HiePaCo: Scalable hierarchical exploration in abstract parallel coordinates under budget constraints*

Gaëlle Richer*, Joris Sansen, Frédéric Lalanne, David Auber, Romain Bourqui

Univ. Bordeaux, LaBRI, UMR5800, F-33400 Talence, France

Abstract

In exploratory visualization systems, interactions allow to manipulate a visual representation and thereby gain insight into its supporting data. The responsiveness of these interactions is crucial, but achieving it on common hardware becomes increasingly difficult with the ever-growing size of datasets. Moreover, the representation of a large dataset itself is challenging since screen space is limited and, past a certain size, the number of items exceeds the number of pixels available or may render the representation unhelpful. The focus of this paper is on multidimensional data and parallel coordinates. For the system to be scalable, we propose a multiscale representation based on hierarchical aggregation on the client-side and distributed computing on a horizontally scalable infrastructure on the server-side. Multiscale visualization builds on several levels of abstraction to provide interactive and incremental changes in the level of detail. Horizontal scalability refers to the ability to increase the resources of the computing infrastructure by connecting additional computers. This paper presents: (1) a graph-based formalism for describing multiscale representations of parallel coordinates and their interactions and (2) a client-server system with a focus+context representation for multiscale parallel coordinates and distributed computation on a remote data-intensive infrastructure. We leverage the proposed formalism to describe several design possibilities for usual interactions in parallel coordinates, hierarchical navigation, and edition. We illustrated the scalability and usage of the representation in a real-world case. Performance experiments demonstrate that on a 15-computer cluster, the prototype system can scale to billion-item datasets while preserving the interactivity for analysis.

Keywords: interactive visualization, big data, large-scale visualization, parallel coordinates, hierarchical aggregation, multi-scale visualization

1. Introduction

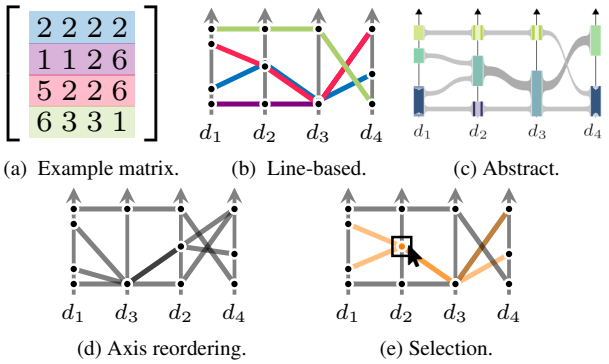
In many application fields, the rapid democratization and development of powerful computers and sensors have led to an increase in the number of collected data and the size of datasets, now reaching petabytes. Visual data exploration systems aim to help analysts gain insights from a dataset through a visualization paired with interactive means to manipulate the data through its representation. The exploratory process is incremental therefore it is essential for interactions to be responsive such that latencies do not interfere with the analyst train of thought. However, achieving responsive interactions on common hardware becomes increasingly difficult with the increasing size of datasets. Latencies may come from the cost of computing the data layout and rendering it or from data processing (filtering, clustering, etc) requested by interaction. Moreover, the representation of a large dataset itself is challenging since display space is inherently limited by screen resolution. Past a certain size, the number of items render the representation unhelpful and even exceeds the number of pixels available. We say that such system requires *(data) processing scalability* and *perceptual (or visual) scalability* [2, 3] that is, it needs techniques to adapt to an ever-growing number of data items while maintaining interactivity and legibility.

In this paper, we take interest in large multidimensional data and parallel coordinate visualization. Multidimensional data is a form common in many application fields. It encompasses all lists of individuals composed of several attributes, possibly temporal. Different aspects of such data may be studied: the particular behavior of individual tuples relative to the whole, the relationship between values from two dimensions, or the distribution of values along each dimension [4]. Parallel coordinates, introduced by Inselberg and Dimsdale [5], is a well-known technique of visualization for multidimensional data. Each data item is represented by a polyline which anchors are positioned on each axis, at the corresponding attribute value of the data item (see Figure 1b which is a parallel-coordinate plot of the data on Figure 1a). Axes are usually aligned in parallel, forming a sequence of two-dimensional subplots sharing one axis with their predecessor. Parallel coordinates usually come with interactive operations for the user to manipulate the data. Usual interactions are *axis reordering* (Figure 1d) to analyze relationships between all dimension pairs and *selection*

*This article is an extended version [1].

*Corresponding author

Email addresses: gaelle.richer@u-bordeaux.fr (Gaëlle Richer), joris.sansen@u-bordeaux.fr (Joris Sansen), frederic.lalanne@u-bordeaux.fr (Frédéric Lalanne), david.auber@u-bordeaux.fr (David Auber), romain.bourqui@u-bordeaux.fr (Romain Bourqui)



(a) Example matrix. (b) Line-based. (c) Abstract.

(d) Axis reordering. (e) Selection.

Figure 1: Examples of (b) line-based and abstract (c) parallel coordinates for a simplistic dataset (a) with two usual interactions: (d) axis reordering and (e) selection. On the line-based plot, colors match those of tuples on (a). On the abstract plot, the height of aggregates corresponds to the number of tuples they cover. On (d), the positions of axes for d_2 and d_3 have been switched. On (e), the first and third tuples have been selected by brushing the second axis.

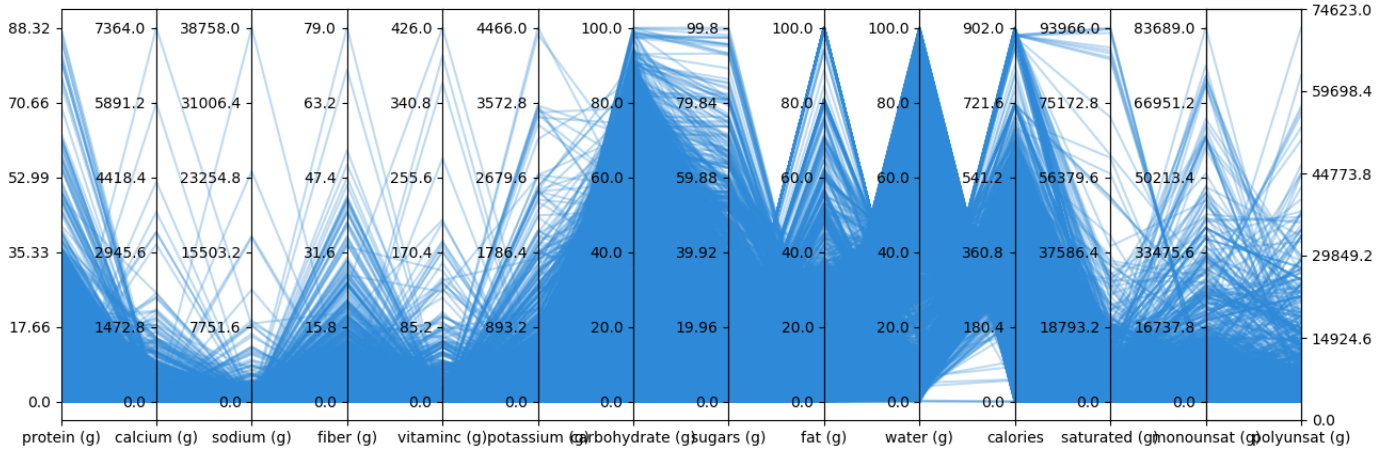
(Figure 1e) to trace subsets of tuples across axes. Axis reordering allows to display different pairs of dimensions, i. e. study different axis-aligned 2D subspaces of the data. Notice that the relationship between d_2 and d_4 is displayed on Figure 1d but not on Figure 1b. Selection relates to interactive means of choosing subsets of tuples and enhancing them such that they can be discriminated from the rest. Highlighting on Figure 1e makes apparent that the tuples sharing the same value for d_2 also overlap on d_3 and d_4.

We are interested in supporting the interactive visual exploration of *large* datasets with a moderate number of dimensions but a challenging number of items or tuples, typically larger than a billion. We built upon the assumption that this number of tuples leads to major *line overplot* and *clutter* when displayed using the traditional line-based parallel coordinate representation (see Figure 2a). Line overplot corresponds to the loss in density information induced by drawing multiple lines over the same pixels. Clutter corresponds to the plot becoming overcrowded which conceal interesting patterns and thus hinder analyses [6]. *Abstraction* is one solution to address this problem [7] that is widely used for multiple visualization techniques. It consists in the display of visual aggregates instead of single lines [8, 9, 10, 11]. Figure 2b and Figure 2c shows examples of *abstract* parallel coordinates using per-dimension clustering to aggregate polylines with two different levels of detail (LoD). Despite the aggregation, these plots successfully provide an overview similar to a traditional plot and perceptually scale for any size of input data. Since they are based on reduced data, they decrease rendering time [9] and suit client/server architecture by bounding the size of the data transferred between client and server [11]. Sansen et al. [11] also leverage aggregation to bound the storage requirements of some precomputed interactions. Precomputing interactions reduces the cost of interactive processing as it remove their dependency on linear scans of the data which is particularly interesting when data does not fit in a desktop computer memory. Indeed, in this case, linear scans over the data affect performance more negatively since access to memory is usually more expensive as it implies e. g. reads on disk or network transfer between several computing units.

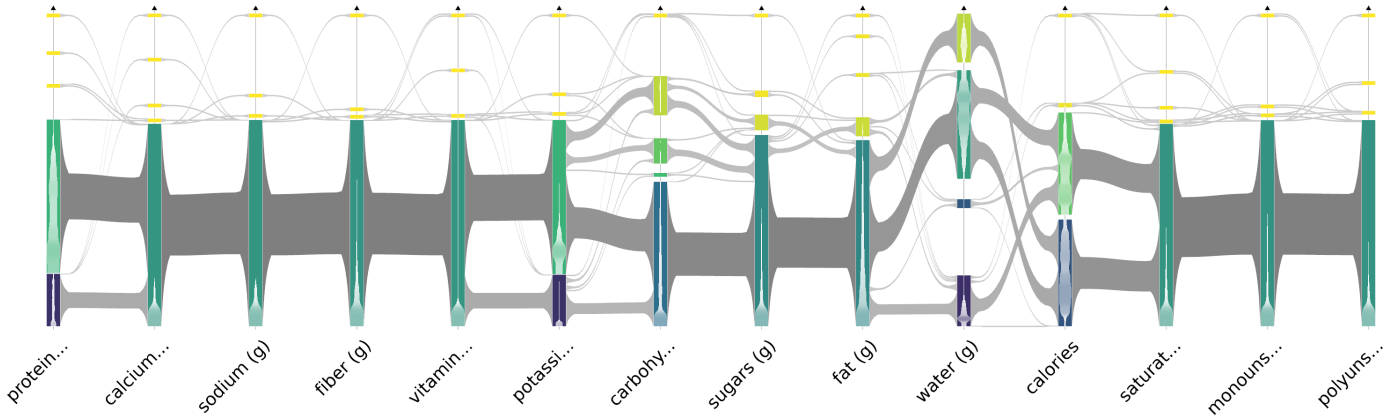
Abstract representations, i. e. representations of aggregated data, is a solution for perceptual scalability but limits in-depth analyses since visual aggregates only convey a reduced amount of information. Specific interactions are commonly provided to manipulate the abstraction and alleviate this limitation: changing the level of detail (show *more* details) and adjusting the aggregation (show *different* details). Supporting these interactions is essential but strongly increases the number of states of the visualization. Then, straightforwardly precomputing and storing these states as proposed by Sansen et al. [11] is no longer efficient. Rübel et al. [12] presented a system based on a modern high-performance computing (HPC) platform for addressing processing scalability. HPC platforms are large and expensive computing systems suited for highly complex and real-time computation. They are composed of multiple processors connected through a fast network and use fast memory. As such, they are particularly adapted to *tightly coupled* tasks where several processors work on the same task and exchange data. Distributed systems, on the contrary, are networks of computing units, usually commodity hardware, connected in a *shared-nothing* architecture (memory and storage are independent to each unit). On these systems, data transfer between computing units uses a slower network connection and thus is critical for performances. Consequently, they are most adapted to *loosely coupled data-parallel* tasks on large amounts of data. In addition to being cheaper alternatives for data-intensive computing, they offer easier *horizontal scalability* (allocation of additional computing units) as their hardware and architecture are less sophisticated. The filtration and aggregation problems at stake in abstract parallel coordinates are data-parallel tasks. In this work, we focus on these less expensive and more accessible platforms to address processing scalability with on-the-fly distributed computing.

This work is motivated by the fact that abstract visualization is inherently limited in the amount of conveyed information and relies on interaction to provide more detail and, in particular, to retrieve item-level information. Without dedicated interactions, increasing the level of detail implies redrawing the whole view at a finer level of detail (for instance from Figure 2b to Figure 2c) which may be cognitively expensive for the user and also tends to result in the same overplotting and clutter issues the line-based plot has. A common manner of addressing this problem is to propose hierarchical focus+context interactions that increase the level of detail *locally* instead of *globally* [13].

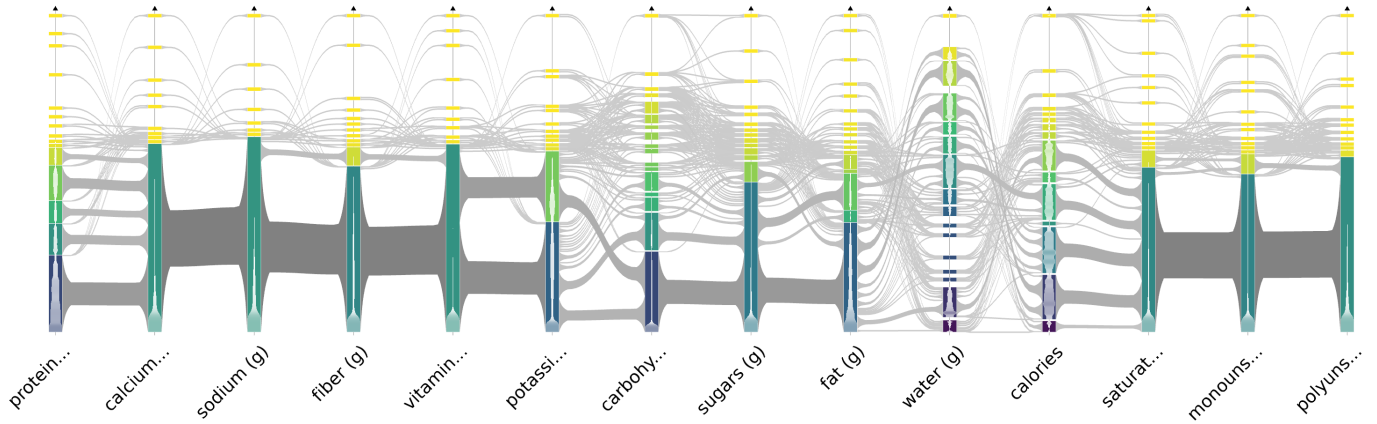
On one hand, several abstract parallel coordinate representations based on aggregation have been proposed [8, 9, 10, 11, 14, 15, 16], some together with hierarchical exploration interactions [14, 15, 16]. On the other hand, parallel-coordinate visualization systems targeting large-scale datasets [11, 12] have not addressed the implementation of such interaction in a remote-visualization setting. The aim of this paper is to address the scalable implementation of such interaction in a remote-visualization context backed by a distributed platform. To this end, we first propose a conceptual model for hierarchical data abstraction in parallel coordinates, that links the base data to



(a) Line-based parallel coordinates with polylines drawn semi-transparent.



(b) Abstract parallel coordinates with 5 clusters per dimension.



(c) Abstract parallel coordinates with 20 clusters per dimension.

Figure 2: The three plots are representations of a medium-sized dataset of 7638 food items and their amount of nutrients. On abstract parallel coordinates (b-c), aggregation allows avoiding the overplotting occurring on line-based parallel coordinates (a) at the expense of some loss of information. c presents an abstract parallel coordinate plots using a larger number of aggregates than (b).

the data supporting the abstract plot. The aim of the model is to (i) facilitate the expression of hierarchical navigation operations, and (ii) allow to evaluate the minimal data necessary to transfer, from server to client, to help compare different interaction designs. Secondly, we present a hierarchical focus+context interaction backed by a distributed platform implementing this interaction as well as other interactions. More precisely, the contributions of this paper are as follows:

- a **graph-based formalism** for data abstraction in abstract parallel coordinate that proposes a graph form for conceptually representing the aggregated data necessary to construct an abstract plot. The model presents a new perspective on several existing works on abstract parallel-coordinate representations, and can be used to devise and evaluate the costs of other interactions for these representations ;
- a comparison of several design possibilities for **interactive operations** for abstract parallel coordinates expressed using the proposed model and compared from a computational perspective. Interactions include hierarchical navigation and edition of the hierarchy.
- a **prototype client/server visualization system** with a focus+context interaction augmenting the representation of Sansen et al. [11]. The server computes interactions, including drill-down and roll-up, by running on-the-fly computation on a horizontally scalable infrastructure. The system guarantees bounded transfer between client and server.

We first present previous work on perceptual scalability in parallel coordinate plots and their interactions with a focus on multi-scale approaches (section 2). Then, we describe the proposed graph-based formalism (section 3). In section 4, we study different interactions under this formalism and compare them under the light of computational complexity and size of data transfer. This yields a prototype implementation and design of a scalable parallel coordinate plot based on hierarchical aggregation, using a so-called *big data* infrastructure, described in section 5. Finally, in section 6 we present a case study and a performance evaluation with some implementation discussion, and section 7 presents directions for future work.

2. Related Work

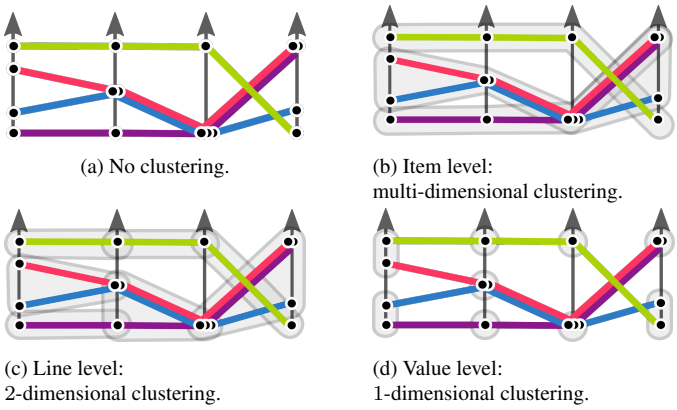
Recent works have focused on the scalability of visualization applications for large-scale data with different techniques, among which are: data reduction, multi-threading, GPU–acceleration, and incremental or approximate data processing. In the case of visualization, the scalability of a system often refers to its capacity to accommodate and handle growing amounts of data. Handling massive datasets brings about two main challenges for exploratory visualization: *perceptual* scalability and *processing* scalability as noted by [17, 2, 3]. The first is concerned with the legibility of visualizations representing numerous items relative to the space available on a screen (so-called *screen real-estate problem*) and human capabilities to apprehend them. The second relates to the computational cost of processing numerous items on *each* user input, that can create latencies responsible for decreasing user performances [18]. A taxonomy of different techniques regarding the perceptual scalability aspect was established by Ellis et al. [6]. A general solution is data reduction, which can be categorized into two approaches: either representing a subset of the data items (sampling, filtering) or meta-items (aggregation, mathematical models). Several works proposed methods (called multiresolution, multiscale, hierarchical or even stratified) for navigating through multiple levels of detail supported by precomputation (e. g. [17, 19, 3]). For graphs, which are the basis of the proposed model, interactive navigation in aggregated views based on hierarchies has been studied in both the database community (e. g. for ontologies [20]) and the visualization community (e. g. TugGraph [21]). This work is related to general techniques addressing perceptual and processing scalability for interactive parallel coordinates.

2.1. Perceptual scalability in parallel coordinates

Various approaches have been proposed to improve the legibility of parallel coordinate plots by either reducing the clutter produced by the multiplicity of overlapping and crossing lines or enhancing their patterns. Approaches can be categorized into *geometry-based* relying on computer graphics techniques and *data reduction* approaches that use approximation or summary of the data. Geometry-based approaches display all items with shape or position modifications to alleviate overdraw in-between axes, for instance by bundling lines (e. g. [22]). These techniques have the advantage of resulting in few losses of information but have the drawback of still being prone to overplot since no reduction of the number of displayed items is performed. Data reduction approaches limit the number of visual items either by sampling items [23] or using meta-items. Model-based approaches mathematically reduce the data to a continuous function, and meta-items usually represents the density of the underlying data (e. g. [24]). Aggregation approaches have been presented for parallel coordinates through different schemes: aggregation of dimensions [25], aggregation of items or values [14, 9, 10] or combinations of both [26, 27].

In this paper, we are interested in scalability relative to the number of items, not dimensions; hence we focus on aggregation over items and their values. Previous work using aggregation have used kernel density estimation, independently applied to dimension [10], different hierarchical clustering algorithms over multidimensional items or dimension [14] but also binning on two-dimensional subspaces [9, 28]. Aggregates have been represented by their statistical properties: extrema [29], cardinality [8], mean [14], or other metrics [10]. As reflected by previous work, parallel coordinates support different levels of grouping: the item level (multi-dimensional), the value level (one-dimensional or per-dimension), the line level (two-dimensional). Figure 3 illustrates how these different clusterings relate to visual elements of parallel coordinates: items relates to polylines, and values to axis points. In the case of 1-dimensional clustering, aggregating dimension points along

the clustering entails an aggregation of lines. The value level



(a) No clustering.

(b) Item level: multi-dimensional clustering.

(c) Line level: 2-dimensional clustering.

(d) Value level: 1-dimensional clustering.

Figure 3: Parallel coordinate visually support different aggregation schemes. Multidimensional clustering correspond to grouping polylines, i. e. items (b), clustering of each axis-aligned 2-dimensional subspace aggregates lines from the same subplot (c), and clustering of values on each dimension aggregates points on the same axis (d).

has the advantages of losing less information, preserving continuity on axes (potentially broken by two-dimensional aggregation) as well as an idea of the pair-wise relationship between dimensions (broken by item-level aggregation). With the value and line levels, together with selection interactions, the user can analyze how the items of one subspace cluster are close in other represented subspaces. Generally, the approach of these aggregation-based methods is to cluster axis-aligned subspaces of the data that hold visual meaning on the parallel coordinate representation (points, lines or polylines) and render the resulting cluster as a visual aggregate.

2.2. Multiscale parallel coordinates

Abstract representations usually depend on a parameter that controls the resolution of the display. Multiscale visualization is common paradigm for navigating between multiple levels of abstraction, using a *drill-down/roll-up* interaction (respectively for increasing and decreasing levels of detail). Elqvist et al. [13] provided general guidelines for multiscale representations and interactions, based on such hierarchical aggregation. Bikakis et al. [30] presented a framework for hierarchical aggregation oriented towards the computational aspects of hierarchical navigation. Interactively changing the level of detail can be integrated into different ways (see [31] for a general review).

A first approach, similar to geometry zooming, either (1) filters part of the representation to maintain a fixed number of visual items on display, or instead (2) purposely displays increasing number of items (e. g. [15, 10, 16]). Filtering has the advantage of being scalable when the amount of displayed items is controlled; however, the overview and context are lost along the way. One drawback of displaying an unlimited number of items is that the screen's pixel limitation could always be reached for a sufficiently massive dataset and then, information would be lost. Additionally, past a certain number, that may overwhelm human cognitive abilities. Consequently, zooming *globally* without filtering does not allow interactive exploration to the item-level complying with perceptual scalability.

For zooming with filtering, an alternative to the loss of context is the *overview+detail* approach which displays both the filtered zoomed view and an overview (e. g. dimension zooming [14]). Among previous work based on hierarchical clustering, HVN (Hierarchical Virtual Nodes) [32] materialized the hierarchy computed over multidimensional items on each axis as dendrograms. The dendrograms allows to directly select groups of items or single items as their nodes are clickable. Then, the line of each selected multidimensional item is drawn routed trough its corresponding parent nodes of the axis dendrograms. Overplot is reduced by displaying item lines solely for the selection since the dendrograms represent the data distribution. For categorical data, Parallel Hierarchies [16] provide an interactive mean to navigate the relationship between different hierarchies by cross-filtering items.

A second approach, called *focus+context*, consists in displaying heterogeneous levels of detail: a selected portion of the data is shown with greater details to the detriment of the rest. This approach increases the level of detail locally while preserving the overview. Fisheye lenses used by Long et al. [33] are an example applied in screen space. In data space, Fua et al. [14] and Novotný and Hauser and [9], although using different aggregation strategies, presented techniques where a subset of items can be enhanced and displayed with fine-scale details layered over the rest of the data, abstracted to some level.

To the best of our knowledge, none of these solutions propose to interactively change the level of detail locally, in a focus+context fashion, while bounding the number of visual items. Additionally, these solutions do not discuss the possibility for the user to interactively edit the aggregation at one specific level such as by merging two aggregates.

2.3. Processing scalability of multi-scale parallel coordinates

For the past ten years, *large-scale* or *massive* has been used to qualify increasingly big data, now up to tera or petabytes in size [2]. For multidimensional data, above about 10^7 items, with a dozen dimensions, perceptual scalability and processing scalability becomes problematic for interactive analysis. Abstract representation and precomputation of interactions are solutions respectively addressing perceptual and processing scalability, as [11] presented. In addition to the enhancement of subsets of items and the reordering of axes, abstract representations require a drilling interaction to allow a fine data analysis.

Processing scalability can be tackled by parallelism, distributed processing and precomputing of complete or partial results for instance. For parallel coordinates, most interactions can be applied to two-dimensional subplots independently and on separate portions of the dataset without needing any communication (they are *pleasantly parallel*). This property has been exploited by Rübel et al. [12] on a HPC platform, and Sansen et al. [11] on a distributed platform. Rübel et al. [12] addressed both scalability challenges with a histogram-based representation adapted from Novotný and Hauser [9]. Histograms are two-dimensional aggregations of the data, precomputed in parallel, potentially for different resolutions. On more affordable hardware, Sansen et al. [11] addressed the same challenges with

a parallel sets representation for a Hadoop+Elasticsearch ecosystem that also relies on full precomputation of certain types of interactions. Both works presented good scalability evaluations relative to the number of computing units used. However, these techniques only support the display of balanced levels of detail i. e. drill-down is *global* and necessarily increases the number of displayed items. With this approach, gaining detail at the item-level is not practical past a certain size of dataset and level of detail as it incur considerable latencies. In contrast, in this paper, we present a technique that supports local drill-down up to the item-level, and edition of the aggregates at a given level of detail.

3. The Graph-Based Formalism

In this section, we detail how we formalize an abstract parallel coordinates plot as a graph, based on the aggregation of tuple values. The goal of this formalism is to connect the data to visualize to the information that support an abstract parallel coordinate representation, and its multiple states resulting from user interactions.

Let us start with a small example made of four tuples and two dimensions as presented on Figure 4a and represented by the red and gray parallel coordinate plot on Figure 4b. By merging together identical values on each axis, we obtain the finest abstract plot represented on Figure 4c, that is essentially a flat version of the original plot. Grouping dimension values using clustering or existing classification allows to generate coarser representations. For instance, Figure 4b represents an example of such grouping in blue, with each blue node defining a group corresponding to its directly connected red nodes and Figure 4d shows the abstract parallel coordinate plot for this grouping. Notice that aggregating values on each dimension leads to aggregation of the lines connecting them, thus forming a structure that can be seen as a graph, here with two nodes on each axis and three edges connecting them. Likewise, the graph of Figure 4c have 7 nodes and 4 edges. This graph object holds the structure of the abstract representation ; together with metadata associated to its nodes and edges, it corresponds to the information sufficient for a visualization client to display the abstract plot to the user. Using the same value grouping, other graphs can be computed by presenting different level of detail between dimensions as on Figure 4e or intra-dimension as Figure 4f. The idea of the formalism described in the following is to show how different interactions on abstract parallel coordinate representation, including hierarchical navigation, translate to construction of such graphs.

On a traditional parallel coordinate plot, only a fraction of the input data 2D subspaces is represented. Here as in the rest of the paper, *subspaces* strictly refers to *axis-aligned subspaces*. Figure 5c and 5d show two examples of representations that display all 2D subspaces at once: respectively the parallel coordinate matrix [34] and the many-to-many plot [35]. To support these representations, the proposed graph model should be able to integrates all 2D subspaces of the data, without assumption on the layout of axes. Additionally, to support abstract

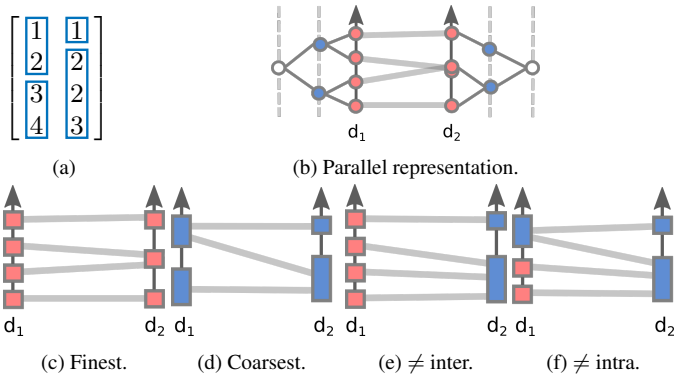


Figure 4: Aggregation at different levels of detail for 4 tuples with 2 dimensions, each augmented by 2-level hierarchies. The finest-level nodes are in red, and coarsest-level nodes in blue. (a) two-dimension data augmented by a dimension grouping, (b) with a parallel coordinate representation with a tree for each dimension grouping. Aggregating tuples along different levels of detail leads to different line aggregation. (c) shows the abstract plot at the finest level of detail, i. e. based on the leaves of the hierarchy trees, (d) shows the abstract plot at the coarsest level of detail, i. e. based on the hierarchy nodes closest to the roots. The levels of detail of dimension clusters can be different *inter* dimensions (e), and *intra* dimension, like d_1 on (f).

representations, it should support aggregation of tuples. Values can also be aggregated along dimensions, however we focus on abstraction based on aggregation of tuples since dimensions hold semantic meaning not embedded in their numerical content. Therefore, automatically and meaningfully aggregate them is not straightforward: some types of values may not make sense aggregated together. Consequently, we target datasets that challenge scalability by their size in tuples and hold moderate amounts of well-chosen dimensions, i. e. $n \gg d$ for datasets with n tuples and d dimensions. .

3.1. Abstraction in parallel coordinates

In parallel coordinates, the values of the input matrix are placed on labeled axes and connected to each other by lines that materialize tuples. As we presented before, this metaphor can be extended to transform an $n \times d$ matrix in an undirected graph which nodes are the matrix values and which edges connect values from the same tuple, i. e. same row in the matrix. For instance, Figure 5e shows such graph for the matrix X of Figure 5a. Each line of a traditional parallel coordinates plot like on Figure 5b translates to a filled edge in this graph. The dashed edges correspond to the *hidden* lines, that only appear when duplicating the axes of a traditional plot, reordering them (Figure 5c) or using alternative representations such as Many-to-Many plots (Figure 5d). Notice that a polyline of a parallel coordinate plot translates to a *clique* in this graph, when hidden lines are included. Thus, this graph is composed of n separate cliques of d nodes, that is, n complete graphs. We note this graph $G = (V, E, w)$, where V denotes the nodes, E the edges, w is the weighting function: $w : V \to \mathbb{R}$ that stores the value of a node. We note $D = \{V_i, i \in [\![1, d]\!]\}$ the partition of V that groups nodes along their origin dimension. Given a sequence of dimension indices, a parallel coordinate plot results from taking the subgraph of G composed of only the edges joining nodes of dimensions consecutive in the sequence. Any repetition in

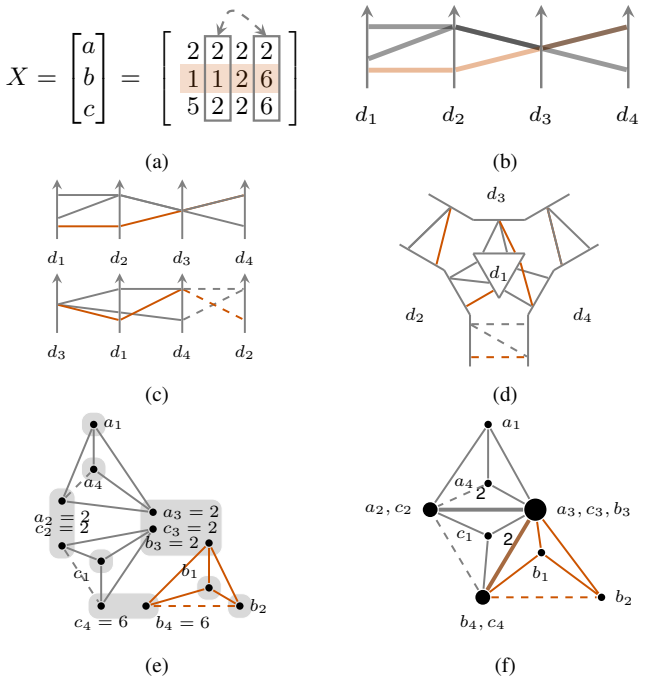


Figure 5: Rationale behind the graph model. (a) Example data. (b) Opacity-based parallel coordinates. (c) Parallel coordinates matrix [34] where all 2-dimensional subspaces are represented. (d) Many-to-many plot [35]. (e) Graph-based representation $G(X)$. Shadowed areas cover identical values from the same dimension and form the partition R. (f) Quotient graph $G_{/R}(X)$ (cardinality mapped to size).

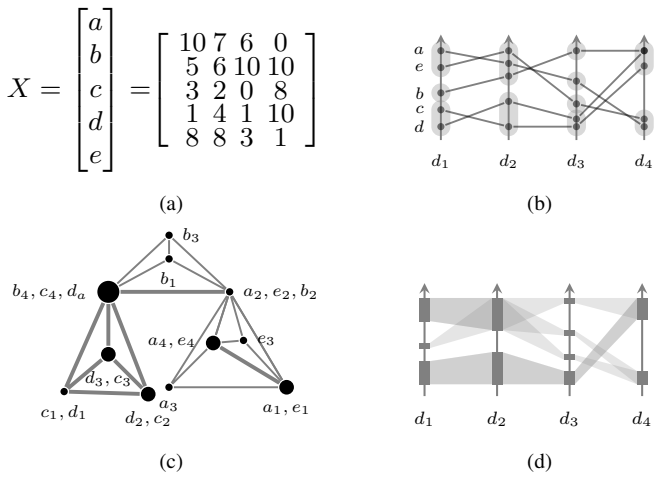


Figure 6: Aggregation of dimension values. (a) Example data (b) Parallel coordinate plot of the example data, augmented by per-dimension partitions. (c) Quotient graph. (d) Value-oriented abstract representation. Visual meta-node covers the vertical space of their interval and the opacity of visual meta-edges encode their cardinality.

the dimension sequence implies that some nodes and edges are replicated. On the corresponding parallel coordinate plot, the nodes are positioned based on the order of their dimension in the sequence (horizontally) and their weight (vertically).

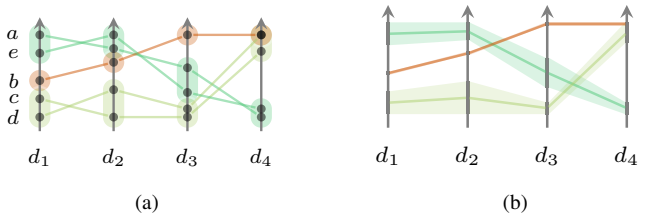
An abstraction of a parallel coordinate plot depends on a clustering or partition of tuples or dimension values. Similarly, abstracting the clique graph depends on a node partition that should be consistent with the partition D introduced before for the result to be drawable onto distinct axes. An example of such node partition is illustrated on Figure 5e by shadowed regions on grouping identical values from the same dimension. A node partition R consistent with D is called a *refinement* of D, noted $R < D$. Abstracting a graph G based on such node partition consists in merging the nodes of G belonging to the same subset of a partition R and merging edges in consequence. For instance, abstracting the graph G from Figure 5e based on its shadowed regions leads to the graph of Figure 5f. This process corresponds to taking the *quotient graph* of G relative to R, defined as the graph whose nodes are the parts of R and where a subset $S \in R$ is adjacent to a subset $S' \in R$, and only if, some node in S is adjacent to a node in S' with respect to the edges of G. The quotient graph of G relative to R is noted $G_{/R}$. We call *meta-nodes* and *meta-edges* the nodes and edges of a quotient graph.

The quotient graph holds the structure of an abstract plot. To draw a complete abstract parallel coordinate plot, properties are computed for *meta-nodes* and *meta-edges* along the aggregation, and then mapped to the visual properties of their visual representations. To this end two aggregation functions are defined to compute the properties of the *meta-nodes* and *meta-edges*. Given a set of edges or nodes as input, they return a sequence of weights (cardinality, extrema, mean, etc) that are subsequently used for assigning visual properties to meta-edges and meta-nodes. For instance, on Figure 5b, the opacity of lines on the parallel coordinate plot depends on the cardinality of meta-edges. In this example, the aggregation of identical values allows finding the optimal number of lines to draw. Indeed, the aggregation done here in data-space relates to the one done in screen-space when rendering the polylines with alpha composing. The same process is applied to obtain an abstract representation given some aggregation functions and a *valid* partition R of V, that is, refinement of D. Note that the obtained quotient graph is d-partite: the nodes of each subset of D are independent since they belong to different tuples and this property remains true for quotient graphs since only nodes that are not connected are contracted into meta-nodes. Figure 6 presents an example of partition obtained with per-dimension clustering and another type of representation based on extrema values. Tuple-oriented approaches are incorporated into the formalism by refining the input tuple partition into a refinement of D. Each subset of the tuple partition can be decomposed into d groups along their membership in D. Figure 7 shows an example of tuple partition refinement and a tuple-oriented representation.

Using the formalism, the abstract representation for some data is entirely defined by: (i) a valid partition R of the nodes of the clique graph G for the data, (ii) two aggregate functions assigning weights to meta-nodes and meta-edges of the quotient graph $G_{/R}$, (iii) an axis layout and visual encodings for meta-node and meta-edge weights (vertical position, color, etc).

3.2. Hierarchical abstraction in parallel coordinates

Hierarchical aggregations are precomputation of different levels of abstraction that naturally support multiscale represen-



(a) (b)

Figure 7: Aggregation of tuples. (a) Translation of the tuple partition (color) into a node partition refining D (shadowed regions). (b) Tuple-oriented abstract representation using mean and extrema values.

tations. Such precomputation outputs a rooted tree structure whose leaves are the data objects to aggregate. Every node of a rooted tree is said to *cover* the leaves of the subtree it is the root of. An *antichain* (also called *cut*) in a tree is a set of nodes S, such that no node of S is an ancestor of another node of S. An antichain is *maximal* if no node can expand it without violating the antichain property. The subsets covered by the nodes of a maximal antichain form a partition of the tree leaves: the antichain property makes them pairwise disjoint, the maximal property ensures that their union is the set of leaves.

In our model, an abstraction is directed by a partition refining the partition D which is equivalent to d partitions, one for each V_i of D. By augmenting each subset V_i with a tree, such partition can be defined by one maximal antichain for each tree. This is equivalent to defining a single maximal and non-trivial antichain in the unified tree where the *root*'s children are the roots of all dimension trees. We note this unified tree $T(V)$, and call its direct subtrees *dimension hierarchies*.

Overall, for some input data modeled by the clique graph $G = (V, E, w)$ and its associated hierarchy $T(V)$, an abstraction is defined as previously described with the valid partition of V induced by a maximal antichain in $T(V)$. This unifying approach accommodates multiple layouts of parallel coordinates (see Figure 5) and several abstract parallel coordinates representations. We presented two abstract representations: per-dimension aggregation (Figure 6d) that corresponds to the technique presented by Palmas et al. [10], and tuple aggregation (Figure 7) like presented by Fua et al. [14]. For tuple aggregation, the hierarchy defined on tuples is used to form all dimension hierarchies. Figure 7 is obtained by cutting all dimension hierarchies at the same level.

4. Envisioning Interactions

For some input data which graph is $G = (V, E, w)$, an abstract view is defined by: a node partition R linked to a hierarchy $T(V)$, a sequence of dimensions, and optionally a selection of items to be emphasized. These parameters all have a preset value (e. g. the predefined hierarchy, the empty selection) that is to be modified incrementally by a corresponding user operation: hierarchy edition and navigation, axis re-ordering and subset selection. Figure 8 shows how these parameters and operations participate in the process of defining an abstraction using the graph model described in the previous section: axis reordering changes which meta-edges are displayed, selection translates to computing a subgraph and hierarchy navigation corresponds to edition of the node partition. Effectiveness and expressiveness are orthogonal objectives when designing an operation and how it modifies a parameter of the process. Expressiveness refers to the flexibility of an operation and can be conceived as the number of states possibly induced from one state by the operation. In our context, limiting the expressiveness of an operation may be motivated by the following: (1) usability, limitation to queries that can be expressed solely over displayed aggregate nodes and edges rather than the whole data, (2) efficiency, limitation of the extents to which an operation changes the abstraction to limit the induced network transfer or computation cost.

In this section, we take advantage of the formalism to explore different designs for these operations. In particular, we investigate the size of the incremental change induced on the abstract view and the complexity of the computation. The ultimate goal is the adequate limitations in expressiveness that guarantee bounded network transfer and linear computation on the server-side. Since computation costs depend on the data structure manipulated by the computing server and the structure of the hierarchy, they are further discussed in the next section. Since the model rests upon an automatically precomputed hierarchy, we consider that the weights (cardinality, extrema, etc) of all of the tree nodes are precomputed at the same time as the hierarchy. Then, operations that do not modify the hierarchy only amount to computing meta-edges for the current parameters.

4.1. Axis reordering

On a regular parallel coordinate plot of a d-dimension data set with d axes, only $d - 1$ of the $\frac{1}{2}(d - 1)$ 2D-subspace of the data are visible. A usual tool to explore the relationship between dimension with non-adjacent axes is axis reordering. Conceptually, a plot of abstract parallel coordinates for a given sequence of axes is obtained by taking the subgraph of the current quotient graph induced by the edges connecting nodes from neighboring axes in the sequence (step 4 on Figure 8). In practice, only the necessary meta-edges can be computed instead. Similarly, axis reordering can be conceived as the computation of the meta-edges that were not displayed on the previous ordering of axes. With axis insertion for instance, this applies to meta-edges joining two pairs of axes at most.

In the general case, computing the meta-edges of a subplot, i. e. the meta-edges connecting nodes from different dimensions, conceptually corresponds to contracting n bottom-level edges of the clique graph, where n is the number of tuples. In practice, it requires one pass over the n tuples to aggregate those that belong to the same meta-node on both dimensions. The size of the transferred data depends on the number of necessary meta-edges. On each subplot, the number of meta-edges depends on the number of meta-nodes of both axes of the subplot and the values of their covered tuples. If the number of meta-nodes per-dimension is bounded by a certain k, then the maximum number of meta-edges per-subplot is k^2.

4.2. Hierarchy navigation: drill-down and roll-up operations

The main limitation of abstract parallel coordinates is the limited amount of information conveyed by a certain level of

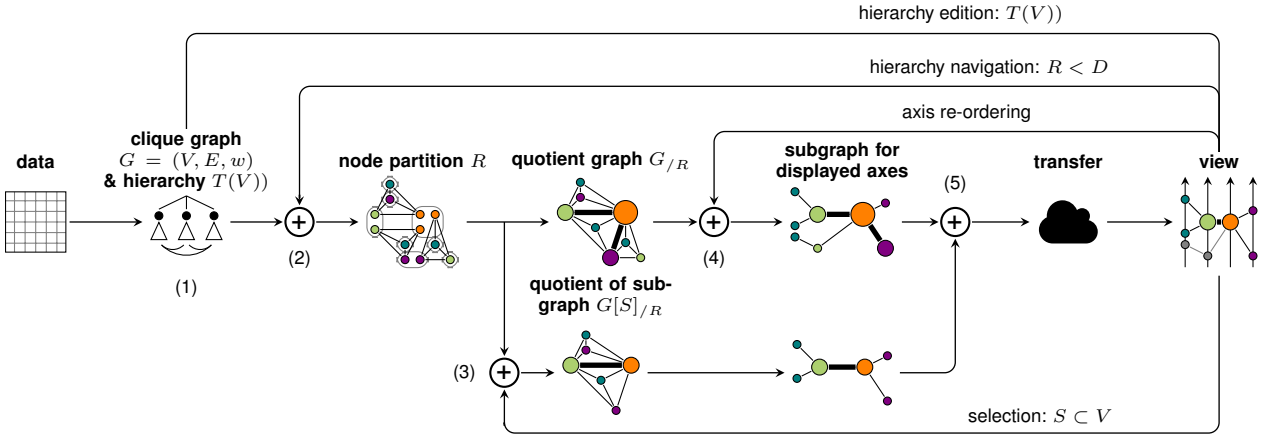


Figure 8: Schematic pipeline of the definition of an abstract view. The parameters of the pipeline are modified incrementally by their corresponding user operation: hierarchy edition and navigation, axis re-ordering and subset selection. (1) Modification of the current hierarchy. (2) Modification of the current node partition. (3) Aggregation functions for the quotient graph are separately computed over the subset of nodes S that correspond to the selected tuples. (4) Filtration of the meta-edges to retain only those shown given the provided ordering of dimensions. (5) The quotient graph is transferred with its associated properties (for the base graph and the selection).

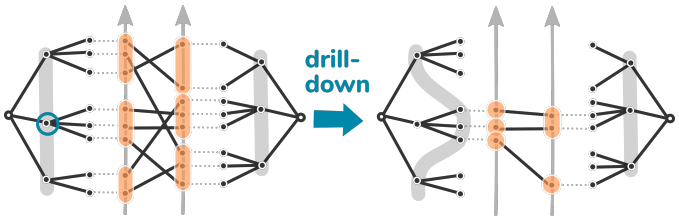
detail. Using a hierarchy that connects different levels of abstraction is a common manner of supporting smooth transition from one level of detail to another but also to enable changing the level of detail of parts of the view resulting in views with varying levels of detail. *Drill-down* and *roll-up* refers to operations that change the level of detail at which the data is rendered. In the context of an abstraction based on a hierarchy, drilling-down means showing nodes sitting deeper in the hierarchy, i. e. closer to the leaves, compared to those shown previously. On the contrary, rolling-up means showing nodes sitting upper in the hierarchy than those presented before. Deeper nodes present a finer aggregation of the base data, while upper nodes present a coarser aggregation of the data. If the aggregate functions are associative, at least parts of a roll-up can be computed based on the previous state rather than from the whole data.

Defining node partitions along a hierarchy, for instance with maximal antichains, helps create coherent changes from one partition to another. In general, the number of maximal antichains of a tree is still exponential relative to the number of leaves. For example, *complete* binary trees, i. e. binary trees with every level full except the last, have $\Omega(2^n)$ maximal antichains where n is their number of leaves. This makes the precomputation of all quotient graphs with respect to every possible maximal antichain for the given hierarchy not possible for the scale of data we aim to tackle.

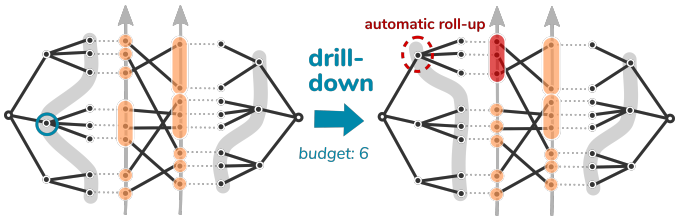
In our model, the displayed level of detail is given by the maximal antichain of the hierarchy on which is based the current node partition R. In general, this level of detail is *unbalanced*, meaning that the nodes of the maximal antichain do not necessarily sit at the same distance from the root in the hierarchy. A drill-down operation that globally increases the level of detail by replacing each meta-node of the antichain with its direct children may result in an uncontrolled number of elements on display, independently of the arity and depth of hierarchies. Moreover, the incremental changes required are exponentially larger as the antichain nodes come closer to the leaves. Therefore, this approach does not offer satisfying guarantees on the size of transferred data. Drilling-down on a single meta-node at once reduces the size of the incremental change but lives to the user the responsibility to manage the number of visual aggregates on display by rolling-up on parts that are no longer of his interest.

In the following, we detail three drilling approaches that aim to limit the increase in the number of visual elements when changing the level of detail by reducing the context. Figure 9 illustrates these approaches on 2-dimensional data. Each tuple is represented by a line linking a point from the left axis (first dimension) to a point from the right axis (second dimension). Each axis is augmented by its corresponding dimension hierarchy with the leaves connected to a tuple point. On hierarchies, we represent in grey the set of nodes that form the current maximal antichain, and the corresponding node partition or meta-nodes are represented in orange on the axes. The *arity* of the tree is defined as an upper bound on the number of children found in the tree and the *depth* of the tree is the maximal distance between a node and the root.

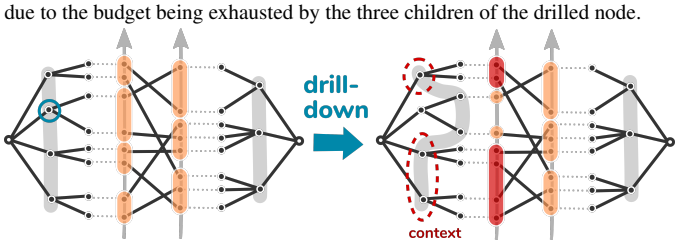
Detail & filter. A first approach is to define drill-down on a meta-node as a filtering operation. Here the context corresponds to all tuples not covered by the drilled node and is removed from all meta-nodes and meta-edges. This corresponds to computing the next quotient graph over the filtered clique-graph that only contains the cliques for the tuples covered by the drilled node. A the same time, the drilled node is replaced by its children in the maximal antichain. Figure 9a presents an example on two axes. The drilled node, circled, covers three tuples therefore only these three tuples are aggregated after drill-down. Notice that on the left axis only the children of the drilled-node remain while on the right axis, one node was removed and the subset of tuples covered by the two others changed. This method effectively bounds the number of aggregates on an axis: throughout drill-down, each axis displays no more nodes than those ini-



(a) Detail & filter. The clique of the tuples not covered by the drilled node are filtered out: the drilled abstraction corresponds to three cliques.



(b) Budgeted detail. The top three nodes are automatically rolled-up (in red) due to the budget being exhausted by the three children of the drilled node.



(c) Dynamic context aggregation detail. Nodes from the maximal antichain that are the deepest in the hierarchy are called focus nodes (in orange). Other nodes from the maximal antichain are merged into context meta-nodes (in red).

Figure 9: Examples of the three drill-down strategies applied to 2-dimensional data. The clique graph and its hierarchies are represented in black. On both sides, the current maximal antichain is shadowed in gray and the corresponding meta-nodes are represented as shadowed regions over their covered leaves. Meta-edge are not represented. On the left, ⊙ denotes the drilled node.

tially presented or the arity of the hierarchy. In turn, this bounds the number of meta-edges which offers a guarantee on the size of the transferred data for this operation and others. The computational and transfer costs of the operation are the same as the cost of a meta-node selection, i. e. a selection of the set of tuples covered by the drilled node. The limit of this approach is that context information is removed from all subplots at once to account for the change of displayed tuples and thus preserve the coherence of the plot. Since the resulting abstraction is computed on less tuples, the weights of the displayed meta-nodes are updates and thus their visual properties as well. For instance, on Figure 9a, the nodes remaining on the right axis after the operation have changed cardinality and extrema which may change the size and position of their visual items. A drawback of this approach is that the whole plot undergoes visual changes which may be unpredictable on all axes other than the one of the drilled node and therefore difficult to apprehend.

Budgeted detail. A second approach lies in constraining the definition of maximal antichains such that their size complies with a predefined budget. Indeed, the size of the current antichain relates to the number of meta-nodes of the quotient graph and therefore the number of aggregates transferred and displayed. With this approach, drilling potentially implies modifying the current maximal antichain in multiple points such that previously acquired detail automatically collapses for new detail to be added when the budget would have been exceeded otherwise. For instance, Figure 9b present an example with a budget of 6 nodes per axis. Before drill-down the left and right axes respectively display 6 and 5 nodes. Without automatic roll-up, the left axis would display 8 nodes after drill-down on the circled node. To meet the budget, the three top nodes are rolled-up, resulting in 6 nodes on the left axis. The right axis remain unchanged. If these changes are restricted to a single dimension hierarchy, the visual changes only affect one or two subplots, which addresses the limitation of the detail & filter method. Computing these subplots costs a pass over all tuples for each subplot, in addition to the cost of finding a suitable maximal antichain. To achieve item-level detail, the current maximal antichain should contain at least one leaf. The minimal budget allowing to define such maximal antichain depends on the arity and depth of the tree. In a k-ary tree with n leaves and depth h, a maximal antichain containing at least one leaf and being minimal in size can have between h and $k \cdot h$ nodes. Considering the example of binary trees which nodes all have either 0 or 2 children, the minimum depth is $\lceil log_2(n) \rceil + 1$ and the maximum depth is n. Consequently, a visual budget allowing gaining detail up to the leaf level has to be chosen with respect to the depth of the hierarchy. For instance, on Figure 9b, the budget has to be at least 5 to display item-level detail on any leaf of the left and right dimensions. Since this depth can reach orders of magnitude same as the number of tuples in general, budgeted detail does not scale without strong constraints on the properties of the hierarchy.

Dynamic context aggregation. An alternative solution to collapsing nodes when focusing on deeper nodes is to aggregate them dynamically. The nodes from the chosen antichain that are not in focus are displayed aggregated which lowers their impact on the visual item budget. In general, these aggregates do not correspond to existing nodes in the hierarchy. For instance, on Figure 9, the bottom context node does not directly correspond to a single node in the hierarchy, but to two. With an additional constraint on the arity of dimension hierarchies, this approach enables drill-down up to the deepest level while complying with a visual budget. Indeed, with f foci per axis and a k-ary hierarchy, an axis hold at most $f - 1$ context nodes and fk *focus* nodes, i. e. non-context nodes.

4.3. Hierarchy edition: split and merge operations

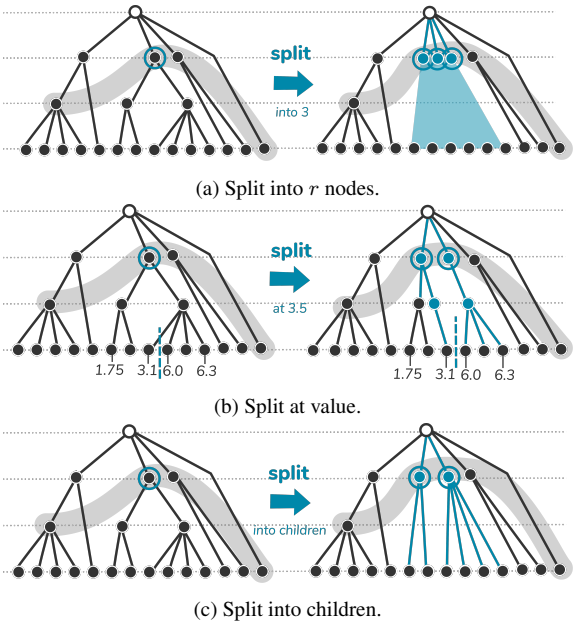
Relying on a predefined or precomputed hierarchy to explore the data presets how items are grouped together at a certain level of detail. To address this limitation, editing tool can be provided to the user The *split* and *merge* operations aim at modifying the current partition of the tuples. They provide the user with a manner of correcting the flaws of the automatically computed hierarchical structure, propose a structure that better reflect the similarities of the values, and more generally, persisting any desired change in the hierarchy. Similarly to drill-down and roll-up, the immediate change on the view is a coars-

ening or refinement of the meta-nodes on an axis (and consequently meta-edges). However, while drill-down and roll-up are merely navigation operations, split and merge are also editing operations which adds another computing step to the operation. Additionally to the computation of the incremental change in meta-nodes and meta-edges needed to update the view, the hierarchy has to be edited to persist the change on the server-side. Changes in the hierarchy may affect large subtrees which can be computationally expensive. In practice, the cost of editing the hierarchy on the back-end is implementation-dependent. In many cases, it is possible to compute the changes necessary to update the view first, and delay the computation of structural changes in the hierarchy such that the interaction delay only reflect the cost of the former computing step.

Split refers to the substitution of a node of the hierarchy with several new others. In general, substituting a node in the hierarchy induces changes in the structure of the subtree it is the root of. Merge refers to the substitution of two or more nodes with a single one. In general, merge is less expensive than split since the incremental changes may be computed directly over the current aggregates rather than the whole data, if the aggregate functions are associative. For both cases, the size of the incremental changes necessary to update the view depends on the number of newly introduced nodes and the current number of meta-nodes on the neighboring axes. For r the number of substitute nodes ($r = 1$ with merge) and k the maximum number of meta-nodes on the neighboring axes, the number of meta-edges to transfer is $O(r \cdot k)$. We present several designs for the split and merge operations and describe their cost regarding hierarchy edition and more precisely tree edge edits. In our model, hierarchy leaves correspond to nodes in the clique-graph and thus are weighted by a value. In each dimension hierarchy, the order of these values induces an order of the leaves and a level-order on each node of the hierarchy, i. e. nodes of the same depth are ordered. Figure 10 and Figure 11 illustrate each version of the presented operations. Trees are drawn with all leaves aligned to emphasize this order. We first describe three strategies for the split operation, from the most expressive to the least.

Recluster into r. The most general manner of splitting a node is to divide it into a chosen number r of nodes. In the general case, the whole subtree of the split node has to be replaced to split the node into r nodes while preserving the rest of the hierarchy tree structure (see Figure 10a). This can be done for instance by running a hierarchical algorithm on the set of the leaves covered by the split node. The drawback of this approach is that obtaining a meaningful hierarchical clustering most likely has a high computational cost for large subtrees, not manageable during interactions. At least, the operation costs a traversal of the r subtrees obtained.

Cut at value x. A second manner to split a meta-node is to replace it by two nodes which subtrees respectively cover values higher and lower than a certain value x that belong to the interval covered by the split meta-node. We say a node of the tree *covers an interval* $[a, b]$ when a and b are the extrema of the values for the leaves it covers. Leaves cover degenerate intervals,



(a) Split into r nodes.

(b) Split at value.

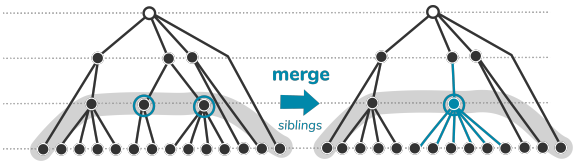
(c) Split into children.

Figure 10: Example of the three split strategies applied on one node of the hierarchy. On the left, ⊙ denotes the split node and on the right it denotes its substitutes. In blue: tree nodes and edges that were modified by the split. In gray: an example of maximal antichain under which the split can be applied.

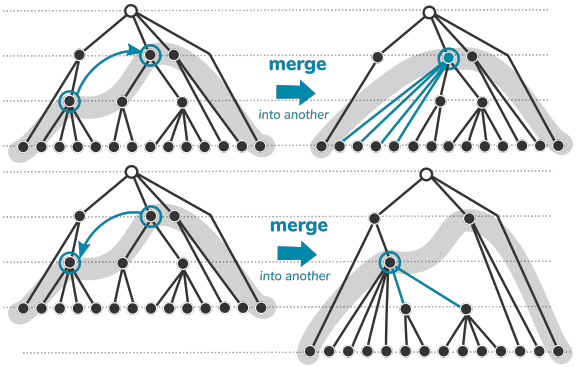
i. e. consisting of a single value. To cut a meta-node m at x, we start by splitting m into two nodes m_- and m_+ and proceed to recursively split each of its descendants that covers an interval including x. The process starts with the children of m. Children of m that covers values higher (resp. lower) than x are assigned to m_+ (resp. m_-). In the worst case (and considering that all leaves of the same value have the same parent), there is only one child of m that covers an interval including x. In that case, that node has to be split as was m and the same process is applied to its children. Supposing the traversed tree is k-ary with depth h. At most, h nodes are split, which amounts to $O(kh)$ edge edits, one for each child of a split node.

Replace by children partition. A third manner to split a meta-node is to replace it by a partition of its children. Conceptually, this corresponds to at most adding the same number of children the split node has. For a k-ary tree, it amounts to editing the $O(k)$ edges to connect the new nodes to their parents and $O(k)$ edges to reassign the original children to the nodes replacing the split node. This strategy is the least expensive of the three. In its effect, this operation is closely related to the drill-down operation, without the reduction of context. Figure 10c illustrates the case when a node is replaced by its children. Depending on how the tree is stored, this may imply passing over as many tuples as the split node covers.

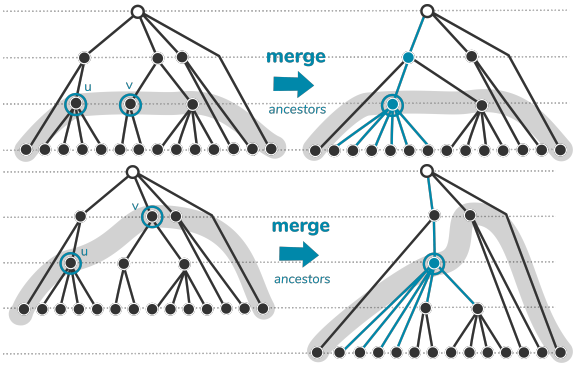
We then detail strategies for the merge operation. Two nodes can be merged only if they cover two consecutive sets of leaves, i. e. if no leaf is positioned between the two sets. If we merge two of these nodes, non-siblings, into a single one without further changes to the structure of the tree, we end up with the resulting node having two parents. We first describe the sim-



(a) Merge siblings.



(b) Merge into.



(c) Merge ancestors.

Figure 11: Examples of the two merge strategies applied on pairs of node of a dimension hierarchy. On the left, ⊙ denotes the merged nodes and on the right it denotes their substitute. In blue: tree nodes and edges that were modified by the merge. In gray: an example of maximal antichain under which the merge can be applied.

ple operation of merging siblings and then present two more general strategies.

Merge siblings. Merging two siblings is straightforward: it consists in merging the two nodes and reassigning the children of the two merged nodes to the resulting node (see Figure 11a). This amount to $O(k)$ edge edits for a k-ary tree.

Merge into another. A first solution for merging non-siblings nodes is to merge one node *into* the other. It amounts to reassigning the children of the node merged into the other, thus $O(k)$ for a k-ary tree. If the two nodes have not the same depth, then the resulting tree may be deeper or flatter than the original one. This strategy has the advantage of introducing limited changes in the structure, however it is not commutative (see Figure 11b) and it may not be straightforward to choose in which way to apply it when only presented with the current tree cut. Put it another way, choosing the way in which to apply the merge is a matter of choosing which of the two chains of ancestors the user which to keep for the children of the merged nodes. This decision is not straightforward when the structure of these ancestors are not displayed. Note that merging siblings can be seen as a particular case of this operation.

Merge ancestors. A second solution for merging non-siblings nodes is to merge their ancestors i. e. the chain of nodes from their parent to the tree root. In practice, the two nodes may share some ancestors, therefore, ancestors are only merged from their parent to their deepest common ancestor. Contrary to the previous strategy, this operation is commutative. In the particular case where the two nodes to merge have the same depth, the number of pairs of ancestors to merge is the same (see top example of Figure 11c). In the general case, the deepest of the two nodes to merge contributes new levels of ancestors to the children of the other. This results in the tree being deepen (see bottom example of Figure 11c). This operation is the opposite of the cut-at-value operation: the cost of merging ancestry depends on the depth of the tree. For a k-ary tree of depth h, at most h pairs of ancestors are merged. This corresponds to $O(kh)$ edge edits, one for each child of a merged node. Note that merging siblings can be seen as a particular case of this operation as well.

4.4. Tuple selection

In general, selection consists in highlighting a part of the tuples across the whole plot as illustrated on Figure 1e. On an abstract representation, since all tuples do not necessarily map to a single visual element, a selection of tuples may be represented by gauges on visual aggregates (meta-nodes and meta-edges) that map to the percentage of selected tuples among their covered tuples. With hierarchical navigation, a dimension could represent nodes from different levels of the hierarchy, including *tuple-level* nodes, i. e. leaves, that represent a single item or items sharing the same values for that dimension.

A selection of m tuples corresponds to computing the quotient graph of the subgraph induced by the nodes of these tuples' cliques. In practice it requires going through m tuples per subplot to compute their contribution to the current meta-edge and meta-node weights (step 3 on Figure 8). The size of the transferred data, the result, is at most of the same order as the whole abstraction, i. e. $O(sk^2)$ meta-edges for s the number of different subplots and k the maximum number of meta-nodes per axis.

5. Scalable System with Focus+Context Representation

Our goal is to enable hierarchical exploration in abstract parallel coordinates while complying with the following scaling properties. On the representation-side, exploration should be possible down to the item level, in a top-down manner, while the number of visual items on display should be limited for any size of input data. On the processing-side, network transfer latency between the displaying unit and the computing unit should be controlled throughout interactions. In this section, we present a prototype client/server system through three aspects:

(1) the conceptual choices that address perceptual scalability and bounded data transfer, (2) the corresponding focus+context representation, and (3) server-side implementation details. In the proposed system, the client is the visualization endpoint and the server is an interface to an on-demand computing and pre-processing back-end. The interactions supported by this prototype are axis reordering, aggregate selection, node drill-down and roll-up.

5.1. Conceptual choices and bounds

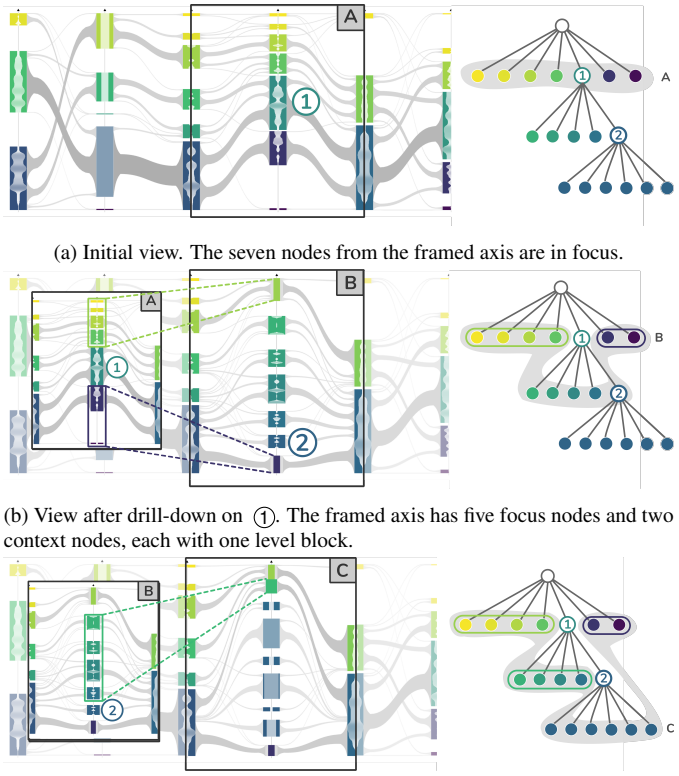
Conceptual choices arise from requirements of the visual representation and from the network transfer bottleneck. First, the targeted visual representation expects meta-nodes of each axis to represent non-overlapping interval of values. Secondly, bounding visual items is essential for perceptual scalability but also ensures that data transfer between our rendering client and back-end unit remains bounded in size thus predictable in time. To this end, we introduce three constraints (C1, C2 and C3) on the hierarchy structure and the drill-down operation.

Hierarchy constraints. We use a user-defined k value that acts as a *resolution parameter*, bounding the number of visual items per displayed axis. This parameter is enforced as the maximal arity of dimension hierarchies, meaning that internal nodes of dimension hierarchies should have between 2 and k children (C1). A k-ary tree can theoretically have its depth in $O(n)$ (degenerate tree), however, we expect the hierarchy to be more *compact* with $h \ll n$, where h is the depth of the tree (C2). Additionally, the order of leaves of each dimension hierarchy should follow the order of their value such that every defined node partition is also an interval partition for each dimension (C3). This property ensures that visual node are drawn without overlap. Binning (equal range partitioning) and adaptive binning (equal size partitioning) are examples of partitioning algorithms that can be applied in a bottom-up fashion to produce hierarchies complying with these requirements. To bound the number of nodes of each quotient graph, we adopt the dynamic context aggregation strategy (cf section 4.2) for drill-down.

Drill-down operation. We define as focus nodes, the nodes that have the maximal depth in the current antichain. The rest of the antichain nodes from one dimension are aggregated into the minimal number of context nodes such that their order is preserved (see Figure 9c). In the proposed implementation, the top-level nodes are initially presented as focus nodes and each drill-down triggers a dynamic aggregation of other nodes. Without consideration for split and merge operations, it means that all focus nodes are necessarily siblings. Consequently, there is no more than k focus nodes and two context nodes at once per axis. Thus, the number of nodes on display is bounded by $k+2$ per axis and the number of edges by $(k+2)^2$ per subplot. Since focus nodes are necessary siblings, the number of focus states of a hierarchy is its number of internal (i. e. non-leaf) nodes.

5.2. Focus+Context representation and interaction

We propose a focus+context view extending the representation presented by Sansen et al. [11]. This representation



(a) Initial view. The seven nodes from the framed axis are in focus.

(b) View after drill-down on ①. The framed axis has five focus nodes and two context nodes, each with one level block.

(c) View after drill-down on ②. The framed axis has six focus nodes and two context nodes. The upper context node has two level blocks and the bottom one has one level block.

Figure 12: Focus+context representation. Example of successive drill-down operation on the same axis. Clicking on ① in state A leads to state B. Clicking on ② on state B leads to state C. The tree on the right illustrates part of the dimension hierarchy of the framed axis with the current antichain in gray. It is represented here for explanation purpose but is not part of the visualization.

included two visual encodings: one oriented towards the distribution of tuples, the other towards the distribution of values themselves. Basically, the first encoding linearly maps aggregate height to their cardinality, while the second maps it to their covered interval size (distance between extrema). In both encodings, we use width to mark a visual node as context or focus: context nodes are displayed with slightly lower width to emphasize focus nodes and mimic a fisheye effect (see Figure 12b). As regions of interest are refined by successive drill-down, focus nodes tend to cover increasingly small interval and increasingly fewer tuples. To ensure focus nodes remain visible throughout drill-down despite their covered region being smaller relative to the whole, node height is rendered with a different scaling factors for context and focus nodes. This produce a distortion in the way node height is mapped and makes the design scalable. Figure 12 presents an example of two consecutive drill-down on the same axis of a representation where node height relates to cardinality. Notice on Figure 12b that the focus nodes approximately occupies 80% of the vertical space while their parent node, annotated ①, occupies roughly 25% of the vertical space on Figure 12a, the parent state.

Context nodes are augmented by a stack of *level blocks* that summarizes the number of drill-down operations (see Fig-

ure 12). The wider these blocks are, the lower in the hierarchy are the nodes they represent. Notice on Figure 12c, the difference in width of the level blocks on the upper context node: the inner block is wider since it corresponds to nodes closer to the focus nodes in the hierarchy. The height and vertical order of level blocks follow the same encoding as those of focus nodes. Level blocks are computed on the client side and thus not transferred nor counted in the visual item budget.

In drilling mode, focus nodes and level blocks are clickable. Clicking on a focus node triggers its animated expansion which split it into its children and merges its siblings into level blocks. Level blocks represent an aggregation of nodes previously in focus, thus they enable going back to this precise previous state. For instance, on Figure 12 clicking on the purple bottom level block in state C or B takes the user to state A.

5.3. Implementation details

The implemented system supports moving axes, selecting nodes and edges, drilling-down by selecting a focus node and rolling-up by clicking on a context part. The back-end was implemented both in a distributed environment for a Hadoop cluster and as a multi-threaded application for single machines (desktop computer or dedicated server). It runs on-the-fly computation. Past a certain number of input records (for a given number of dimensions and resolution parameter), a distributed platform should be more efficient while facilitating load expansion (see subsection 6.2).

The client is a WebGL application that displays the representation, computes level blocks and context aggregation on drill-down, and queries the supporting back-end for other interactions. The back-end server is a long-lived Spark [36] application which runs distributed job on demand while keeping prepared data in memory. It computes dimension hierarchies in an initialization step and stores the resulting node weights (extrema and cardinality) in a HBase distributed database. The hierarchies result from a hierarchical clustering computed in a bottom-up manner using an adaptation of Canopy clustering [37] on each column of the input data. The memberships of each input data value are stored in a *hierarchy matrix* of the same size as the input data, where each value holds the list of computed ancestors for the matching input data value. This matrix is kept in memory and split among computing units which will pass over their slice of the data to filter and aggregate results on demand. Upon user interaction and if necessary, the client requests the server which in turn runs a distributed operation and merges the partial results returned by computing units. All server responses correspond to an abstraction or parts of an abstraction. As such, they amount to a bounded number of float values. Finally, the client receives the incremental changes in plain text and updates the view consequently.

6. Use Case & Performance Evaluation

To validate the proposed implementations, we first present a use case showing the application of the focus+context representation and hierarchical navigation to a real-world dataset. Secondly, we present a performance evaluation of the server-side component using synthetic data.

6.1. Use Case

The 1990 US Census dataset from the UCI Machine Learning Repository [38] is a 1%-sample of the Public Use Microdata Samples person records drawn from the full 1990 census. The whole dataset has 125 attributes and about 2.5 millions records corresponding to individuals from the sample. Individuals have various attributes: most are binary flags (e. g. *Worked in 1989*, *Language Other Than English*), others are categorical (e. g. *Place of Birth*), ordinal (e. g. *Ability to Speak English*) or numerical (e. g. *Age*). An appendix gives hierarchies for several categorical attributes. To be able to exemplify the hierarchical interaction and focus+context representation, we mainly focused on numerical attributes and categorical attributes have hierarchies provided. We looked at the entire set of individuals and selected the 8 following individual attributes: country of birth (`POB`), work environment/field (`INDUSTRY`), job title (`OCCUPY`), `AGE`, `SEX`, number of children (`FERTIL`), and the poverty level (`POVERTY`).

In this study, we looked into the relationship between different job titles and environment (based on the `OCCUPY` and `INDUSTRY` attributes) and other individual characteristics following the analysis previously carried out by Vosough et al. [16] on the same dataset. Figure 13 presents the initial view: the `POB`, `INDUSTRY`, and `OCCUPY` attributes are clustered following the provided hierarchies. `SEX` and `FERTIL` are flat categorical attributes with `FERTIL` having 14 values ordered from bottom to top. The `AGE` and `POVERTY` are numerical attributes, hierarchically clustered so that each hierarchy node has 15 children. The initial view presents an overview of the distribution of dimension values and the the relationship between dimensions, with the height of nodes and thickness of edges conveying the number of covered individuals. It immediately shows that most individuals were born in the United States and that the sample seems to have about the same number of individuals in each sex and age category. It also directly shows that the `FERTIL` attribute only has values for female individuals since no link connect the male node to `FERTIL` node other than the one labeled n/a.

On Figure 14, we drilled-down on the three first axes to focus on individuals born in the United States, and holding managerial and professional specialty occupations in manufacturing. The categories just under the focused nodes in the hierarchies are depicted vertically enlarged compared to the rest of the nodes that are consequently shrunk on the sides in addition to being merged together. This distortion renders the links between nodes of these axes discernible, although they correspond to only a small number of individuals of the whole dataset. In particular, it allows to select the link corresponding to individuals having professional specialty occupations in manufacturing of durable goods which corresponds to 11215 individuals (0.46%). From a certain level of detail, selected groups of individuals may be too small compared to other levels of detail for the portion of covered individuals to be readable and comparable. In that case, hovering over nodes and edges allows

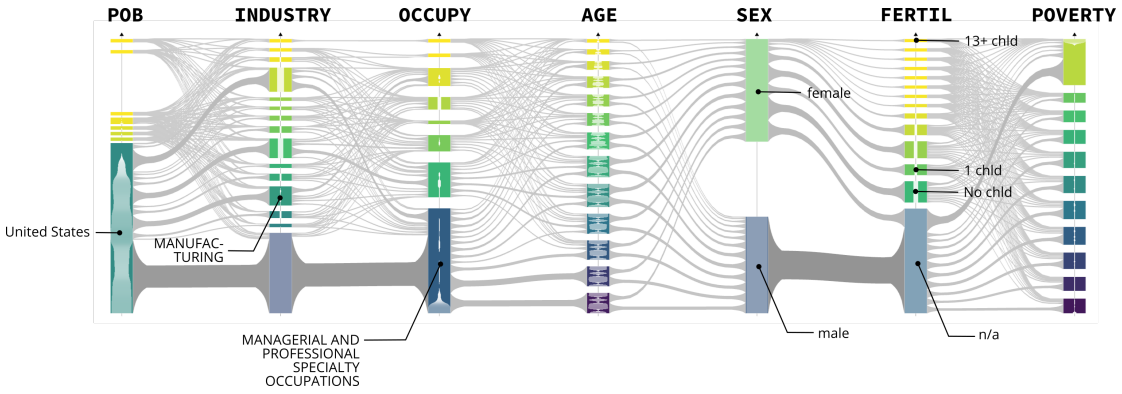


Figure 13: Initial view of the 1990 US census data for 2.5 millions individuals.

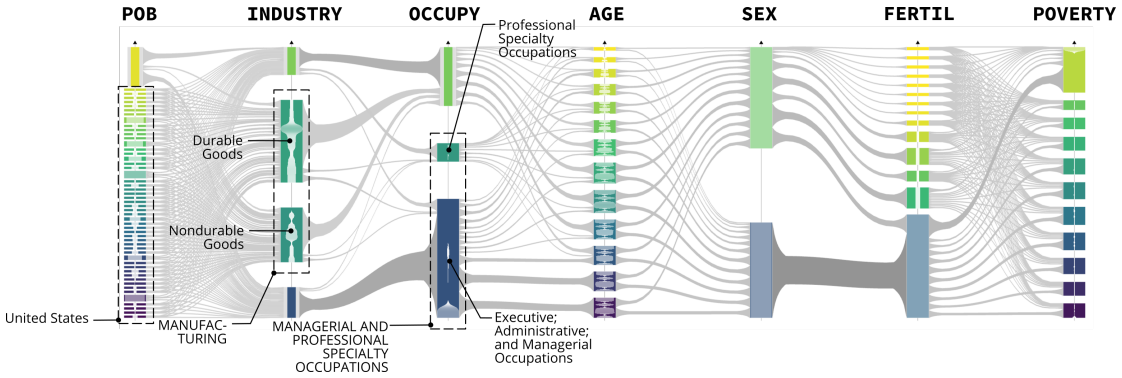


Figure 14: View following 3 drill-down operations on: the "United States" (`POB`), "Manufacturing" (`Industry`) and "Managerial and professional specialty occupations" (`OCCUPY`).

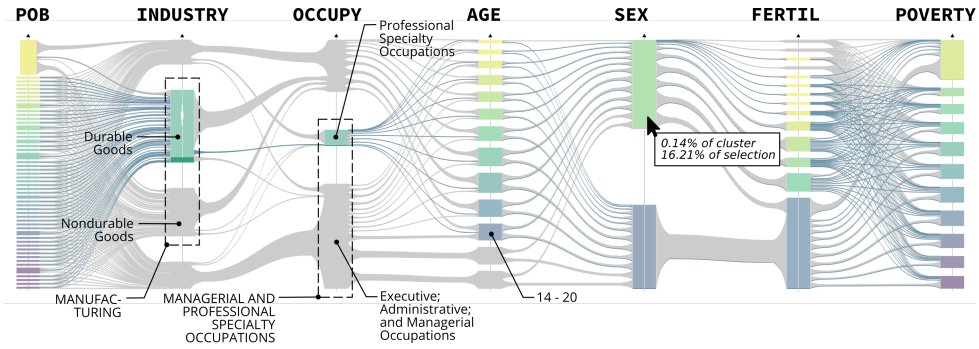


Figure 15: Selection of the individuals having professional specialty positions in manufacturing of durable goods. Hovering over the female node indicates that about 16% of these individuals are female.

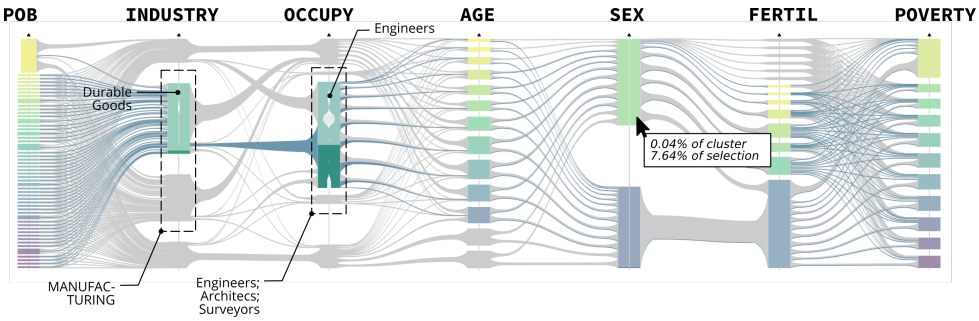


Figure 16: Selection of the individuals being engineers positions in manufacturing of durable goods. Hovering over the female node indicates that about 8% of these individuals are female.

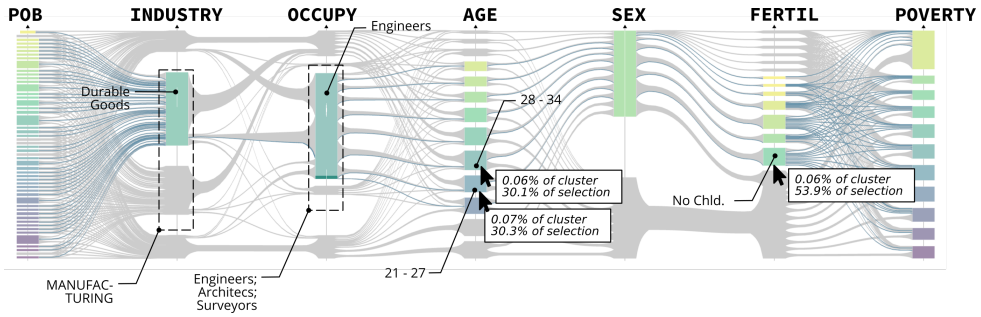


Figure 17: Selection of women working as engineers in durable good manufacturing.

to access the numerical information. Here, the resulting view shows that the selected individuals are 16 years old and older, which is expected for any selection of work position, and are primarily male since only 16% are female (see Figure 15).

After further drilling-down in the `OCCUPY` hierarchy, we selected the individuals working as engineers in manufacturing of durable goods which corresponds to 7255 individuals (0.30%), 7.64% of which are female. From the selection, we see that these individuals generally do not have many children (see Figure 16). By comparing Figure 15 and Figure 16, we also see that while some women had over 6 children among individuals working in the more general category of professional specialty occupations, in the engineer category, no woman has more than 6 children.

To refine the previous observations, we queried the intersection of the previously selected link (durable goods manufacturing and engineers) with the female node of the `SEX` axis. On the resulting view, presented on Figure 17, it appeared that most women working as engineers in durable good manufacturing have between 21 and 34 years old, and half of them do not have children. The small portion of women in this line of work is also apparent from the thickness of the shading of the edge for manufacturing durable goods and engineers. By comparing this view with the previous one, we can also see that, this work categories includes male individuals older than all female individuals.

The advantage of pushing context aside instead of filtering it is show on Figure 15), where the navigation depth is represented on the three first axes by their one-level context nodes. Context node and out-coming edges also provide information, although in reduced form, during selection. For instance, on Figure 16 and still on Figure 17, we see that the sample hold individuals – women – working as engineer on manufacturing of durable goods who were born outside the United State.

6.2. *Performance evaluation*

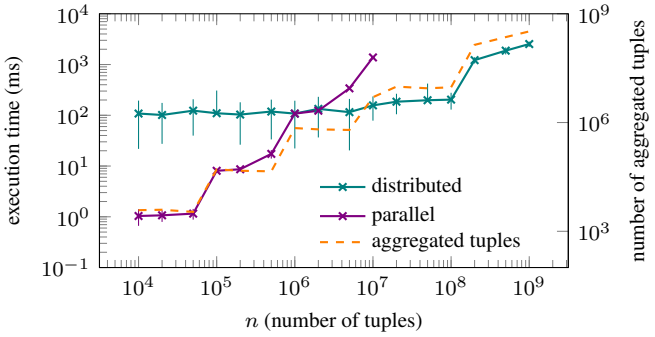
Since we are interested in supporting "$n \gg d$" datasets, we measure the execution times of the implemented operations for varying number of tuples n. The primary goal is to confirm that we observe a linear increase in computation time as the workload increases. To demonstrate the scalability of the distributed implementation, we also evaluate performance relative to the resources allocated for computation.

We performed tests for two different implementations: a multi-threaded single-computer implementation and a distributed implementation. The distributed implementation has the form of a long-lived Spark application running on a cluster of 15 computers. The parallel implementation is a C++/OpenMP application running on a high-end laptop computer. Each node of the distributed platform has 64GB RAM and 2x6 hyper-threaded cores at 2.10GHz each, connected via a 1Gbit/s network. The single computer used for running the multi-threaded implementation has 4 hyper-threaded cores at 2.7GHz and 32GB RAM. Test datasets are generated for varying n (from 10^4 to 10^9), with $d = 15$ and $k = 32$. Test datasets are generated such that pairs of dimensions present a close to null correlation factor [39] which tends to create close to the maximum number of edges between dimensions. Dimension hierarchies are generated using Canopy clustering [37] applied in a bottom-up manner. For all experiments, time measurements for execution are averaged over 1000 runs. For each type of operation, the most expensive operation is timed, i. e. the one that aggregates the largest number of tuples.

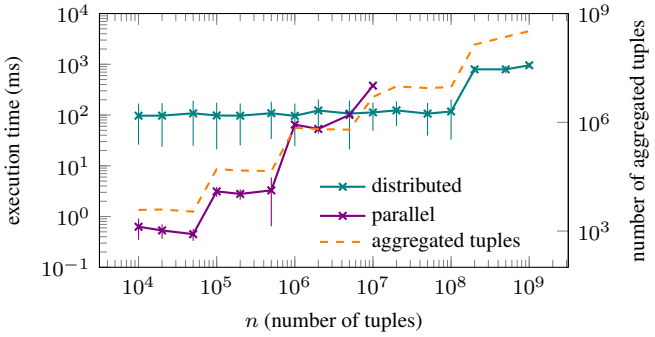
Execution time for two implementations

We first compare the execution time between the two implementations (distributed and parallel) for the four types of operations (selection of edge, of node, drill-down and roll-up) for varying size n of the generated dataset. Figure 18 shows the mean execution time and standard deviation for these experiments. Overall, for the four experiments, for n less than approximately 10^6, the distributed implementation underperforms the parallel implementation despite having more resources. We also note that, up until $n = 10^8$ the four operations run under 1s for the distributed implementation with the 15 executors used in these experiments. We remark that under $n = 2 \cdot 10^8$ (about 10^8 aggregated tuples), the performances of the distributed implementation are stable despite the increasing workload. Up until this limit, they also display very high variation, especially for the two selection operations. Along the two curves for execution, we plotted the number of aggregated tuples for each operation (with its values on the right axis). The curve for the number of aggregated tuples has a distinctive stairs shape which brings out the fact that the parallel implementation seems to scale linearly with respect to the number of aggregated tuples. We find the same trend for the distributed implementation past $n = 10^8$.

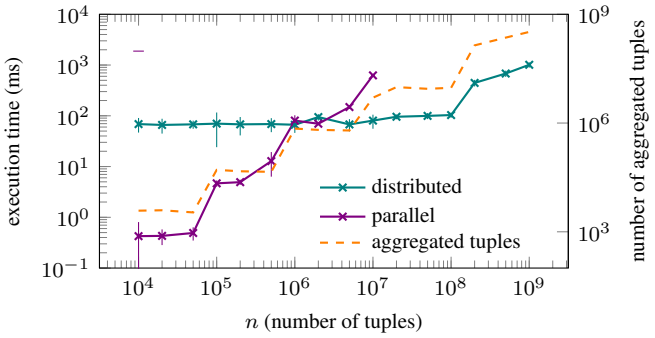
Due to the hierarchy constraints and the bottom-up approach



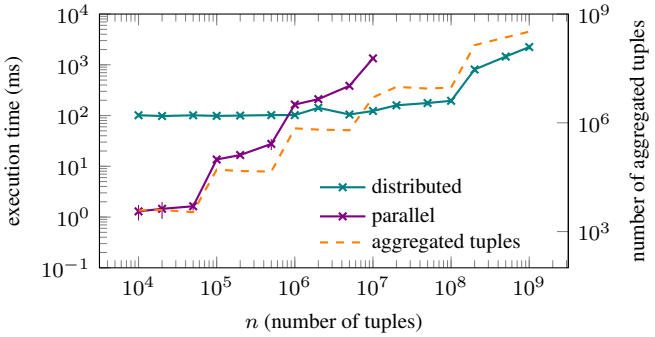
(a) Meta-node selection.



(b) Meta-edge selection.



(c) Drill-down.

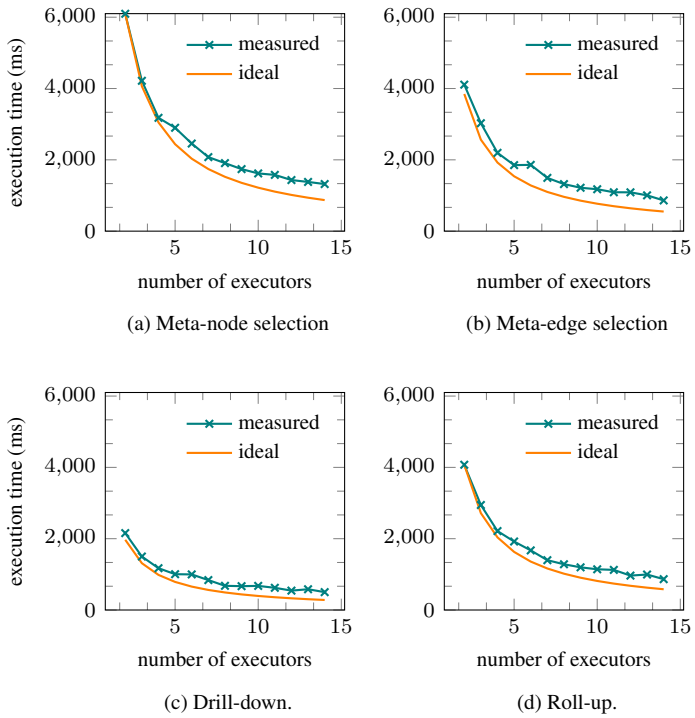


(d) Roll-up.

Figure 18: Execution times of different operations relative to the size of the dataset in number of tuples (n). Experimental results are averaged over 1000 runs on two implementations: distributed and parallel. The distributed version uses 15 executors. Errors bars are not symmetrical due to the log-log scale.

for clustering, the number of nodes on the top-level varies and the cardinality of the largest nodes does not increase linearly with n. This can explain the stairs-shaped curve for the number of aggregated tuples relative to n. The stability of the distributed implementation performances for the smallest datasets is not surprising and suggests that execution time for these datasets is dominated by costs related to network and disk I/O, or by merging all executor results by the *driver* unit. Indeed, the cost of merging results remains approximately the same as it is a function of the output size and the number of tasks running in parallel. Therefore, unless other assets are considered, it is preferable to use a classical implementation for datasets smaller than about 10^6 tuples since there is no efficiency gain in using the distributed infrastructure. An asset of the distributed infrastructure that may be considered is resilience. The high variation observed in the distributed execution time for the small datasets compared to larger ones may be explained by garbage collection: when the allocated memory is oversized, garbage collection incurs a substantial delay when triggered. Another explanation is that these variations are due to variations in network and disk I/O since computation time is dominated by their costs at these scales. The output of selection operations is larger than those of drill-down/roll-up operation, and consequently, the data transferred between the executors and the driver during the reduction step is also larger. This would explain why selection computations present more variation than the others.

Scalability of the distributed implementation



(a) Meta-node selection

(b) Meta-edge selection

(c) Drill-down.

(d) Roll-up.

Figure 19: Scalability test for the distributed implementation with $n = 2 \cdot 10^8$ and the number of executors varying from 2 to 14. Executors each have 12 cores and 31GB memory.

We then investigate the scalability of the distributed system

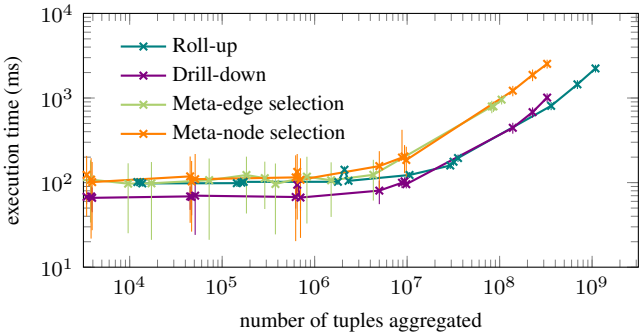


Figure 20: Comparison of the mean execution time between operations relative to the number of aggregated tuples for the distributed implementation.

using a dataset larger of $n = 2 \cdot 10^8$, that is a size for which execution time is not dominated by a fixed cost. One goal of our system is to limit interactive operations to linear complexity such that we fully benefit from the scalability of the distributed infrastructure. Therefore, we expect experimental results to be close to linear speedup (ideal). The speedup of execution is said linear when the execution time is halved when the number of executors is doubled. Figure 19 presents the mean execution time and standard deviation for the distributed implementation of four operations on a $2 \cdot 10^8$-tuple dataset with varying numbers of executors. The plotted *ideal* execution time corresponds to a linear speedup derived from the execution time using two executors. Overall, the results show that the system performs close to the ideal. Selection operations perform slightly worst, probably because their output is larger in size which implies more network transfer between executors and a more expensive merge between executor results.

Comparison between operations

Finally, we look at the execution time for the four operations with respect to the number of aggregated tuples. Figure 20 show the curves for the four operations for the distributed implementation. Above 10^7 aggregates tuples, the measures split into two groups: selection operations on one side and drill-down/roll-up on the other side. Although they follow the same trend, selection operations are more expensive by a significant factor. This relates to the fact that drill-down and roll-up correspond to changes on one dimension and therefore affect one or two subplots. Selections operations, however, affect all subplots at once and thus are more computationally expensive by a constant factor related to the number of axes on display. In our experiments, the representation displayed 15 different axes which correspond to 14 different subplots. If we were to implement edit operations, we expect the performances of the computation of incremental changes for merge and split to be similar to those of drill-down and roll-up for the same reason.

6.3. Predefined hierarchies and resolution parameter k

In this work, we aimed to design a system such that a strict upper bound on the number of visual items was guaranteed throughout interactions. This upper bound depends on a parameter k, that is chosen before exploration and used to initialize the precomputed hierarchy such that it is a k-ary tree. Precomputing a hierarchy has the advantages of providing a structure over which the user can navigate from the start; however, it is also limiting. By introducing hierarchical edition interactions to remedy this limit, we essentially allow the user to modify the arity of the hierarchy and therefore the upper bound on the number of visual items and therefore transferred data over the network upon interaction. At a certain point in exploration, this limit still exists but depends on the arity k' of the edited hierarchy. We can assume that, through interactions, the user changes the arity without necessarily changing its order of magnitude. In fact, even at the initialization, the effective arity of the hierarchy may be lower than k, depending on the used clustering algorithm, the approach (bottom-up or top-down) and the data. What is the most important, is not to guarantee a sharp bound on the number of visual items but rather to ensure that the arity of the hierarchy remains much less than n by several orders of magnitude.

7. Conclusion

We have presented a graph-based formalism for hierarchical aggregation on multidimensional data for parallel-coordinate representations. This conceptual model formalizes hierarchical aggregation over multidimensional data at the value level. This approach treats all dimensions equivalently which matches the way dimensions are handled in parallel coordinates. Moreover, we have shown that two types of aggregation used in previous work can be expressed with this model (tuple aggregation and 1D aggregation). The model closely corresponds to the representation and is useful for devising other abstractions and envisioning their interactions. In particular, we used the model to evaluate the number of visual elements transiting between the computing and rendering components of the system through interactions. We also presented several design possibilities for hierarchy navigation and edition.

The second contribution is a scalable system for visually exploring multidimensional data with a interactive multiscale parallel coordinates. The system uses a focus+context representation that allows displaying arbitrary detailed focus regions on each axis while maintaining a bounded number of visual items on display. The strength of this approach is that it supports exploration down to the item-level while ensuring perceptual scalability which also addresses the limiting lack of navigation of the previous system by Sansen et al. [11]. On the client-side, we proposed to use a focus+context view relying on distortion to ensure perceptual scalability. Focus nodes are expanded from the hierarchies and context regions are summarized and presented with simple navigation cues. One focus of the design is the bounded data transfer between the computing back-end and the visualization client. This bound relies on (1) a resolution parameter k, of small orders of magnitude, that bounds the hierarchy's arity and (2) a focus+context approach. A second focus of the design is that all operations have a linear computing complexity relative to the number of tuples to process. Experimental results demonstrate the scalability of the system relative to the size of the input data and to the resources

allocated for computation. The results suggest that the proposed system can support increasingly large datasets by expanding its network of computing units. To a certain extent, adding computing resources can also reduce interaction latencies.

A first direction of future work would be the evaluation of hierarchical parallel coordinates and its drill-down interaction from a user performance point of view. To do so, it would be necessary to measure user performance with different resolution parameters. Indeed, theoretically, the higher k is, the more information is displayed on the screen at the same time which should improve user performance. At the same time, we can also expect more latencies for higher k which could lead to worse user performance. With smaller k, interactions should be faster but it requires more user input to access an information, e. g. more drill-down to reach a certain level, and there would be less information on screen in general.

Secondly, methods for latency reduction other than horizontal scaling could be developed. A first possibility is using space as a trade-off for computation time as it was used by Sansen et al. [11]. The number of possible meta-edges makes their total precomputation unfeasible and less relevant if the hierarchy can be interactively edited. However, partial precomputation could be investigated: either beforehand or as a background process targeting meta-edges that are the most likely to be requested given the current state. The resulting precomputation would most likely need to be updated after hierarchy edition interactions. A second possibility is to consider switching the computation model to a progressive model following the type of system proposed by Moritz et al. [40].

Regarding the graph-based formalism, the model could be further generalized to incorporate other aggregations: 2D-subspace aggregation like presented by Novotný and Hauser [9] or dimension aggregation like presented by Andrews et al. [25] and Lex et al. [26]. Another direction would be to leverage existing work on graph mining for envisioning new interactions, for instance a selection interaction based on motif detection. Regarding the prototype system, we could implement the different strategies for split and merge and evaluate their usability in a user study.

Acknowledgments

This work has been carried out as part of the "REQUEST" (PIAO18062-645401) and "SpeedData" (PIAO17298-398711) projects supported by the French "Investissement d'Avenir" Program (Big Data – Cloud Computing topic).

References

[1] G. Richer, J. Sansen, F. Lalanne, D. Auber, R. Bourqui, Enabling hierarchical exploration for large-scale multidimensional data with abstract parallel coordinates, in: Proceedings of the Workshops of the EDBT/ICDT 2018 Joint Conference (EDBT/ICDT 2018), Vienna, Austria, March 26, 2018., 2018, pp. 76–83 (2018).

[2] D. Fisher, Big data exploration requires collaboration between visualization and data infrastructures, in: Proceedings of the Workshop on Human-In-the-Loop Data Analytics, HILDA@SIGMOD 2016, San Francisco, CA, USA, June 26 - July 01, 2016, ACM, 2016, p. 16 (2016). doi:10.1145/2939502.2939518.

[3] P. Godfrey, J. Gryz, P. Lasek, Interactive visualization of large data sets, IEEE Trans. Knowl. Data Eng. 28 (8) (2016) 2142–2157 (2016). doi:10.1109/TKDE.2016.2557324.

[4] J. H. T. Claessen, J. J. van Wijk, Flexible linked axes for multivariate data visualization, IEEE Trans. Vis. Comput. Graph. 17 (12) (2011) 2310–2316 (2011). doi:10.1109/TVCG.2011.201.

[5] A. Inselberg, B. Dimsdale, Parallel coordinates: A tool for visualizing multi-dimensional geometry, in: Proceedings IEEE Visualization '90, San Francisco, California, USA, October 23-26, 1990., IEEE Computer Society Press, 1990, pp. 361–378 (1990). doi:10.1109/VISUAL.1990.146402.

[6] G. P. Ellis, A. J. Dix, A taxonomy of clutter reduction for information visualisation, IEEE Trans. Vis. Comput. Graph. 13 (6) (2007) 1216–1223 (2007). doi:10.1109/TVCG.2007.70535.

[7] J. Heinrich, D. Weiskopf, State of the art of parallel coordinates, in: Eurographics 2013 - State of the Art Reports, Girona, Spain, May 6-10, 2013, Eurographics Association, 2013, pp. 95–116 (2013). doi:10.2312/conf/EG2013/stars/095-116.

[8] R. Kosara, F. Bendix, H. Hauser, Parallel sets: Interactive exploration and visual analysis of categorical data, IEEE Trans. Vis. Comput. Graph. 12 (4) (2006) 558–568 (2006). doi:10.1109/TVCG.2006.76.

[9] M. Novotný, H. Hauser, Outlier-preserving focus+context visualization in parallel coordinates, IEEE Trans. Vis. Comput. Graph. 12 (5) (2006) 893–900 (2006). doi:10.1109/TVCG.2006.170.

[10] G. Palmas, M. Bachynskyi, A. Oulasvirta, H. Seidel, T. Weinkauf, An edge-bundling layout for interactive parallel coordinates, in: IEEE Pacific Visualization Symposium, PacificVis 2014, Yokohama, Japan, March 4-7, 2014, IEEE Computer Society, 2014, pp. 57–64 (2014). doi:10.1109/PacificVis.2014.40.

[11] J. Sansen, G. Richer, T. Jourde, F. Lalanne, D. Auber, R. Bourqui, Visual exploration of large multidimensional data using parallel coordinates on big data infrastructure, Informatics 4 (3) (2017) 21 (2017). doi:10.3390/informatics4030021.

[12] O. Rübel, Prabhat, K. Wu, H. Childs, J. S. Meredith, C. G. R. Geddes, E. Cormier-Michel, S. Ahern, G. H. Weber, P. Messmer, H. Hagen, B. Hamann, E. W. Bethel, High performance multivariate visual data exploration for extremely large data, in: Proceedings of the ACM/IEEE Conference on High Performance Computing, SC 2008, November 15-21, 2008, Austin, Texas, USA, IEEE/ACM, 2008, p. 51 (2008). doi:10.1145/1413370.1413422.

[13] N. Elmqvist, J. Fekete, Hierarchical aggregation for information visualization: Overview, techniques, and design guidelines, IEEE Trans. Vis. Comput. Graph. 16 (3) (2010) 439–454 (2010). doi:10.1109/TVCG.2009.84.

[14] Y. Fua, M. O. Ward, E. A. Rundensteiner, Hierarchical parallel coordinates for exploration of large datasets, in: IEEE Visualization 1999, 24-29 October 1999, San Francisco, CA, USA, Proceedings., IEEE Computer Society and ACM, 1999, pp. 43–50 (1999). doi:10.1109/VISUAL.1999.809866.

[15] K. S. Candan, L. D. Caro, M. L. Sapino, PhC: Multiresolution visualization and exploration of text corpora with parallel hierarchical coordinates, ACM TIST 3 (2) (2012) 22:1–22:36 (2012). doi:10.1145/2089094.2089098.

[16] Z. Vosough, M. Hogräfer, L. A. Royer, R. Groh, H.-J. Schulz, Parallel hierarchies: A visualization for cross-tabulating hierarchical categories, Computers & Graphics 76 (2018) 1–17 (Nov. 2018). doi:10/gd82d2.

[17] Z. Liu, B. Jiang, J. Heer, *imMens*: Real-time visual querying of big data, Comput. Graph. Forum 32 (3) (2013) 421–430 (2013). doi:10.1111/cgf.12129.

[18] Z. Liu, J. Heer, The effects of interactive latency on exploratory visual analysis, IEEE Trans. Vis. Comput. Graph. 20 (12) (2014) 2122–2131 (2014). doi:10.1109/TVCG.2014.2346452.

[19] A. Perrot, R. Bourqui, N. Hanusse, F. Lalanne, D. Auber, Large interactive visualization of density functions on big data infrastructure, in: 5th IEEE Symposium on Large Data Analysis and Visualization, LDAV 2015, Chicago, IL, USA, October 25-26, 2015, IEEE Computer Society, 2015, pp. 99–106 (2015). doi:10.1109/LDAV.2015.7348077.

[20] G. Troullinou, H. Kondylakis, K. Stefanidis, D. Plexousakis, Exploring RDFS KBs using summaries, in: The Semantic Web – ISWC 2018, Vol. 11136, Springer International Publishing, 2018, pp. 268–284 (2018). doi:10.1007/978-3-030-00671-6_16.

[21] D. Archambault, T. Munzner, D. Auber, TugGraph: Path-preserving hierarchies for browsing proximity and paths in graphs, in: IEEE Pacific Visualization Symposium, 2009. PacificVis' 09., IEEE, 2009, pp. 113–120 (2009). doi:10/cxf92h.

[22] J. Wang, X. Liu, H. Shen, G. Lin, Multi-resolution climate ensemble parameter analysis with nested parallel coordinates plots, IEEE Trans. Vis. Comput. Graph. 23 (1) (2017) 81–90 (2017). doi:10.1109/TVCG.2016.2598830.

[23] G. P. Ellis, E. Bertini, A. J. Dix, The sampling lens: making sense of saturated visualisations, in: Extended Abstracts Proceedings of the 2005 Conference on Human Factors in Computing Systems, CHI 2005, Portland, Oregon, USA, April 2-7, 2005, ACM, 2005, pp. 1351–1354 (2005). doi:10.1145/1056808.1056914.

[24] H. Nguyen, P. Rosen, DSPCP: a data scalable approach for identifying relationships in parallel coordinates, IEEE Trans. Vis. Comput. Graph. 24 (3) (2018) 1301–1315 (2018). doi:10.1109/TVCG.2017.2661309.

[25] K. Andrews, M. Osmic, G. Schagerl, Aggregated parallel coordinates: integrating hierarchical dimensions into parallel coordinates visualisations, in: Proceedings of the 15th International Conference on Knowledge Technologies and Data-driven Business, I-KNOW '15, Graz, Austria, October 21-23, 2015, ACM, 2015, pp. 37:1–37:4 (2015). doi:10.1145/2809563.2809588.

[26] A. Lex, M. Streit, C. Partl, K. Kashofer, D. Schmalstieg, Comparative analysis of multidimensional, quantitative data, IEEE Trans. Vis. Comput. Graph. 16 (6) (2010) 1027–1035 (2010). doi:10.1109/TVCG.2010.138.

[27] J. Lu, Y. Zhang, J. Xu, G. Xiao, Q. A. Liang, Data visualization of web service with parallel coordinates and nodetrix, in: IEEE International Conference on Services Computing, SCC 2014, Anchorage, AK, USA, June 27 - July 2, 2014, IEEE Computer Society, 2014, pp. 766–773 (2014). doi:10.1109/SCC.2014.104.

[28] Z. Geng, Z. Peng, R. S. Laramee, J. C. Roberts, R. Walker, Angular histograms: Frequency-based visualizations for large, high dimensional data, IEEE Trans. Vis. Comput. Graph. 17 (12) (2011) 2572–2580 (2011). doi:10.1109/TVCG.2011.166.

[29] G. Andrienko, N. Andrienko, Parallel coordinates for exploring properties of subsets, in: Proc. of Int. Conf. on Coordinated & Multiple Views in Exploratory Visualization, 2004, pp. 93–104 (2004).

[30] N. Bikakis, G. Papastefanatos, M. Skourla, T. Sellis, A hierarchical aggregation framework for efficient multilevel visual exploration and analysis, Semantic Web 8 (1) (2017) 139–179 (2017). doi:10.3233/SW-160226.

[31] A. Cockburn, A. K. Karlson, B. B. Bederson, A review of overview+detail, zooming, and focus+context interfaces, ACM Comput. Surv. 41 (1) (2008) 2:1–2:31 (2008). doi:10.1145/1456650.1456652.

[32] M. L. Huang, T. Huang, X. Zhang, A novel virtual node approach for interactive visual analytics of big datasets in parallel coordinates, Future Generation Comp. Syst. 55 (2016) 510–523 (2016). doi:10.1016/j.future.2015.02.003.

[33] T. V. Long, L. Linsen, Multiclustertree: Interactive visual exploration of hierarchical clusters in multidimensional multivariate data, Comput. Graph. Forum 28 (3) (2009) 823–830 (2009). doi:10.1111/j.1467-8659.2009.01468.x.

[34] J. Heinrich, J. Stasko, D. Weiskopf, The Parallel Coordinates Matrix, in: M. Meyer, T. Weinkaufs (Eds.), EuroVis - Short Papers, The Eurographics Association, 2012 (2012). doi:10.2312/PE/EuroVisShort/EuroVisShort2012/037-041.

[35] M. Lind, J. Johansson, M. D. Cooper, Many-to-many relational parallel coordinates displays, in: 13th International Conference on Information Visualisation, IV 2009, 15-17 July 2009, Barcelona, Spain, IEEE Computer Society, 2009, pp. 25–31 (2009). doi:10.1109/IV.2009.43.

[36] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012, USENIX Association, 2012, pp. 15–28 (2012).

[37] A. McCallum, K. Nigam, L. H. Ungar, Efficient clustering of high-dimensional data sets with application to reference matching, in: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, Boston, MA, USA, August 20-23, 2000, ACM, 2000, pp. 169–178 (2000). doi:10.1145/347090.347123.

[38] D. Dua, C. Graff, UCI machine learning repository, Web page. URL http://archive.ics.uci.edu/ml

[39] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: D. Georgakopoulos, A. Buchmann (Eds.), Proceedings of the 17th International Conference on Data Engineering, April 2-6, 2001, Heidelberg, Germany, IEEE Computer Society, 2001, pp. 421–430 (2001). doi:10.1109/ICDE.2001.914855.

[40] D. Moritz, D. Fisher, B. Ding, C. Wang, Trust, but verify: Optimistic visualizations of approximate queries for exploring big data, in: CHI, ACM, 2017, pp. 2904–2915 (2017).