Extending the View Composition Algebra to Hierarchical Data

Eugene Wu

Abstract—Comparison is a core task in visual analysis. Although there are numerous guidelines to help users design effective visualizations to aid known comparison tasks, there are few formalisms that define the semantics of comparison operations in a way that can serve as the basis for a grammar of comparison interactions.

Recent work proposed a formalism called View Composition Algebra (VCA) that enables ad hoc comparisons between any combination of marks, trends, or charts in a visualization interface. The key idea is to model a view \( V = R(Q(D)) \) as a thin visual mapping function over the result of a query \( Q \). \( Q \) is responsible for all data processing, aggregation, and transformations. In this way, composition operations over multiple views such as differencing and union can be defined as expressions over the underlying queries of the views.

For instance, Figure 1 illustrates the two core operators. (a) computes the difference between SFO and OAK matching bars with the same days and subtracting each day’s Oakland delay with the SFO delay on that day. (b) illustrates the union operation, which juxtaposes the bars for easier visual comparison.

However, VCA limits comparisons to visual representations of data that have an identical schema, or where the schemas form a strict subset relationship (e.g., comparing price per state with price, but not with price per county). In contrast, the majority of real-world data—temporal, geographical, organizational—are hierarchical. To bridge this gap, this paper presents an extension to VCA (called VCA\(^H\)) that enables ad hoc comparisons between visualizations of hierarchical data. VCA\(^H\) leverages known hierarchical relationships to enable ad hoc comparison of data at different hierarchical granularities. We illustrate applications to spatial visualizations and Tableau visualizations.

Index Terms—Visualization, Algebra, Comparison, Databases

1 Introduction

Comparison is a core task in visual analysis. Although there are numerous guidelines to help users design effective visualizations to aid known comparison tasks, there are few formalisms that define the semantics of comparison operations in a way that can serve as the basis for a grammar of comparison interactions.

Recent work proposed a formalism called View Composition Algebra (VCA) [17] that enables ad hoc comparisons between any combination of marks, trends, or charts in a visualization interface. The key idea is to model a view \( V = R(Q(D)) \) as a thin visual mapping function over the result of a query \( Q \). \( Q \) is responsible for all data processing, aggregation, and transformations. In this way, composition operations over multiple views such as differencing and union can be defined as expressions over the underlying queries of the views.

For instance, Figure 1 illustrates the two core operators. (a) computes the difference between SFO and OAK matching bars with the same days and subtracting each day’s Oakland delay with the SFO delay on that day. (b) illustrates the union operation, which juxtaposes the bars for easier visual comparison.

However, VCA limits comparisons to views that both render the same data attributes, or where their data attributes have a subset relationship (e.g., comparing price per state with price). In contrast, the majority of real-world data are hierarchical, and users may wish to compare e.g., price per state in 2000 with price per county in 2001.

To bridge this gap, we extend VCA to enable ad hoc comparisons between visualizations of hierarchical data. We call this extension VCA\(^H\). VCA\(^H\) leverages known hierarchical relationships to enable ad hoc comparison of data at different hierarchical granularities, and we base the formalisms on functional dependencies, a core concept in the relation model [2]. We also illustrate applications to the hierarchical language HIVE [11] and Tableau visualizations.

2 Related Work

2.1 Visualization Languages for Hierarchical Data

Hierarchical structures are integral to visualization algebras such as VizQL [5][14]. VizQL defines algebraic operators over data attributes to compose small-multiple display, which are manifested as interactions to drag-and-drop attributes onto x- and y-axis “shelves”. The \( \text{cross}(Q_tr \times \text{State}) \) operator creates one small multiple for each combination of quarter, state values. However, \( \times \) is not appropriate when composing attributes within a hierarchy, because not all combinations of the attribute domains will exist in the dataset. For instance, \( Q_tr \times \text{Month} \) will return \( (Q_tr, \text{Dec}) \), which will never contain data. The hierarchy-aware \( \text{nest}(Q_tr/\text{Month}) \) operator obeys the containment relationship, and ill only return pairs where the month is in the appropriate quarter.

Slingsby et al. [11] define a grammar for specify spatial layouts of hierarchical data. Much like how graphical grammars map data attributes (e.g., day, cost) to a mark’s visual attributes (e.g., x- and y-positions), HIVE proposes a visual mapping from attributes in a data hierarchy to levels in a spatial hierarchy. In addition to this mapping, users also specify the desired layout algorithm at each level of the spatial hierarchy. For instance, using a simplified notation, \( \text{state(profit} \to \text{size)/year(profit} \to \text{size) creates a nested tree map; the top level uses a space filling layout sized by price, and each rectangle is further subdivided by year and sized by that year’s profits (Figures 5). HIVE can express a rich set of spatial visualizations, including calendars, tree maps, and cartograms. Similar ideas were explored in product plots [16], and probabilistic graphical grammars [9].

Although these languages enable users to create visualizations that are aware of hierarchies, they do not support ad hoc comparisons across different hierarchical levels.

2.2 Composition in Visualization Grammars

Graphical grammars, such as ggplot2 [15], Vega-lite [10], and VizQL [12] model visualizations as mappings from data attributes to the visual attributes of the rendered marks. These grammars implicitly perform data transformations, such as grouping and aggregation, based on the data types. For instance, VizQL is a table algebra to compose faceted, multi-layer visualizations, and used to navigate multidimensional data cubes [4][13]. VCA is compatible and composable with these grammars—VCA operators can take their views as input, and emits views as output. VCA defines unambiguous data transformations so that interaction designers can focus on design choices.

Fig. 1. Example VCA operations. (a) statistical composition computes the difference between two views, and (b) union composition juxtapose or superimposes data from both views.
### 2.3 Design Strategies for Comparison

Javed and Elmquist [6] propose a design space for composite visualizations, describe four visual composition designs (juxtapose, superpose with and without shared axes, and nested views). Gleicher et al. [3] characterize comparison by the target elements being compared, and the actions to compare them. They similarly propose juxtaposition, superposition, and explicit encoding as design strategies. VCA [8] goes beyond design guidelines, and defines formal composition rules and notions of safety.

### 3 Preliminaries

How to model hierarchical data so that an algebra can symbolically manipulate it? This section presents a model for hierarchical data based on functional dependencies, and also introduces VCA.

### 3.1 Functional Dependencies and Hierarchical Data

SQL and most query languages are grounded in the relational model [2]. The relational model defines a database as a set of relations. Each relation (table) consists of a set of tuples (records) that adhere to a schema—a list of attribute, type pairs. Following VizQL [12] and other visualization formalisms, we assume that a table $T$'s schema $A_T$ consists of dimensions (used for filtering, grouping, and database joins) and measures (used to compute statistics); without loss of generality, we will assume that each table contains a single measure.

**Functional Dependencies:** The relational model defines a functional dependency constraint ($FD$) $X \rightarrow Y$ as a set of attributes $X$ that functionally determines the set of attributes $Y$. In other words, if two records have the same $X$ values, they must also have the same $Y$ values. For instance, $zip \rightarrow state$ says that two records with the same zipcode must have the same state. Foreign key references are also a special case of functional dependencies because both values are equal, thus the dependency is in both directions.

**Hierarchies are FDs:** Attributes in real-world data typically form a hierarchical structure. Hierarchies are used in multi-dimensional databases to drill-down or roll-up, and in visualization systems to zoom in or zoom out. Visualization formalisms like VizQL [5] rely on such hierarchies to define operators like nest.

Hierarchies are a special case of functional dependencies. When we say that district is a child of state in a hierarchy, is expressed as the FD district $\rightarrow$ state.

Thus, we define a hierarchy $H = \{F_1, \ldots, F_p\}$ as a set of FDs whose edges form a directed acyclic graph. In contrast to a colloquial hierarchy, where edges point from coarser to finer granularities (e.g., state to county), the edges in $H$ are reversed, and point from finer to coarser granularities.

For instance, Figure 2 shows tables that contain hierarchical data. $T_3$ stores each census block’s daily Profit, and contains Day, Month, Qtr, and Year attributes. $T_2$ stores the geographical data associated with each block, and can be joined with $T_1$ on block (dotted line). $T_3$ contains the Nation name, and joins with $T_2$ on nid.

**Fig. 2. Tables with two example hierarchies; arrows are FDs.**

### 3.2 How VCA models Views

VCA models a view as $V_i = R_i(T_i)$, where $T_i$ is a table (a raw table or a SQL query result) and $R_i$ is a visual encoding specification from a subset of $T_i$’s schema to visual attributes. In practice, $T_i = Q_i(D)$ is the output of a query $Q_i$ over an underlying database table $D$. Let $D(a_1, \ldots, a_n)$ contain $n$ attributes and $A_D = [a_1, \ldots, a_n, a_0]$ denote its schema, where $a_0$ is the measure, and the rest are dimensions. Let $A_i$ be shorthand for $A_{Q_i}$.

It is important that $Q$ encapsulates all computations and transformations needed to produce $T$, and that the renderer is only responsible for visual design—mapping data rows in $T$ to marks (or other objects) in the view and performing layout. This is because the result of comparison operators depends on knowing how $T$ is computed.

### 3.3 VCA Operators

VCA defines a library of composition operators. For space constraints, we focus on its two core operators: statistical composition, and that the renderer is only responsible for visual design—mapping data rows in $T$ to marks (or other objects) in the view and performing layout. This is because the result of comparison operators depends on knowing how $T$ is computed.

### 3.3.1 Statistical Composition

$V^* = V_1 \circ_{\sigma_{op}} V_2$ joins rows from $Q_1$ and $Q_2$ and computes a new measure $Q_1.y \circ_{op} Q_2.y$ from the two views’ measures (Figure 1(a)):

$$Q^* = \pi_{A_{Q_1} \cup A_{Q_2}} (Q_1 \circ_{\sigma_{op}} Q_2)$$

$R^* = R_1$ $Q^*$ first computes the outer join between $Q_1$ and $Q_2$ by matching records from each query whose grouping attributes $A_{Q_1}$ have the same values. An outer join ensures that rows in either table have at least one output row even if there is no match. Since the input queries, by definition, were grouped on $A_{Q_1}$, we are guaranteed exactly one output row for each group in $Q_1$ and $Q_2$. Finally, $\pi$ copies the join attributes, computes $\pi_{\{y\}} (Q_1 \circ_{\sigma_{op}} Q_2)$, and renames it as $y$.

$op$ is defined as “$+$” by default, however any binary arithmetic function is allowed. As shorthand, $\oplus$ and $\otimes$ denote $+$ and $\cdot$, respectively. $\circ_{op}$ is symmetric iff $y_1 \circ_{op} y_2 = y_2 \circ_{op} y_1$. 

---

Table 1. Description of relational algebra operations

| Operator | Description |
|----------|-------------|
| $\gamma_{i,f(a)}$ | Group by attributes $A$, and compute $f(a)$ for each group |
| $\pi_{i \rightarrow a_i}$ | Compute expressions $\pi_i$ and rename them as $a_i$. |
| $T.$ | * copies all attributes from input table $T$. |
| $\sigma_{p}$ | Filter records using boolean function $p(row)$ |
| $S \circ_{op} T$ | Join $S$ and $T$ rows with the same attribute values in $A$. |
3.3.2 Union Composition $\cup$

Union $V^* = V_1 \cup_{qid \neq} V_2$ composes the marks from both views into the same output view (Figure 3(b)):

$$Q^* = \pi_{qid}(Q_1) \cup \pi_{qid}(Q_2)$$

$$R^* = \{ qid \rightarrow a \} \cap R_1 \quad \text{s.t.} \; a \text{ is an unmapped vis attr}$$

Each query $Q_i$ is augmented to track a unique identifier $qid$, so that rows from each query can be distinguished in $Q^*$. $R^*$ additionally maps $qid$ to a visual attribute $a$ that is not already mapped in $R_1$.

3.4 Safety

Since VCA operators may take arbitrary combinations of marks as input, it is possible that the user may try to compose “incomparable” data together. For instance, comparing stock price per hour with average stock price is sensible (safe), but comparing stock price with trading volume is not (unsafe).

VCA defines safety based on the ability to map table $R_1$’s schema $A_{R_1}$ to table $R_2$’s schema $A_{R_2}$. Specifically, if their measures are the same, and if there is a unique mapping from each dimension in $A_{R_1}$ to a dimension in $A_{R_2}$ (or null). Dimensions can be mapped if they are the same. In short, $R_1$ is a subset of $R_2$.

Although correct, this strict definition rejects many natural comparisons. For instance, VCA cannot compare price per hour with price per day, because hour and day are different dimension attributes. Similarly, VCA cannot compare votes per district with votes per state.

4 VCA for Hierarchical Data

This section extends statistical and union composition to hierarchical-aware comparisons. Given two views $V_1 = R_1(Q_1(D))$ and $V_2 = R_2(Q_2(D))$ and a hierarchy $H$, we will define the semantics for statistical composition $\odot$ and union composition $\cup$.

Our examples will be based on the visualizations in Figure 3 which builds on the hierarchy and data in Figure 2. The calendar chart renders daily profits over all geographies and products, while the bar chart renders average monthly profits for toy products. The goal is to define composition for any combination of comparisons between visualization’s labeled subviews (1 to 4).

![Calendar view of overall Daily Profits and Bar chart of average Toy Profits by Month](image)

(a) Two views of hierarchical data

(b) 1 ⊙ 2

(c) 3 ⊙ 4

Fig. 3. A hierarchical calendar visualization that renders daily profits, and a bar chart that renders average monthly profits. All labeled subviews correspond to hierarchy levels that are comparable in VCA$^\#$.

4.1 Safety

To begin, we must extend VCA’s safety rules to allow hierarchy-aware comparisons. For space constraints, we focus on the case where both view’s schemas $(A_1, A_2)$ have the same number of dimensions, as this is the core challenge.

$A_1$ and $A_2$ are safe to compose if there exists a bijective mapping:

$$\{(a_i, a_j) | a_i \in A_1, a_j \in A_2, a_i = a_j \rightarrow H \quad a_j \rightarrow H \quad a_i\}$$

Where $a_i \rightarrow a_j$ given a directed path from $a_i$ to $a_j$ in $H$. This relaxes VCA’s rules so $a_i$ can map to $a_j$ if one is an ancestor of the other.

We also define $a_i \sim a_j$ to denote that the value of $a_i$ (e.g., “California”) functionally determines $a_j$ (e.g., “USA”) or vice versa. If $a_i \rightarrow a_j$, we can translate $a_i$ into $a_j$. Consider block $\rightarrow$ nation in Figure 2. We can translate block 1 to “USA” by joining $T_2$ and $T_3$ so that both attributes are in the same table, and then simply lookup the nation attribute value.

4.2 Statistical Composition $\odot$

In VCA, $V^* = V_1 \circ_{op} V_2$ matches rows in $Q_1$ and $Q_2$ using a join and computes a new query $Q_{1, y} op Q_{2, y}$. Extending $\odot$ to hierarchical data presents two challenges.

First, it potentially requires reaggregating data in one of the views in order to compute the new measure. For instance, 1-2 in Figure 2 which computes the difference between average monthly toy profits and overall daily profits. The views are safe to compare, because day=month. However, we need to reaggregate the daily profits to the month granularity, and the specific aggregation function may depend on the application needs. For this reason, we extend statistical composition $\circ_{op, reagg}$ to take an additional reagg() parameter for reaggregation. reagg defaults to $Q_2$’s aggregation function if not specified (e.g., $\circ_{\oplus, \oplus}$).

Second, $\odot$ is asymmetric, and reaggregation is not always necessary nor desired. For instance, 1-4 subtracts e.g., March’s profits from each day’s profits in March.

For these reasons, we define $\circ_{op, reagg}$ under two cases based on whether or not reaggregation is required. To simplify the discussion, we assume that $A_1$ is identical to $A_2$, except for a single pair of attributes $a_1 \in A_1$ and $a_2 \in A_2$, where $a_1 \rightarrow a_2$ or $a_2 \rightarrow a_1$.

4.2.1 Case 1: $a_1 \rightarrow a_2$

If the attribute $a_1$ is finer granularity than $a_2$, then we do not need to perform reaggregation. Thus we define the output view $V^* = V_1 \circ_{op, reagg} V_2$ as follows:

$$Q^* = \pi_{A_{gb}} Q_1, y \circ_{op} Q_2, y \sim (y \circ_{op} Q_1, \sigma_{gb} Q_1, a_1 \sim a_2, Q_2)$$

$$R^* = R_1$$

$$A'_{gb} = A_{gb} - \{ a_1, a_2 \}$$

$$\text{joinp} = \land_{a \in A'_{gb}} Q_1, a = Q_2, a$$

where $A_{gb}$ are the grouping dimension attributes in $Q_1$, joinp is identical to the join conditions in the definition in Section 3.3.1, but only for the attributes that are identical in $Q_1$ and $Q_2$. The join also checks that the value of $a_1$ functionally determines the value of $a_2$.

Example 1 Alice compares the difference between daily profits and monthly toy profits, so specifies 1-2, which results in Figure 2(b). The join condition only matches rows where translating $Q_1$ day into its month is equal to corresponding $Q_2$ month. Each day’s profit is reduced by that month’s average profits. For instance, January 1st’s profits is reduced by the average January profit.

4.2.2 Case 2: $a_2 \rightarrow a_1$

If the attribute $a_1$ is coarser than $a_2$, then we need to reaggregate the data in $V_2$. Recall that queries have the form $Q_i = \gamma_{\lambda_{gb}, f(a), y \rightarrow} (\sigma_f(D))$.

We define the output view $V^*$ as follows:

$$Q^* = \pi_{A_{gb}} Q_1, y \circ_{op} Q_2, y \sim_{gb} (Q_1, \circ_{op} \gamma_{\lambda_{gb}, \text{reagg}(a), y \rightarrow} \sigma_f(D))$$

$$Q_2 = \gamma_{A_{gb}, \text{reagg}(a), y \rightarrow} (\sigma_f(D))$$

$$R^* = R_1$$

The main idea is to reaggregate $Q_2$ at the same granularity as $Q_1$ (by using its grouping attributes $A_{gb}$), and then perform statistical composition as normal. We reaggregate the base table $D$ rather than the output of $Q_2$ to avoid potentially misleading statistics such as the average of averages.
Example 2 Alice now compares the difference between monthly toy profits with the daily profits, so specifies $\text{J} \cup \text{F}$ (Figure 3(c)). To compute this, $\text{VCA}^R$ first aggregates e.g., March’s profits using avg, and then subtracts the average profits from March’s toy profits.

4.3 Union Composition $\cup$

Although $\cup$ was a symmetric operator in $\text{VCA}$, it is asymmetric in $\text{VCA}^R$. This is because union composition requires two definitions based on the need for reaggregation. Using the same notations and assumptions as above, we define $V^* = V_1 \cup V_2$ under two cases. $\text{qid}$ is a query identifier, $a$ is an available visual attribute.

4.3.1 Case 1: $a_1 \rightarrow a_2$

In the first case, $V_2$ is rendered as a coarser granularity than $V_1$, so we duplicate each record in $V_2$ for each matching record in $V_1$. For instance, $\{a_1 \cup a_2\}$ will duplicate January’s average profits for each day in January. This is accomplished by defining $Q'_2$, which performs a left outer join, so that there is one output row for every unique value of $a_1$ in $Q_1$. We then keep the attributes in $A_1$ so that $Q'_2$ has the same schema as $Q_1$.

$$Q'_2 = \pi_{A_1}(\pi_{a_1}(Q_1) \leftjoin_{a_1 \leftarrow a_2} Q_2)$$

$$Q = \pi_{\text{qid}}(Q_1) \cup \pi_{\text{qid}}(Q_2)$$

$$R^* = \{\text{qid} \rightarrow a\} \cup R_1 \quad \text{s.t. a is an unmapped vis attr}$$

4.3.2 Case 2: $a_2 \rightarrow a_1$

In the second case, $V_2$ is at a finer granularity than $V_1$, so we reaggregate its data to $V_1$’s granularity. This is similar to the reaggregation procedure for statistical composition, and also requires a reagg() function, which defaults to the aggregation function in $Q_1$.

$$Q^* = \pi_{\text{qid}}(Q_1) \cup \pi_{\text{qid}}(Q_2)$$

$$\bar{Q}'_2 = \pi_{\text{qid}, \text{reagg}(a_1) \rightarrow a_2}(\sigma_{a_1}(D))$$

$$A_{gb} = A_{gb} - \{a_2\} \cup \{a_1\}$$

$$R^* = \{\text{qid} \rightarrow a\} \cup R_1 \quad \text{s.t. a is an unmapped vis attr}$$

5 Examples

$\text{VCA}^R$ directly benefits from $\text{VCA}$’s compositionality, its well-defined semantics, and that it can compare data independently of how they are visually encoded. This section presents examples of how $\text{VCA}^R$ complements existing visual analysis systems.

5.1 Tableau

$\text{VCA}^R$ can enable novel comparison interactions in Tableau-like exploration interfaces. Figure 4 shows small multiples of a bar chart for each quarter, and each chart renders costs per month. For all four example interactions, the user can drag component $V_2$ in the interface over $V_1$ to express $V_1 \cup V_2$. $\text{A}$ composes $Q_1$’s bar chart with all Quarters. It aggregates the bar chart data to the quarter level (average of $Q_1$) and subtracts that value from each quarter’s average cost. The horizontal line is the zero line. $\text{B}$ drags the Q2 header to the the month attribute in the x-axis shelf. This subtracts Q2’s average cost from all small multiples charts. The output is the same set of small multiples, but with each bar reduced by Q2’s average cost. $\text{C}$ drag the Q2 label onto the Q1 bar chart. This is akin to $\text{D}$, but only for a single target bar chart. $\text{E}$ drags the Qtr attribute in the x-axis shelf onto the Month attribute. This computes each quarter’s average cost, and subtracts it from the corresponding quarter’s bar chart.

5.2 Hierarchical Visualizations in HIVE

Figure 5 illustrates an example where the left view renders the HIVE statement $\text{state(profit} \rightarrow \text{size)}/\text{year}$, and the right view plots profit by quarter. Logically, HIVE computes the spatial layouts one level at a time. It first computes average price by state, and uses it to size the state-level red rectangles. Then, for each state, it computes the average price by year and uses it to size the per-year rectangles. Since these are all group-by aggregation queries, $\text{VCA}^R$ naturally applies. The line chart renders price per week in NY, and week$\rightarrow$Year.

6 Conclusion

We showcase three composition interactions and their outputs in Figure 5(c). $\text{X}$ drags Texas (TX) to the state component of the HIVE statement. This will subtract Texas’ average price from each state’s price and re-render the entire spatial visualization. Notice that TX disappears because subtracted price is now 0, and is allocated no space. $\text{Y}$ drags the selection range in the line chart over the TX label. This aggregates the data in the selection range to the year granularity and removes the aggregated values from the corresponding TX year. Only the TX subset of the spatial visualization is returned. $\text{Z}$ drags the selection range onto the year component of the HIVE statement. This again aggregates the selected data, and subtracts it from each state’s corresponding years. The rectangles for 2001 are allocated more space in both states, however the proportions for the states do not change because the same amount is removed from both states.

View Composition Algebra (VCA) is a recent formalism for composing entire parts of visualizations to aid adhoc comparison tasks. Users can select values, marks, legend elements, and entire charts as targets, and use composition operators to summarize or compare the targets. This paper extend View Composition Algebra to support comparisons between views that render data at different hierarchical granularities. This enables users to easily compare data within spatial visualizations, between different visualizations. We presented the formal semantics that are compatible with the underlying queries supported by visual analytic systems such as HiDE and Polaris. We further illustrated example interactions in the context of small multiples visualizations and spatial hierarchical visualizations.
References

[1] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. SIGMOD Rec., 26:65–74, 1997.
[2] E. F. Codd. A relational model of data for large shared data banks. Communications of the ACM, 26(1):64–69, 1983.
[3] M. Gleich. Considerations for visualizing comparison. In TVCG, 2018.
[4] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatramo, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Mining and Knowledge Discovery, 1:29–53, 2004.
[5] P. Hanrahan. Vizql: a language for query, analysis and visualization. In Proceedings of the 2006 ACM SIGMOD international conference on Management of data, pp. 721–721. ACM, 2006.
[6] W. Javed and N. Elmqvist. Exploring the design space of composite visualization. 2012 IEEE Pacific Visualization Symposium, pp. 1–8, 2012.
[7] N. Kamat, P. Jayachandran, K. Tunga, and A. Nandi. Distributed and Interactive Cube Exploration. In ICDE, 2014.
[8] C. A. L. Pahins, S. Stephens, C. Scheidegger, and J. Comba. Hashed-cubes: Simple, low memory, real-time visual exploration of big data. IEEE Transactions on Visualization and Computer Graphics, 23:671–680, 2017.
[9] X. Pu and M. Kay. A probabilistic grammar of graphics. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, pp. 1–13, 2020.
[10] A. Satyanarayanan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. IEEE Transactions on Visualization and Computer Graphics, 23:341–350, 2017.
[11] A. Slingsby, J. Dykes, and J. Wood. Configuring hierarchical layouts to address research questions. IEEE transactions on visualization and computer graphics, 15(6):977–984, 2009.
[12] C. Stolte and P. Hanrahan. Polaris: a system for query, analysis and visualization of multi-dimensional relational databases. IEEE Symposium on Information Visualization 2000. INFOVIS 2000. Proceedings, pp. 5–14, 2000.
[13] C. Stolte, D. Tang, and P. Hanrahan. Multiscale visualization using data cubes. IEEE Trans. Vis. Comput. Graph., 9:176–187, 2003.
[14] C. R. Stolte. Query, analysis, and visualization of multidimensional databases. 2003.
[15] H. Wickham. ggplot2: elegant graphics for data analysis. Springer, 2016.
[16] H. Wickham and H. Hofmann. Product plots. IEEE Transactions on Visualization and Computer Graphics, 2011.
[17] E. Wu. Vca: View composition algebra for ad hoc comparison. In TVCG, 2022.