentiate among the path blocking alternatives. Instead, as we discuss in the next section, we need to expand the definition of *gain* in a way that accounts also for the leakages of the conditioning attributes.

### 6.2 Issue #2: Noisy Causal Information Passing

Imperfect causal information passing is not limited to only the blocked nodes. Even open nodes may be imperfect conduits of causality. A common causal model [44, 61], for example, represents the causal relationship $X \rightarrow Y$ as $Y = c_{xy}X + \epsilon_y$, where $c_{xy}$ denotes the causal impact of $X$ on $Y$, whereas $\epsilon_y$, the noise that acts on attribute $Y$ along this causal edge. When the noise, $\epsilon_y \sim 0$, the causal impact of $X$ and $Y$ is clear. When $|\epsilon_y| \gg 0$, the causal relationship between $X$ and $Y$ will be noisy and the impact of $X$ on $Y$ may be hidden, even for large values of $c_{xy}$.

### 6.3 Leaky Negative Path Blocking

We address leaky blocking and noisy information passing problems by blocking the negative paths, while preserving the positive ones in the underlying causal graph, in a way that takes into account leaky conditioning due to cluster-based *similar-value grouping* in *Step 1* of the algorithm. The algorithm shares the same outline as the *gain-based negative path blocking skyline (gnSky)* algorithm in Section 5, but instead of treating each node in the graph as open or closed, it considers a degree

of information passing, $\pi$, on a conditioned attribute node: A conventionally open/non-blocked node passes complete information ($\pi = 1$), whereas a conventionally (i.e., *single-value grouping* based) blocked node passes no information at all ($\pi = 0$). In contrast, a *noisy open* node passes $\pi = \lambda_o$ information, whereas a *similar-value grouping* based blocked node passes $\pi = \lambda_b$ information. Here $0 < \lambda_b \ll \lambda_o < 1$ are the corresponding degrees of information passage for imperfectly open and blocked nodes, respectively. The outline of the *leaky negative path blocking skyline (lnSky)* algorithm is as follows:

(1) **Path counting.** In the first step, we consider all pairs of $a_i, a_j \in P$ and for each such pair, we enumerate the set of paths from $a_i$ to $a_j$ using exhaustive BFS or DFS. Let $Paths_{i,j}$, $Open^+_{i,j}$, and $Open^-_{i,j}$ be defined as before.

(2) **Conditioning set enumeration.** Let $\mathcal{Z} \subseteq A$ be a subset of the attribute set of the given relation.

(a) *Preference pair assessment.* For this subset, consider all pairs of $a_i, a_j \in P$

(i) *Identify new paths that provisionally open* between $a_i$ and $a_j$. As before, this can happen if a path that was originally blocked through one or more colliders now gets un-blocked, because each collider on the path (or one of its descendants) gets conditioned. Let $Prov^+_{i,j}(\mathcal{Z})$ and $Prov^-_{i,j}(\mathcal{Z})$ be defined as before.
Note that, in this case, how much information passes on a path $p \in Prov^+_{i,j}(\mathcal{Z}) \cup Prov^-_{i,j}(\mathcal{Z})$ depends on how much information passes on the conditioned colliders.

(ii) *Revise the sets of open paths with the provisional set.* Given the above, we can revise the sets of open paths between $a_i$ and $a_j$ considering the paths that provisionally open thanks to the un-blocked conditioners in $\mathcal{Z}$:
- $Open^+_{i,j}(\mathcal{Z}) = Open^+_{i,j} \cup Prov^+_{i,j}(\mathcal{Z})$, and
- $Open^-_{i,j}(\mathcal{Z}) = Open^-_{i,j} \cup Prov^-_{i,j}(\mathcal{Z})$,

(iii) *Identify the paths that get blocked by $\mathcal{Z}$.* Next, we consider the open paths between $a_i$ and $a_j$ and see if any of these get blocked by the conditioning of any mediators or forks:
- $Blocked^+_{i,j}(\mathcal{Z}) \subseteq Open^+_{i,j}(\mathcal{Z})$ denotes the set of positive paths that were (provisionally) open between $a_i$ and $a_j$ that are blocked by conditioning of $\mathcal{Z}$; and
- $Blocked^-_{i,j}(\mathcal{Z}) \subseteq Open^-_{i,j}(\mathcal{Z})$ denotes the set of negative paths that were (provisionally) open between $a_i$ and $a_j$ that are blocked by conditioning of $\mathcal{Z}$.

In this case, conditioning does not entirely eliminate information passage on the paths, but reduces it due to leakages.

(iv) Compute noise and leakage adjusted causal weights of the paths. Given an open or closed path $p$, let the causal weight $cw(p)$ of the path be determined as the multiplication of the weights of the causal edges on the causal path; i.e, $cw(p) = \prod_{e \in p} cw(e)$. Let $o_p(\mathcal{Z})$ be the number of open nodes on $p$ and let $b_p(\mathcal{Z})$ be the number of blocked nodes on $p$, given $\mathcal{Z}$. The noise and leakage adjusted causal weight of $p$ is computed as

$$\hat{cw}(p) = cw(p) \cdot \lambda_o^{o_p(\mathcal{Z})} \cdot \lambda_b^{b_p(\mathcal{Z})}.$$

(v) *Evaluate the leaky impact of $\mathcal{Z}$ on positive and negative paths* between $a_i$ and $a_j$, in terms of the amount of information flow on them:
- $\mathfrak{O}^+_{i,j} = \sum_{p \in Open^+_{i,j}} \hat{cw}(p)$,
- $\mathfrak{O}^-_{i,j} = \sum_{p \in Open^-_{i,j}} \hat{cw}(p)$,
- $\mathfrak{OB}^+_{i,j}(\mathcal{Z}) = \sum_{p \in Open^+_{i,j}(\mathcal{Z}) \cup Blocked^+_{i,j}(\mathcal{Z})} \hat{cw}(p)$,
- $\mathfrak{OB}^-_{i,j}(\mathcal{Z}) = \sum_{p \in Open^-_{i,j}(\mathcal{Z}) \cup Blocked^-_{i,j}(\mathcal{Z})} \hat{cw}(p)$,
- $l\_imp^+_{i,j}(\mathcal{Z}) = \mathfrak{OB}^+_{i,j}(\mathcal{Z}) - \mathfrak{O}^+_{i,j}$, and

**Algorithm 2** Skylines with Leaky Negative Path Blocking (*lnSky*)

**Input** $R(A)$; $P \subseteq A$, $D$,$m$
**Output** Skyline set $S$, on preference attributes, $P$

**procedure** LNSKY($R(A)$, $P$, $D$, $m$)

▷ *Step 0:* Leaky Gain-based Negative Path Blocking

$C_P \leftarrow correlation\_matrix(P, D)$
**for** $a_i, a_j \in P$ **do**
$(Paths_{i,j}; Open^+_{i,j}; Open^-_{i,j}) \leftarrow paths(G, a_i, a_j)$
**end for**
**for** $\mathcal{Z} \subseteq A \backslash P$ **do**
$I(\mathcal{Z}) \leftarrow \emptyset$
**for** $a_i, a_j \in P$ **do**
$(Prov^+_{i,j}; Prov^-_{i,j}) \leftarrow l\_prov\_open(G, \mathcal{Z}, a_i, a_j)$
$Open^*_{i,j}(\mathcal{Z}) \leftarrow Open_{i,j} \cup Prov^*_{i,j}(\mathcal{Z})$
$Blocked^*_{i,j}(\mathcal{Z}) \leftarrow get\_l\_blocked(\mathcal{Z}, Open^*_{i,j}(\mathcal{Z}))$
$l\_imp^*_{i,j}(\mathcal{Z}) \leftarrow compute\_l\_impact(Open^*_{i,j}(\mathcal{Z}), Blocked^*_{i,j}(\mathcal{Z}), Open^+_{i,j})$
$I(\mathcal{Z}) \leftarrow I(\mathcal{Z}) \cup l\_imp^*_{i,j}(\mathcal{Z})$
**end for**
$l\_gain(\mathcal{Z}) \leftarrow compute\_l\_gain(I(\mathcal{Z}), C_P)$
**end for**
$\mathcal{Z}^- \leftarrow argmax_{\mathcal{Z}}\ gain(\mathcal{Z})$

▷ *Step 1:* Conditioning step

$D_1, \ldots, D_m \leftarrow \texttt{cluster}(D, \mathcal{Z}^-, m)$

▷ *Step 2:* Candidate skylines per conditioning value

$\forall_i S_i \leftarrow skyline(P, D_i)$

▷ *Step 3:* Merging of candidate skylines

$C \leftarrow \bigcup_i S_i$

▷ *Step 4:* Computing of the final skylines

$S \leftarrow skyline(P, C)$
**end procedure**

- $l\_imp^-_{i,j}(\mathcal{Z}) = -(\mathfrak{DB}^-_{i,j}(\mathcal{Z}) - \mathfrak{D}^-_{i,j})$.

  Note that, a (leaky) impact value < 0 indicates a drop of information flow on the corresponding open paths, whereas a value > 0 indicates an increase.

(b) *Compute the overall leaky gain provided by $\mathcal{Z}$.* The overall (leaky) impact of the conditioning of $\mathcal{Z}$ is a function of the changes of the amount of positive and negative information flow in the causal graph.

$$l\_gain(\mathcal{Z}) = \sum_{a_i, a_j \in P} [l\_imp^+_{i,j}(\mathcal{Z}) - l\_imp^-_{i,j}(\mathcal{Z})]$$

(3) **Conditioning set selection.** Given all subsets of the attribute set, we select the one, $\mathcal{Z}$, that provides the largest value of the leaky gain, $l\_gain(\mathcal{Z})$.

Note that, as before, if we are working with graphs where edges are not labeled with ±, we can assume that all open paths are negative paths. Note also that, if we have edge impact values associated with the edges, we can also accommodate them (multiplicatively).

This gives us the following *leaky negative path blocking causal skyline (lnSky)* algorithm: Once the conditioning set is selected based on the leaky gain as described above (we refer to the above process as *Step 0* of the *lnSky* algorithm), we proceed to identify the skylines for the input data set $D$ using the same 4 steps in the baseline algorithm (Section 4.2) and the gain-based negative path blocking skyline (*gnSky*) algorithm.

The outline of the algorithm is presented in Algorithm 2.

### 6.4 Complexity Analysis

Both *gnSky* and *lnSky* algorithms consist of 5 steps (Steps 0 through 4). We analyze the complexity of each step separately:

- *Step 0 - Gain computation and conditioning set selection:* The cost of this step depends on the number of paths considered which is a function of the number of preference variables and the number of paths in the causal graph between pairs of these variables. More specifically, in the worst case, the algorithm would enumerate $((|P| \cdot |P| - 1)/2) \cdot 2^{|A|}$ number of paths. Each of these paths would then be evaluated for each possible conditioning set. Given the above, the overall worst case cost of the algorithm is $O(|P|^2 \cdot 2^{2|A|})$ or equally $O(|P|^2 \cdot 4^{|A|})$.
- *Step 1 - Data conditioning:* GROUP BY would require $O(N)$ time for hashing if the values are categorical or $O(Nlog(N))$ time if the values are ordinal and we need to sort the data for similarity based grouping. The cost of the clustering based implementation would depend on $m$, $N$, and the clustering algorithm used.
- *Step 2 - Candidate skylines:* Assuming that the data is uniformly divided into $m$ subsets, the cost of this step is $m \cdot cost\_skyline(N/m)$; i.e., $O(m \cdot (N/m)^2)$. Note that, in the worst case (where one of the subsets is almost as large as the original data), of course, the cost would be $\sim cost\_skyline(N)$, which is, generally, $O(N^2)$.
- *Step 3 - Merging of candidate skylines:* Let $s_i$ be the number of skylines returned for the $i^{th}$ subset. The cost of this step is $O(\sum_i s_i)$.
- *Step 4 - Obtaining final skylines:* The cost of this step is $cost\_skyline(\sum_i s_i)$, which is $O((\sum_i s_i)^2)$.

Note that the cost of Step 0 depends on the size and complexity of the causal graph and, while the worst case size is exponential in the number of attributes, in practice, the number of paths that need to be enumerated is much smaller and, as we see in Section 7, the cost of Step 0, (which is independent of the data size, $N$) is generally negligible relative to the cost of skyline computations, which is $O(N^2)$. Moreover, once the impact values for a causal graph is computed, the gain values can be re-used without any recomputation for a new data set with the same causal structure. Consequently, the overall cost of Steps 1 through 4 depends on (a) how uniformly the data conditioning step splits the data into $m$ subsets and (b) how well the conditioning reduces the number of skyline objects (through elimination of negative correlations among the preference attributes).

### 6.5 Computation of the Scaling Parameters

The Algorithm #2 (*lnSky*) alleviates the shortcomings of Algorithm #1 (*gnSky*) by accounting for the leaky blocking and noisy causal information passage by leveraging two scaling factors, $\lambda_b$ and $\lambda_o$, respectively. We therefore need a mechanism to quantify these degrees of leakage and noise as a function of the structure of the causal graph. Let $p$ be a path between two preference variables $X$ and $Y$ and $\mathcal{Z}$ be the set of conditioning attributes; to compute the conditioning gain for the path, $p$, we need to compute two terms:

- $corr_p(X, Y)$; i.e., the correlation between $X$ and $Y$, implied by the causal path $p$, before the conditioning step, and
- $corr_{p,\mathcal{Z}}(X, Y)$; i.e., the correlation between $X$ and $Y$, implied by the causal path $p$, after the conditioning of attributes in $\mathcal{Z}$.

Given these, the conditioning gain for the path given the conditioning set $\mathcal{Z}$ can be computed as $c\_gain(p, \mathcal{Z}) = corr_{p,\mathcal{Z}}(X, Y) - corr_p(X, Y)$. While outside the scope of this paper, we discuss how to compute these terms, under the assumption that the sets of variables have multi-variate Gaussian distributions, in the extended version of this paper [37].

Table 1. Experiment parameters (bold indicate default values)

| Parameter | Values |
|---|---|
| Number of data elements | 50K, 100K, **200K**, 400K |
| Number of preference attributes | 2, 3, 4 |
| Number of de-correlation clusters | 2 - 9, **10**, 11 - 20 |
| $\langle \lambda_o, \lambda_b \rangle$ pairs | $\langle 1.0, 0.0 \rangle$, $\langle 0.9, 0.1 \rangle$, $\langle 0.8, 0.2 \rangle$, $\langle 0.7, 0.3 \rangle$, $\langle \mathbf{0.6}, \mathbf{0.4} \rangle$ |

Table 2. Real Causal Graphs

| Name | # Variates | # Tuples |
|---|---|---|
| Abalone [41] | 8 | 4177 |
| Adult [9] | 14 | 48842 |
| Auto MPG [47] | 7 | 398 |
| Concrete Compressive Strength [63] | 9 | 1030 |
| Dry Bean [2] | 16 | 13611 |
| Hong Kong Weather [23] | 10 | 18262 |
| Individual Household Power Consumption [22] | 9 | 2075259 |
| Seoul Bike Demand [3] | 13 | 8760 |
| Wine Quality (White) [19] | 11 | 4898 |

## 7 Experiments and Analysis

Next, we present experiment results evaluating the effectiveness of the proposed causally-informed skyline (CSS) search[4].

### 7.1 Experiment Setup

*7.1.1 Causal Graphs and Data Sets.* We consider data sets with both real and synthetic causal graphs for evaluating CSS. For each causal graph, we consider different preference sets, with varying numbers of preference attributes.

- *Real Causal Graphs:* Table 2 presents the real causal graphs considered in these experiments. Since in this section we also want to investigate the impact of the data size, we varied the data set size for these graphs by leveraging data augmentation using perturbed oversampling with Gaussian noise [10, 14].
- *Synthetic Causal Graphs:* In addition to these real causal graphs, we also considered a number of synthetic causal graphs, with varying complexities (number of nodes, edge densities, and different causal primitives). The causal edge weights for these causal graphs were set to +1.0 or +0.5 for strongly and weakly positive causal edges, respectively. Similarly, −1.0 and −0.5 were used for strongly and weakly negative causal edges, respectively. The data for these causal graphs were also generated synthetically.
- *Inferred Causal Graphs:* We also experimented with causal graphs inferred from the data, rather than being provided as input by the user. While there are many causal discovery algorithms [17, 51, 52], in the experiments, we used the NOTEARS [67] causal inference algorithm, which assumes a simple linear structural equation model, to discover the causal graph and the edge weights.

*7.1.2 Skyline Algorithms.* As competitors to CSS, we consider the following skyline algorithms:

[4]The source code, along with all causal graphs used in our experiments and their corresponding preference sets and dominance criteria are available at https://github.com/EmitLab/Causal-Search-for-Skylines.

- *Block Nested Loop Skylines (BNL) [13]* BNL scans the dataset once, keeping a window of currently non-dominated tuples – no specific ordering of tuples is assumed. Every new tuple is compared against this window: the tuple is discarded if it is dominated or inserted into the window, evicting any tuples it dominates.
- *Sort-First Skylines (SFS) [18]* SFS sorts the data on a monotone ranking criterion ($\langle sum \rangle$ in the experiments), then performs a scan in that order, maintaining a window of candidates[5]. The presort reduces the number of comparisons.
- *Sort-and-Limit Skyline (SaLSa) [8]* SaLSa builds on SFS by introducing "lazy" sorting: it interleaves (partial) sorting with skyline testing, in such a way that any removed tuple cannot re-enter the skyline. For SaLSa, we have used the $\langle max, sum \rangle$ monotonic function as recommended by the authors.
- *Branch and Bound Skyline (BBS) [42]* BBS operates on an R-tree index, exploring nodes in best-first order using a priority queue pruning whole sub-trees if their bounding rectangles are dominated by skyline tuples already found.
- *Divide and Conquer Skylines (D&C) [13, 29]* D&C partitions the space along the median values into sub-regions, computes the skyline of each partition independently, and merges the partial skylines discarding any tuples dominated by another from any partition. The divide-and-conquer strategy of the D&C algorithm can be seen as pure de-correlation of the preference attributes.

*7.1.3 CSS Implementation.* For each of the above skyline algorithms, we consider the causal version as follows: (a) given a causal graph and the set of preference attributes, we compute the expected gains for each possible conditioning set; (b) given the target number of de-correlation clusters, we apply Euclidean-distance based $k$-means on the conditioning attributes, normalized to zero mean and unit variance; (c) we compute the causal skylines for each cluster[6], and (d) obtained skylines from the clusters are unioned and the skyline is searched among these tuples.

Note that the leaky negative path blocking (*lnSky*) algorithm presented in Section 6 uses two parameters $\lambda_b \ll \lambda_o$, denoting the degrees of information passage on imperfectly open and blocked nodes. Table 1 lists the values considered in the experiments.

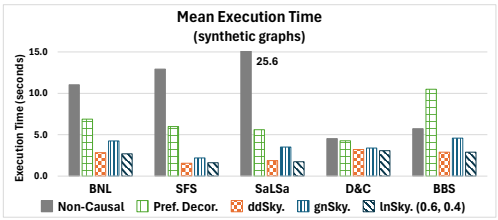
In our implementation, all algorithms use the main memory to compute the skylines, and no temporary file was ever needed (i.e., BNL, SFS, and SaLSa always completed in a single pass). The skyline algorithms, their corresponding causal equivalents, and K-means clustering have been implemented in Python 3.12.7 with Numpy 2.1.2. The experiments were executed using Chameleon [27] on AMD EPYC 7763 64-Core Processor, 256GB RAM, and 500GB SSD.

*7.1.4 Baselines.* In addition, for each of the above skyline algorithms, we consider the following baselines:

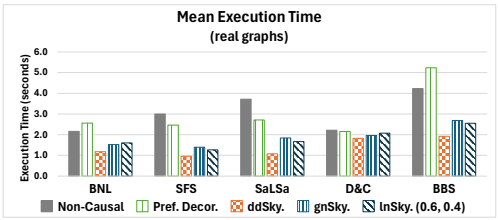
- *Non-causal:* This is the original, non-causal algorithm.
- *Data-driven (ddSky):* Here, we utilize the data-driven strategy to compute the expected gains for each possible conditioning set and subsequently follow the clustering based approach for computing skylines as outlined in the leaky negative path blocking skyline (*lnSky*) algorithm.
- *Preference de-correlation:* In this baseline, we consider a non-selective, preference attribute based de-correlation strategy: we apply K-means clustering directly on the set of preference attributes, thereby de-correlating the data without paying attention to whether this would eliminate positive or negative correlations.

[5]Although LESS [21] improves upon SFS, we opted for SFS instead due to its lesser complexity and fewer hyper-parameters. In fact, the primary optimization in LESS, the elimination filter (EF), is applied during the external sorting stage, which does not conflict with our proposal.

[6]While both D&C and CSS allow for parallelization, in this section we assume sequential executions of these algorithms.

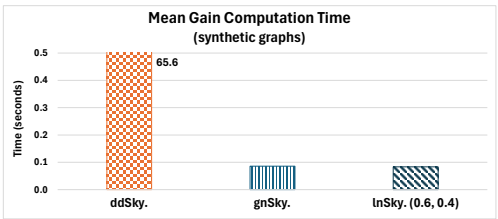



(a) Synthetic causal graphs

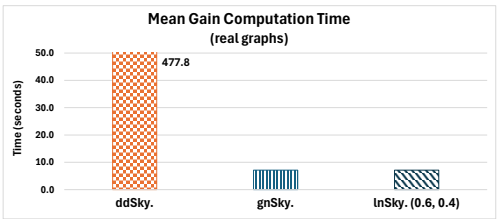



(b) Real-world causal graphs

Fig. 9. Cost of skylines with (a) synthetic and (b) real-world causal graphs (the lower, the better) – 200K data

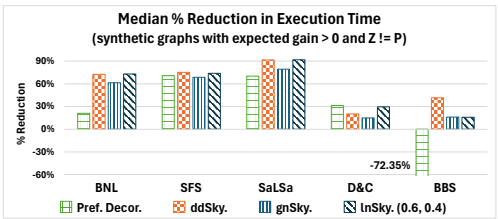



(a) Synthetic causal graphs

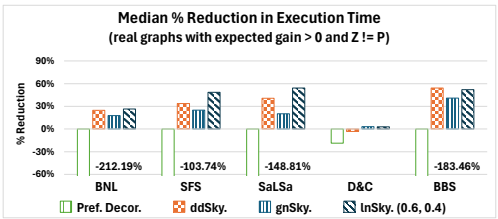



(b) Real-world causal graphs

Fig. 10. Cost of the expected gain computation step (the lower, the better) – 200K data




(a) Synthetic causal graphs



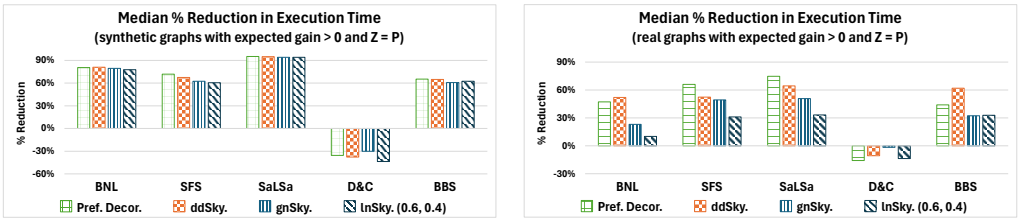

(b) Real-world causal graphs

Fig. 11. Median cost reduction (the higher, the better); for these experiments, expected gain is positive and larger than the gain predicted for the preference attribute set (i.e., *we expect to see benefits over both vanilla skylines and preference attribute de-correlation*) – 200K data

*7.1.5 Evaluation Criteria.* As evaluation criteria, we consider both (a) *the number of dominance checks* and (b) *execution time* of the skyline algorithm (wall clock), which also includes the time to de-correlate the data based on the selected de-correlation attribute set. We separately report the time to compute the de-correlation set. Since the dominance checks show similar patterns to execution times, we report them in the extended version of the paper [37].

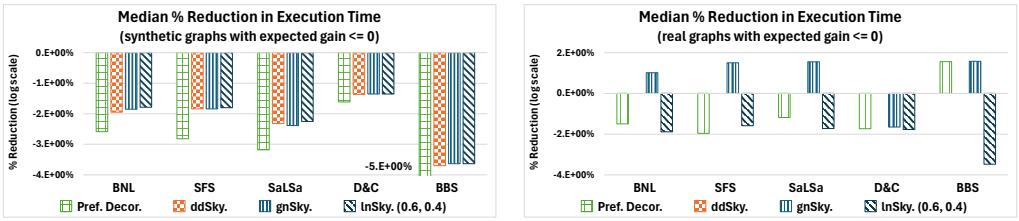
## 7.2 Experiment Results

In this section, we present and interpret the experiment results on both synthetic and real-world causal graphs. Importantly, since the divide-and-conquer (D&C) algorithm can be seen as the pure de-correlation of the preference attributes, we distinguish between scenarios where there is a conditioning attribute set predicted to perform better than conditioning on the preference attribute



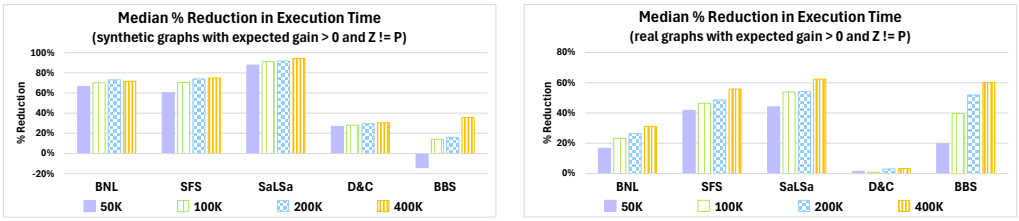

(a) Synthetic causal graphs (b) Real-world causal graphs

Fig. 12. Median cost reduction (the higher, the better); for these experiments, expected gain is positive and equal to the gain predicted for the preference attribute set (i.e., we expect to see benefits over vanilla skylines, but we do not expect to beat preference attribute de-correlation) – 200K data




(a) Synthetic causal graphs (b) Real-world causal graphs

Fig. 13. Median cost reduction (the higher, the better); for these experiments, expected gain is negative (i.e., *we do not expect to see benefits from de-correlation*) – 200K data



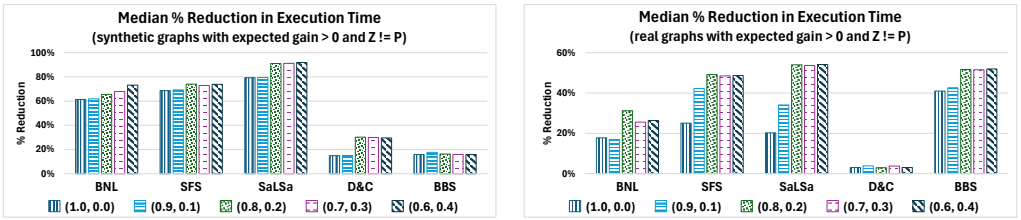

(a) Synthetic causal graphs (b) Real-world causal graphs

Fig. 14. Median cost reduction (the higher, the better) for different numbers of tuples when using *lnSky(0.6, 0.4)*; for these experiments, the expected gain is positive and larger than the gain predicted for the preference attribute set (i.e., *we expect to see benefits over both vanilla skylines and preference attribute de-correlation*)

set and the cases where the causal analysis predicts that the preference attribute set itself is the best conditioning attribute set.

### 7.2.1 *Effectiveness of Causal Search for Skylines (CSS).*

**Overview.** The way CSS would be used in practice is to leverage the causal version of the algorithm if there exists a conditioning attribute set with positive predicted gain and revert to the vanilla version otherwise. The benefits of this approach is visualized in Figure 9: As we see in the figure, assuming that the expected gains have already been computed, both *ddSky* and *lnSky* substantially outperform the vanilla versions, indicating that this adaptive strategy effectively




(a) Synthetic causal graphs (b) Real-world causal graphs

Fig. 15. Median cost reduction (the higher, the better) for $\langle \lambda_o, \lambda_b \rangle$ pairs; for these experiments, expected gain is positive and larger than the gain predicted for the preference attribute set (i.e., *we expect to see benefits over both vanilla skylines and preference attribute de-correlation*) – 200K data

leverages CSS when it is predicted to be advantageous, while defaulting to the vanilla approach in scenarios where CSS is not expected to be ideal.

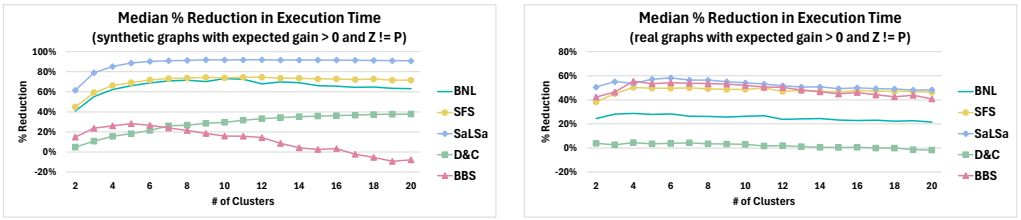
Interestingly, when the *lnSky* algorithm is considered, SFS and SaLSa become the fastest algorithms due to the significant reduction of dominance checks from which they benefit, even beating more recent algorithms, such as D&C and BBS.

**Cost of the Expected Gain Computation Step.** Figure 10 shows the time required to identify the best conditioning attribute set based on different strategies (Step 0). As we expected (see Section 4.2), the gain computation phase of *ddSky* is prohibitively expensive. This prevents it from being applicable in practice, even though it can provide significant gains in terms of execution time as we have already seen in Figure 9. Importantly, the cost of gain computation for *lnSky* (which is independent of the data size, $N$) is negligible relative to the cost of skyline computations, making *lnSky* to be the more practical de-correlation strategy.

**Effectiveness of the Gain Prediction.** In Figure 9, we have seen that gain prediction can be used to decide whether and how to de-correlate the data to obtain significant reductions in the cost of skyline search. Figure 11 through 13 deep dives in the investigation of the effectiveness of the gain prediction by focusing on the cases where (a) the expected gain is positive and better than that of the preference attributes, (b) gain is positive but not better than that of the preference attributes, and (c) expected gain is negative:

**(a)** Figure 11 presents the results for scenarios in which there exists a conditioning attribute set with a positive expected gain, *predicted to outperform conditioning on the preference attribute set.* As we see here, in these scenarios, both *ddSky* and *lnSky* significantly outperform vanilla (non-causal) baseline algorithms as well as preference-attribute set based de-correlation. Interestingly, in the case of real-world causal graphs, the preference-attribute based de-correlation performs even worse than the vanilla skyline algorithms, while the proposed data- and causality-driven extensions provide significant gains.

**(b)** Figure 12 shows the results for 200K data samples in scenarios where a positive expected gain exists, but the best conditioning attribute set is *predicted to coincide with the preference attribute set itself.* As expected, in the synthetic case, all strategies perform similarly with the preference-attribute based de-correlation or *ddSky*. In the real data case, we see that *lnSky* has a somewhat lower reduction in execution time, indicating that the causal gain prediction does not perfectly identify these scenarios.




(a) Synthetic causal graphs (b) Real-world causal graphs

Fig. 16. Median cost reduction (the higher, the better) for different numbers of de-correlation clusters when using *lnSky(0.6, 0.4)*; for these experiments, the expected gain is positive and larger than the gain predicted for the preference attribute set (i.e., *we expect to see benefits over both vanilla skylines and preference attribute de-correlation*) – 200K data

**(c)** Figure 13 shows the results for 200K data samples in scenarios where no conditioning attribute set with positive expected gain exists[7]. As expected, conditioning on an attribute set with negative expected gain significantly hurts performance. Interestingly, *gnSky* shows improvements in certain cases with real causal graphs. This likely occurs because *gnSky* misclassifies conditioning attribute sets: it sometimes assigns negative gain to sets that actually have positive gain. Conversely, this also explains why *gnSky* performs worse than *lnSky* in cases with true positive gain, as it incorrectly predicts negative conditioning attribute sets to yield positive gain. In contrast, if *lnSky* predicts that a conditioning attribute set has negative gain, it should be avoided. In such cases, the non-causal version is preferable.

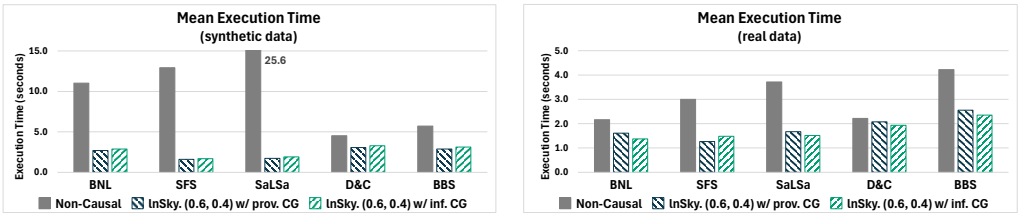
*7.2.2 Impact of the Number of Tuples.* As we see in Figure 14, for both synthetic and real-world graphs, performance gains improve as the size of the data gets larger, indicating that the proposed *lnSky* de-correlation strategy enables the skyline algorithms to scale better with the data size.

*7.2.3 Effect of the lnSky Parameters* $\langle \lambda_o, \lambda_b \rangle$. Figure 15 shows the effect of leakage parameters on *lnSky* search time. For synthetic graphs, *lnSky(0.6, 0.4)* performs the best. For real-world graphs, the best option is *lnSky(0.8, 0.2)*. In the rest of this section, we use the leakage parameter pair $\langle 0.6, 0.4 \rangle$ for both synthetic and real-world graphs for overall experimental setup consistency, even though that pair is not the best option for real-world graphs.

*7.2.4 Effect of Number of De-correlation Clusters.* As outlined in Section 4, the proposed CSS approach relies on K-means clusters of (suitably normalized) data to achieve selective de-correlation. Figure 16 illustrates how the number of de-correlation clusters affects the skyline search efficiency. As we see in the figure, a small number of clusters (on the order of 10-15 clusters for 200K data) is sufficient to obtain the necessary de-correlation – a larger number of clusters either do not provide additional benefits or (in the case of causal BNL) actually starts hurting the performance due to the increasing merge overhead.

*7.2.5 Impact of Inferred Causal Graphs.* In the preceding experiments, we assumed that both the causal graph structure and the corresponding edge strengths were user provided. In practice, however, such causal knowledge may not be available. In such scenarios causal inference and causal effect estimation techniques can help infer the underlying causal graph and causal strengths. In this section, instead of using the user provided graphs, we use the causal graphs inferred by

[7]Note that for real causal graphs, we did not observe any scenario where *ddSky* finds a conditioning attribute set with negative gain.




(a) causal graphs inferred from synthetic data (b) causal graphs inferred from real data

Fig. 17. Cost of skyline computation with causal graphs inferred from (a) synthetic data (i.e., data generated from synthetic causal graphs) and (b) real data (the lower, the better), when using *lnSky(0.6, 0.4)* – 200K data

the NOTEARS [67] causal inference algorithm from (a) synthetic data (i.e., data generated from synthetic causal graphs) and (b) real-world data. Figure 17 compares the performance of *lnSky(0.6, 0.4)* when using causal graphs inferred by the NOTEARS algorithm with its performance when using the user provided causal graphs, as well as with the baseline variants of the skyline algorithms. As we see in the figure, whether the causal graphs are user provided or inferred, *lnSky* provides significant gains. For data obtained through synthetic causal graphs, using inferred graphs performs slightly worse than using the true causal graphs. Interestingly, for real causal graphs, *lnSky* using the inferred causal graphs outperforms *lnSky* using domain-expert graphs, further demonstrating that a perfect causal graph is not required for CSS to provide significant benefits.

## 8 Conclusions

In this paper, we introduced Causal Search for Skylines (CSS), a novel approach that leverages pruning opportunities arising from positively aligned data distributions in skyline queries. CSS exploits the causal structure of the data to selectively de-correlate attributes, preserving positive correlations while removing negative ones. Our experiments showed that CSS achieves greater efficiency than existing methods, and we discussed its potential limitations. We also outlined a strategy for applying CSS in cases where it offers clear benefits while avoiding it otherwise.

One line of work we will explore in the future is to see how our divide-and-conquer style approach can be further improved by methods, such as parallelizing the skyline computation of individual clusters. Also, in CSS, we use the same skyline algorithm for computing both the skyline of each cluster and also to compute the final skyline after combining the skyline output of all the clusters; however, the skyline merge step can potentially be implemented more efficiently. In addition to these, we plan to explore other ways of "selective de-correlation", such as arranging the order of the variates in a causally informed manner and, generating new attributes through combinations of the preference attributes to produce causally derived conditioning sets.

## Acknowledgments

We sincerely thank the anonymous reviewers for their constructive feedback. This work is supported by NSF grant 2311716, "CausalBench: A Cyberinfrastructure for Causal-Learning Benchmarking for Efficacy, Reproducibility, and Scientific Collaboration", along with NSF grants 2309030, 2412115, 2435886 and a USACE grant GR40695.

Github Co-Pilot was used to assist in the code development. ChatGPT was used for minor editorial support to improve spelling, grammar, punctuation, and clarity, as specified in ACM guidelines [1].

## References


[1] Association for Computing Machinery [n. d.]. *ACM Policy on Authorship*. Association for Computing Machinery. https://www.acm.org/publications/policies/new-acm-policy-on-authorship

[2] 2020. Dry Bean. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C50S4B.

[3] 2020. Seoul Bike Sharing Demand. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5F62R.

[4] Ruhul Amin, Taufik Djatna, Annisa Annisa, and Sukaesih Sitanggang. 2025. Development of Skyline Query Algorithm for Individual Preference Recommendation in Streaming Data. *Journal of Applied Data Sciences* 6, 2 (2025), 1012–1025.

[5] Fahim Tasneema Azad, K. Selçuk Candan, Ahmet Kapkiç, Mao-Lin Li, Huan Liu, Pratanu Mandal, Paras Sheth, Bilgehan Arslan, Gerardo Chowell-Puente, John Sabo, Rebecca Muenich, Javier Redondo Anton, and Maria Luisa Sapino. 2024. (Vision Paper) A Vision for Spatio-Causal Situation Awareness, Forecasting, and Planning. *ACM Trans. Spatial Algorithms Syst.* 10, 2 (2024), 14. doi:10.1145/3672556

[6] Elias Bareinboim, Andrew Forney, and Judea Pearl. 2015. Bandits with Unobserved Confounders: A Causal Approach. In *NIPS*, Vol. 28.

[7] Ole Barndorff-Nielsen and Milton Sobel. 1966. On the distribution of the number of admissible points in a vector random sample. *Theory of Probability & Its Applications* 11, 2 (1966), 249–269.

[8] Ilaria Bartolini, Paolo Ciaccia, and Marco Patella. 2006. SaLSa: computing the skyline without scanning the whole sky. In *CIKM*.

[9] Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

[10] Jacqueline Beinecke and Dominik Heider. 2021. Gaussian noise up-sampling is better suited than SMOTE and ADASYN for clinical decision making. *BioData Mining* 14, 1 (2021), 49.

[11] Jon L Bentley, Kenneth L Clarkson, and David B Levine. 1993. Fast linear expected-time algorithms for computing maxima and convex hulls. *Algorithmica* 9 (1993), 168–183.

[12] Jon Louis Bentley, Hsiang-Tsung Kung, Mario Schkolnick, and Clark D Thompson. 1978. On the average number of maxima in a set of vectors and applications. *Journal of the ACM (JACM)* 25, 4 (1978), 536–543.

[13] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. 2001. The Skyline Operator. In *ICDE*. 421–430.

[14] Paula Branco, Luís Torgo, and Rita P. Ribeiro. 2016. A Survey of Predictive Modeling on Imbalanced Domains. *ACM Comput. Surv.* 49, 2, Article 31 (Aug. 2016), 50 pages. doi:10.1145/2907070

[15] Chee-Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. 2006. Finding k-dominant skylines in high dimensional space. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. ACM, 503–514. doi:10.1145/1142473.1142530

[16] Surajit Chaudhuri, Nilesh Dalvi, and Raghav Kaushik. 2006. Robust Cardinality and Cost Estimation for Skyline Operator. In *ICDE*. 64.

[17] David Maxwell Chickering. 2002. Optimal structure identification with greedy search. *Journal of machine learning research* 3, Nov (2002), 507–554.

[18] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with presorting. In *ICDE*. 717–719.

[19] Paulo Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis. 2009. Wine Quality. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C56S3T.

[20] Bin Gao and Yuehua Cui. 2015. Learning directed acyclic graphical structures with genetical genomics data. *Bioinformatics* 31, 24 (09 2015), 3953–3960.

[21] Parke Godfrey, Ryan Shipley, Jarek Gryz, et al. 2005. Maximal vector computation in large data sets. In *VLDB*, Vol. 5. 229–240.

[22] Georges Hebrail and Alice Berard. 2006. Individual Household Electric Power Consumption. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C58K54.

[23] Hindy. 2021. Hong Kong Weather (1970-2019). https://www.kaggle.com/datasets/hindy51/hong-kong-weather-20082016/data

[24] Wan Ting Hsu, Yu Ting Wen, Ling Yin Wei, and Wen Chih Peng. 2014. Skyline Travel Routes: Exploring Skyline for Trip Planning. In *2014 IEEE 15th International Conference on Mobile Data Management*, Vol. 2. 31–36. doi:10.1109/MDM.2014.64

[25] Jin Huang, Jian Chen, Qing Du, and Jian Yin. 2010. A load balancing skyline query algorithm in high bandwidth distributed systems. In *FSKD*, Vol. 5. 2076 –2080.

[26] Dominik Janzing. 2019. Causal Regularization. In *Advances in Neural Information Processing Systems*, Vol. 32. Curran Associates, Inc., Vancouver, BC.

[27] Kate Keahey, Jason Anderson, Zhuo Zhen, Pierre Riteau, Paul Ruth, Dan Stanzione, Mert Cevik, Jacob Colleran, Haryadi S. Gunawi, Cody Hammock, Joe Mambretti, Alexander Barnes, François Halbach, Alex Rocha, and Joe Stubbs. 2020. Lessons Learned from the Chameleon Testbed. In *Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC '20)*. USENIX Association.

[28] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In *VLDB*.

[29] Hsiang-Tsung Kung, Fabrizio Luccio, and Franco P. Preparata. 1975. On Finding the Maxima of a Set of Vectors. *J. ACM* 22 (1975), 469–476.

[30] Ken CK Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. 2007. Approaching the skyline in Z order.. In *VLDB*, Vol. 7. 279–290.

[31] Ken C. K. Lee, Baihua Zheng, Huajing Li, and Wang-Chien Lee. 2007. Approaching the skyline in Z order. In *VLDB*. 279–290.

[32] Mao-Lin Li, K. Selçuk Candan, and Maria Luisa Sapino. 2023. CTT: Causally Informed Tensor Train Decomposition. In *IEEE International Conference on Big Data, BigData 2023, Sorrento, Italy, December 15-18, 2023*. IEEE, 1180–1187. doi:10.1109/BIGDATA59044.2023.10386626

[33] Mao-Lin Li, K Selcuk Candan, and Maria Luisa Sapino. 2024. Causally Informed Factorization Machines . In *2024 IEEE International Conference on Big Data (BigData)*. IEEE Computer Society, Los Alamitos, CA, USA, 448–455. doi:10.1109/BigData62323.2024.10825754

[34] Xuemin Lin, Yidong Yuan, Wei Wang, and Hongjun Lu. 2005. Stabbing the Sky: Efficient Skyline Computation over Sliding Windows. In *ICDE*.

[35] David Lopez Paz, Robert Nishihara, Soumith Chintala, Bernhard Scholkopf, and Leon Bottou. 2017. Discovering Causal Signals in Images. In *CVPR*.

[36] Cheng Luo, Zhewei Jiang, Wen-Chi Hou, Shan He, and Qiang Zhu. 2012. A sampling approach for skyline query cardinality estimation. *Knowledge and information systems* 32, 2 (2012), 281–301.

[37] Pratanu Mandal, Abhinav Gorantla, K. Selçuk Candan, and Maria Luisa Sapino. 2026. Causal Search for Skylines (CSS): Causally-Informed Selective Data De-Correlation. https://doi.org/10.48550/arXiv.2603.14339. (Extended version).

[38] Jiri Matoušek. 1991. Computing dominances in $E^n$. *Inform. Process. Lett.* 38 (1991), 277–278.

[39] Xiaoye Miao, Yangyang Wu, Jiazhen Peng, Yunjun Gao, and Jianwei Yin. 2023. Efficient and Effective Cardinality Estimation for Skyline Family. *Proc. ACM Manag. Data* 1, 1, Article 104 (May 2023), 21 pages. doi:10.1145/3588958

[40] Mithila Nagendra and K. Selçuk Candan. 2012. Skyline-sensitive joins with LR-pruning. In *EDBT*. 252–263.

[41] Warwick Nash, Tracy Sellers, Simon Talbot, Andrew Cawthorn, and Wes Ford. 1994. Abalone [Dataset]. UCI Machine Learning Repository. doi:10.24432/C55C7W

[42] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive Skyline computation in database systems. *ACM Trans. Database Systems* 30 (2005), 41–82.

[43] Sungwoo Park, Taekyung Kim, Jonghyun Park, Jinha Kim, and Hyeonseung Im. 2009. Parallel Skyline Computation on Multicore Architectures. In *Proceedings of the 25th IEEE International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 760–771. doi:10.1109/ICDE.2009.42

[44] Judea Pearl. 2000. *Causality: Models, Reasoning, and Inference* (1st ed.). Cambridge University Press.

[45] Judea Pearl. 2009. Causal inference in statistics: An overview. *Statist. Surv.* 3 (2009), 96–146.

[46] Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. 2017. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, Cambridge, MA.

[47] R. Quinlan. 1993. Auto MPG. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5859H.

[48] Georg Stefan Schlake and Christian Beecks. 2024. The Skyline Operator to Find the Needle in the Haystack for Automated Clustering. In *2024 IEEE International Conference on Big Data (BigData)*. 6117–6122. doi:10.1109/BigData62323.2024.10825416

[49] Haichuan Shang and Masaru Kitsuregawa. 2013. Skyline Operator on Anti-correlated Distributions. *Proc. VLDB Endow.* 6, 9 (2013), 649–660. doi:10.14778/2536360.2536365

[50] Paras Sheth, Ruocheng Guo, Lu Cheng, Huan Liu, and Kasim Selçuk Candan. 2023. Causal Disentanglement for Implicit Recommendations with Network Information. *ACM Trans. Knowl. Discov. Data* 17, 7 (2023), 94:1–94:18. doi:10.1145/3582435

[51] Shohei Shimizu, Patrik O Hoyer, Aapo Hyvärinen, Antti Kerminen, and Michael Jordan. 2006. A linear non-Gaussian acyclic model for causal discovery. *Journal of Machine Learning Research* 7, 10 (2006).

[52] Peter Spirtes, Clark N Glymour, and Richard Scheines. 2000. *Causation, prediction, and search*. MIT press.

[53] R. E. Steuer. 1986. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley, New York, 546 pp.

[54] Ivan Stojmenović and Masahiro Miyakawa. 1988. An optimal parallel algorithm for solving the maximal elements problem in the plane. *Parallel Comput.* 7 (1988), 249–251.

[55] Dalie Sun, Sai Wu, Jianzhong Li, and Anthony K. H. Tung. 2008. Skyline-join in distributed databases. In *ICDE Workshops*.

[56] Kian-Lee Tan, Pin-Kwang Eng, and Beng Chin Ooi. 2001. Efficient Progressive Skyline Computation. In *VLDB*. 301–310.

[57] Yufei Tao and Dimitris Papadias. 2006. Maintaining Sliding Window Skylines on Data Streams. *IEEE TKDE* 18 (2006), 377–391. Issue 3.

[58] Akrivi Vlachou, Christos Doulkeridis, and Yannis Kotidis. 2008. Angle-based space partitioning for efficient parallel Skyline computation. In *SIGMOD Conference*. 227–238.
[59] Akrivi Vlachou, Christos Doulkeridis, and Neoklis Polyzotis. 2011. Skyline query processing over joins. In *SIGMOD*.
[60] Yu-Ting Wen, Kae-Jer Cho, Wen-Chih Peng, Jinyoung Yeo, and Seung-won Hwang. 2015. KSTR: Keyword-Aware Skyline Travel Route Recommendation. In *2015 IEEE International Conference on Data Mining*. 449–458. doi:10.1109/ICDM.2015.37
[61] Sewall Wright. 1921. Correlation and Causation. *Journal of Agricultural Research* 20 (1921), 557–585.
[62] Ping Wu, Caijie Zhang, Ying Feng, Ben Y. Zhao, Divyakant Agrawal, and Amr El Abbadi. 2006. Parallelizing Skyline queries for scalable distribution. In *EDBT*. 112–130.
[63] I-Cheng Yeh. 1998. Concrete Compressive Strength. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5PK67.
[64] Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang, and Jeffrey Xu Yu. 2009. Probabilistic Skyline Operator over Sliding Windows. In *ICDE*. 1060–1071.
[65] Xichen Zhang and Hasan Cavusoglu. 2026. Skyline operators in multi-criteria decision making: A review of characterization, comparison, and perspectives. *Expert Systems with Applications* 306 (2026), 130889. doi:10.1016/j.eswa.2025.130889
[66] Zhenjie Zhang, Yin Yang, Ruichu Cai, Dimitris Papadias, and Anthony Tung. 2009. Kernel-based skyline cardinality estimation. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data* (Providence, Rhode Island, USA) *(SIGMOD '09)*. Association for Computing Machinery, New York, NY, USA, 509–522. doi:10.1145/1559845.1559899
[67] Xun Zheng, Bryon Aragam, Pradeep K Ravikumar, and Eric P Xing. 2018. Dags with no tears: Continuous optimization for structure learning. *Advances in neural information processing systems* 31 (2018).

Received October 2025; revised January 2026; accepted February 2026