Exploring Data Partitions for What-if Analysis

Nguyen Quoc Viet Hung[1], Kai Zheng[2], Matthias Weidlich[3], Bolong Zheng[4],
Hongzhi Yin[5], Nguyen Thanh Tam[6], Bela Stantic[1]

[1]*Griffith University* [2]*University of Electronic Science and Technology of China* [3]*Humboldt-Universität zu Berlin*
[4]*Aalborg University* [5]*The University of Queensland* [6]*École Polytechnique Fédérale de Lausanne*

***Abstract*—What-if analysis is a data-intensive exploration to inspect how changes in a set of input parameters of a model influence some outcomes. It is motivated by a user trying to understand the sensitivity of a model to a certain parameter in order to reach a set of goals that are defined over the outcomes. To avoid an exploration of all possible combinations of parameter values, efficient what-if analysis calls for a partitioning of parameter values into data ranges and a unified representation of the obtained outcomes per range. Traditional techniques to capture data ranges, such as histograms, are limited to one outcome dimension. Yet, in practice, what-if analysis often involves conflicting goals that are defined over different dimensions of the outcome. Working on each of those goals independently cannot capture the inherent trade-off between them. In this paper, we propose techniques to recommend data ranges for what-if analysis, which capture not only data regularities, but also the trade-off between conflicting goals. Specifically, we formulate a parametric data partitioning problem and propose a method to find an optimal solution for it. Targeting scalability to large datasets, we further provide a heuristic solution to this problem. By theoretical and empirical analyses, we establish performance guarantees in terms of runtime and result quality.**

I. Introduction

Given a simulation model of some complex system, *what-if analysis* aims at exploring the dependencies imposed by the model between input parameters and outcomes [12]. It is performed if the outcomes cannot be easily captured as a mathematical function over the input parameters. What-if analysis has diverse applications, e.g., in data warehousing [27], system workload profiling [17], policy design and forecasting [16], and source code management [4].

Traditional What-if Analysis. What-if analysis involves (i) a set of input *parameters* for which values are chosen by a user; (ii) a simulation *model* that, given a set of values for the input parameters, produces an *outcome*; and (iii) a *goal* according to which some outcomes are preferred over others [12]. What-if analysis enables a user to understand the dependencies between parameters and outcomes, exploring how the outcome improves (w.r.t. the given goal) when changing particular parameters [12]. Although a goal is often specified as an optimisation problem, its actual solution is typically not the primary concern of a user. Rather, a user strives for insights that go beyond knowing an optimal solution in order to take well-informed decisions.

A common approach to what-if analysis is interactive and iterative, driven by a visualisation of the relation between input parameters and outcomes. One of the parameters is profiled by varying the remaining parameters and relying on the model to derive the outcomes. The obtained results are then visualised, e.g., as a histogram [11, 24]. By exploring its shape, a user derives insights on the regularities and patterns of the system represented by the model. However, having a histogram show the outcomes for all parameter values is overwhelming for users. Hence, models such as maxdiff-histograms [39] or v-optimal histograms [23] partition parameter values into ranges and visualise aggregated outcomes. The partitioning and aggregation, in turns, aim at preserving characteristics and patterns in the relation between parameters and outcomes.

What-if Analysis with Conflicting Goals. Methods for what-if analysis based on histograms do not generalise to scenarios, in which multiple goals are defined over different dimensions of outcomes. Such goals are often conflicting: an outcome may be preferred over another one according to one goal, but not according to a second goal. What-if analysis enables users to explore and thereby understand these trade-offs between goals.

As an example, we consider the scenario of a user exploring flight connections between two cities as captured in Table I. Here, the travel date, changed explicitly by the user, and various implicit factors (not shown, e.g., airline and number of stops) serve as input parameters. The outputs are specific connections, evaluated along two dimensions: price and duration. The user's goal is to minimize both, the price and the duration. Using a model given as a query over some database of flight data, these goals turn out to be conflicting—some short connections are more expensive than those with a long duration.

TABLE I: Data for what-if analysis of flight connections.

Travel Date	Duration (min)	Price (USD)
01/12/2016	480	485.65
03/12/2016	540	534.87
04/12/2016	620	1616.2
06/12/2016	600	362.1
06/12/2016	600	398.57

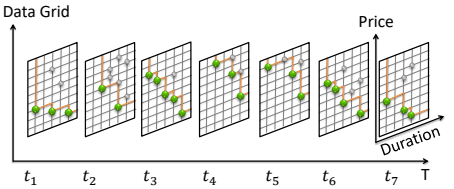


Fig. 1: Grids of possible outcomes

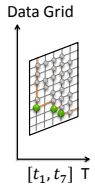


Fig. 2: Merged grids

To explore this trade-off, a user may rely on the plot in Fig. 1. Here, the x-axis represents possible values of the explicit parameter, the travel date. Each value is associated with a *data grid* of possible outcomes. Each outcome is a data point in two dimensions: price and duration of the flight connection.

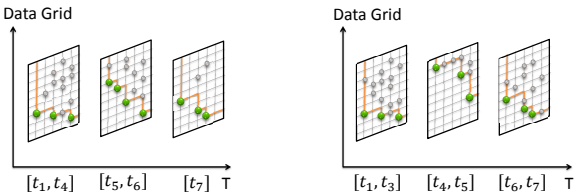


Fig. 3: Sample data range partitions

It is apparent that the visualisation offered in Fig. 1 does not scale in the number of explored parameter values, e.g., a few years of flight data. Therefore, what-if analysis requires data aggregation methods to provide a more compact representation of the data, while preserving specific patterns in the data. However, simply merging all grids (Fig. 2) hides data patterns and is thus not particularly useful for data exploration [22]. Rather, users shall be supported in the precise definition of their interests, which requires illustration of existing data patterns.

Challenges. To enable effective what-if analysis in the presence of conflicting goals, the number of visualised data grids needs to be reduced, as done by maxdiff-histograms or v-optimal histograms for scenarios with a single goal. Parameter values need to be merged into ranges and their associated grids shall be summarised by a new representative grid. The extent of data aggregation is then controlled by a user selecting a pre-defined number b (*buckets*) of data ranges to appear in the result.

When summarising data grids for what-if analysis, two questions need to be answered:

(1) Which data to aggregate? Aggregation incurs information loss. Yet, effective analysis requires the characteristics of the data to be preserved as much as possible, raising an optimisation problem: *data range partitioning* aims at finding a partition of the domain of an input parameter, so that the resulting aggregation of data grids has minimal information loss.

(2) How to aggregate data? Once parameter ranges have been identified, their associated data grids need to be aggregated. The challenge here is the construction of a representative data grid that effectively summarises the original grids.

Answering the above questions is difficult not only from a conceptual point of view, but also in terms of the induced scalability challenges. What-if analysis often faces large datasets. The parameter domains to explore may show up to 100K values [35], meaning that a large number of data grids, each comprising up to millions of data points [5], need to be aggregated. Also, there is a combinatorial explosion of the number of possible aggregations. Since the lengths of the data ranges in the result are arbitrary, there is a factorial number of possible partitions of the domain of a parameter. For example, Fig. 3 shows two possible partitions of Fig. 1. Fig. 3b turns out to be more informative than Fig. 3a, since it shows a pattern of flights on t_4 and t_5 (i.e., weekend days) being worse in price and duration compared to the remaining dates.

Approach. We argue that for large-scale what-if analysis with conflicting goals, it is neither feasible nor reasonable to consider all available data. In fact, many data points are dominated by other points regarding *all* goals. In Fig. 1, there are flight options (grey dots) that are dominated by other options (green dots) in price and duration. Such redundant points are not of interest to a user and shall be neglected in data aggregation.

Against this background, we ground what-if analysis in skylines [5] (or Pareto sets) of data grids, i.e., the set of their non-redundant data points (i.e., the green points in Fig. 1). This has several advantages. It enables us to compute the similarity of two grids as the distance of their skylines, measured efficiently based on the multi-dimensional area bounded by them. Such similarity assessment then guides the selection of data grids to be aggregated. Furthermore, the construction of a representative data grid becomes another skyline computation based on all non-redundant data points of the original data grids. Then, the information loss implied by the aggregation is measured by the total distances between the skyline of the representative grid and the skylines of the original grids.

Based on this general idea, the contributions of this paper are summarised as follows:

- We present a computational model to measure the similarity of data grids, to construct a representative grid, and to quantify the information loss induced by aggregation. This model is based on skylines of data grids.
- To find an optimal data range partition, we propose an algorithm with a time complexity of $O(n^2 \cdot b)$ and space complexity of $O(n^2)$, b being the number of data ranges and n being the number of original data grids. We show that the problem has an optimal sub-structure property, which gives rise to a dynamic programming procedure.
- Analysing properties of our computational model, we propose improvements of the algorithm to solve the data range partitioning problem. By amortizing the computation of the information loss, we are able to reduce the time complexity to $O(n^2)$ and the space complexity to $O(n \cdot b)$.
- We propose a heuristic solution to data range partitioning, which exploits that highly similar neighbouring data grids typically end up in the same representative grid. This heuristic solution has pseudo-linear complexity, i.e., it runs in $O(b \cdot m \cdot n)$ time and $O(b \cdot m)$ space with $m \ll n$.
- Finally, for interactive data exploration scenarios, we present techniques that support parameter domain extensions, dynamic adaptation of the number of buckets, and drill-down into specific ranges.

Structure. The next section reviews related work. The data range partitioning problem is introduced in §III, before §IV describes the measurement of information loss, the construction of a representative data grid, and the computation of an optimal solution for data range partitioning. §V presents scalability improvements and a heuristic algorithm. We then discuss interactive data explorations scenarios in §VI, before §VII presents an experimental evaluation. §VIII concludes the paper.

II. Related Work

What-if analysis studies the effects of parameters on the outputs of a complex system [12]. Types of what-if analysis vary in how a complex system is modelled, including hypothetical modifications [9], stochastic simulation [2], or

statistical methods [28]. In this work, we study what-if analysis for data exploration, where the model of a system is non-trivial. Although the setting is similar to sensitivity analysis [41], traditional techniques consider only one outcome dimension and, thus, one goal. Our use cases are scenarios with multiple, conflicting goals, for which skylines turn out to be an appropriate representation of data grids.

Our problem setting is similar to the one of database exploration techniques [22], especially in the context of multi-objective optimisation problems [7]. That is, users cannot formulate their interests as a query until the data of interest is shown to them. However, in what-if analysis, users commonly share some constraints to optimize (e.g., the price), whereas their requirements differ in other dimensions (e.g., setting a threshold for the maximal duration). Moreover, what-if analysis cannot be tracked back to multi-objective optimisation as the trade-off between conflicting goals cannot be encoded. For instance, there is no deterministic weighting of goals to apply.

Histograms. Data exploration scenarios with a single goal often employ histograms to approximate the distribution of data [11, 14, 15, 24, 25]. Different types of histograms have been studied in the literature, such as: (i) equi-width histograms [35], (ii) equi-depth histograms [35], (iii) v-optimal histograms [23], (iv) maxdiff-histograms [39], (v) compressed histograms [39]. The quality of histogram construction is measured by error functions that are specific to application domains and data characteristics. Examples include the sum of squares of absolute errors [46], maximum error metrics, and relative error metrics [24].

Histograms have also been studied in signal and image processing [26] under the name of *wavelets*. Here, the idea is to apply hierarchical decomposition functions to transform raw data into wavelet coefficients. Despite having different concepts and techniques, wavelet construction shares some similar complexity and quality results with histogram construction.

The state-of-the-art in histogram construction is limited to a single outcome dimension. So-called multi-dimensional histograms [32, 40] are either based on dimensionality reduction (SVD and Hilbert numbering) and lack guarantees on the result quality, or partition each dimension incrementally (MHIST [40]). In the latter case, a dimension is partitioned only based on the partition of the previous dimension, which neglects any trade-off between outcome dimensions.

Skyline Computation. The computation of skylines (or Pareto sets) has been investigated in the context of databases [5] and decision-support [34]. A first skyline algorithm (solving the maximal vector problem) was presented in the seminal work by Kung et al. [30]. More recently, skyline computation exploits block-nested loop and divide-and-conquer algorithms [5]. For the special case of modelling each skyline point by a monotone function over all dimensions, a nearest neighbour algorithm may also be used [29]. A state-of-the-art algorithm for skyline computation over static data is based on branch-and-bound-search [36]. Most algorithms to compute skylines utilise R*-trees and have a time complexity of $O(n \cdot \log n)$.

Recently, variations of skylines, e.g., dynamic skylines and reverse skylines [33], gained popularity in P2P applications [31] and for streaming and uncertain data [3, 51]. Also, to query a skyline without scanning the whole input data, progressive algorithms [36, 44] and parallelisation schemes [37] have been proposed. Yet, skylines have not yet been studied in what-if analysis. We will demonstrate that what-if analysis motivates novel techniques for merging skylines and computing the distance between them.

III. Model and Problem Statement

Consider a parameter T that a user wants to explore. T takes a finite sequence of possible values $\langle t_1, \ldots, t_n \rangle$ that defines a natural order of the domain R of T, where $t_1 < \ldots < t_n$ and $t_i \in R$, $1 \leq i \leq n$. A user is interested in profiling this parameter by running the simulation model for each parameter value, while varying other parameters to generate possible outcomes. As a result, each parameter value t_i is associated with a data grid π_i. Each data grid is a set of possible outcomes, i.e., $\pi = \{p_1, \ldots, p_m\}$. In this work, we consider outcomes to be two-dimensional as a starting point for generic what-if visualisation [42]. Technically, an outcome is a data point $p = (x, y) \in R \times R$ representing the trade-off along two outcome dimensions. As a short-hand, we use $p_{|x} = x$ and $p_{|y} = y$ to denote the values of the outcome. The domain of x and y is discrete and finite and, without loss of generality, assumed to be the same as for the parameter. User preference is encoded for x and y in an increasing order, such that the smaller the value of x or y, the more preferred the respective outcome.

Denote by $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ the finite sequence of data grids associated with the value sequence of T. By $\Pi^{\cup} = \{\pi_1, \ldots, \pi_n\}$, we denote the set of the grids in Π. Further, $P = \bigcup_{i=1}^{n} \pi_i$ is the set of all possible outcomes of T. To limit the cognitive load imposed on a user, outcomes are not considered for all possible values of T, but for a predefined number b of data ranges. This requires construction of a representative data grid for a set of original data grids by a function $f : 2^{\Pi^{\cup}} \rightarrow 2^{P}$.

The construction of a representative data grid implies information loss. To capture this information loss, we define the distance between two data grids (aka information distortion) as $d : 2^P \times 2^P \rightarrow R$. Then, the information loss for a given set of grids $A \subseteq \Pi^{\cup}$ is measured by the total distances between the representative data grid and the original grids:

$$\zeta(A) = \sum_{\pi \in A} d(f(A), \pi). \tag{1}$$

Recommending b data ranges of the parameter T for a user is equivalent to constructing a non-overlapping partition of parameter values. Since the value domain of T is a sequence, we can define the partition over the indices for simplicity's sake. That is, a partition $V = \langle v_0, v_1, \ldots, v_{k-1}, v_k \rangle$ is valid if $k = b$ and $0 = v_0 < v_1 < \ldots < v_{k-1} < v_k = n$, which distributes the original values $\langle t_1, \ldots, t_n \rangle$ to subsets $\langle \bigcup_{i=v_0+1}^{v_1} t_i, \ldots, \bigcup_{i=v_{b-1}+1}^{v_b} t_i \rangle$. With $\mathcal{V}$ as the set of all valid partitions, we define to the problem of data range partitioning for what-if analysis:

Problem 1 (Data Range Partitioning): Given a sequence of data grids $\Pi = \langle \pi_1, \ldots, \pi_n \rangle$ and a number of data ranges b, find

a valid partition $V^* \in \mathcal{V}$ with minimal total information loss:

$$V^* = \underset{V=\langle v_0,v_1,\ldots,v_{b-1},v_b\rangle \in \mathcal{V}}{\arg\min} \sum_{k=0}^{b-1} \zeta\left(\bigcup_{i=v_k+1}^{v_{k+1}} \pi_i\right) \quad (2)$$

Using a solution V^*, the original data grids in each data range of the partition $\langle \bigcup_{i=v_0+1}^{v_1} \pi_i, \ldots, \bigcup_{i=v_{b-1}+1}^{v_b} \pi_i \rangle$ replaced by representative data grids $\langle f(\bigcup_{i=v_0+1}^{v_1} \pi_i), \ldots, f(\bigcup_{i=v_{b-1}+1}^{v_b} \pi_i)\rangle$. Table II summarizes important notations.

Example 1: The example from Fig. 1 can be represented in this model as follows: The parameter T is a possible start date of the trip. For each date, Fig. 1 shows a data grid with several flight options in two dimensions, x being the duration and y being the price. The example also illustrates the data range partitioning problem: The price for connections on a weekend (t_4 and t_5) is higher than for the other days. Hence, a good partition with $b = 3$ might be $\langle t_1, t_4, t_6\rangle$ since the range $[t_4, t_5]$ would capture the high-price pattern for the weekend days.

TABLE II: Overview of notations.

$\Pi_k = \langle \pi_1, \ldots, \pi_k \rangle$	A sequence of data grids
$\Omega_k = \langle \omega_1, \ldots, \omega_k \rangle$	A sequence of skylines
$V = \langle v_0, v_1, \ldots, v_{b-1}, n \rangle$	A valid partition of data ranges
$d : 2^P \times 2^P \to R$	The distance between two data grids
$f : 2^{\Pi^\cup} \to 2^P$	Function to merge data grids into a single grid
$\zeta : 2^{\Pi^\cup} \to R$	Information loss of a subset of data grids
$I(\omega)$	Dominated value space of a skyline ω

IV. A Method for Data Range Partitioning based on Skylines

This section instantiates the problem of data range partitioning for what-if analysis with conflicting goals. Based on skylines, we show how to measure the similarity of data grids (§IV-A), how to construct a representative data grid (§IV-B), and how to quantify the information loss incurred by aggregation (§IV-C). Finally, we present a dynamic programming algorithm to solve the data range partitioning problem (§IV-D).

A. Distance between data grids

We capture the amount of difference in user interest in the outcomes of data grids by measuring their distance. In the presence of conflicting goals, a distance measure shall be agnostic to redundant outcomes, which are dominated by other outcomes in all dimensions and thus not of interest to a user. Consequently, we trace back the distance between data grids to the distance between their skylines.

Skyline construction. Given a set of data points, a skyline [5] comprises only points that are not dominated by other data points. Formally, a point p *dominates* another distinct point q, denoted as $p \preceq q$, if and only if the values of p are equal or smaller than those of q, $p_{|x} \le q_{|x} \wedge p_{|y} \le q_{|y}$. A point is *redundant*, if it is dominated by at least one point.

Using existing techniques [36], a skyline is computed in $O(|\pi| \log |\pi|)$ time, where $|\pi|$ is the number of data points. That is, each data grid π is reduced into a skyline ω of non-dominated points, i.e., $\omega = \{p \in \pi : \nexists\ p' \in \pi, p' \preceq p\}$, such that $\forall\ p' \in \pi \setminus \omega, \exists\ p \in \omega, p \preceq p'$. A skyline is generally much smaller than the original data grid [5]. Below, we assume skylines to be precomputed for all data grids, with their points sorted by their x-values (breaking ties by the y-values) [44].

We further define Σ as the space of all possible points and, based thereon, the *dominated value space* of a skyline ω as $I(\omega) = \{p \in \Sigma : \exists\ p' \in \omega, p' \preceq p\}$. This notion induces a partial ordering on distinct skylines: We write $\omega \preceq \omega'$ if and only if $\forall\ p' \in \omega', \exists\ p \in \omega, p \preceq p'$. Hence, it holds that $\omega \preceq \omega' \Leftrightarrow I(\omega') \subseteq I(\omega)$.

Distance between skylines. A skyline defines a bound for the outcomes of a data grid that can be represented by a staircase line [36]. Data points not belonging to the skyline are above this line. The distance between two skylines can thus be measured by the area bounded by the staircase line. Formally, we define this distance as the symmetric difference between the dominated value spaces: $d(\omega, \omega') = |(I(\omega) \cup I(\omega')) \setminus (I(\omega) \cap I(\omega'))|$.

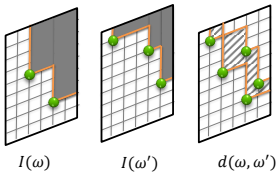


Fig. 4: Distance between two data grids

Example 2: The distance between data grids π and π', with skylines ω and ω', is illustrated in Fig. 4. The solid areas show the dominated value spaces. Intuitively, the distance between ω and ω' is the difference between these areas (shown as dashed).

We present an algorithm to compute the distance between two skylines in the supplement [45] of this paper. It sweeps through two skylines in the order of increasing x-values and finds crossings of line segments to measure the overlap of the dominated value spaces. Using this algorithm, the distance between skylines ω and ω' is computed in $O(|\omega| + |\omega'|)$ time.

B. Construction of a representative data grid

To construct a representative data grid from a set of data grids, we need to merge their data points, retaining dominating points and discarding the redundant ones, which again yields a skyline. A naive construction would take the union of all data points and then compute the skyline. However, since the skylines of the original data grids are available, it is more efficient (and equivalent) to work on the original skylines.

Formally, the representative data grid of a set of grids $\Pi_k^\cup = \{\pi_1, \ldots, \pi_k\}$ is obtained by constructing their skyline. With $\Omega_k^\cup = \{\omega_1, \ldots, \omega_k\}$ as a set of skylines, ω_i being the skyline of grid π_i, the representative grid as a skyline is defined as:

$$f(\Pi_k^\cup) = f(\Omega_k^\cup) = \left\{ p \in \bigcup_{\omega \in \Omega_k^\cup} \omega \mid \nexists\ q \in \bigcup_{\omega \in \Omega_k^\cup} \omega, q \neq p, q \le p \right\} \quad (3)$$

such that $|f(\Omega_k^\cup)|$ is maximal.

However, merging skylines can introduce new redundant points, which need to be removed when performing the merge. Another challenge is to preserve the order of data points in the x-values for further computations.

To address these challenges, we propose an optimal solution in Alg. 1. The input is given as k arrays of skyline points

sorted by their x-value. The main idea is that, when merging skylines, one keeps track of the minimal y-value and discards all redundant points. To preserve the order in the x-values, a priority queue tracks the minimal value of the currently traversed elements (i.e., heads) of input arrays. According to the definition of a skyline, we ensure that there are no points with the same x-value or the same y-value (lines 2, 3 and 11), since two skylines may have two different points with the same x- or y-values. More precisely, we iterate over all points in increasing order of their x-values (line 6) with the help of a priority queue Q. As part of each iteration, we compare the y-value of the current point with y_{min} (line 11). If it is larger, p is redundant due to the definition of a skyline. The output is a new skyline $\hat{\omega}$ which is also sorted by the x-values.

Correctness of the algorithm follows directly: the result contains only points of the original skylines that are not dominated by another point in the result.

Theorem 1: The time complexity of Alg. 1 is $O(k \log k)$, with k being the number of skylines.

All proofs are part of the supplement [45] of this paper.

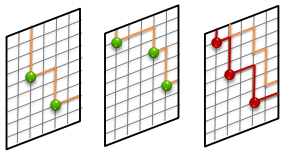


Fig. 5: Merging of skylines

Example 3: For skylines ω and ω' of two data grids, Fig. 5 illustrates the merged skyline of the representative data grid. It contains solely non-redundant points of the original skylines.

C. Computation of information loss

We now show how to compute the information loss induced by the construction of a representative data grid. Following the above rationale to neglect redundant data points, we work directly on the skylines. That is, we measure information loss based on the distances between the skyline of the representative data grid and those of the original data grids:

$$\zeta(\{\pi_1, \ldots, \pi_k\}) = \zeta(\{\omega_1, \ldots, \omega_k\}) = \frac{1}{k} \sum_{i=1}^{k} d(f(\bigcup_{j=1}^{k} \omega_j), \omega_i) \qquad (4)$$

where ω_i is the skyline of data grid π_i.

As detailed above, merging k skylines by means of Alg. 1 takes $O(\sum_{i=1}^{k} |\omega_i| \log k)$ time, while measuring the k distances requires $O(\sum_{i=1}^{k} |\omega_i|)$ time. Hence, computing information loss requires $O(m \cdot k \log k)$ time, where m is the average size of skylines. If m is small or fixed, we can consider it as a constant.

D. Optimal data range partitioning

Using the notions introduced above, we now present an algorithm to solve the problem of data range partitioning (Problem 1). Our algorithm exploits the optimal sub-structure property of the problem: the solution for a sequence of skylines, and thus data grids, can be computed from the solutions of sub-sequences. Such an approach, however, requires knowing the information loss induced by merging a set of skylines that fall into a particular range. We therefore first focus on the pre-computation of information loss for sets of skylines.

Algorithm 1: Merging skylines

input : A set of skylines $\Omega_k^{\cup} = \{\omega_1, \ldots, \omega_k\}$, each sorted by their x-dimension
output: A new skyline $\hat{\omega} = f(\Omega_k^{\cup})$

1 $\hat{\omega} = \langle\rangle$;
2 $y_{min} = +\infty$;
3 $x_{max} = -\infty$; ▷ Ensure no duplicate
// Initialise priority queue
4 $Q[i] = \omega_i[1]_{|x}$ for all $1 \le i \le k$; ▷ Head elements of k arrays
5 $pivot[i] = 1$ for all $1 \le i \le k$; ▷ Pivots for k arrays
6 **while** Q *is not empty* **do**
// Select the array whose pivot element has minimal x-value
7 $m = \arg\min_{1 \le i \le k} Q[i]$; ▷ Remove the head of priority queue
8 $p = \omega_m[pivot[m]]$;
9 $pivot[m] = pivot[m] + 1$;
// Insert next element into priority queue
10 **if** $pivot[m] \le |\omega_m|$ **then** $Q[m] = \omega_m[pivot[m]]_{|x}$;
// Check the redundancy
11 **if** $p_{|x} > x_{max}$ *and* $p_{|y} < y_{min}$ **then**
12 $\hat{\omega} = \hat{\omega}.\langle p \rangle$; ▷ Append p to output
13 $y_{min} = p_{|y}$;
14 $x_{max} = p_{|x}$;
15 **return** $\hat{\omega}$;

Pre-computation of information loss. Given a sequence of skylines $\Omega = \langle \omega_1, \ldots, \omega_n \rangle$, we construct an $n \times n$ matrix of information loss. Each element $L[j,i]$ is the information loss of merging the j-th to i-th skylines ($L[j,j] = 0$ by definition):

$$L[j,i] = \zeta(\bigcup_{t=j}^{i} \omega_t) \qquad (5)$$

For a static number of data points in the skylines, the matrix is constructed in $O(n^3 \log n)$ time and with $O(n^2)$ space, since we iterate over all possible ranges ($1 \le j < i \le n$).

Finding optimal boundary positions. Our algorithm is motivated as follows. Let $V[i,k]$ be an optimal partition for the sequence $\langle \pi_1, \ldots, \pi_i \rangle$ of data grids into k data ranges. Then, the respective total of information loss $Q[i,k]$ (referred to as optimal information loss) is given as:

$$Q[i,k] = \sum_{l=0}^{k-1} \zeta(\bigcup_{t=v_l+1}^{v_{l+1}} \pi_t) = \sum_{l=0}^{k-1} L[v_l + 1, v_{l+1}] \qquad (6)$$

where $v_t \in V[i,k]$ and $v_k = i$, while $Q[i,1] = L[1,i]$ by definition.

We note that the optimal information loss of the sequence from 1 to i with k data ranges can be computed from the sequence from 1 to $i-1$ with $k-1$ data ranges:

$$Q[i,k] = \min_{1 \le j < i} \{Q[j,k-1] + L[j+1,i]\} \qquad (7)$$

That is, the solution for k data ranges can be reduced to the case of $k-1$ data ranges by considering all possible partitions of the rightmost (k-th) data ranges. The correctness of this observation is proved by induction and contradiction: By definition, the optimal information loss $Q[i,k]$ is the sum of information losses over all data ranges $L[v_t + 1, v_t]$ of an optimal partition.

Exploiting this observation, Alg. 2 solves the data range partitioning problem using dynamic programming. The algorithm calculates the information loss matrix bottom-up: It computes

Algorithm 2: Obtaining an optimal solution

```
   input : An n × n information loss matrix L, a number of data ranges b
   output: An optimal partition V* and the minimal information loss Q[n,b]
1  Initialise an empty matrix Q[i,k]_{n×b} ;
2  Initialise an empty matrix V[i,k]_{n×b} ;
   // Dynamic programming
3  for k = 1 to b do
4      for i = 1 to n do
5          min = +∞ ;
6          for j = i-1 to 1 do
7              if Q[j,k-1] + L[j+1,i] < min then
8                  min = Q[j,k-1] + L[j+1,i] ;
9                  V[i,k] = j ;
10         Q[i,k] = min ;
   // Construct the boundary positions of the partition
11 V* = ⟨⟩ ;
12 j = n ;
13 k = b ;
14 while j > 0 do
15     j = V[j,k] ;
16     k = k-1 ;
17     V* = ⟨j⟩.V* ;                          ▷ Insert j into the head of V*
18 return V*, Q[n,b];
```

$Q[i,k]$ for $1 \le i \le n$ and $1 \le k < b$, in increasing order of k, for any fixed k, in increasing order of i. These values are stored, so that they can be retrieved if needed in the computation of later values. The result parts are obtained by maintaining an additional matrix $V[i,k]$ to keep track of the data range boundary positions of evaluated partial solution.

Theorem 2: The time complexity of Alg. 2 is $O(n^2 \cdot b)$ with n and b being the numbers of skylines and data ranges.

Again, the proof can be found in the supplement [45]. Furthermore, the output of Alg. 2 is an optimal solution with *at most b* data ranges in the result. To generate *exactly b* ranges, we skip the computation of $Q[i,k]$ for any $i < k$ and set it to $+\infty$. Then, these solutions will not be chosen when searching the optimal boundary position j.

V. Performance Improvements

Targeting large-scale what-if analysis, this section focuses on solving the data range partitioning problem more efficiently than the above algorithm. We first identify properties of our model that help to reduce the complexity of our algorithm, while maintaining optimality (§V-A). Second, we present a heuristic solution with pseudo-linear complexity (§V-B).

A. Improvements of the optimal algorithm

Efficient pre-computation. The naive approach to pre-compute the information loss matrix outlined in §IV-D has cubic time complexity. However, both functions, $f(.)$ to merge skylines and $d(.,.)$ to measure the distance between skylines, can be computed incrementally. Hence, the computation of the information loss matrix can be amortized.

Specifically, for $\Omega_1^{\cup}$ and $\Omega_2^{\cup}$ as sets of skylines, we exploit the following properties:

- Merging of skylines is associative, i.e., the result of applying the function to a set of skylines is equivalent to applying it to the result for any subset and the remaining elements: $f(\Omega_1^{\cup} \cup \Omega_2^{\cup}) = f(f(\Omega_1^{\cup}) \cup \Omega_2^{\cup})$.

Algorithm 3: Pre-computing information loss

```
   input : A sequence of skylines Ω = ⟨ω_1, ..., ω_n⟩
   output: An information loss matrix L
1  for 1 ≤ i ≤ n do
2      L' = 0 ;
3      ω̂' = ω_i ;
4      for i > j ≥ 1 do
5          ω̂ = f(ω̂', ω_j) ;                          ▷ Merge two skylines
6          L' = L' + (i-j)d(ω̂, ω̂') + d(ω̂, ω_j) ;     ▷ Compute information loss
7          ω̂' = ω̂ ;
8          L[j,i] = L';
9  return L;
```

- Computation of the information loss induced by merging can be done recursively. If $\Omega_1^{\cup} = \Omega_2^{\cup} \cup \{\omega\}$, it holds that: $\zeta(\Omega_1^{\cup}) = \zeta(\Omega_2^{\cup}) + |\Omega_2^{\cup}| d(f(\Omega_1^{\cup}), f(\Omega_2^{\cup})) + d(f(\Omega_1^{\cup}), \omega)$.

Proofs of these properties along with auxiliary results needed to derive them can be found in the supplement [45].

Exploiting these properties, Alg. 3 computes the information loss matrix, while amortizing the cost of both, merging skylines and computing the information loss induced by the merged skyline. The algorithm takes as input a sequence of skylines and returns the information loss matrix, which is computed incrementally. We use a variable $\hat{\omega}$ to keep track of the merge result of $i-j$ skylines. Adding a skyline ω_j, we compute the new merge result in line 5. Similarly, information loss induced by merging $j-i+1$ skylines is computed from the result for $j-1$ skylines in line 6 using the above recursion property.

Theorem 3: The time and space complexity of Alg. 3 are $O(n^2)$, with n being the number of skylines.

Search space pruning. Alg. 2 to solve the data range partitioning problem includes a search step (the loop in line 6) to find the boundary position j (from $i-1$ to 1), such that the value of $Q[j,k-1] + L[j+1,i]$ is minimal. The search space for this step can be reduced based on the observation that the sum of two values ($Q[j,k-1] + L[j+1,i]$ in line 7) is always greater than each individual value. Hence, the minimal value found so far for the sum serves as an upper bound for each individual value. In general, however, we do not know which values of j can be pruned in the search, since $Q[j,k-1]$ or $L[j+1,i]$ may increase or decrease when changing j.

To tackle this challenge, we exploit the monotonicity of information loss. That is, for i,j,t with $0 \le i \le t < j \le n$, it holds that $L[i,j] \ge L[i,t] + L[t+1,j]$ (we provide a proof in [45]). Intuitively, this means that merging two data ranges into one will incur an information loss greater or equal than the total of the information loss induced by the original ranges.

We use the monotonicity of information loss to guide the search step in Alg. 2 by two pruning heuristics. Both heuristics do not change the worst case complexity of the algorithm, but are likely to yield significant speed-ups in practice.

Bootstrapping: The monotonicity property tells us that $L[j+1,i]$ monotonically increases when j decreases, from $i-1$ down to 1 (line 6 of Alg. 2). Thus, knowing the minimum solution Q_0 computed for $Q[j,k-1] + L[j+1,i]$ so far, enables us to stop the search when arriving at a j_0 with $L[j_0+1,i] > Q_0$. Smaller values of j would lead to a larger information loss. We

exploit this idea by bootstrapping a value of Q_0 by executing the respective loop (line 6) in the algorithm a few times, e.g., i/k times following the maximum entropy principle [1]. Using the determined value of Q_0, binary search is then performed to find $j_0 \in [1, i-i/k]$ with $L[j_0+1,i] > Q_0$. Finally, the search is completed on the shorter interval $[j_0, i-i/b]$ to find the minimum value of $Q[i,k]$.

Bounding: Our second heuristic improves the first one by exploiting that $Q[j,k-1]$ monotonically increases when j increases. Thus, the lower bound of the search range can be lifted, recursively reducing the search interval by means of binary search. To realise this idea, we first bootstrap an initial value Q_0 as the suspected upper bound of $Q[i,k]$, again, following the maximum entropy principle. Binary search finds j_0, the minimum j such that $L[j_0+1,i] > Q_0$, ensuring that the optimal solution is some $j > j_0$. Then, $Q[j_0,k-1]$ is a lower bound for any $Q[j,k-1]$ with $j > j_0$ and, due to dynamic programming, the value of $Q[j_0,k-1]$ is known already. We now define $Q_1 = Q_0 - Q[j_0,k-1]$ and perform binary search to find the minimum $j_1 \in [j_0, i-i/k]$, such that $L[j_1+1,i] > Q_1$, meaning that the optimal solution is some $j > j_1$. In general, we define $Q_m = Q_0 - Q[j_{m-1},k-1]$ and repeat this process until $j_m = j_{m-1}$. We then use this j_m as the lower limit and complete the search in the interval $[j_m, i-i/k]$.

Online computation of information loss. The need to store the information loss matrix implies that our approach has quadratic space complexity. As detailed above, the information loss induced by a certain data range can be computed from the one induced by smaller ranges in constant time. Hence, Alg. 2 can be adapted, such that only the information loss induced by a single skyline per data range is stored. Intuitively, the *i*-loop (line 1) and the *j*-loop (line 4) in Alg. 3 for the pre-computation of information loss become the *i*-loop (line 4) and the *j*-loop (line 6) in Alg. 2. Then, information loss is computed online, amortized into constant time and space, reducing the space complexity to $O(n \cdot b)$. In that case, the above heuristics to prune the search space are no longer applicable, though.

B. Heuristic solution

Despite the above improvements, the complexity of computing information loss ($O(n^2)$ time and $O(n^2)$ space) and the construction of an optimal partition ($O(n^2 \cdot b)$ time and $O(n \cdot b)$ space) may still be intractable for large-scale what-if analysis. Datasets may define parameter domains with up to 100K values [35] for data grids containing up to 1M data points [5]. We thus propose a heuristic solution with pseudo-linear complexity that is based on the following observation. Aggregation of similar data grids implies comparatively low information loss. Hence, adjacent grids that are highly dissimilar are likely to induce a boundary of the result partition.

The above algorithms ignore this aspect in order to guarantee optimality. Exploring all possible boundary positions, their time complexity is at least quadratic in n, the number of considered data grids or skylines, respectively. When compromising optimality, however, a scalable solution may be derived by exploring a small number m of 'promising' boundary positions

Algorithm 4: Finding candidate boundary positions

```
   input : A sequence Ω = ⟨ω_1, ... ω_n⟩ of skylines, the number of candidate
           boundary positions (scalable parameter) m
   output: The most promising boundary positions V̂
   // The first m−1 positions
1  for 1 ≤ i ≤ m−1 do
2      T[i] = d(ω_i, ω_{i+1});                          ▷ Priority queue
3      V[i] = i;                          ▷ m most promising boundary positions
   // Investigate remaining positions and replace minimal distance
4  for m ≤ i < n do
5      k_min = arg min_k T[k] ;
6      if d(ω_i, ω_{i+1}) > T[k_min] then
7          T[k_min] = d(ω_i, ω_{i+1}) ;
8          V[k_min] = i;
9  V̂ = ⟨V[1], ..., V[m−1], V[m] = n⟩ ;
10 return V̂
```

($b < m < n$). Then, m becomes a *trade-off parameter*: higher values lead to a better result and higher time complexity.

Below, we exploit this observation and first outline how to find the m most promising boundary positions before showing how to use these positions to find the near-optimal solution.

Finding candidate partition boundary positions. Given a sequence of skylines $\Omega = \langle \omega_1, \ldots \omega_n \rangle$, we consider the skyline at position i to be a promising candidate for a partition boundary, if the distance between ω_i and ω_{i+1} is large. Here, it suffices to focus on partition end-points, since start-points are given by their subsequent positions.

An approach to identify the m most promising candidate boundary positions for a given sequence of skylines is presented in Alg. 4. This algorithm finds the top-m largest distances between two consecutive positions in the sequence. It maintains a priority queue T of distances and tracks the positions of skylines in the original sequence in V. That is, $T[i]$ contains the distance between skylines at positions $V[i]$ and $V[i]+1$ in the original sequence. Iterating over all skylines, the m positions with the largest distances are derived.

Theorem 4: The time and space complexity of Alg. 4 are $O(n \cdot \log m)$ and $O(m)$, respectively, with n being the number of skylines and m being the trade-off parameter.

Merging skylines with minimal information loss. To scale up the solution to the data range partitioning problem based on the determined boundary positions, we adapt the dynamic programming procedure of Alg. 2 used to compute an optimal solution. Instead of exploring all possible positions, we evaluate solely the candidate positions when identifying the best partition. Moreover, Alg. 2 assumes that the information loss matrix has been pre-computed. Since this pre-computation step takes $O(n^2)$ time (Theorem 3), it may dominate the overall time complexity of the heuristic. Thus, we adapt Alg. 2 to amortize the computation of information loss by exploiting its optimal sub-structure property (see §V-A). Yet, incremental computation of information loss is not possible for the first position. Instead, we need to spend n/m iterations on average for incremental computation.

Against this background, Alg. 5 takes as input a sequence of skylines, the number of data ranges in the result, and a sequence of candidate boundary positions derived by Alg. 4. It

Algorithm 5: Heuristic solution

```
input : A sequence Ω = ⟨ω_1, ... ω_n⟩ of skylines, a sequence of m candidate
        boundary positions V̂, a number of data ranges b (b < m < n)
output: The scalable solution with b candidate boundary positions V̂*
1  Initialise an empty matrix Q̂[i,k]_{m×b} ;
2  Initialise an empty matrix V[i,k]_{m×b};
3  L = 0                                          ▷ Temporary variable as in Alg. 3
4  for 1 ≤ k ≤ b do
5      for 1 ≤ i ≤ m do
6          if k = 1 then
7              Calculate L = ζ(⋃_1^i ω_l) from ζ(⋃_1^{i-1} ω_l) ;
8              Q̂[i,k] = L;
9          else
10             Q̂[i,k] = +∞ ;
11             t = i − 1 ;
12             for V̂[i] − 1 ≥ j ≥ 1 do
13                 Calculate L = ζ(⋃_{j+1}^{V̂[i]} ω_l) from ζ(⋃_{j+2}^{V̂[i]} ω_l) ;
14                 if j = V̂[t] then
15                     Q̂[i,k] = min(Q̂[i,k], Q̂[t,k−1] + L) ;
16                     V[i,k] = t if Q̂[i,k] takes value from second term ;
17                     t = t − 1 ;
   // Construct the boundary positions of the partition
18 Determine V̂* from V analogously to V* in Alg. 2 (lines 11- 17) with j = m;
19 return V̂*, Q̂[m,b];
```

returns a partition $\hat{V}$ represents a scalable solution to the data range partitioning problem, along with the induced information loss $\hat{Q}$. The solution is stored in a matrix $\hat{Q}_{m\times b}$ (line 8, 10, 15), such that $\hat{Q}[i,k]$ denotes the minimal information loss for the input sequence from position 1 to position $\hat{V}[i]$ using k data ranges. Here, the idea is that $\hat{Q}[i,k]$ is computed by iterating over all the preceding candidate boundary positions ($1 \leq t \leq i-1$) and computing the sum of the sub-structural solution $\hat{Q}[t,k-1]$ and information loss for the data range $[\hat{V}[t]+1, V[i]]$ in the input sequence. Since we amortize the computation of information loss as in Alg. 3, we only need to keep a single variable L (lines 7 and 13). While this still requires iteration over n positions in the original sequence of skylines (line 12), we only evaluate an optimal solution for each of the candidate boundary positions (line 15).

Theorem 5: The time and space complexity of Alg. 5 are $O(b \cdot m \cdot n)$ and $O(b \cdot m)$, respectively, with n being the number of skylines, m being the trade-off parameter, and b being the number of data ranges.

While Alg. 5 yields a heuristic solution to data range partitioning, it finds an optimal solution under the given set of positions. Hence, if the optimal positions are subsumed by the candidate positions, the resulting solution is no longer an approximation, but an optimal solution (proofs can be found in the supplement [45] of this paper):

Lemma 1: Let $V^* = \langle v_0, v_1, \ldots, v_{b-1}, v_b \rangle$ the partition returned by Alg. 2 and let $\hat{V}^*$ be the partition returned by Alg. 5, given some candidate positions $\hat{V} = \langle v'_1, \ldots, v'_m \rangle$. If for all $0 \leq i \leq b$ there exists $1 \leq j \leq m$ with $v_i = v'_j$, then $\hat{V}^* = V^*$.

Finally, we observe an inherent trade-off between the result quality obtained with this heuristic and its computational complexity. By increasing the number of candidate boundary positions ($m \to n$), we get closer to the optimal solution, yet the worst-case complexity of the heuristic ($O(b \cdot m^2)$) converges to the one of optimal algorithm ($O(b \cdot n^2)$).

Theorem 6: Let $\hat{Q}[m_1,b]$ and $\hat{Q}[m_2,b]$ be the information loss of scalable solutions returned by Alg. 5 for candidate boundary positions $\hat{V}_1 = \langle v_1, \ldots, v_{m_1} \rangle$ and $\hat{V}_2 = \langle v'_1, \ldots, v'_{m_2} \rangle$ derived by Alg. 4, respectively. If $m_1 \leq m_2$, then it holds that $\hat{Q}[m_2,b] \leq \hat{Q}[m_1,b]$.

VI. Supporting Data Exploration

Having introduced algorithms for data range partitioning, we now turn to methods to support interactive data exploration [38].

Extending the domain of the profiled parameter. During data exploration, a user may extend the domain of the profiled parameter, thereby enlarging the sequence of considered data grids. If the number of parameter values is increased from n to n', computation is rather efficient, taking $O((n'^2 - n^2) \cdot b)$ time. The reason is that $n' - n$ further rows need to be computed for the information loss matrix ($Q[.,.]$), each element of which needs $n + n'$ iterations, while matrix $L[.,.]$ is amortized in $O(n'^2 - n^2)$ time. As such, users may be encouraged to start with a restricted domain of the profiled parameter, subsequently extending this domain with little computational overhead.

Adapting the number of data ranges. The number of data ranges in the result is typically chosen by a user and may be adapted after obtaining initial results. Our approach supports such a dynamic adaptation without re-computing the entire result. If a user decreases the number of data ranges in the result from b to b', our approach can rely on the solutions $L[n,b']$ (n being the number of values of the profiled parameter) to obtain the optimal result for the data range partitioning problem.

If a user increases the number of ranges from b to b'', our approach can derive the result by running Alg. 2 with the outer loop starting at $b+1$ and ending at b'', which takes $O(n^2 \cdot (b'' - b))$ time.

Drilling down. After exploring a partition of b data ranges, a user might want to expand a single range into smaller ones. Since this is a time-consuming tasks, a user shall be guided to 'interesting' ranges for exploration. To this end, for each generated data range, we compute a measure of interestingness and rank ranges accordingly.

We quantify the interestingness of a skyline by the area bounded by it. The rationale is that the larger this area, the higher the preference (in terms of the goals of what-if analysis) for the data points in the skyline. Using the distance of skylines introduced earlier, the interestingness of a skyline ω is:

$$g(\omega) = d(\omega, \{(x_{max}, y_{max})\}) \tag{8}$$

where $\{(x_{max}, y_{max})\}$ is a skyline containing only the point with maximum x-value and maximum y-value in the data space. Based thereon, data ranges are ranked in decreasing order of the interestingness of the respective skylines.

Maintenance of skylines. Finally, during data exploration, new data points may become available and shall be considered in the analysis. Being based on skylines of datasets, such new data can be incorporated incrementally in our approach. That is, online computation of skylines is exploited, so that

the update complexity is at most proportional to the size of skyline, see [36] and the work discussed in §II.

VII. Experimental Evaluation

We evaluated our approach to what-if analysis using real-life as well as synthetic datasets. Below, we first clarify the setup (§VII-A) in terms of datasets and evaluation measures, before turning to an analysis of the efficiency of our methods (§VII-B). We then present results on the quality of the obtained summaries of data grids (§VII-C), also compared to baseline partitioning methods. Finally, we evaluate the efficiency of our methods to support interactive exploration (§VII-D).

A. Setup

Real-world datasets. We rely on two real-world datasets from the domain of travel planning, which are publicly available.[1]

Airline dataset: This dataset contains information on airline ticket bookings for 213 days (June to December 2016), crawled from Skyscanner. Each record is made up of an entire trip and comprises information on the price, total duration, date, origin, destination, number of connecting airports, etc. We target what-if analysis that explores the *price* and *duration* of bookings and spans all departure dates. We further choose all connections between five major cities. Each connection between two cities can be seen as a separate dataset that contains a data grid for each of the 213 days. The average size of data grids is 1000 data points. The skylines are of size 31 on average, a common size in multi-criteria databases [5]. In the remainder, we average the results over all pairs of cities.

Hotel dataset: We consider accommodation offers by hotels in the five major cities for the same period of time, also crawled from Skyscanner. Here, what-if analysis focuses on the *price* and the *distance to the city centre* of the offers. Each city is a separate dataset, with the average sizes of data grids and skylines being 100 and 22, respectively. Again, we report average results over all cities.

Synthetic data. To test our approach in a controlled manner, we also used synthetic data. To this end, we combine a strategy to generate points for data grids with a strategy to distribute these data points across parameter values.

Data grid generation: To control the size and elements of skylines, we follow state-of-the-art procedures [5, 37], fixing data size to 1M and generating 2-dimensional data points that are either (i) *independent*, (ii) *correlated*, or (iii) *anti-correlated*. By doing so, we control the average size of the skylines, since a correlated (anti-correlated) distribution induces rather small (large) skylines. Each distribution has a seed factor $\overline{\omega}$ to control its skewness and the value domain of each dimension.

Sequence simulation: We control the distance between data grids and their order in the input sequence by means of common distributions [6, 13, 35] (e.g., normal, zipfian, exponential). However, these distributions are one-dimensional, whereas we consider two-dimensional data grids. Therefore, we generate the seed factor $\overline{\omega}$ by a sequence distribution, before it is used as input for data grid generation. We study the following sequence distributions that are meaningful in practice: (1) *no shift*, so that we use the original normal, zipfian, and exponential distributions; (2) *normal shifts* that simulate the recurring patterns in real-world data by shifting the mean of a normal distribution multiple times; (3) *random shifts* that simulate multiple shifts and for each shift, randomly select between a normal and an exponential distribution.

[1]The real-world datasets can be downloaded at https://goo.gl/UyJQUk.

Metrics. To assess how well a partition V represents the original sequence of data grids Π, the *distortion* of V is measured by the relative distance between an original grid and the representative grid averaged by the sequence length n:

$$\Delta(V) = \frac{1}{n}\sum_{k=0}^{b-1}\sum_{\pi\in p_k}\frac{d(f(p_k),\pi)}{g(\pi)} \tag{9}$$

where $p_k = \bigcup_{i=v_k+1}^{v_{k+1}}\pi_i$ is the $(k+1)$-th partition of V and $g(\pi) = d(\pi,\{(x_{max},y_{max})\})$ is the interestingness of π, measured by its distance with the maximum point (x_{max},y_{max}).

To evaluate the heuristic solution, we measure the *approximation ratio*, the relative difference in total information loss between the obtained partition $\hat{V}^*$ and an optimal one V^*:

$$\varepsilon(V^*,\hat{V}^*) = \sum_{k=0}^{b-1}\zeta(p_k)/\sum_{l=0}^{b-1}\zeta(\hat{p}_l) \tag{10}$$

where $p_k = \bigcup_{i=v_k+1}^{v_{k+1}}\pi_i$ is the $(k+1)$-th partition of V^* and $\hat{p}_l = \bigcup_{i=v_l+1}^{v_{l+1}}\pi_i$ is the $(l+1)$-th partition of $\hat{V}^*$.

Experimental environment. All results have been obtained on an Intel Core i7 system (3.4Ghz, 16GB RAM).

B. Evaluating the efficiency

Using synthetic data, we first evaluate the efficiency of our methods to compute the information loss matrix and to search for the optimal partition.

1) Computing the information loss: We compare the computation of information loss using two algorithms: (i) *naive:* constructs the information loss matrix directly using Eq. 5; (ii) *amortize:* constructs the matrix incrementally by Alg. 3. In the experiment, we vary the length of the input sequence, since the complexity of both approaches depends on the number of data grids. The results are averaged over different data grid distributions with an average size of skylines being 1000.

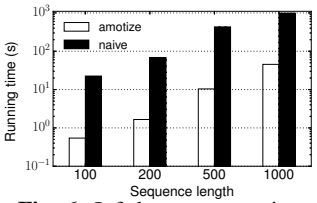


Fig. 6: Inf. loss computation

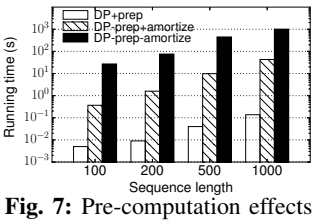


Fig. 7: Pre-computation effects

The results in Fig. 6 highlight that the incremental approach (*amortize*) outperforms the *naive* one. The performance difference is more than an order of magnitude, which illustrates the importance of considering the properties of information loss that enable efficient computation.

We further study the benefits of the information loss matrix pre-computation on the efficiency of the dynamic programming (DP) approach (Alg. 2) to derive an optimal partition. We consider three ways to incorporate this pre-computation: (i) *DP+precomp:* the information loss matrix is pre-computed by Alg. 3 exploiting incremental computation; (ii) *DP-precomp-amortize:* Alg. 2 is executed without pre-computation of information loss and not exploiting incremental computation; and (iii) *DP-precomp+amortize:* Alg. 2 no pre-computation, but incremental computation of information loss.

We use the synthetic dataset, setting $b = 20$ following studies on human cognitive load [8, 10, 18, 19, 42, 50]. The observed runtimes, when varying the length of the input sequence, are shown in Fig. 7. Pre-computation of information loss greatly reduces the time to construct an optimal partition. The fastest algorithm (*DP+precomp*) outperforms the slowest one (*DP-precomp-amortize*) by three orders of magnitude. Around one order of magnitude is due to incremental computation, as illustrated by the results for *DP-precomp+amortize*. Yet, computing information loss on-the-fly still involves complex operations, such as merging of skylines and the computation of their distance, so that pre-computation is beneficial.

2) Obtaining a partition: We next turn to the efficiency of obtaining a partition and compare our method based on dynamic programming with the two pruning heuristics (§V-A) and the heuristic solution (§V-B): (i) *DP:* derives the optimal partition; (ii) *Bootstrap:* applies the bootstrapping heuristic (§V-A) to prune the search space; (iii) *Bound:* prunes based on the bounding heuristic (§V-A); and (iv) *Approx:* is the heuristic solution (Alg. 4 and Alg. 5). The information loss matrix is pre-computed as required by the pruning heuristics.

Impact of the sequence length. We first set $b = 20$ and vary the length of the input sequence from 100 to 10000. The number of candidate boundary positions in the heuristic solution is fixed to $m = 100$. Note that the overall runtime of the heuristic solution depends on the sequence length, as the latter affects Alg. 4 to find candidate positions.

Fig. 8a illustrates that the heuristic solution yields the lowest runtime and scales linearly with the sequence length. Yet, the pruning heuristics also help to improve the runtime considerably compared to the standard approach (*DP*).

Impact of the number of data ranges. Next, we vary the number of data ranges from 10 to 100, while the sequence length is fixed to $n = 5000$ and, as above, we set $m = 100$. To avoid any influence of the length of the input sequence in this experiment, the runtime for the heuristic solution does not include the time to extract candidate boundary positions.

Fig. 8b shows that the runtime improvement achieved by the pruning heuristics compared to the standard approach (*DP*) increases with the number of data ranges. This is attributed to the look-ahead applied by the heuristics: The large the number of data ranges, the more information loss values are accumulated, which results in a higher chance of pruning redundant split positions. The heuristic solution still outperforms the other algorithms.

Impact of the scalability level. We further investigate the heuristic solution (*Approx*) and vary the number of considered candidate boundary positions m from 100 to 5000 (fixing $n = 5000$ and $b = 100$).

As illustrated in Fig. 8c, when m increases, the heuristic algorithm converges to the optimal one. This is expected as, approaching $m = n$, the candidate positions include all possible positions, so that both algorithms consider the same input. Furthermore, we observe the runtime to scale sublinear. This is because the time complexity of Alg. 5 to return an approximate solution becomes $O(b \cdot m^2)$ instead of $O(b \cdot m \cdot n)$, if the information loss matrix is pre-computed.

Real-world datasets. Finally, we test the efficiency of all algorithms for the real-world datasets, using $b = 20$ for the number of data ranges and setting $m = 100$ for the approximation scheme. The results in Fig. 8d confirm the trends observed for the synthetic data in the above experiments. Note that the scalability of heuristic approaches does not only depend on the size of datasets, but is also influenced by dataset characteristics. Absolute runtimes are very low ($\approx 10ms$), which indeed enables support for interactive data exploration.

C. Evaluating the partition quality

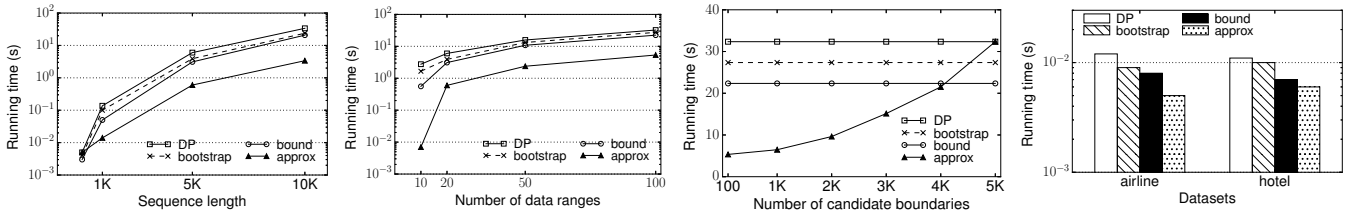
We now turn to the quality of the data summaries obtained by solving the data range partitioning problem. We first consider optimal partitions, before turning to approximate solutions.

1) Representativeness of an optimal partition: We start by evaluating the distortion (the lower, the better) of an optimal partition. To this end, we consider the 'compression ratio' b/n between the number of data ranges in the result and the number data ranges used as input (the sequence length) as a measure to capture the extent of summarisation. We vary the compression ratio from 10% to 100%. For the real-world datasets, this is equivalent to varying parameter b, since the sequence length is fixed. For the synthetic datasets, we vary n from 100 to 1000 and set b such that the respective ratio is obtained.

A uniform partition that divides the input sequence into equal-size parts serves as a baseline for this experiment. With x as the quotient of n/b, we construct x parts of equal size and an additional part for the remaining data grids. For a fair comparison, our methods use a solution with $b+1$ data ranges.

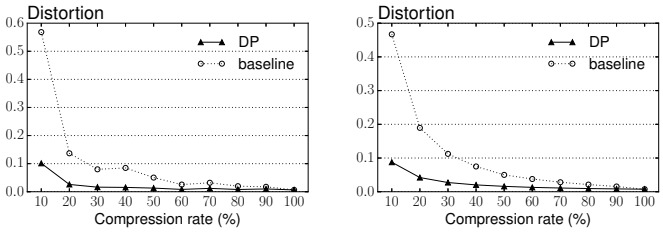
Fig. 9a and Fig. 9b show the average results over the real-world datasets and synthetic datasets, respectively. Our technique (*DP*) turns out to be robust against changes in the compression ratio across datasets. For the representative grids, the average distortion w.r.t. the original data grids is small. Even with a low compression ratio (10%), the distortion is less than or equal 10% of the original data.

Also, the baseline performs worse for small compression ratios (e.g. 10%), highlighting the practicality of our approach: Acknowledging cognitive load limits of users ($b \leq 20$ according to [20, 21, 42, 43, 47–49]), our approach helps to identify important data regularities, outperforming a (naive) uniform partitioning. Even with a compression ratio of 90%, the distortion of uniform partitioning is two times higher than our approach.



(a) Varying the sequence length **(b)** Varying # data ranges **(c)** Varying the scale level **(d)** Real-world datasets

Fig. 8: Experiments on algorithms to obtain a solution for the data range partitioning problem



(a) Real-world datasets **(b)** Synthetic datasets

Fig. 9: Representativeness of an optimal partition

2) Error of the heuristic solution: Having evaluated the efficiency of the heuristic solution already, we now turn to the introduced error and compare the observed information loss to the one obtained for an optimal partition. Specifically, we test the approximation ratio (Eq. 10) depending on the trade-off rate m/n, defined over the sequence length n and the trade-off parameter m. We rely on this rate to provide a guideline on how to choose m for a particular sequence length. We further set $b = 20$ and vary $r = m/n$ from 10% to 100%. For real-world datasets, this is equivalent to varying m only, since n is fixed. For the synthetic datasets, for each value of r_i, we increase the sequence length n from b/r_i (keeping $m \geq b$) to 1000 and set $m = n \cdot r_i$ accordingly. The approximation ratio reported for each value r_i is averaged over all values of n.

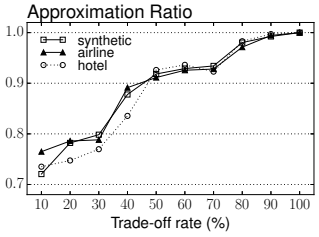


Fig. 10: Approximation – Effects of candidate boundary positions

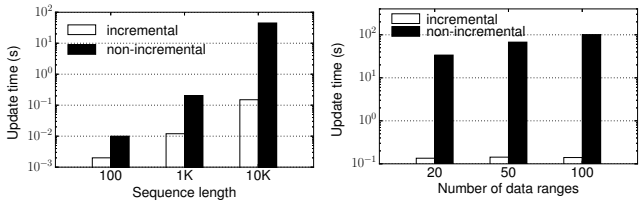
Fig. 10 shows that the approximation ratio increases, over all datasets, with larger trade-off ratios. For example, when 40% of the input sequence are chosen as potential boundary positions, the approximation ratio is higher than 80%, i.e., the relative difference in information loss compared to the optimal partition is less than 20%. Combined with the results on the approximation efficiency, we conclude that the heuristic solution can help to speed-up what-if analysis significantly under a moderate reduction in the result quality.

D. Efficiency of interactive data exploration

Finally, we consider the efficiency of the techniques to support interactive data exploration.

Extending the domain of the profiled parameter. In this experiment, we compare the runtime of the *incremental* approach introduced in §VI with the *non-incremental* computation when extending the parameter domain by one value. We set $b = 20$ and vary the initial domain size from 100 to 10K.

As shown in Fig. 11a, the *non-incremental* approach has the expected quadratic time complexity. The *incremental* version, in turn, scales linearly and thus better supports interactive data exploration, which is important in what-if analysis.



(a) Extending parameter domain **(b)** Adapting data ranges

Fig. 11: Efficiency of interactive data exploration

Adapting the number of data ranges. We also test the *incremental* approach (§VI) against its *non-incremental* counterpart, when increasing the number of data ranges by one. In this adaptation, the information loss matrix is reused and, thus, does not incur any computational overhead. We vary the initial number of data ranges from 20 to 100 and set the length of the input sequence to $n = 10K$.

The *non-incremental* method has comparably long runtimes, which increases with the number of data ranges, see Fig. 11b. The *incremental* approach is fast in all configurations, which is particularly useful in practice, since in what-if analysis, users tend to iteratively examine diverse configurations.

VIII. Conclusions

In this paper, we studied how to support what-if analysis by effective and efficient aggregation of data grids. We argued that what-if analysis with conflicting goals shall be grounded in skylines, i.e., sets of non-redundant data points. For this model, we presented a method to find an optimal partition of the domain of an explored parameter, enabling aggregation of the data grids under minimal information loss. We proposed several optimisations for this method and also introduced a heuristic solution with pseudo-linear complexity. The algorithms are complemented by techniques to support interactive data exploration scenarios. Our experimental results underline the scalability and utility of the presented techniques. Compared to baseline methods, our approach significantly lowers the data distortion introduced by the aggregation.

ACKNOWLEDGMENT

This work is partly funded by the National Environmental Science Programme (NESP) Tropical Water Quality Hub and the Association of Marine Park Tourism Operators (AMPTO) and partially supported the Natural Science Foundation of China (Grant No. 61532018, 61502324).

REFERENCES

[1] D. R. Anderson. *Inf. Theory & Entropy*. Springer, 2008.
[2] H Arsham et al. ""What-if" analysis in computer simulation models: A comparative survey with some extensions". In: *MCM* (1990), pp. 101–106.
[3] M. J. Atallah et al. "Computing all skyline probabilities for uncertain data". In: *PODS*. 2009, pp. 279–287.
[4] C. Bird et al. "Assessing the value of branches with what-if analysis". In: *FSE*. 2012, pp. 45–55.
[5] S. Borzsony et al. "The skyline operator". In: *ICDE*. 2001, pp. 421–430.
[6] S. Chaudhuri et al. "Random sampling for histogram construction: How much is enough?" In: *SIGMOD*. 1998, pp. 436–447.
[7] K. Deb. *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, 2001.
[8] X. Deng et al. "A Novel Centrality Cascading Based Edge Parameter Evaluation Method for Robust Influence Maximization". In: *IEEE Access* (2017), pp. 22119–22131.
[9] D. Deutch et al. "Caravan: Provisioning for What-If Analysis." In: *CIDR*. 2013.
[10] C. T. Duong et al. "Provenance-Based Rumor Detection". In: *ADC*. 2017, pp. 125–137.
[11] H. Ehsan et al. "MuVE: Efficient Multi-Objective View Recommendation for Visual Data Exploration". In: *ICDE*. 2016, pp. 731–742.
[12] M. Golfarelli et al. "What-if simulation modeling in business intelligence". In: *IJDWM* (2009), pp. 24–43.
[13] S. Guha et al. "Approximation and streaming algorithms for histogram construction problems". In: *TODS* (2006), pp. 396–438.
[14] S. Guha et al. "Data-streams and histograms". In: *STOC*. 2001, pp. 471–475.
[15] S. Guha et al. "REHIST: Relative error histogram construction algorithms". In: *VLDB*. 2004, pp. 300–311.
[16] P. J. Haas et al. "Data is Dead... Without What-If Models." In: *PVLDB* (2011), pp. 1486–1489.
[17] H. Herodotou et al. "Profiling, what-if analysis, and cost-based optimization of mapreduce programs". In: *VLDB*. 2011, pp. 1111–1122.
[18] N. Q. V. Hung et al. "Answer validation for generic crowdsourcing tasks with minimal efforts". In: *VLDB J.* (2017), pp. 855–880.
[19] N. Q. V. Hung et al. "Argument discovery via crowdsourcing". In: *VLDB J.* (2017), pp. 511–535.
[20] N. Q. V. Hung et al. "Computing Crowd Consensus with Partial Agreement". In: *TKDE* (2018), pp. 1–14.
[21] N. Q. V. Hung et al. "Minimizing Efforts in Validating Crowd Answers". In: *SIGMOD*. 2015, pp. 999–1014.
[22] S. Idreos et al. "Overview of data exploration techniques". In: *SIGMOD*. 2015, pp. 277–281.
[23] P. Indyk et al. "Approximating and testing k-histogram distributions in sub-linear time". In: *PODS*. 2012, pp. 15–22.
[24] Y. Ioannidis. "The history of histograms (abridged)". In: *VLDB*. 2003, pp. 19–30.
[25] H. V. Jagadish et al. "Optimal histograms with quality guarantees". In: *VLDB*. 1998, pp. 24–27.
[26] B. Jawerth et al. "An overview of wavelet based multiresolution analyses". In: *SIAM review* (1994), pp. 377–412.
[27] S. Klauck et al. "Interactive, Flexible, and Generic What-If Analyses Using In-Memory Column Stores". In: *DASFAA*. 2015, pp. 488–497.
[28] J. P. Kleijnen. "Sensitivity analysis of simulation models: an overview". In: *PSBS* (2010), pp. 7585–7586.
[29] D. Kossmann et al. "Shooting stars in the sky: An online algorithm for skyline queries". In: *VLDB*. 2002, pp. 275–286.
[30] H.-T. Kung et al. "On finding the maxima of a set of vectors". In: *JACM* (1975), pp. 469–476.
[31] J. Lee et al. "Supporting efficient distributed skyline computation using skyline views". In: *ISCI* (2012), pp. 24–37.
[32] J.-H. Lee et al. "Multi-dimensional selectivity estimation using compressed histogram information". In: *SIGMOD*. 1999, pp. 205–214.
[33] K. C. Lee et al. "Z-SKY: an efficient skyline query processing framework based on Z-order". In: *JVLDB* (2010), pp. 333–362.
[34] D. Mindolin et al. "Preference elicitation in prioritized skyline queries". In: *JVLDB* (2011), pp. 157–182.
[35] H. Mousavi et al. "Fast and accurate computation of equi-depth histograms over data streams". In: *EDBT*. 2011, pp. 69–80.
[36] D. Papadias et al. "Progressive skyline computation in database systems". In: *TODS* (2005), pp. 41–82.
[37] Y. Park et al. "Parallel computation of skyline and reverse skyline queries using mapreduce". In: *VLDB*. 2013, pp. 2002–2013.
[38] D. A. Peixoto et al. "Scalable and Fast Top-k Most Similar Trajectories Search Using MapReduce In-Memory". In: *ADC*. 2016, pp. 228–241.
[39] V. Poosala et al. "Improved histograms for selectivity estimation of range predicates". In: *SIGMOD*. 1996, pp. 294–305.
[40] V. Poosala et al. "Selectivity estimation without the attribute value independence assumption". In: *VLDB*. 1997, pp. 486–495.
[41] A. Saltelli et al. "Variance based sensitivity analysis of model output. Design and estimator for the total sensitivity index". In: *CPHYSICS* (2010), pp. 259–270.
[42] J. Sweller. "Cognitive load theory, learning difficulty, and instructional design". In: *LI* (1994), pp. 295–312.
[43] N. T. Tam et al. "Retaining Data from Streams of Social Platforms with Minimal Regret". In: *IJCAI*. 2017, pp. 2850–2856.
[44] K.-L. Tan et al. "Efficient progressive skyline computation". In: *VLDB*. 2001, pp. 301–310.
[45] *https://goo.gl/dc9rhn*.
[46] Y. Wang et al. "A novel approach for approximate aggregations over arrays". In: *SSDBM*. 2015, 4:1–4:12.
[47] H. Yin et al. "Adapting to User Interest Drift for POI Recommendation". In: *TKDE* (2016), pp. 2566–2581.
[48] H. Yin et al. "Discovering interpretable geo-social communities for user behavior prediction". In: *ICDE*. 2016, pp. 942–953.
[49] H. Yin et al. "Mobi-SAGE: A Sparse Additive Generative Model for Mobile App Recommendation". In: *ICDE*. 2017, pp. 75–78.
[50] H. Yin et al. "SPTF: A Scalable Probabilistic Tensor Factorization Model for Semantic-Aware Behavior Prediction". In: *ICDM*. 2017, pp. 585–594.
[51] L. Zhu et al. "Distributed skyline retrieval with low bandwidth consumption". In: *TKDE* (2009), pp. 384–400.