Semantic Snapping for Guided Multi-View Visualization Design

Yngve S. Kristiansen, Laura Garrison, and Stefan Bruckner, *Member, IEEE Computer Society*

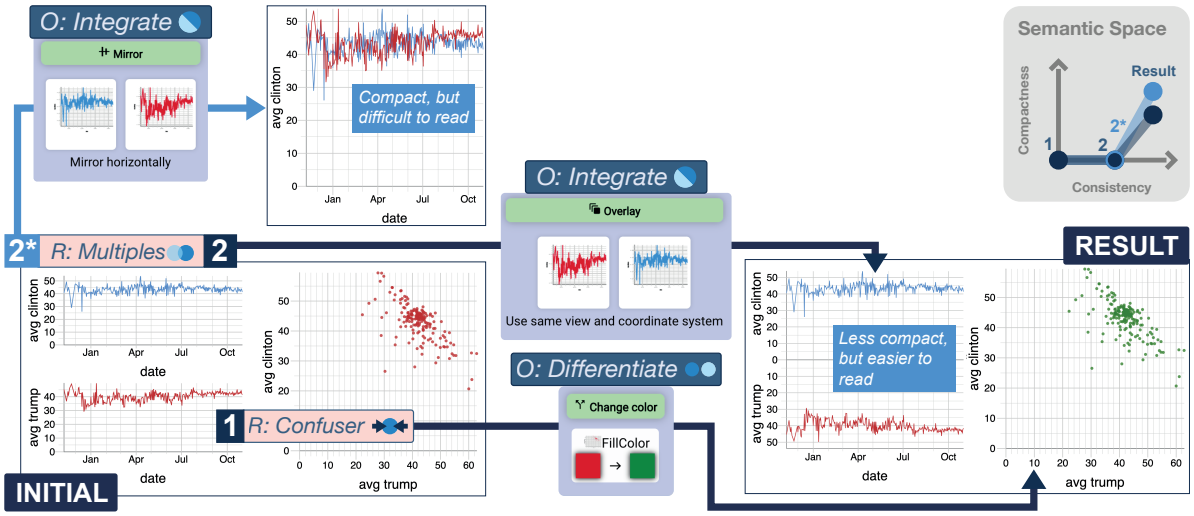

Fig. 1. Semantic snapping allows the user to perform iterative operations to improve the compactness and consistency of a multi-view visualization. Underlying algebraic rules called *relations* define the available *operations* for each iteration. In this example showing the 2016 US Election poll percentages and pollsters, from our initial composition we (1) identify a *confuser* relation between the bottom left and rightmost views showing the same color (red). We *differentiate* these views by selecting green as the fill color for the scatter plot. We next identify a *multiples* relation in the two left views. We resolve this through one of two *integration* operations. (2*) *Overlay* produces an unsatisfactory result, so we revert and (2) perform a *mirroring* operation to arrive at our resulting composition. The semantic map to the right illustrates our path through semantic space.

Abstract—Visual information displays are typically composed of multiple visualizations that are used to facilitate an understanding of the underlying data. A common example are dashboards, which are frequently used in domains such as finance, process monitoring and business intelligence. However, users may not be aware of existing guidelines and lack expert design knowledge when composing such multi-view visualizations. In this paper, we present semantic snapping, an approach to help non-expert users design effective multi-view visualizations from sets of pre-existing views. When a particular view is placed on a canvas, it is "aligned" with the remaining views–not with respect to its geometric layout, but based on aspects of the visual encoding itself, such as how data dimensions are mapped to channels. Our method uses an on-the-fly procedure to detect and suggest resolutions for conflicting, misleading, or ambiguous designs, as well as to provide suggestions for alternative presentations. With this approach, users can be guided to avoid common pitfalls encountered when composing visualizations. Our provided examples and case studies demonstrate the usefulness and validity of our approach.

Index Terms—Tabular data, guidelines, mixed initiative human-machine analysis, coordinated and multiple views

1 Introduction

Multi-view visualizations are frequently utilized to present and analyze data. Dashboards, for example, are commonly employed for monitoring and related tasks in a wide variety of fields. Popular visualization systems like Tableau [39] and PowerBI provide galleries of carefully crafted templates in order to enable the quick and easy generation of such visualizations. However, when non-expert users would like to extend, modify, or customize such a multi-view visualization, they may easily fall prey to a number of pitfalls that can result in potentially misleading or otherwise problematic results. Expert knowledge to guide such tasks is mostly available in the form of guidelines from the visualization literature, which are not readily accessible to novice users. Common examples include Qu and Hullman's constraints, C1 (the same data should be shown the same way) and C2 (different data should be shown in different ways) [30]. In this paper, we present a method that detects and helps users to resolve such potential problems in multi-view visualization design in a semi-automatic and guided fashion.

Suppose a user of a visualization system wants to create or extend a multi-view visualization from a set of pre-existing charts that are individually well-designed (e.g., based on a gallery). If the user wishes to use these visualizations in combination (e.g., in a dashboard), there are non-obvious design opportunities and pitfalls. Views may show the same data in different ways, or different data in the same way. A single view may be highly informative and take up a modest amount of screen space. However, when used in combination with other views, they may show overlapping information, or use too much screen space. These issues can be remedied by showing the same data with fewer views, i.e., making the overall design more *compact*. A multi-view visualization can be made more compact, or less conflicting, by manually redesigning and tweaking single views. However, manually detecting and resolving conflicts, and coming up with alternate representations of views, is

• *The authors are with the Department of Informatics, University of Bergen, Norway. E-mail: ykr088@uib.no, laura.garrison@uib.no, stefan.bruckner@uib.no.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

cumbersome, error-prone and time-consuming. Our method lets the user perform this process via high-level design-altering operations.

Our approach uses a semantic space with two axes that represent the degree of consistency and compactness of a multi-view visualization. We examine *relations* between views to identify opportunities to improve the overall visualization with respect to these criteria. We then provide the user with a set of operations to realize the corresponding changes. For instance, two views may employ the same color map for different quantities. In such a case, our approach may suggest to *differentiate* the two views by modifying one of the mappings to increase the overall consistency. Likewise, when the same data are shown differently in multiple views, our method suggests different ways to *homogenize* them. In other cases it may be possible to *integrate* multiple views in order to improve the compactness of the visualization.

The contributions of our work can be summarized as follows. Based on a synthesis of existing guidelines from the literature, we present a novel approach for identifying and applying potential improvements of multi-view visualizations. We use predicate logic to represent relations between individual views and propose operations to improve the consistency and compactness of the underlying visualization based on the identified relations. Furthermore, we propose a simple workflow and user interface for presenting and selecting the suggested operations.

2 Related Work

Visualization design and measures. Bertin's Semiology of Graphics [2] and Wilkinson's Grammar of Graphics [44] were two of the early influential works focusing on formal aspects of reasoning about the effectiveness of visualizations. Munzner [28] later consolidated and refined existing concepts and terminology, leading to a comprehensive framework for thinking about visualization in terms of principles and design choices. Bolte & Bruckner [3] survey measures focusing on different aspects of the visualization process: perceptual characteristics, task-oriented quality measures, structure-oriented measures, and meta-perceptual processes. Perceptual characteristics such as Cleveland & McGill's experiments on graphical perception [6] are based on human performance in elementary tasks such as comparing positions on a common scale. Other measures express desirable relationships between the data and its visual representations. For example, Tufte's data-to-ink ratio [41] describes the proportion of pixels used to represent data versus the total amount of available pixels. Furthermore, Correll et al. [7] address the issue that designs may appear to be showing the data completely, while hiding important details. They propose actions to remedy discovered vulnerabilities for different chart types. Behrisch et al. [1] categorized different quality measures from around 250 papers. Most of these measures were specific to a certain combination of underlying data, task and visualization technique.

Through literature review, Zhu [50] points out why existing definitions of visualization effectiveness are often incomplete: they usually take either a data-centric, or task-centric view on what an effective visualization is. Data-centric effectiveness measures deal with how accurately a visualization is showing its underlying data. An example of a data-centric framework for measuring visualization effectiveness is Kindlmann and Scheidegger's algebraic framework [18]. By considering symmetries between changes in data space and resulting changes in visualization space, they describe three principles that should ideally be true for any data-to-visualization mapping: unambiguous data depiction, representation invariance, and visualization-data correspondence. We draw inspiration from this model and adopt a similar line of reasoning in the context of multi-view visualizations. Based in part on the concept of algebraic visualization design, McNutt and Kindlmann [26] present a linting mechanism for the process of designing a chart. Their linting is realized as a Python library that evaluates visualizations created with matplotlib, and returns a list of rules that are violated. While our work is based on similar fundamental considerations, we focus on multi-view visualizations and expose potential revisions through a user interface.

Many approaches take into account both data and tasks. Cantu et al. [4] outline an approach to identify relationships between visualization challenges and representation components (e.g., data transformations, filtering techniques, visual variables). They argue that these relationships can further our understanding of the mechanisms behind visualization components, which could eventually be used to build visualization recommendation tools. Silva et al. [38] survey work done on using different color scales in visualization, with a focus on desired properties and guidelines for choosing the right colors. They highlight that it is important to consider factors such as the type of data, type of visualization, type of task, and audience. As pointed out by Zhu [50], there are multiple disjoint, sometimes conflicting sets of guidelines and measures. Efforts have been made to facilitate convergence and understanding between different viewpoints. Diehl et al. [9] initiated the VisGuides forum both to facilitate collection and discussion of visualization guidelines, and knowledge about visualization in general. Engelke et al. [11] highlight that there is a gap between the communities who propose visualization guidelines, and those who need them. They provide a conceptual model called VISupply that highlights problems and opportunities with how guidelines are currently "shipped" to non-experts.

Authoring tools and visualization recommendation. Visualization authoring tools help users creatively express a wide range of individual charts. While these systems have much design freedom, they also rely on the expertise of the user. Zhu et al. [49] survey different tools for automatically generating infographics and visualization recommendations. Examples of systems that mostly focus on authoring and design flexibility include Charticulator [33], Lyra [35], iVisDesigner [32], Data Illustrator [22], Data Driven Guides [17]. These systems all use varying underlying frameworks for representing visualizations. We provide a set of relations and operations specified at a high enough level so that they can be expressed in terms of most individual frameworks.

Several efforts have been made to make expert knowledge available through software. Among them, visualization recommendation systems can potentially take into account expert knowledge to steer which revised designs are presented to the user. MacKinlay's APT (A Presentation Tool) [23] was among the first of these systems. He used a composition algebra for designing visualizations, and evaluated their effectiveness in accordance with Cleveland & McGill's effectiveness metrics [6]. Wongsuphasawat et al. proposed CompassQL [46] as a general language for querying over the space of visualizations, to be used in visualization recommender systems. Voyager [45] allows for exploring data via automatically generated visualizations. With Voyager 2 [47], the user is able to partially specify what a view should show by using wildcards and also see automatically-generated charts showing data related to the existing views. Data2Vis [8] is a trainable neural translation model for automatically generating visualizations from datasets. It is powered by formulating visualization generation as a language translation problem, where data specifications are mapped to Vega-Lite specifications [36]. Grammel et al. [13] explore how novices construct visualizations. Their findings suggest the need for a tool that supports iterative refinements, and explanations that help with learning. Our method shares a similar line of thought by enabling incremental refinement of a multi-view visualization. Show Me [24] is a set of user interface commands that provide a way to display an additional data attribute within a view, as well as high-level commands for building views for multiple fields. Draco [27] makes visualization design guidelines available for a wider audience by formalizing the knowledge into precise constraints, which can then be used and accessed in an Answer Set Programming environment. They model single visualizations as sets of logical facts, and represent design guidelines as hard and soft constraints over these facts. Dziban [21] further extends Draco with anchoring mechanisms to help drive specification queries with increased user agency. These works all represent different ways of representing and reasoning about visualizations. Our method differs from these approaches in that it focuses in the incremental refinement of multi-view visualizations.

Multi-view visualization design. One of the most common use-cases of multi-view visualizations are dashboards. Sarikaya et al. [34] construct a design space of dashboards, by analyzing multiple examples of dashboards found "in the wild." QualDash [10] is a task-oriented

dashboard generation engine that enables the mapping of specific user task sequences in healthcare quality improvement to a view composition. For dashboards and multi-view visualizations in general, multiple views must be laid out on a single screen, or even multiple screens. PanoramicData [48] is a visual analysis tool using a canvas metaphor to explore and combine data views. We use a similar metaphor in our approach, although we focus on semantics rather than filtering and linking the views. Vistribute [14] is a framework that automatically distributes visualizations and user interface components among multiple heterogenous devices. Scout [40] is a system that helps interface designers to create layouts by using high-level constraints based on design concepts such as semantic structure, emphasis and order. While our approach currently does not address layout, we believe that our method could be combined with similar approaches to also take into account layout considerations.

Composed views such as small multiples [42] allow for comparing visualizations. Gleicher et al. [12] provide a general taxonomy of visual designs for comparing visualizations, with three categories: juxtaposition, superposition and explicit representation of relationships. Elzen and van Wijk [43] leverage small multiples so that they are not only informative, but also helpful for the data exploration process itself. Through a series of graphical perception experiments, Ondov et al. [29] investigated which compositions of multiple charts are the most effective for different tasks. From 360 images of multi-view visualizations collected from IEEE VIS, EuroVis and PacificVis publications from 2011 to 2019, Chen et al. [5] identify common multi-view visualization practices, including typical view layouts, view types, and correlations between view types and layouts. The patterns found among these views are made available through a multi-view visualization recommendation system, allowing users to interactively browse different designs. We draw inspiration from these approaches by enabling the transformation of, for example, two bar charts into an item-wise grouped or chart-wise juxtaposed mirrored bar chart in order to increase the compactness of the overall visualization, as in Figure 5.

Conventional snapping creates a "gravity field" around geometric objects, making it easier to place them together in certain ways. Hudson [15] introduced the notion of *semantic snapping* as an interaction technique for geometrically snapping objects together only if the objects are specified to be semantically related. Our work is a continuation of this basic concept, extending it to the scenario of multi-view visualization design and focusing on the semantic rather than geometric aspects. Shadoan & Weaver [37] explore semantic relations in multi-view visualizations using a hypergraph querying system. While such queries are constructed similarly to *relations* in our approach, the former are driven through cross-filtering on attribute relationship graphs, while ours draws from rules heavily inspired by Kosslyn's principles [19] and Kindlmann and Scheidegger's algebraic framework [18]. The latter framework has been used to identify effective visualization types for certain user tasks, e.g., table cartograms [25]. Kim et al. characterize responsive visualization strategies via their targets, i.e., element(s) of a design that change, and actions, i.e., how element(s) are changed [16]. This semantics-based characterization parallels our notion of *relations* and *operations*, although the underlying models differ.

Qu and Hullman [30] discuss how to operationalize Kosslyn's principles [19] with the two following constraints: C1 (encode the same data in the same way), and C2 (encode different data in different ways). These two constraints are further detailed by specifying lower-level constraints on encodings across two views. In a later paper [31], they found through a Wizard-of-Oz study that Tableau users unknowingly, and with some exceptions, respected their constraints C1 and C2. They found that study participants were positive to having a consistency checker tool to surface such warnings. Similarly to Qu and Hullman, we operationalize the principles C1 and C2 on an encoding-level, but we do so by using a model inspired by Kindlmann and Scheidegger's algebraic framework [18]. Furthermore, we present a practical realization of this concept that both shows how to identify potentially problematic relations and introduces a set of concrete operations to address the relations, i.e., remove the relation itself or a problem caused by the relation.

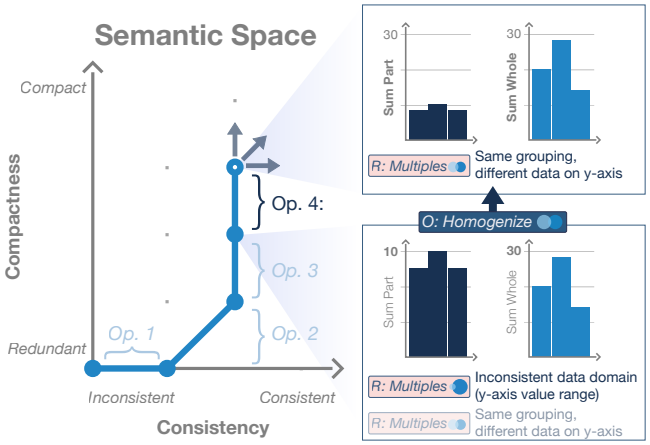

Fig. 2. Conceptual figure showing the semantic space with relations and operations for our semantic model. Operation 4 displays the *homogenize* operation which is available as a result of a *multiples* relation (the two axis scales are different, but should be the same if the underlying data represents the same quantity).

3 Semantic Snapping Model

Semantic snapping is the process of incrementally modifying a multi-view visualization by aligning its individual views with respect to their semantic, rather than their geometric, attributes. The main underpinning of our method is that a multi-view visualization, and its potential revisions, can be placed into a semantic space with two dimensions representing the degree of compactness and degree of consistency. We provide a conceptual overview of this semantic space in Figure 2. In other words, a potential revision of a design is either more or less compact, or more or less consistent, than the original design. Our method identifies these potential revisions, and presents them to the user as operations. By executing these operations, the user is able to intuitively navigate the semantic space of revised designs.

Achieving high-level goals by piecing together low-level modifications can be tedious, especially for novice users. In other software, such as word processors, semi-automatic tools help with this workload by highlighting errors, and suggesting corrections to these errors. Previous approaches, such as McNutt and Kindlmann's linting mechanism [26], have already explored this direction by providing functionality akin to a spell checker for a *single* visualization. In contrast, semantic snapping can be seen as more similar to a grammar checker, since it focuses on relationships *between* visualizations, just as a grammar checker analyzes relationships among words or phrases. Errors or potential errors represent detected inconsistencies or redundancies between views, and error corrections are represented as suggested operations to revise the composition of views.

Our method identifies existing and potential semantic inconsistencies and redundancies, so-called *relations*, between single views. Each relation identifies a potential problem which can be resolved by an *operation*. Thus, each operation is a high-level modification to the overall design. It is necessary to have the user involved in each modification to the design, since consistency and compactness are sometimes traded off for other design considerations [31]. The cycle of finding relations to infer available operations is repeated every time the design is altered.

3.1 Semantic Space

We begin with defining the terms that comprise our semantic space. A *canvas* is composed of multiple views of a single tabular dataset, where each individual view displays a single chart of a certain type (for example a bar chart, scatter plot, line chart, etc.). A *chart grouping* is a data dimension by which a chart is grouped, similar to SQL's `GROUP BY` command. For example, a bar chart grouped by country will have one bar for each distinct country in the dataset. Each chart has a set of *channels*, which may or may not be mapped to data. Data shown by a channel is denoted as a *data mapping*, which may also be

	Relation	Specification	Illustration	Possible Operations	Illustration
(a)	Full redundancy	$g = 1 \wedge \forall c(c = 1 \rightarrow d = 1)$	D1 D1	Delete one	D1
(b)	Partial redundancy	$g = 1 \wedge \forall c((c = 1 \wedge d \neq 1) \rightarrow \exists! D = 0)$	D1 D2 D1∈D2	Integrate, or delete D1 view	D2
(c)	Multiples (1) same grouping	$g = 1 \wedge \exists c((c = 1 \wedge \exists D \neq 0) \rightarrow d \neq 1)$	D1 D2	Integrate or homogenize	D1 D2
(d)	Multiples (2) same data	$g \neq 1 \wedge \exists c((c = 1 \wedge \exists D \neq 0) \rightarrow d = 1)$	D1 D1	Homogenize	D1 D1
(e)	Hallucinator	$g = 1 \wedge \exists c(c = 1 \wedge d = 1 \wedge \exists D \neq 0 \wedge v \neq 1)$	D1	Homogenize	D1
(f)	Confuser	$\exists c(c = 1 \wedge d \neq 1 \wedge v = 1)$	D1 D2	Differentiate	D1 D2

Table 1. All relations specified in terms of our model. The lower case letters: g, c, d, and v represent equalities (1) or inequalities (0) between chart groupings, channels, data mappings and visual outputs, and the specifications are predicate logic expressions operating primarily on these (lower case) equalities or inequalities. The uniqueness quantifier on $\exists!D$ indicates that there exists exactly one data mapping that satisfies a certain condition, for example being unmapped ($= 0$) for (b). For the *partial redundancy* relation, $D1 \in D2$ signifies that all data shown by one view ($D1$) is also shown by the other view ($D2$).

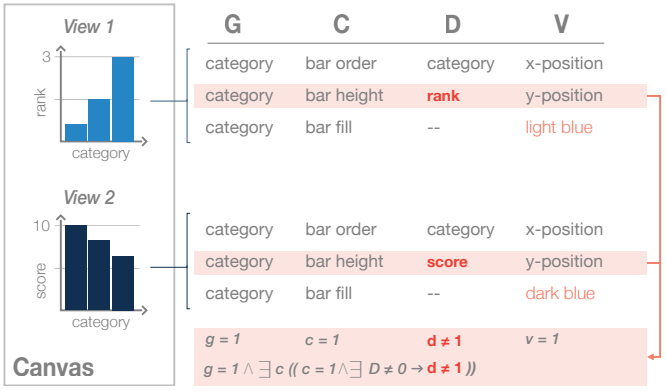

Fig. 3. Here we see two views, and the (G, C, D, V) tuples corresponding to the three channels representing the y-axis (bar height), x-axis (bar order), and fill color. The comparison of the two highlighted tuples are what identifies these two views as *multiples (1) same grouping*.

empty (indicating an unmapped channel). For example, consider the fill color channel in a scatter plot, which can optionally be mapped to data to display an additional quantitative data attribute for each mark. When a channel has a data mapping, the data are scaled from a *data domain* to a resulting *visual output* which directly or indirectly affects the appearance of the chart. The data domain denotes the minimum and maximum value of a certain attribute, or attribute aggregate, and is used as an input for the scale from data to a resulting visual output. When groupings differ, sampled data domains may also differ. Furthermore, domains may be different due to custom configuration of individual views. While there are different ways to arrange and transform tabular data, we limit the scope of chart groupings and channel mappings to single data dimensions. Summarized, a view has one chart grouping and multiple channels. Each channel has a data mapping, a data domain (if the data mapping is non-empty), and a resulting visual output. A view is part of a canvas, and a canvas has a certain position in the semantic space.

The semantic space has two axes, representing (1) redundancy/compactness (shorthand: compactness axis) and (2) inconsistency/consistency (shorthand: consistency axis). The compactness axis ranges from redundant to compact, whereas the consistency axis ranges from inconsistent to consistent. For example, if a canvas is made more compact by turning two views into one, the canvas becomes less redundant, thus moving up along the compactness axis. Semantic snapping corresponds to movement along one of these semantic axes.

It is important to note that more consistency and compactness is not always desirable. For example, Qu and Hullman [31] found that in cases, homogenizing axis domains is undesirable due to the extra white space it generates. Conversely, a compact design is not always more readable, or the most ideal for telling a story. Our operations make it possible for the designer to explore this space of alternative designs more rapidly, one semantic axis at a time. Relations between individual views identify not only where the canvas is currently located, but also which changes (denoted operations) are possible.

3.2 Algebraic Relations

Relations are explicit specifications of redundancies or inconsistencies between views. Although relations themselves do not indicate whether a design is good or bad, they are available to help the user identify potential problems in their overall visual design.

Our specification of relations draws both from Qu and Hullman's evaluation constraints [30] and a generalization of the principles established in Kindlmann & Scheidegger's algebraic model of visual design [18]. Originally developed in the context of only a single visualization, their model describes the relationships between three elements of the visualization process: the data, the representation of the data, and the resulting visualization. We adopt two principles from this model, which are easily framed within the two high-level constraints stated by Qu and Hullman [30]:

C1 Encode the same data in the same way. A violation of this constraint corresponds to representation invariance in Kindlmann & Scheidegger's algebraic model, which states: if the data of two visualizations are the same, the resulting visualizations should also be the same. A violation of this is called a *hallucinator* (Table 1e).

C2 Encode different data in different ways. A violation of this constraint has a corollary again in Kindlmann & Scheidegger's model as an unambiguous data depiction, which states: if the resulting visualizations are the same, the data should also be the same. If this principle is violated, we say that there is a *confuser* (Table 1f).

We represent aspects of a single view with the following four elements: the chart grouping (G), a channel (C), the data shown by the channel (D), and the resulting visual output (V). Typically, the term "channel" can denote the entire mapping from data to visual output. However, in our case C simply represents the name of the channel, e.g., fill color, and we use the other lower-level elements to concisely specify relations between views as predicate logic expressions as specified in Table 1.

A single view has a grouping (G), which is the first element in our model. Each view has multiple channels, with (C) referring to a *single* channel, and correspondingly each single channel has a data mapping (D) and a resulting visual output (V). Thus, each view has one (G, C, D, V) tuple *per channel*, where G is always the same, while (C, D, V) is unique to each channel as shown by Figure 3. Consider a bar chart grouped by category, showing average rank on the y-axis. Since G=category, C=bar height, D=average rank, and V=y-position, the tuple for the channel is then (category, bar height, average price, y-position)

as seen in Figure 3. If a channel does not have a data mapping, this is expressed as $D = 0$.

By considering a single view to be a set of (G, C, D, V) tuples (see Figure 3), we can establish relations between two views by using predicate logic on the tuples and their equalities. When comparing the tuples of two views, we use the same lower case letter to denote equality or inequality. For instance, if two views have the same chart grouping, the relation between G_1 and G_2 is the identity: $g = 1$. Conversely, if the groupings are different, then $g \neq 1$. If a relation exists between the views A and B, and between A and C, it also exists between B and C.

A relation exists between two views if there are two tuples (one from each view) that satisfy the predicate logic formula. For example, consider the predicate logic expression of the *multiples* relation: $g = 1 \wedge \exists c((c = 1 \wedge \exists D \neq 0) \rightarrow d \neq 1)$. This relation exists between two views if there is a pair of channels (one from each view) that satisfy this expression. As illustrated in Figure 3, the two highlighted views have the same grouping (category), but are showing different quantities on the y-axis (rank vs. score), making the *multiples* expression come true.

As discussed by Qu and Hullman [30], two encodings are showing the *same field* when the fields are *semantically* the same. We use this definition. Thus, if two fields are semantically the same, $d = 1$. To confirm semantic sameness, the user is asked to confirm if fields are the same, as seen in Figure 5.1b if this cannot be directly determined. We also specify that $d \neq 1$ if both data mappings are empty, but the grouping is different. For example, suppose two pie charts are respectively grouped by gender, and age group, and are both colored red. A sector of the pie chart can then represent either an age group, or a gender, yet they are colored the same. This is a potential confuser since each are showing different data, but are colored the same.

We define that the stroke color channel of charts without filled shapes (e.g., a line chart), and the fill color channel for a any chart with a fill (e.g., bar chart, scatterplot), is the same. For example, in the example shown in Figure 1.1 we see a line chart and a scatter plot both using the color red. With our notion of channel equality, $c = 1$ since the stroke color of the line chart is the same as the fill color of the scatterplot. Furthermore, they are grouped differently ($g \neq 1$), and both of the color channels are not mapped to data. As a result of these two factors, the data mappings of the two channels are seen as different: $d \neq 1$.

The degree of redundancy and compactness in a view can be measured by the number, and severity, of detected relations. A design is more compact if it has fewer relations indicating redundancy, and more consistent if it has fewer relations indicating inconsistency. For our method, it is only necessary to know that a relation exists, and that it can be resolved. However, generating a quantitative score from these relations would be possible, and useful for many other problems. These relations are specified and visually summarized in Table 1. We discuss each of these relations in detail in the remainder of this section.

R1: Full Redundancy. If two views are showing exactly the same data, there is a full redundancy relation between them. The full redundancy relation is present when two views have the same grouping ($g = 1$) for all channel pairs ($\forall c$). If the channels are the same ($c = 1$), then they also show the same data ($d = 1$). For example, if two bar charts are both grouped by number of cylinders ($g = 1$), and their bar height is mapped to average price, then ($c = 1 \rightarrow d = 1$) is true, i.e., there is a full redundancy relation between them.

R2: Partial Redundancy. Two views A and B are partially redundant if A is showing all data shown by B, as well as some data not shown by B. More formally, two views are considered partially redundant if they have the same grouping ($g = 1$), and for all pairs of channels showing different data ($c = 1 \wedge d \neq 1$), one of the channels is unmapped, and all the unmapped channels consistently belong to the same view ($\exists! D = 0$). For example, consider two bar charts, both grouped by number of cylinders ($g = 1$) and with bar height mapped to average price, but with one chart also indicating the number of cylinders via its fill color channel. When comparing the fill color channels of the charts ($c = 1$), we see that they have different data mappings ($d \neq 1$), and that one of them is not mapped to anything ($\exists! D = 0$). Thus, all the data shown by the one chart is also shown by the other chart.

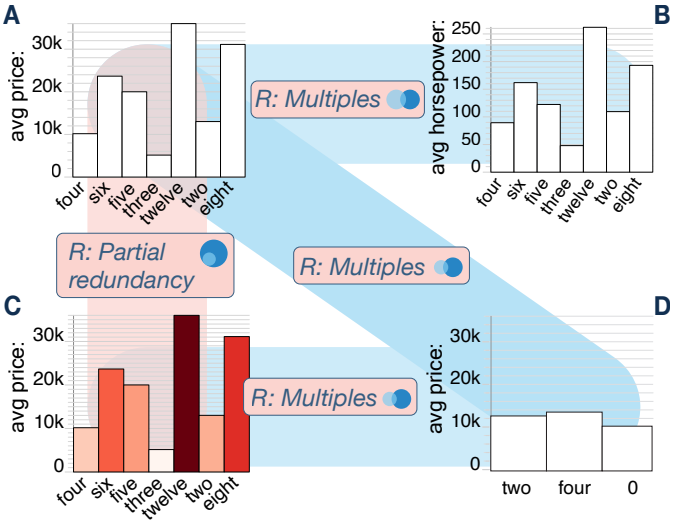

Fig. 4. These four figures illustrate all degrees of redundancy. There is a multiples relation between (a) and (b) since they both have the same grouping, but are showing different data via the bar height channel. Between a and c we see there is a partial redundancy, since (a) is showing the exact same data as (c), but (c) is also showing more data via the fill color. (a) and (d), as well as (c) and (d) are differently grouped multiples showing the same data via the bar height.

R3: Multiples. There are two kinds of multiples: (1) views with same grouping but different data, or, conversely, (2) views with different groupings but same data. As an example of the former, suppose two equally grouped bar charts showing a different quantity via the bar height channel as illustrated in Figure 4a and b. Multiples with different groupings could for example be two differently grouped bar charts showing the same aggregated dimension via the bar height channel (see Figure 4a and d). The multiples relation is specified more precisely in Table 1c-d.

R4: Hallucinator. Corresponding to Kindlmann and Scheidegger's model, a hallucinator is present when the same data are shown in different ways. A hallucinator exists on a canvas if two views with the same chart grouping ($g = 1$), have a common channel ($c = 1$) showing the same data ($d = 1$), but with different visual output ($v \neq 1$).

R5: Confuser. A confuser exists on a canvas if the same channel ($c = 1$) of two views has the same visual output ($v = 1$), but different data mappings ($d \neq 1$). As an example, consider charts using the same fill color (for example, reds) to show different data.

Relations identify redundancies, inconsistencies, and alternative design opportunities. They are detected by iterating over all view permutations and checking whether the permutation satisfies the predicate logic expression that corresponds to the rule. If the expression is satisfied, the relation exists between the views. Each relation has corresponding "resolutions" – operations which resolve the given relation by altering one or more of the affected views.

3.3 User Operations

Operations resolve potential problems between views of a canvas. The main idea behind each operation is to resolve a certain relation by changing or removing one or more views. At a low level, operations can, for example, transfer a data mapping from one view onto another, or replace two views with another view showing the same data. Our set of operations do not exhaustively express all possible combinations of low-level changes, but serves to demonstrate the wide range of possible operations to navigate the semantic space of revised designs.

Our model specifies the following classes of operations, which we summarize in Table 1 along with associated their associated relation(s).

O1: Delete. The first operation is the most simple. If there is a *full redundancy* relation between two views, the user may delete one view. With one of the views removed, the formula for *full redundancy*, $g = 1 \wedge \forall c(c = 1 \rightarrow d = 1)$, will not evaluate to true for that pair of views, since the pair no longer exists.

O2: Homogenize. The *homogenize* operation resolves *hallucinator*, as well as *multiples* relations, where the same data are shown differently. On a high level, the *homogenize* operation makes dissimilar views more similar. For example, consider two bar charts that are showing the same data dimension with a different color scheme (hallucinator), as in Figure 7.1a, or that are using different data domains (multiples), as shown in Figure 6.2. The *homogenize* operation resolves these conflicts by making the visual outputs, or data domains equal for the two views. If visual outputs are made equal, the $v \neq 1$ portion of the *hallucinator* will evaluate to false and thus remove the relation. For multiples with different domains, equalizing the domains will make the views consistent. If only the data domains are different, the operation is presented to the user as *homogenize* data. When the visual outputs differ, the user will see the operation as *homogenize* style, although it also implicitly homogenizes the data domains.

O3: Differentiate. The *differentiate* operation addresses a *confuser*, where two views show different data in the same way. This is achieved by making the views to show different data in different ways (Table 1f). For instance, if different data are shown using the same color scheme, the *differentiate* operation will assign different color schemes to the views. An example of this operation could take two views showing different data, such as age and income, where both views are mapped to the color red. This is ambiguous. Our solution is to use a different color scheme for one of the views. When the visual outputs are made different, i.e., from $v = 1$ to $v \neq 1$, the formula for a *confuser*, $c = 1 \wedge d \neq 1 \wedge v = 1$, will evaluate to false, since $v \neq 1$.

O4: Integrate. The *integrate* operation resolves redundancy to create a visually compact canvas, and can be used to resolve *partial redundancies* and certain *multiples* relations. With a *partial redundancy* relation, there are two possible solutions: delete the view showing the least data, or *integrate* the "missing" mapping into this view while deleting the other (Table 1b). Views sharing a *multiples* relation where the data grouping is the same (Table 1c) can be *integrated* in several ways. Since the multiples relation can only exist between two views, the act of combining these views by integration also removes the relation from the canvas. Views that are highly semantically similar are sensible to *integrate*, provided that: (1) the chart type is the same, and (2) if $d = 1$ for the channel representing the x-axis. There are four ways to perform this *integration*: *overlay, group, stack,* and *mirror*. *Overlay* integrates multiple views into the same coordinate system. This operation can be applied to scatter plots and line charts. Figure 1.2* shows an example of an *overlay* operation when applied to a line chart. The *mirror* operation can be applied to line charts, area charts, and bar charts. This operation first aligns the two views and then mirrors one of them, causing their marks diverge from a common origin in a manner similar to violin plots. We demonstrate an example of this in Figure 5.2b. Similar to the group operation, the *stack* operation stacks views into a single view, turning, for instance, a set of bar charts into a stacked bar chart as shown again in Figure 5.2c. The *group* integration bundles several views into one single view. It can, for example, turn multiple bar charts into a grouped bar chart, as illustrated in Figure 5.2d.

Operations make high-level changes to the canvas, making it more compact or more consistent. For an operation to be applicable to a design, a certain relation must exist. When the user selects a view in the interface, our method reveals available operations to resolve a given relation. When the operation is performed, the corresponding relation is addressed.

Listing 1. Pseudocode of the general execution flow of semantic snapping.

```
def findRelations(views):                                          1
    byView = { view: [] for view in views }                        2
    subsets = permutations(views)                                  3
    for view1, view2 in subsets:                                   4
        for relationFn in allRelations:                            5
            if relationFn(view1, view2):                           6
                for view in subset:                                7
                    byView[view].append({                          8
                        'subset': [view1, view2],                  9
                        'relation': relation })                    10
    return byView                                                  11
                                                                   12
def semanticSnap(views):                                           13
    relationsByView = findRelations(views)                         14
    view = userInput() # User selects a view                       15
    relations = relationsByView.get(view)                          16
    operations = [ findOperation(r) for r in relations ]           17
    display(operations) # User sees operations                     18
    selectedOperation = userInput() # User selects                 19
    newViews = selectedOperation.execute()                         20
    return newViews                                                21
```

Listing 2. Pseudocode of an example relationFn, invoked at Listing 1 line 6, modelling a hallucinator as specified in Table 1e.

```
def isHallucinator(view1, view2):                                  1
    if isSameGrouping(view, view2):                                2
        pairs = findChannelPairs(view1, view2,{                    3
            'd': 1, # same data                                    4
            'v': 0, # different visual output                      5
            'mappedToData': 1  }) # D != 0                         6
        return pairs.length > 0                                    7
    return 0                                                       8
```

3.4 Snapping Algorithm

The goal of our approach is to provide the user with a set of available design-altering operations upon selection of a single view. The outlined algorithm in Listing 1 achieves this goal by identifying all relations and mapping them to operations for any selected view. When an operation is selected, a revised set of views is generated. The relations correspond to the descriptions in Table 1, and can in practice be modeled as constraints or functions.

The first step of the algorithm is to identify all relations between all subsets of views. The logic of each relation is outlined in Table 1, and is mapped to a relation function that takes in two views, and returns `1` if the relation exists between the views, or `0` if the relation does not exist between the views, as exemplified in Listing 2. Consider line 5 in Listing 1. Here we loop over each `relationFn` (relation function), and invoke it using two views as arguments. If this invocation returns `1`, the relation exists between the two views. When relations are identified for all views, they are grouped by the views they affect. When the user selects a view, the view's relations are looked up and used to identify which operations are possible. Line 17 of Listing 1 illustrates how relations are mapped to corresponding operations. When an operation is executed, a new set of views is generated and displayed to the user. With this new set of views, the algorithm is re-run, recomputing relations and potential operations.

4 Workflow & Implementation

Our method improves and refines canvas designs incrementally. In order to create a canvas, single visualizations must also be generated. While the creation of single visualizations is not a part of our method, we used the existing Visception visualization editor environment [20] as a basis to realize and demonstrate our method. Our semantic snapping interface enables the user to build a canvas using simple drag & drop operations from a visualization gallery and to optimize the design with semantic snapping step by step. In order to build a canvas, the user places individual views into a grid layout and is presented with a set of potential operations at every step. While browsing the operations,

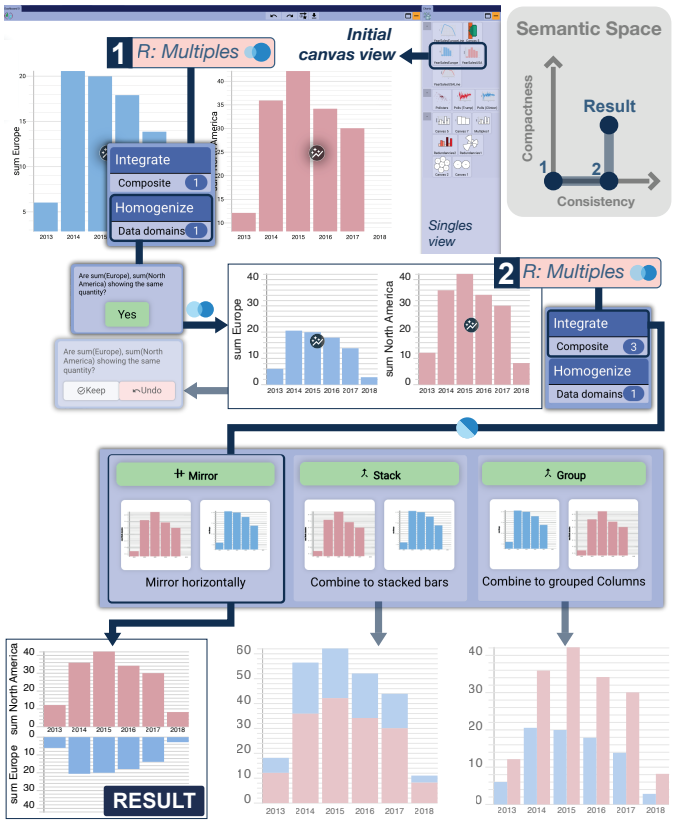

Fig. 5. Semantic snapping interface and workflow. The semantic snapping interface presents the user with a canvas to place single views within a larger layout. A clickable button on top of each view exposes the possible operations available to resolve a relation between two views, in this case a *multiples* relation (1). The user *homogenizes* the two views after confirming that the y-axes are semantically the same. This step represents a move towards increased consistency in semantic space. The user may choose to keep or undo the result of any performed operation. On execution of an operation, we recompute the set of possible operations. The user may next perform any one of three available *integration* operations to resolve a *multiples* relation in this view (2). The user selects the *mirror* operation to increase the compactness of the visualization in semantic space.

the user is presented with information about what they do and what potential problems in the design they resolve.

We use two primary views in our interface: the *singles view* and the *canvas view*. Both are shown in Figure 5.1. The *singles view* is a view from where the user can drag single visualizations into the *canvas view*. Each tile in the singles view represents a data source, which, when clicked, expands to more tiles–one for each single visualization of that dataset. We highlight two of these single visualization tiles in Figure 5, which have been dragged into the canvas view.

4.1 Workflow

The main workflow of using semantic snapping is integrated into the *canvas view*. We demonstrate this workflow in Figure 5. As the user is constructing a canvas with multiple views, our approach detects relations and makes the corresponding operations available along every step of the way. Whenever an existing view is added to the design, or modified by an operation, relations are re-detected and the corresponding operations are updated. In order to see possible operations, the user clicks on the view of interest. When the view is clicked, a menu appears, showing how many operations are available per category (homogenize data, homogenize style, differentiate, and integrate). Only categories with available operations are displayed. The user can then click on a category and see all available operations as tiles. Each operation tile informs the user of the potential problem and its solution. Consider the canvas in our workflow example where there is a *multiples* relation between two bar charts (Figure 5.1), showing sum of sales in Europe, and North America. To verify that the fields are semantically equal, the operation tile will ask the user "Are sum(Europe) and sum(North America) representing the same quantity?". If the user clicks "Yes", the domains are made consistent using a *homogenize operation*. When any operation is executed, the user is given the option to undo or keep it as shown in Figure 5.2. If "keep" is clicked, the current canvas is re-evaluated and the user can proceed to explore other operations, add new views, or otherwise customize the setup.

4.2 Implementation

We implemented semantic snapping within the framework of Visception [20], an application written in Javascript ES6 using VueJS for user interface components and D3 for SVG rendering. The underlying framework of the authoring tool was leveraged to realize the relations and operations to support semantic snapping.

The relations are specified as functions taking in two views as parameters, returning true if they match the given relations. Operations are also defined by functions that take in a set of views, and a detected relation. From this set of views and the detected relation, we can infer what operations are possible, and also take into account the specific chart type and other edge cases. When the user modifies a design, a pipeline of four steps is run. First, all relations are detected for all sets of relations as shown in Listing 1, and the relations are stored so that they can be looked up on a per-view basis. When the detection is done, the editor is ready for the user to specify which view to change. When the user clicks on a view, all relations and corresponding sets of views are looked up, and all possible operations are computed. When the user selects an operation, it is executed, and the existing set of views is modified, and relations are recomputed.

5 Case Studies

We next demonstrate our semantic snapping method workflow in three case studies. These studies include data from the 2016 US Election Results, Nightingale's historic Soldier Morbidity & Mortality, and a COVID-19 dataset. We selected these particular datasets as they are both representative of the type of data we expect to be used for our approach, as well as for their familiarity and applicability to the visualization community. Each case study represents a possible pathway through semantic space from an initial to a more compact and consistent design. We illustrate such pathways through semantic space with a semantic space map positioned in the upper right of each associated figure.

5.1 2016 Election Results

In this case study we demonstrate a user flow that identifies *confuser* and *multiples* relations that are resolved via *differentiate* and *integrate* operations. We also use this study case to demonstrate a flexible workflow whereby the user may perform and then revert an operation to arrive at their preferred final design.

In Figure 1 we see an initial canvas comprised of three views depicting data from the 2016 US Election. These views show election polls over time for the two main candidates (left, top view: Democrats, bottom view: Republicans), as well as average pollster ratings for the two candidates (right scatter plot view). We localize our position in semantic space at the origin (pos. 1) in the map in the upper right of Figure 1. We quickly identify a *confuser* relation between the bottom line chart and the right scatter plot (Figure 1.1). This is because the color channels of the two views are using the color red as visual output. This is particularly misleading in the right scatter plot view, where each dot represents a pollster, since red may indicate that all pollsters are advocates for the Republican party. Since these charts are using the same visual output to represent different data domains, our model recommends a *differentiate* operation to change their respective visual outputs. We change the color of the scatter plot to green, as this is color is more neutral. We keep the red color in the lower left view; this makes sense to remain red, as this is the color of the US Republican

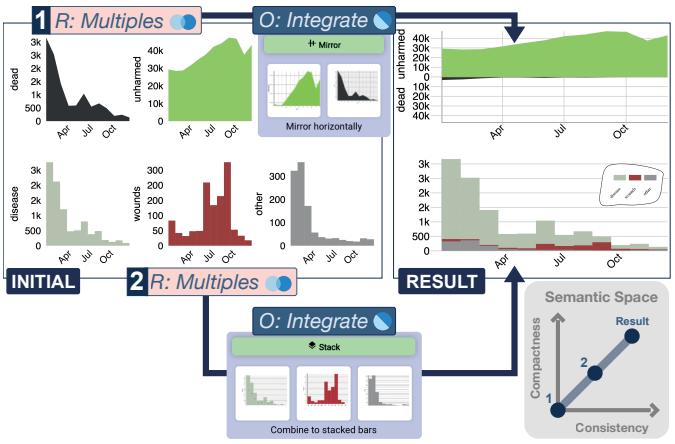

Fig. 6. Case study workflow demonstrating semantic snapping to resolve two *multiples* relations in the canvas depicting 1858 solider morbidity & mortality from the Nightingale dataset. This end result is a semantically consistent and compact visualization.

Party. In our semantic map we have increased the consistency of our canvas and are now at pos. 2.

We next observe a correspondence between the two leftmost views. These views share the same x-axis, but show a different quantity on the y-axis (bottom view: avg. Trump, top view: avg. Clinton). In other words, they share a *multiples* relation. We consequently can *integrate* them to produce a more compact visualization using the *mirroring* (Figure 1.2) or *overlaying operation* (Figure 1.2*). Integrating these charts additionally produces a more consistent visualization, because both *overlay* and *mirror* perform an implicit axis homogenization step. To mirror or overlay, we simply select and execute either operation. In this case we first try *overlay* (Figure 1.2*). However, while the result is very compact, it is difficult to read. We choose to do a different *integrate* operation to resolve the *multiples* relation. We revisit the available operations for this relation and select this time to *mirror* the two views (Figure 1.2). This path in semantic space leads us to an equally consistent, while slightly less compact visualization. The resulting chart composition, however, is easier to read, which illustrates the flexibility of our approach in incorporating user goals and decision processes.

5.2 Nightingale Soldier Morbidity & Mortality in 1858

In this case study we use the popular Nightingale solider morbidity & mortality dataset to illustrate the use of additional operations to resolve *multiples* between canvas views.

Figure 6 shows the fate of British soldiers in the year of 1858 in the Crimean War. In our traversal through semantic space we begin again at the origin in our semantic map at the upper right (pos. 1). The top two views of the initial canvas display area charts. The left view plots the number of soldier deaths over time while the right plots the number of unharmed soldiers over time. Because these two views comprise the same grouping (soldier morbidity & mortality) with a different data domain plotted onto the y-axis, we can say that these views share a *multiples* relation (Figure 6.1). We can compact the views by *mirroring* (integrating) the views. When the charts are mirrored, the domains are also implicitly homogenized, which additionally increases the consistency of the resulting chart. This brings us to pos. 2 in our semantic map. It would also be possible to keep the multiples relation, and only homogenize the data domains on the y-axis. However, our choice to compact the views with mirroring illustrates that, while our model can show potential problems in a canvas, it is ultimately up to the user to decide how they wish to design their visualization.

We may also compact the three bar chart views arrayed along the bottom of the canvas that show different causes of soldier death (Figure 6, bottom of initial canvas). Each of the three views plots by number of deaths caused by disease, wounds, or other, respectively, over time. Because these views share the same grouping (cause of solider death) and data domain on the x-axis (time), but their data domain plotted to the y-axis is different, we identify a *multiples* relation (Figure 6.2). However, by looking at the data, we also know that the y-axes represent the same *semantic* quantity – i.e., number of deaths. As a consequence of this, they can be integrated via a *group* or *stack* operation. Since our goal is to produce a maximally compact visualization, we choose to *stack* the views. This step additionally homogenizes the data domains for increased consisstency. This brings us to the result point in our semantic map.

5.3 COVID-19 in Germany

COVID-19 dashboards are now ubiquitous in society with great importance for public health. However, integration of numerous charts to demonstrate various data aspects in a dashboard may introduce numerous possible conceptual and perceptual pitfalls. For our final case study we demonstrate the ability of our approach to assist in resolving the complexities of creating a semantically consistent COVID-19 dashboard. We demonstrate an overview of this workflow in Figure 7.

The initial layout as shown in the central part of Figure 7 shows six charts. The first chart column shows COVID-19 deaths grouped by age, where the top bar chart represents total deaths and the bottom streamgraph indicates deaths over time. The second column displays COVID-19 cases that are again grouped by age, with the top bar chart indicating summed cases while the below streamgraph shows case load over time. The rightmost column shows two pie charts grouped by gender, where the top chart shows cases while the bottom shows deaths. Our goal is create a dashboard using multiple chart types that clearly presents the COVID-19 cases and deaths distributed by age group and gender in Germany.

As in the prior case examples we have a number of different routes through which we can traverse the semantic space, as indicated in map in the lower middle of Figure 7. In this case study we describe the navy blue indicated route, beginning with the *confuser* that our system identifies between the pie chart showing COVID-19 deaths and the COVID-19 cases over time chart (Figure 7.1). This is a *confuser* because the female segment in the pie chart uses the same blue as for the color mapping in the chart showing cases grouped by age over time. We perform the suggested *differentiate* operation to clarify the different groupings by changing the color mapping of genders to a light green for males and pink for females. This increases consistency in semantic space. We next resolve the *hallucinator* relation between the two pie charts by *homogenizing* the color mapping of both charts so the cases chart receives the same green and pink color mapping to males and females, respectively, for increased consistency in semantic space (Figure 7.2).

Two additional *hallucinators* exist, one between the age-grouped COVID-19 cases charts (Figure 7.3) and the second between the age-grouped COVID-19 deaths charts (Figure 7.4). Each are classified as *hallucinators* because the chart data and groupings are identical but they do not share the same color mapping. We resolve the first *hallucinator* between the two case charts with a *homogenize* operation that applies the same continuous blue color mapping in the streamgraph to the bar chart. We resolve the second *hallucinator* the same way for the deaths chart, by applying the continuous red color mapping in the deaths over time streamgraph to the corresponding bar chart. Each of these operations sequentially improves consistency in semantic space.

We may compact our dashboard visualization by resolving two *multiples* relations that our system identifies. The first *multiples* relation exists between the two bar charts, which we *integrate* into a single grouped row chart (Figure 7.5). A second compacting step in semantic space *integrates* the two streamgraphs in a mirroring operation to resolve their *multiples* relation (Figure 7.6). In both *integration* steps the system implicitly *homogenizes* the data domains as well. The resulting COVID-19 dashboard in the right of Figure 7 is a much more compact and semantically consistent visualization with the aid of our approach.

6 Discussion & Limitations

We realized our method by implementing and embedding it into the Visception visual authoring system [20]. For specifying rules, it is

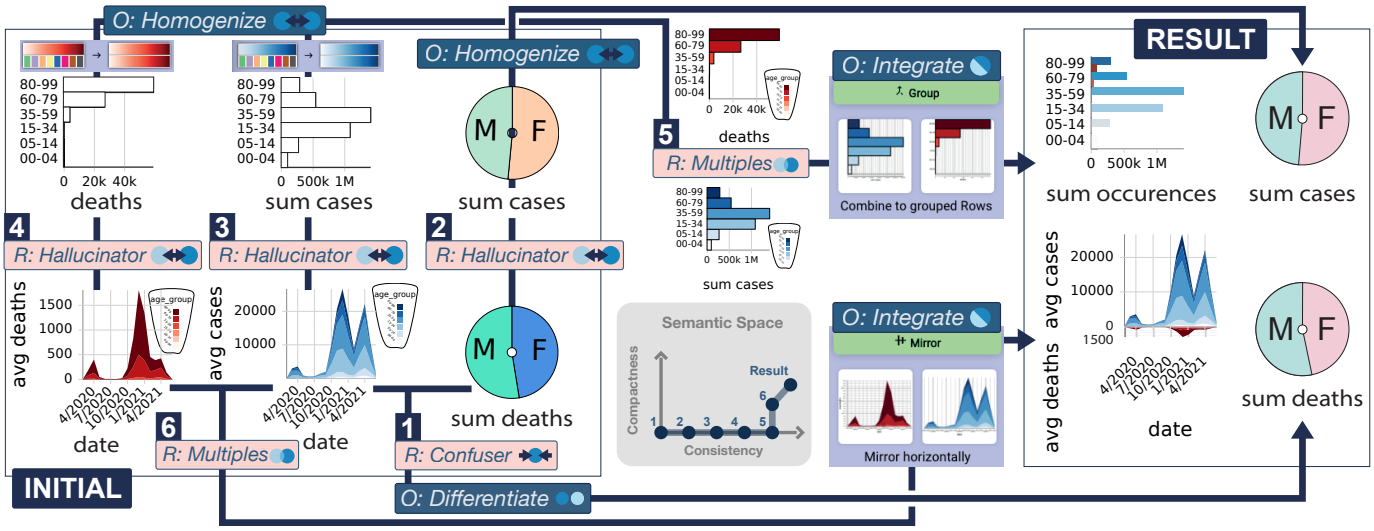

Fig. 7. Case study workflow demonstrating semantic snapping to resolve a COVID-19 dashboard with a *confuser*, several *hallucinators* and two *multiples* relations. We resolve these relations via a series of operations that include *homogenize*, *differentiate*, and *integrate*.

necessary to be able to retrieve detailed information about each channel mapping, as well as the chart type and grouping. We believe this information should be accessible in most frameworks. Specifying chart operations requires more knowledge about the underlying architecture and programming interface. For instance, our framework had support for nesting visualizations, which was highly useful for generating grouped and stacked charts. For example, two bar charts grouped the same, showing two different quantitative attributes can be grouped into a stacked bar chart, where the outer bar is grouped the same, and the inner grouping is one bar for each of the two attributes. In principle, however, we believe that our approach is applicable to a variety of different visualization systems, even if the specifics of how individual operations are implemented will differ. At present, our prototype only supports a limited number of common chart types: line charts, bar charts, pie charts, scatter plots, and streamgraphs. For addressing a wider range of charts, we believe that a framework for unifying the reasoning about these charts could allow for a more generally applicable realization of semantic snapping. We believe that frameworks that allow for expressing and modifying charts on a general level are ideal for implementing our semantic snapping concept.

At present, a canvas is limited to views of a single tabular dataset. For dealing with more advanced multi-table setups, our approach would have to be built on top of an additional abstraction over these different data topologies. Such an abstraction layer would enable a more general implementation of semantic snapping that could also facilitate the incorporation of other dataset types such as network data. Likewise, techniques such as interactive linking & brushing and crossfiltering are frequently used in multi-view visualizations but currently not explicitly supported in our framework, which also represents an interesting challenge for future research.

The layout of the views is an important factor in an overall design, which is currently not addressed in our approach but is definitely worthy of further investigation. It would be possible to specify more advanced relations by incorporating the spatial arrangement of individual views. For instance, if two views are sufficiently spatially separated, a *confuser* could be classified as less severe. Likewise, taking into account spatial arrangement could extend the space of operations as, for example, a *differentiate* operation could move views further apart or even add graphical separators or visual groupings. This is an important direction for future research. Related to this, since currently the number of possible operations is sufficiently small, we do not perform any explicit sorting. However, a larger number of possibilities would necessitate to incorporate an appropriate mechanism for prioritizing operations. We believe that such a sorting of potential revisions using for example Qu & Hullman's effectiveness preservation score [30] would be useful when there are many potential solutions.

While the operations of our method alter the design and resolve potential inconsistencies, it would provide more flexibility and design freedom if they were customizable. For example, the mirror operation could be parameterized by letting the user decide the spacing between the views and the placement of the labels. A general assumption of our method is that the existing views are already well-designed individually. However, when a view is placed into a multi-view design, the aspect ratio and size will change. Keeping font sizes and other styles consistent across a design becomes tedious. While our operations do combine views and optimize design, they do not at present allow for a final fine-tuning of, for instance, font sizes. Such global controls are not a part of our method, but would be highly helpful in any multi-view visualization design process.

Finally, our method is based on general principles in the sense of Kindlmann and Scheidegger [18] and thus does not take into account an explicit task specification. While this focus was deliberate, since meaningfully characterizing user tasks is a significant challenge of its own that would also explode the design space, we still believe that exploring how different types of general user tasks could guide the evaluation of relationships and the presentation of operations is an important topic for future research.

7 Conclusion

We presented semantic snapping, a semi-automatic guided method that allows for incrementally refining multi-view visualizations. While previous work on multi-view visualizations has given us guidelines and constraints for reasoning about and improving visualizations, we further operationalized these concepts by (1) specifying relations between views precisely, and (2) proposing how each relation can be resolved by an operation. Each operation is a step in the semantic space with two axes representing the consistency and compactness. Furthermore, we presented a prototype implementation of our method, where users can perform operations to gradually refine a multi-view visualization design. In the future, believe that our approach to specifying relations and corresponding operations can be applied to more elements of multi-view visualizations such as their layout. Furthermore, many additional rules and guidelines for single visualizations could be adapted to or extended for multi-view visualizations.

Acknowledgments

The research presented in this paper was supported by the MetaVis project (#250133) funded by the Research Council of Norway as well as the VIDI project (#813558) funded by the the Trond Mohn Foundation in Bergen, Norway.

REFERENCES

[1] M. Behrisch, M. Blumenschein, N. W. Kim, L. Shao, M. El-Assady, J. Fuchs, D. Seebacher, A. Diehl, U. Brandes, H. Pfister, et al. Quality metrics for information visualization. *Computer Graphics Forum*, 37(3):625–662, 2018. doi: doi.org/10.1111/cgf.13446

[2] J. Bertin. *Semiology of Graphics*. University of Wisconsin Press, 1983.

[3] F. Bolte and S. Bruckner. Measures in visualization space. In *Foundations of Data Visualization*, pp. 39–59. Springer, 2020. doi: 10.1007/978-3-030-34444-3_3

[4] A. Cantu, O. Grisvard, T. Duval, and G. Coppin. Identifying the relationships between the visualization context and representation components to enable recommendations for designing new visualizations. In *Proc. International Conference on Information Visualisation*, pp. 20–28, 2017. doi: 10.1109/iV.2017.55

[5] X. Chen, W. Zeng, Y. Lin, H. M. Al-Maneea, J. Roberts, and R. Chang. Composition and configuration patterns in multiple-view visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):1514–1524, 2021. doi: 10.1109/TVCG.2020.3030338

[6] W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American statistical association*, 79(387):531–554, 1984. doi: 10.1080/01621459.1984.10478080

[7] M. Correll, M. Li, G. Kindlmann, and C. Scheidegger. Looks good to me: Visualizations as sanity checks. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):830–839, 2018. doi: 10.1109/TVCG.2018.2864907

[8] V. Dibia and Ç. Demiralp. Data2vis: Automatic generation of data visualizations using sequence-to-sequence recurrent neural networks. *IEEE Computer Graphics and Applications*, 39(5):33–46, 2019. doi: 10.1109/MCG.2019.2924636

[9] A. Diehl, A. Abdul-Rahman, M. El-Assady, B. Bach, D. A. Keim, and M. Chen. Visguides: A forum for discussing visualization guidelines. In *Proc. EuroVis (Short Papers)*, pp. 61–65, 2018. doi: 10.2312/eurovisshort.20181079

[10] M. Elshehaly, R. Randell, M. Brehmer, L. McVey, N. Alvarado, C. P. Gale, and R. A. Ruddle. Qualdash: Adaptable generation of visualisation dashboards for healthcare quality improvement. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):689–699, 2020. doi: 10.1109/TVCG.2020.3030424

[11] U. Engelke, A. Abdul-Rahman, and M. Chen. Visupply: A supply-chain process model for visualization guidelines. In *Proc. International Symposium on Big Data Visual and Immersive Analytics*, pp. 1–9, 2018. doi: 10.1109/BDVA.2018.8534029

[12] M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts. Visual comparison for information visualization. *Information Visualization*, 10(4):289–309, 2011. doi: 10.1177/1473871611416549

[13] L. Grammel, M. Tory, and M.-A. Storey. How information visualization novices construct visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):943–952, 2010. doi: 10.1109/TVCG.2010.164

[14] T. Horak, A. Mathisen, C. N. Klokmose, R. Dachselt, and N. Elmqvist. Vistribute: Distributing interactive visualizations in dynamic multi-device setups. In *Proc. ACM CHI*, pp. 1–13, 2019. doi: 10.1145/3290605.3300846

[15] S. E. Hudson. Adaptive semantic snaping—a technique for semantic feedback at the lexical level. In *Proc. ACM CHI*, pp. 65–70, 1990. doi: 10.1145/97243.97253

[16] H. Kim, D. Moritz, and J. Hullman. Design patterns and trade-offs in responsive visualization for communication. *Computer Graphics Forum*, 40(3):459–470, 2021. doi: 10.1111/cgf.14321

[17] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic, and H. Pfister. Data-driven guides: Supporting expressive design for information graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):491–500, 2017. doi: 10.1109/TVCG.2016.2598620

[18] G. Kindlmann and C. Scheidegger. An algebraic process for visualization design. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2181–2190, 2014. doi: 10.1109/TVCG.2014.2346325

[19] S. M. Kosslyn. Understanding charts and graphs. *Applied cognitive psychology*, 3(3):185–225, 1989. doi: 10.1002/acp.2350030302

[20] Y. S. Kristiansen and S. Bruckner. Visception: An interactive visual framework for nested visualization design. *Computers & Graphics*, 92:13–27, 2020. doi: 10.1016/j.cag.2020.08.007

[21] H. Lin, D. Moritz, and J. Heer. Dziban: Balancing agency & automation in visualization design via anchored recommendations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–12, 2020. doi: 10.1145/3313831.3376880

[22] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proc. ACM CHI*, pp. 123:1–123:13, 2018. doi: 10.1145/3173574.3173697

[23] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Transactions On Graphics*, 5(2):110–141, 1986. doi: 10.1145/22949.22950

[24] J. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1137–1144, 2007. doi: 10.1109/TVCG.2007.70594

[25] A. McNutt. What are table cartograms good for anyway? an algebraic analysis. *Computer Graphics Forum*, 40(3):61–73, 2021. doi: 10.1111/cgf.14289

[26] A. McNutt and G. Kindlmann. Linting for visualization: Towards a practical automated visualization guidance system. In *Proc. IEEE VIS Workshop on the Creation, Curation, Critique and Conditioning of Principles and Guidelines in Visualization*, 2018.

[27] D. Moritz, C. Wang, G. Nelson, H. Lin, A. M. Smith, B. Howe, and J. Heer. Formalizing visualization design knowledge as constraints: Actionable and extensible models in draco. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):438–448, 2019. doi: 10.1109/TVCG.2018.2865240

[28] T. Munzner. *Visualization Analysis and Design*. A K Peters/CRC Press, 2015. doi: 10.1201/b17511

[29] B. Ondov, N. Jardine, N. Elmqvist, and S. Franconeri. Face to face: Evaluating visual comparison. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):861–871, 2019. doi: 10.1109/TVCG.2018.2864884

[30] Z. Qu and J. Hullman. Evaluating visualization sets: Trade-offs between local effectiveness and global consistency. In *Proc. BELIV*, pp. 44–52, 2016. doi: 10.1145/2993901.2993910

[31] Z. Qu and J. Hullman. Keeping multiple views consistent: Constraints, validations, and exceptions in visualization authoring. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):468–477, 2018. doi: 10.1109/TVCG.2017.2744198

[32] D. Ren, T. Höllerer, and X. Yuan. iVisDesigner: Expressive interactive design of information visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2092–2101, 2014. doi: 10.1109/TVCG.2014.2346291

[33] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):789–799, 2019. doi: 10.1109/TVCG.2018.2865158

[34] A. Sarikaya, M. Correll, L. Bartram, M. Tory, and D. Fisher. What do we talk about when we talk about dashboards? *IEEE Transactions on Visualization and Computer Graphics*, 25(1):682–692, 2018. doi: 10.1109/TVCG.2018.2864903

[35] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum*, 33(3):351–360, 2014. doi: 10.1111/cgf.12391

[36] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer. Vega-lite: A grammar of interactive graphics. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):341–350, 2017. doi: 10.1109/TVCG.2016.2599030

[37] R. Shadoan and C. Weaver. Visual analysis of higher-order conjunctive relationships in multidimensional data using a hypergraph query system. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2070–2079, 2013. doi: 10.1109/TVCG.2013.220

[38] S. Silva, J. Madeira, and B. S. Santos. There is more to color scales than meets the eye: a review on the use of color in visualization. In *Proc. International Conference on Information Visualization*, pp. 943–950. 10.1109/IV.2007.113, 2007.

[39] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1):52–65, 2002. doi: 10.1109/2945.981851

[40] A. Swearngin, C. Wang, A. Oleson, J. Fogarty, and A. J. Ko. Scout: Rapid exploration of interface layout alternatives through high-level design constraints. In *Proc. ACM CHI*, pp. 1–13, 2020. doi: 10.1145/3313831.3376593

[41] E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics

Press, 1986.
[42] E. R. Tufte. *Envisioning Information*. Graphics Press, 1990.
[43] S. van den Elzen and J. J. van Wijk. Small multiples, large singles: A new approach for visual data exploration. *Computer Graphics Forum*, 32(3):191–200, 2013. doi: 10.1111/cgf.12106
[44] L. Wilkinson. *The Grammar of Graphics*. Springer-Verlag New York, 2005. doi: 10.1007/0-387-28695-0
[45] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):649–658, 2015. doi: 10.1109/TVCG.2015.2467191
[46] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Towards a general-purpose query language for visualization recommendation. In *Proc. Workshop on Human-In-the-Loop Data Analytics*, pp. 4:1–4:6, 2016. doi: 10.1145/2939502.2939506
[47] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proc. ACM CHI*, pp. 2648–2659, 2017. doi: 10.1145/3025453.3025768
[48] E. Zgraggen, R. Zeleznik, and S. M. Drucker. PanoramicData: Data analysis through pen touch. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2112–2121, 2014. doi: 10.1109/TVCG.2014.2346293
[49] S. Zhu, G. Sun, Q. Jiang, M. Zha, and R. Liang. A survey on automatic infographics and visualization recommendations. *Visual Informatics*, 4(3):24–40, 2020. doi: 10.1016/j.visinf.2020.07.002
[50] Y. Zhu. Measuring effective data visualization. In *Proc. International Symposium on Visual Computing*, pp. 652–661, 2007. doi: /10.1007/978-3-540-76856-2_64