Semi-Automatic Layout Adaptation for Responsive Multiple-View Visualization Design

Wei Zeng, *Member, IEEE*, Xi Chen, Yihan Hou,
Lingdan Shao, Zhe Chu, and Remco Chang, *Member, IEEE*

Abstract—Multiple-view (MV) visualizations have become ubiquitous for visual communication and exploratory data visualization. However, most existing MV visualizations are designed for the desktop, which can be unsuitable for the continuously evolving displays of varying screen sizes. In this paper, we present a two-stage adaptation framework that supports the automated retargeting and semi-automated tailoring of a desktop MV visualization for rendering on devices with displays of varying sizes. First, we cast layout retargeting as an optimization problem and propose a simulated annealing technique that can automatically preserve the layout of multiple views. Second, we enable fine-tuning for the visual appearance of each view, using a rule-based auto configuration method complemented with an interactive interface for chart-oriented encoding adjustment. To demonstrate the feasibility and expressivity of our proposed approach, we present a gallery of MV visualizations that have been adapted from the desktop to small displays. We also report the result of a user study comparing visualizations generated using our approach with those by existing methods. The outcome indicates that the participants generally prefer visualizations generated using our approach and find them to be easier to use.

Index Terms—Multiple-view visualization, responsive design, layout adaptation, mobile devices

1 Introduction

Recent years have witnessed an increased demand for visual data access using different displays beyond the traditional desktop environment [48]. However, with few exceptions (*e.g.*, [26], [38]), existing guidelines for visualization design remain primarily tailored for the desktop, which do not transfer well to small displays such as mobile phones and tablets. In response to this knowledge gap, researchers have promoted the concept of *responsive visualization* – designs that aim to maintain the readability of a visualization when adapting its use across displays of varying sizes [34].

However, existing systems for responsive visualization design focus on adapting a single chart (*e.g.*, [25], [35], [59], [61]) while neglecting multiple-view (MV) visualizations that composite multiple views in a cohesive manner [47]. There has been an increasing demand for MV visualizations on mobile devices, such as dashboards (*e.g.*, [20], [31], [63]) that arrange multiple visualizations together [50]. It is becoming increasingly common to communicate information through MV visualizations on mobile devices [25]. Unfortunately, although it is clear that researchers and developers are cognizant of the importance of adapting MV visualizations to different display devices [51], developers today are still often required to manually design for both desktop and mobile devices. This practice of having separate designs of MV visualizations for various displays increases the burden for development, updates, and maintenance.

To address this need, in this work, we advocate for responsive MV visualization design across displays with different sizes, with a focus on *layout design* of MV visualizations. Our algorithm aims to preserve view properties and topology of the input MV visualization in order to minimize the adaptation effort for users when switching from the source to the target display. This is a non-trivial task: MV visualization design includes the consideration of a number of factors, including view properties, coordination types, and display viewport [52]. Existing approaches that do not consider the problem holistically may result in generating unsatisfactory layouts, such as having too much white space by uniform scaling, or distorting aspect ratios of the views by scale-and-stretch (Figure 2(a)). Other techniques such as vertical stacking (Figure 2(b)) and horizontal scrolling (Figure 2(c)) may result in visualizations requiring substantially more interactions that hinder visual analysis since users need to recall and connect information across viewports [16].

To improve upon previous techniques, we consider the challenge of responsive layout for MV visualizations, and distill a list of design requirements and considerations (Sect. 3.2). Based on the requirements, we present a two-stage framework that retargets and tailors layouts of desktop MV visualizations for small displays. In the first stage, we cast layout retargeting as an optimization problem and develop an automated layout retargeting algorithm, *AdaMV*, that includes a tree traversal method that first organizes views into a hierarchical tree structure and traverses the tree to form view groups into target viewports (Sect. 4.1). Next, using a simulated annealing (SA) technique, the algorithm iteratively adjusts the view layout, and finally produces an optimal solution that matches the geometric and topological properties of the source MV visualization (Sect. 4.2). In the second stage, we provide a semi-automatic view tailoring approach (Sect. 4.3) that allows a designer to fine-tune the views using a rule-based auto configuration method and supports propagation of view layout tailoring with a visual interface. We demonstrate the feasibility and expressivity of our proposed approach using a gallery of desktop MV visualizations adapted to small-size displays such as mobile phones and tablets

- *W. Zeng and Y. Hou are with the Hong Kong University of Science and Technology (Guangzhou) and the Hong Kong University of Science and Technology. E-mail: {weizeng@, yhou073@connect}.hkust-gz.edu.cn.*
- *X. Chen, L. Shao, and Z. Chu are with Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, and also with University of Chinese Academy of Sciences. E-mail: {xi.chen2, ld.shao, zhe.chu}@siat.ac.cn.*
- *R. Chang is with Tufts University. E-mail: remco@cs.tufts.edu*

Manuscript received xx xx, 202x; revised xx xx, 202x.

(Sect. 5.1). We conducted a user study showing that our method can generate more user-preferable designs than existing methods of scale-and-stretch, vertical stacking, and horizontal scrolling (Sect. 5.2). We also present the results of a run-time analysis of our approach and discuss possible solutions for improving the efficiency of our algorithm (Sect. 5.3). Lastly, we discuss the participants' feedback on the tailoring interface (Sect. 5.4).

In sum, our work makes the following contributions:

- We summarize the requirements and considerations for responsive MV visualization design and model them as an optimization problem.
- We design *AdaMV*—an automated algorithm using simulated annealing to iteratively adjust the view layout based on the geometric and topological properties of the views. *AdaMV* is complemented with a semi-automatic interactive interface that consists of a rule-based auto configuration method and a visual interface that allows for propagating view edits. The project is available at https://hkust-cival.com/projects/adaMV/.
- We conducted a user study and demonstrate that the MV visualizations generated using our approach are preferred by the participants and rated to be more suitable for analytics than those generated using previous methods.

2 Related Work

Responsive Visualization. Various types of displays with different screen sizes, including mobile phones and tablets, are becoming commercially available. The trend gives rise to the demand for visual data access beyond the desktop [48]. Recent years have witnessed increasing efforts on improving visualization design for these varied display sizes and modalities. On the one hand, some studies aimed to curate design guidelines for different displays. For example, Blascheck et al. [10] found a user preference for bar charts and donut charts over radial bar charts for data comparison on a smartwatch. Brehmer et al. [12] compared the performance of animation and small multiples for trend visualization on mobile phones. These studies enrich the nuanced understanding of visualization design on small displays, yet most of them focus on specific data types (*e.g.*, range data [11]), chart types (*e.g.*, bar, donut, and radial bar charts [10]), or application contexts (*e.g.*, public transit [32]).

On the other hand, some other studies aimed to scale up desktop visualization design to different displays, so that the visualizations remain informative and legible. The process of adapting the visualization design across multiple display sizes is referred as *responsive visualization* [24], [25], [34]. Naïve techniques such as uniform or proportionate rescaling may cause issues like overflow, overlapping content, and tiny text, due to the limited display space. Instead, with responsive visualization, the contents of the visualizations are taken into consideration [62]. Content-aware methods decompose a visualization into declarative formats, which can be intelligently configured to fit small displays using automatic algorithms [59] or interactive user interfaces [25]. Recently, Wu et al. [61] applied deep learning models to learn optimal chart parameters (including aspect ratio and orientation) that can automatically recommend chart layout, while Kim et al. [35] developed an automated design recommender for the responsive transformation of a source visualization by approximating the loss of support for identification, comparison, and trend tasks. Kim et al. [34] further summarized key considerations when adapting visualization designs across display sizes, especially from large screens to small screens, and argued that trade-offs among competing goals, such as graphical density, layout challenges, interaction complexity, and loss of information, must be carefully considered.

Our work shares a similar motivation as these prior works. However, instead of focusing on designing responsive visualization for a single chart, our work emphasizes the adaptation of MV visualizations. In particular, we contribute an automatic layout adaptation technique (Sect. 4.2) based on the geometric and topological properties of the source MV visualization, and an authoring system (Sect. 4.3) that allows propagating design edits of MV visualizations across different screen sizes.

Multiple-View Visualization. MV visualization is a specific visual data exploration technique that combines multiple views into a single cohesive representation [47]. A well-designed MV visualization can support a user in exploring complex data from different perspectives [47]. A series of systems (*e.g.*, Snap-Together Visualization [42], Improvis [56], and ComVis [41]) and commercial software (*e.g.*, Tableau [4], Spotfire [3], and PowerBI [2]) make use of MV designs. The wide use of MV has led to research that can improve the understanding of the design space of MV visualization. Javed and Elmqvist [29] classified view composition patterns into five categories: *juxtaposition*, *superimposition*, *overloading*, *nesting*, and *integration*, based on which they defined a unified design space for composite visualization. Sarikaya et al. [51] analyzed the design space of dashboards that typically arranges multiple views together, while Ma et al. [40] developed LADV that can automatically recognize view types and layouts in dashboard images and sketches. Recently, Chen et al. [14] identified composition and configuration patterns in MV visualizations by analyzing MV visualizations in practice. Shao et al. [52] analyzed the impacts of various design factors on MV layouts, showing that view and coordination types had a significant impact on layout design. Chen et al. [13] reviewed the coordination patterns from existing theories and applications and developed a coordination framework that allows users to easily construct coordination among multiple coordinated views.

However, while there have been extensive research and development on MV visualizations, the design guidelines are mainly curated for desktop displays (*e.g.*, [9], [46]). Due to the emerging ubiquity of mobile devices, there is also a growing interest in developing MV visualizations on small screens. For instance, Sadana and Stasko [50] identified multiple layout design goals for MV visualization on tablets, such as maximizing the size of each view and keeping all views on screen. However, they considered only vertical stacking and grid-based view layouts, which underutilizes the richness of view layouts identified in practice [14]. Vistribute [27] further considers properties and relationships of multiple views, devices, and user preferences. As a result, Vistribute can automatically generate view distribution with compelling quality as manual designs by experts.

Our work differs from these prior works in that we aim to develop to develop a responsive MV visualization tool that adapts the layout of a MV visualization designed for the desktop to small displays, rather than designing MV visualizations for mobile devices from scratch. Badam and Elmqvist [8] recently showed that *boundary* layout is more preferable than *overview* layout when adapting to MV visualizations to small screens. However, boundary layout requires scalable and space-efficient visualizations, such as horizon graphs [23] for filled line charts and space-filling bar chart [15] for bar chart, which may not be always available for any

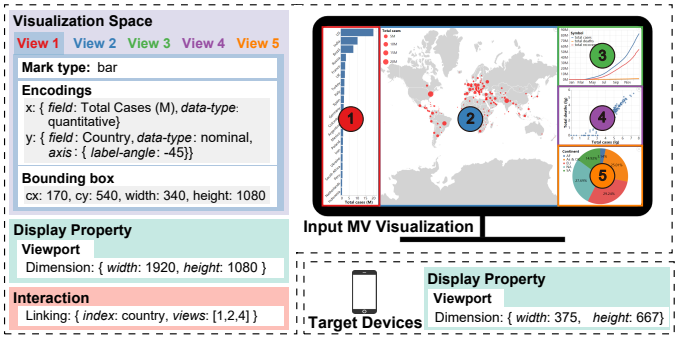


Fig. 1. This work considers the design space of a MV visualization with regards to *visualization space*, *display property*, and *interaction*.

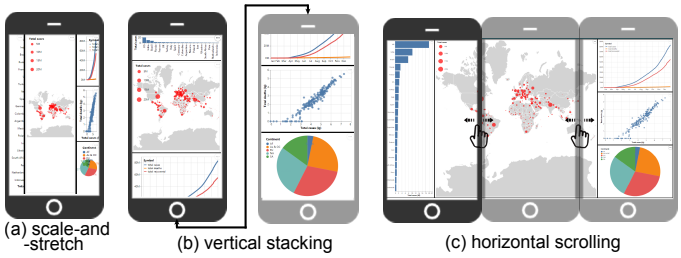


Fig. 2. Examples of common layout adaptations by existing methods. (a) Scale-and-stretch heavily distorts aspect ratios of the views. (b) Stacking views up vertically alters view topology in the input MV visualization. (c) Horizontal scrolling requires substantial user interactions.

chart type. Following Vistribute [27], we consider both properties of the views in the source MV visualization, and those of the target display. In this way, this work provides a more generic solution for dealing with all types of visualizations.

Computational UI Retargeting. Beyond visualization layout, there have been efforts in retargeting general user interfaces (UI) across displays with different sizes. These techniques can be categorized as model-based and data-driven approaches. Model-based approaches typically adopt a two-stage process. First, the approaches propose formal abstractions of user interfaces to describe the interface and its properties, operation logic, and relationships to other parts of the system [18]. For example, Jacobs et al. [28] proposed a set of grid-based document templates that know how to adapt to a range of page sizes and other viewing conditions. Park et al. [44] further considered device capabilities, user roles, preferences, and access rights when adapting graphical UIs for collaborative environments. Next, the approaches adopt certain optimization methods to produce UIs that fulfill the constraints. Agrawala and Stolte [5] used simulated annealing to search for an optimal route map over a space of layouts, while Kieffer et al. [33] utilized greedy heuristics to produce human-like orthogonal network layout. Conversely, data-driven approaches automatically transform desktop-optimized UIs to other devices by learning from existing device-customized UIs as examples of good designs. The effectiveness of data-driven approaches can be seen in past work that successfully adapt webpages [36], graphic designs [43], and accessibility interfaces [21] for different display sizes.

Despite the diversity of application domains, these computational UI retargeting approaches share a common design goal. Specifically, these methods consider both the content of the input UIs and the context of the target displays. We build on this observation in this work, by formulating a series of retargeting heuristics from design patterns of the input MV visualization and output display, and adopting a simulated annealing method to produce optimal MV layouts on the target display.

3 Overview

In this section we first present the research focus of this work (Sect. 3.1). Next, we summarize the requirements and considerations (Sect. 3.2). Finally, we present a formal problem statement and an overview of our approach (Sect. 3.3).

3.1 Research Focus

This work focuses on *layout adaptation* for MV visualizations across displays of varying sizes. A typical scenario is to adapt dashboards that are consisted of multiple visualization views to tablets and mobile devices. Sarikaya et al. [51] observed that many dashboards in use face the challenge of "adapting to multiple platforms including mobile devices." For example, the COVID-19 dashboard [31] by the Johns Hopkins University presents two versions for desktop and mobile devices. The same design requirement of maintaining two dashboard designs is also reported in a variety of applications such as business intelligence [63] and citizen science [20]. Commercial software such as Tableau [4], Spotfire [3], and ArcGIS Dashboards [1] offer user-friendly dashboard adaptation to mobile devices. However, the tools mostly adopt simple layouts such as vertical stacking (*e.g.*, Tableau and Spotfire) or tabs (*e.g.*, ArcGIS Dashboards), when adapting desktop MV visualizations to small displays; see examples in Supplementary Table S3 and the video.

Responsive MV visualization needs to consider *layout*, *visual mapping*, *underlying data*, and *interaction* [8]. Sadana and Stasko [49] found that the number of data items should be limited to less than 1000 for a scatterplot on tablets. Andrews et al. [6], [7] showed examples of responsive visualizations including line charts, bar charts, parallel coordinates, and scatterplots, by adapting their layouts, display densities, and interactions. However, there are always trade-offs. As noted by Kim et al. [34], care must be taken when maintaining the balance between the graphical density of visualizations with intended takeaways for users. For example, when adapting the MV visualization in Figure 1 to small displays, one may consider using sampling to adapt underlying data or using aggregated heatmap to adapt visual mapping. These strategies however distort information for certain data samples, which may mislead users when understanding their data.

Instead, this work considers the design space of a MV visualization with regards to visualization space, display property, and interaction, as illustrated in Figure 1. As a proof of concept, we develop and demonstrate the expressivity of *AdaMV* on adapting desktop MV visualizations to small displays of tablets and mobile phones, which are common in modern sensing making processes beyond the desktop [48].

3.2 Requirements and Considerations

Based on prior work in MV visualization design and our synthesis of prior literature, we summarize the following aspects of requirements for responsive MV visualization design.

R1. Keep all views. All views incorporated in a MV visualization depict certain aspects of information and complement each other to support a comprehensive understanding of the data [14], [47]. As such, all views must be kept when adapting to the target display. Ideally, the views can be packed together

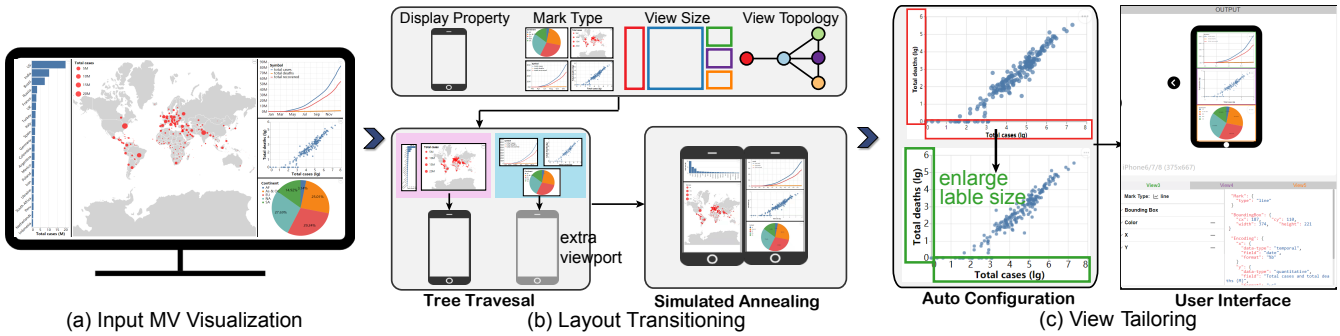


Fig. 3. Overview of our proposed approach for retargeting and tailoring a desktop MV visualization (a). We first design an automatic layout retargeting method that traverses all possible view grouping options based on the geometric and topological properties of the multiples views and computes an optimal layout for the target display using a simulated annealing algorithm (b). We further enable fine-tuning of the visual encodings of a view in its allocated display estate, using a rule-based auto configuration method complemented with an interactive user interface (c).

in one viewport and allocated enough display estates. However, a MV visualization may consist of too many views for a small target viewport. In such scenarios, extra viewports may be added to allow the user to switch between the viewports using page flipping or tabs.

R2. **Avoid overlap**. Overlap can cause severe occlusions that affect local navigation of a view coordinate space, and consequently, affect the data interpretability [19], [57]. Thus, overlapping views are strictly prohibited. Note that some views may include multiple-layer graphics, such as glyphs on top of maps. These graphics as considered as one superimposed view [29], rather than several views occluding each other.

These two requirements are hard constraints that must be met by a responsive MV visualization. However, satisfying these two constraints is often not sufficient. Figure 2 shows examples of common layout adaptations, including scale-and-stretch, vertical stacking, and horizontal scrolling. The layouts fulfill the aforementioned hard constraints, yet user studies show that they are not preferred (see Sect. 5.2). Towards generating effective MV adaptations, we distill the following considerations and include them as additional soft constraints.

C1. **Avoid unnecessary extra viewports**. Extra viewports should only be added if and only if necessary. Multiple viewports form a complex display ecology that coordinates, transfers, and connects information across viewports for visual analysis [16]. For example, users need more efforts to compare COVID-19 cases when views 1, 2 & 4 in Figure 1 are distributed across different viewports, using layouts by horizontal scrolling as shown in Figure 2(c).

C2. **Minimize white space**. White space should be minimized in the target display. White space, or negative space, is a fundamental consideration for readability and aesthetics in visual design [58]. Visualization design shall achieve effective space usage for the full range of functions since white space occupy display estate that hinders visual exploration [59].

C3. **Maintain aspect ratios and relative sizes in target layouts.** Aspect ratios and relative sizes of views in desktop MV visualizations are often determined after thoughtful consideration. Studies have shown that relative size may influence the importance of views that the designer wants to emphasize [43]. Scale-and-stretch can generate layouts that heavily distort aspect ratio of each view and consequently damage view readability (see Figure 2(a)).

C4. **Preserve view topology**. The distribution and layout of views in a visualization interface are not arbitrary [27], but often indicate visual information flow [39] that reflects the underlying semantic structure linking the graphical data elements. As an example, views near each other may be more related or placed together for the purpose of comparison [22], [27]; see the coordinated views 1, 2 & 4 in Figure 1. When adapting to mobile phones, vertically stacking the views alters the view topology in the desktop MV visualization (Figure 1). As an example, in Figure 2(b), users will need to scroll up and down to locate countries in the map (view 2) and identify case-fatality ratios in the scatterplot (view 4). The substantial interactions are not preferred by users, as we will demonstrate in the user study described later in Sect. 5.2.

C5. **Improve view readability.** An individual view may suffer from unreadable labels, cluttered text, distorted layout, etc. on small displays [59]. The issues negatively affect view readability and should be fixed after the display estate is assigned to each view.

We note that these considerations (soft constraints) may conflict with each other. For example, to adapt the MV visualization in Figure 1, one may choose to proportionately rescale the layout to keep relative sizes and aspect ratios (**C3**), but it will leave much white space (against **C2**). Since not all considerations can always be satisfied, our approach is to strictly enforce the two requirements (**R1** and **R2**) as hard constraints, satisfy as many considerations (**C1** to **C5**) as possible, and minimize the number and the severity of the violations and conflicts.

3.3 Method Formalization and Overview

To formalize our problem of MV adaptation, we assume the inputs of: 1) a set of views in a desktop-optimized MV visualization $MV := \{v_1, v_2, \cdots, v_n\}$, where each view is specified by *mark*, *encodings*, and *bounding box* attributes; and 2) a target display of viewport vp specified by the *dimension* attributes. Our goal for this work is *to derive an optimal MV design* $\overline{V} := \{\bar{v}_1, \bar{v}_2, \cdots, \bar{v}_n\}$ *that meets the requirements* ***R1 & R2****, while striving to satisfy the considerations* ***C1*** $-$ ***C5***.

Figure 2 presents some example results by existing methods including scale-and-stretch (Figure 2(a)), vertical stacking (Figure 2(b)), and horizontal scrolling (Figure 2(c)), for adapting the desktop MV visualization shown in Figure 1. The results violate some of the considerations described above and can affect user preference and understandability as found in our user study (Sect. 5.2). To improve upon these existing techniques, we develop a two-stage framework: *layout retargeting* and *view tailoring* to

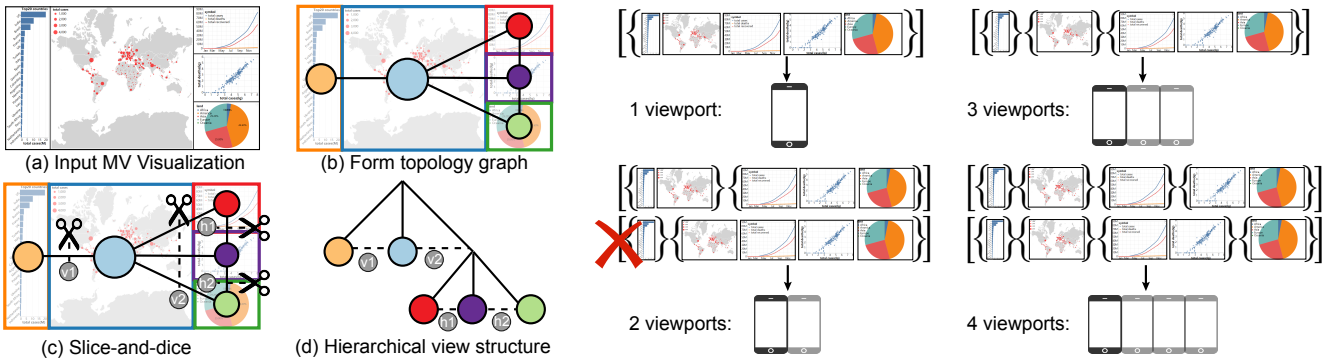


Fig. 4. Left: From the input MV visualization (a), we create a topology graph (b) and slice-and-dice the MV (c), yielding a hierarchical view structure (d). Right: We form possible view grouping options for different numbers of viewports based on view hierarchy. Note that the second grouping option for 2 viewports is omitted because it breaks view hierarchy, and the same applies to other grouping options for 3 and 4 viewports.

semi-automatically retarget and tailor a desktop-optimized MV visualization for small displays (see Figure 3).

- *Layout Retargeting* (Sects. 4.1&4.2). The goal of this process is to derive an optimal layout that fulfills the requirements **R1 & R2** and strive to satisfy considerations **C1**−**C4** on the target display. We first employ a tree traversal method that organizes the views in a hierarchical tree layout based on their geometric and topological properties and traverses the tree to form all possible view grouping options (Sect. 4.1). Next, we cast the layout retargeting requirements as energy terms, and design a simulated annealing (SA) algorithm that achieves the optimal view layout with the least global cost (Sect. 4.2).
- *View Tailoring* (Sect. 4.3). We further incorporate a view tailoring process to optimize the readability of each view (**C5**). Here, we first develop a fine-tuning method that can automatically adapt the visual encodings of each view in the context of its allocated display estate. The algorithm detects issues like out of viewport, unreadable labels, and cluttered text, and adjusts the encodings accordingly by adjusting the visualization specifications. We further develop an interactive interface that enables users to customize the visual appearance of each view, and to provide visual feedback for users to evaluate their adjustments.

4 MV Layout Adaptation

This section presents the two-stage framework for MV layout adaptation. First, we describe the *layout retargeting* approach that includes: a view grouping method that divides views into groups and assigns view groups to viewports (*View Grouping*, see Sect. 4.1), and a simulated annealing algorithm that derives optimal view layout (*Layout Retargeting*, see Sect. 4.2). Second, the retargeting is complemented with a layout and view tailoring approach (*View Tailoring*, see Sect. 4.3) that allows the user to customize the layout and improve readability.

4.1 View Grouping

Depending on the target display, a MV visualization sometimes needs to be split into multiple viewports (*e.g.*, accessed via tabs or with page-flipping). To accommodate for these situations, we first construct a hierarchical view structure based on view properties and layout of an input MV visualization (Figure 4(a)), as follows:

1) *Form topology graph* (Figure 4(b)). We first construct an undirected graph $G = (V, E)$ to represent a view topology. Each node $v \in V$ represents a view in the input MV visualization. We add an edge e_{ij} to connect a pair of nodes v_i & v_j if the views are adjacent in the original layout. Connected nodes reflect the underlying semantic structure that links the views, and they may form potential view groups.
2) *Slice-and-dice* (Figure 4(c)). We apply slice-and-dice operations to separate the views recursively. We employ a specialized binary space partition (BSP) algorithm, in which every cut is either a horizontal or vertical line. Here, the algorithm will first perform vertical cut v_1 that breaks the edge connectting the bar chart and the map, followed by v_2 that further breaks the edges connecting the map and the others. Horizontal cuts h_1 and h_2 can only be performed afterward since the cuts will divide the bar chart and the map into two parts. Views cut by consecutive horizontal cuts or vertical cuts will be put in the same level, whilst views cut in different directions will be put in different levels. The recursive partitioning splits the views until all edges in G have been cut.

In this way, we construct a tree structure (Figure 4(d)) corresponding to the hierarchical view hierarchy. Our algorithm takes into account the special case of 'small multiples', which is described as *a series of graphics, showing the same combination of variables, indexed by changes in another variable* [54]. We consider neighboring views of the same size and view type as small multiples, and they are treated as a whole and will not be split.

Next, we separate views into a list of view groups $[S_1, \cdots, S_l], 1 \leq l \leq n$, where each view group S_i contains a subset of views in V, *i.e.*, $S_i \subseteq V$. We denote a view $v_j \in S_i$ as $v_j^{S_i}$. Here l ranges from one to the number of views in the input MV visualization n, indicating that every view will be put on an individual viewport. Figure 4 (right) presents some examples of possible grouping results based on the constructed view hierarchy. In the case of one viewport (where $l = 1$), all views are in the same group and put in the same viewport. When two viewports are available (where $l = 2$), there is still only one option: bar chart (orange circle) and map (light blue circle) in one group, and the others in another group. Similarly, we can form grouping options for three viewports, and so on. Some grouping options, such as bar chart, map, and line chart (red circle) in one group, and the remaining two views in another group, are not permitted. This is because the option will break the hierarchical structure, and puts views at the same level in different viewports whilst view at different hierarchy levels in one viewport.

4.2 Layout Retargeting

We formulate the layout retargeting requirements and considerations (Sect. 3.2) as a set of cost functions and employ a simulated

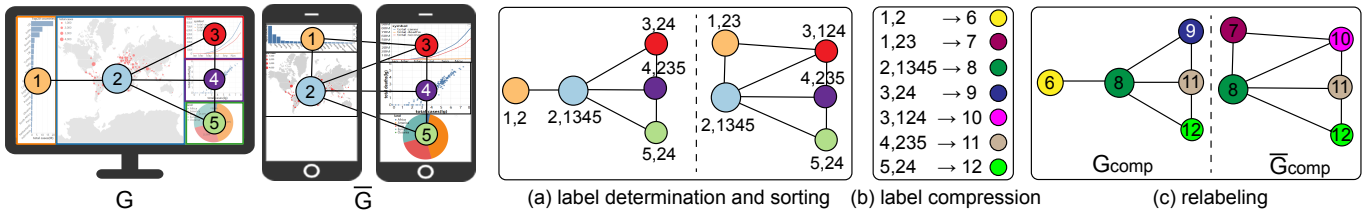


Fig. 5. Computing change between two topology graphs using the Weisfeiler-Lehman test for the input and retargeted MV layouts (left). The algorithm consists of three stages: label determination and sorting (a), label compression (b), and relabeling (c).

annealing technique to derive the optimal layout.

4.2.1 Cost Functions

The cost functions are defined based on: (1) the input MV visualization $V := \{v_i\}_{i=1}^{n}$ and its topology $G = (V, E)$, (2) a list of view groups $[S_1, \cdots, S_l]$, (3) a list of viewports $VP := \{vp_i\}_{i=1}^{l}$, where we assign a view group S_i to a viewport vp_i, and (4) the retargeted MV visualizations $\overline{V} := \{\bar{v}_i\}_{i=1}^{n}$ and its topology graph $\overline{G}$. The following costs capture the retargeting considerations.

Space utility (f_{su}). To minimize the white space (**C2**), we incorporate a metric of overall space utility f_{su} as:

$$f_{su} = \sum_{i=1}^{l}(1 - \frac{\sum_{j=1}^{|S_i|} size(v_j^{S_i})}{size(vp_i)}), \tag{1}$$

where $size(\cdot)$ denotes the size of a view or a viewport. The value of f_{su} ranges in [0, l): close to 0 values indicate the display space is well utilized, whilst large values indicate the display space is wasted. The metric also punishes unnecessary viewports (**C1**), since f_{su} increases as l increases. Notice that f_{su} is computed based on the premise that there is no overlap among views (**R2**), and no view extends beyond its containing viewport.

Relative size (f_{rs}). Relative view size is an important factor for preserving layout consistency (**C3**). We measure relative sizes of a view in the input and retargeted MV visualizations and compute their differences. The overall relative size change is computed as:

$$f_{rs} = \sum_{i=1}^{n} |\frac{size(\bar{v}_i)}{\sum_{i=1}^{n} size(\bar{v}_i)} - \frac{size(v_i)}{\sum_{i=1}^{n} size(v_i)}|. \tag{2}$$

The value of f_{rs} ranges in [0, 2): values close to 0 indicate minimal changes to the relative sizes, whilst values close to 2 indicate that the relative view sizes are distorted heavily. Note here the relative size metric is computed among the views, but not the viewports. Together with the *space utility* metric, the algorithm tends to derive maximized and consistent view sizes.

Aspect ratio (f_{as}). To fulfill consideration **C3**, we strive to also preserve the aspect ratio of a view, defined as $as(v_i) = w(v_i)/h(v_i)$. Here we include two-aspect considerations. First, a readable view should not be overly long and narrow, *i.e.*, $as(v_i)$ shall fall in a certain range, denoted as $[as_{min}, as_{max}]$. Views of different types have distinct ranges of aspect ratio. We find empirical mark-oriented ranges that cover over 95% views from the MV visualization dataset [14] as follows: *bar* chart [1/7, 7], *line* chart [1/5, 5], *area chart* [1/6, 6], and other types [1/2, 2]. We pay a large penalty ψ to prevent the aspect ratio of a view from going beyond the range. Second, certain view types are transposable, *i.e.*, the views can be displayed in either vertical or horizontal orientation, without effects on view readability. For example, a bar chart and a scatterplot can be displayed in either horizontal or vertical orientation.

We define f_{as} by summing up aspect ratio change θ of all views as $f_{as} = \sum_{i=1}^{n} \theta(\bar{v}_i, v_i)$, where

$$\theta(\bar{v}_i, v_i) = \begin{cases} \psi, & \text{for } as(\bar{v}_i) > as_{max} \text{ or } as(\bar{v}_i) < as_{min} \\ |as(v_i) - as(\bar{v}_i)|, & \text{for non-transposable views} \\ min(|as(v_i) - as(\bar{v}_i)|, |as(v_i) - \frac{1}{as(\bar{v}_i)}|). & \text{otherwise} \end{cases} \tag{3}$$

We empirically set ψ to 7, which corresponds to the largest range for a bar chart. As such, the value of f_{as} ranges in [0, $7 \times n$], and small values are preferred to keep the aspect ratios of the views. The introduction of ψ to penalize aspect ratios beyond the range promotes views of proper aspect ratios. The right figure shows a contrary example of the resulting layout without penalizing aspect ratios out of range when adapting the input MV visualization (Figure 1) to a mobile phone. Here, the aspect ratio of the bar chart is close to that in the input. However, the long and narrow bounding box leaves little space for the bars. Instead, adding the penalty ψ will generate a more readable layout (see Figure 10 (C1)).

Fig. 6. A contrary example of resulting layout without penalizing aspect ratios out of range.

Topology (f_{topo}). To reserve view topology (**C4**), we measure the difference between the input G and the retargeted $\overline{G}$ topology graphs using Weisfeiler-Lehman test of isomorphism on graphs [53]. As illustrated in Figure 5, the algorithm first assigns a label to each node by concatenating the node index and its neighboring node indices in ascending order (Figure 5(a)). Next, the algorithm compresses each node label to a new index using a hashing function (Figure 5(b)), and relabels all nodes to the hashing values, yielding compressed graphs G_{comp} and $\overline{G}_{comp}$ (Figure 5(c)). If the node indices in G_{comp} and $\overline{G}_{comp}$ are different, and algorithm halts. Otherwise, the processes (a-c) will be repeated until different node indices in the compressed graphs are identified. We can then represent the topology graphs as one dimensional array, by counting the number of node labels in the original (G & $\overline{G}$) and the compressed (G_{comp} & $\overline{G}_{comp}$) graphs. For instance, the input topology graph G is represented as a vector $\varphi(G) = (1,1,1,1,1,1,0,1,1,0,1,1)$, where the first five ones denotes the occurrence of labels $1-5$ in G, while zeros at the 7^{th} and 10^{th} positions indicate G_{comp} does not contain labels 7 & 10. Similarly, we can represent $\overline{G}$ as $\varphi(\overline{G}) = (1,1,1,1,1,0,1,1,0,1,1,1)$. In this way, the graph isomorphism between G and $\overline{G}$ is computed as $k_{WL}(G,\overline{G}) =< \varphi(G), \varphi(\overline{G}) >= 8$.

We then measure the topology change f_{topo} as:

$$f_{topo} = 1 - \frac{< \varphi(G), \varphi(\overline{G}) >}{size(\varphi(G))}, \tag{4}$$

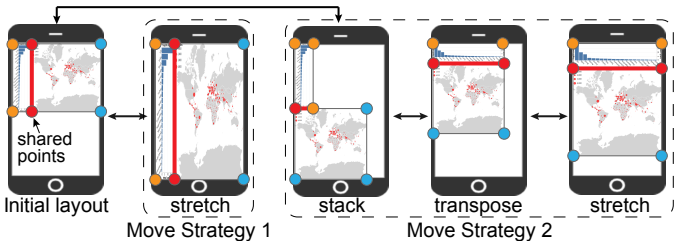


Fig. 7. Illustration for the layout retargeting process. The initial layout scales according to the viewport. Strategy 1 applies vertical *stretch* move that fills up the viewport, whilst Strategy 2 applies *stack*, *transpose*, and *stretch* moves sequentially. Red dots here indicate shared points. Layout by Strategy 2 has a smaller cost than that by Strategy 1.

where $size(\varphi(G))$ indicates the size of $\varphi(G)$. f_{topo} ranges in [0, 1), and small values are preferred to reserve view topology.

Global cost function (F). We normalize all the costs described above, and define the global cost function F as a simple combination of the different cost functions:

$$F(\overline{V}) = \omega_{su} f_{su} + \omega_{rs} f_{rs} + \omega_{as} f_{as} + \omega_{topo} f_{topo}, \quad (5)$$

where the weights ω_{su}, ω_{rs}, ω_{as}, and ω_{topo} are all positive and can be tuned to achieve a suitable tradeoff between the different costs. All weights are set to 1 by default.

4.2.2 Retargeting Algorithm

Our goal is now to produce the optimal layout $\overline{V}$ by minimizing the global cost function F. Here we model each view as a rectangle represented by four points. The points can be positioned anywhere in the viewport, but not inside rectangles of other views to ensure non-overlap. Two neighboring views have two shared points if the rectangles share an entire edge, or one shared point if the rectangles partially share an edge and one side aligns. Operations on the shared points will affect both views. The goal is then to find optimal positions for the points. We can derive the optimal layout by moving the points in the following ways:

- *Stretch*: randomly selects an edge of a view and shifts it to identify optimal width/height in the range of zero to viewport width/height. If one point is shared with another view, the other view will also be stretched in the same direction.
- *Stack/Unstack*: stacks/unstacks the adjacency between two views from horizontal to vertical or vice versa. Note that a *Stack/Unstack* operation can break shared points.
- *Transpose*: switches the width and height of a view, and adjusts the neighboring views accordingly. The move can only be applied to views that can be displayed in both vertical and horizontal orientations, *e.g.*, a bar chart or a scatterplot.

We then employ a SA approach to reach the quasi-optimal solution imitating the annealing process, which has been successfully demonstrated in many applications, such as route map generation [5], interior layout [64], and infographic design [45]. As illustrated in Figure 7, the SA algorithm works as follows:

1) *Set the initial layout*. The method first scales the views in proportion to fit the viewport, yielding an initial layout $\overline{V}_0$.
2) *Propose a candidate layout*. Next, the method proposes a candidate layout $\overline{V}_{s+1}$ by applying one of the above mentioned moves on the current layout $\overline{V}_s$. For example, the method can either stretch an edge to fill up the viewport (Strategy 1), or flip the rectangles that stack up the views vertically (Strategy 2).

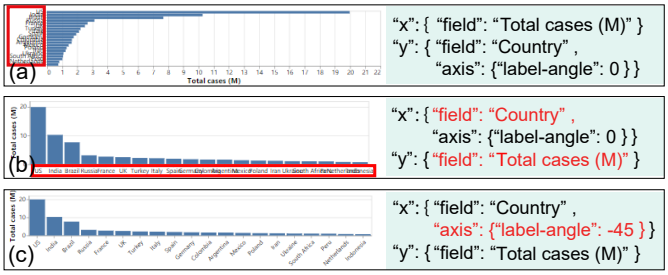


Fig. 8. Rule-based auto configuration of encoding attributes to improve view readability: inherit encodings from the input view (a); switch fields of x and y channels based on *transpose* move in the layout retargeting stage (b); rotate labels by 45° to address text cluttering issue (c).

3) *Accept or reject a candidate layout*. We adopt the metropolis criterion to measure the probability of a candidate layout $\overline{V}_{s+1}$ being accepted, as $\exp(-\frac{F(\overline{V}_{s+1})-F(\overline{V}_s)}{T_s})$ where T_s is the current temperature. For example, the candidate layout by Strategy 1 increases spatial utilization, but heavily distorts aspect ratios. The overall cost $F(\overline{V}_1)$ is higher than that of the initial layout $F(\overline{V}_0)$, thus the layout by Strategy 1 has a small probability of being accepted. In comparison, the first move by Strategy 2 has a high probability of being accepted.
4) *Update layout*. The method updates the current layout upon acceptance of the candidate, *i.e.*, $\overline{V}_s \leftarrow \overline{V}_{s+1}$, and repeats steps 2-3 until no better layout is produced. As in Figure 7, the stack move in Strategy 2 is accepted, and the method continues to generate lower-cost layouts via transpose and stretch operations.

This process is applied to each of the view grouping options from $l = 1$ to $l = n$ (see Sect. 4.1). The lowest-cost option from all view grouping options is selected as the final layout.

4.3 View Tailoring

The layout retargeting stage computes the optimal layout based on geometric and topological properties of the input MV visualization. However, default encoding configurations of a view may cause issues like unreadable labels and cluttered text. To address these issues and improve readability (**C5**), we develop a rule-based auto configuration method and an interactive user interface to further tailor and customize the layout and encodings of each view.

Rule-Based Configuration. The configuration follows these general rules: 1) *No changes to the data and mark type.* Though some pilot studies have been conducted (*e.g.*, [7], [24], [25], [34], [35]), there is still a lack of a general solution for adapting data density and visual marks to fit a display's DPI. 2) *Avoid information loss.* The adapted views on the target display shall present the same information as the original views. 3) *Avoid overlapping and cluttered texts.* The purpose of view tailoring is to improve readability (**C5**), while overlapping and cluttered texts are the main reason for poor readability [59].

Based on these rules, we allow the configuration to transpose axes for certain view types, and adapt *orientations* and *font sizes* of title, ticks and legend only. We adjust visual appearances of each individual view in its allocated display estate, as follows:

- The layout retargeting method identifies some views that can be transposed to achieve better aspect ratios. Here, the configuration algorithm first performs the transposition. For example, in Figure 8(a), the fields are 'Total cases (M)' and 'Country' for x and y respectively, corresponding to those in the input MV visualization. Figure 8(b) shows the result after the view has

been transposed. Note that the labels are more readable in this transposed view.

- Second, the algorithm adjusts font sizes in linear proportion according to the display estates allocated in the input and the retargeted viewports. To keep the multiple views consistent, we use the same font size for labels across different views. The consistency rule also applies to title and tick sizes. To avoid unreadable labels, we specify a minimum font size that is empirically set to 12 points.
- We further adjust the display of axis labels to address the most common issue of cluttered text as in Figure 8(b). In the case of overlapping labels, we consider two possible operations: 1) we apply a line break to overlong labels composed of two or more words, and 2) we try a label rotation of 45° if overlapping still exists, and increase the rotation to 90° if necessary. Margins and graphic sizes are automatically adjusted based on the changes made to axis labels. Take Figure 8(c) for example, labels of the x-axis are rotated in 45°, resulting in non-overlapping text.
- Last, we scale sizes of legend marks and labels in linear proportion according to the display estates. We also overlay legends on top of the graphics to save more space. The legend position is set to the top-left corner by default, which can be adjusted using the user interface as shown in Figure 9.

The process stops when no overlap is identified. Alternatively, the method will select the configuration with the least overlap if all configurations present overlaps.

Interactive User Interface. The rule-based auto configuration method may fail to produce a suitable design depending on the user's needs. For example, the legend placed at the top-left corner may be undesirable because it occludes other graphical elements. To overcome the deficiency of the automated method, we develop an interactive user interface that allows a user to further fine-tune the encodings of each view. As shown in Figure 9, the interface consists of three components:

- **Control Panel**: The panel allows a user to specify the property of the target display, and change weights for layout retargeting costs. In the device control panel (Figure 9(a1)), users can specify the viewport dimension by selecting one from a list of options including iPhone 7/8 (375×667), iPhone X/XS/11/11Pro (375×812), iPhone XR/XS Max/11Pro Max (414×896), iPad Mini/Air (768×1024), and Samsung Galaxy Tab (800×1280), or specifying the width and height. The display orientation can also be set to *landscape* or *portrait*.
  The layout weighting panel (Figure 9(a2)) allows users to control the weights of *space utility*, *relative size*, *aspect ratio*, and *topology* used in the SA algorithm (see Equation 5). All weights are normalized to the range of [0, 1]: 1 indicates that the cost is fully taken into consideration, while 0 indicates that the cost is ignored. A special note is that the interface provides individualized control for each of the views for controlling the weights for relative size and aspect ratio. By controlling the weights, the interface allows users to explore different layout results, including reproducing results from existing methods. For example, if a user specifies the weight for space utility to 0, the output is the same as that by uniform scaling.
- **Input Panel**: The input panel presents a visual representation of the input MV visualization, and design information of each individual view. In the representation panel (Figure 9(b1)), a MV visualization optimized is uniformly scaled to fit within the panel. It is relevant to note that the MV visualization is not a static image, but a fully functioning visualization – interactions like linking and coordination still function in the representation panel. Bounding boxes of the views are encoded as rectangles and highlighted in different colors. Detailed design information of a view, in terms of *mark type*, *bounding box*, and *encodings*, are arranged in tabs in the attribute panel (Figure 9(b2)). Users can examine different views by clicking on the view in the representation panel or selecting the corresponding view tab. Users can choose to remove or modify the title, set tick count, or change label sizes and orientations of the view, and select a position for *legend*. Changes to the specification are shown in the representation panel for direct visual feedback.
- **Output Panel**: The output panel provides the same functionality as the input panel, except it shows the target display environment. The user can manually adjust the resulting layout by selecting a view and dragging its bounding box within the viewport. Adjacent views of the adjusted view are automatically updated as well. If the output includes two or more viewports, they are arranged in a pagination fashion, and the attribute panel will only show current views in the display; see Figure 9(c1 & c2) for example. Alternatively, if a display is duplicated to provide extra viewports, a navigation button is overlaid over the display icon for switching viewports.

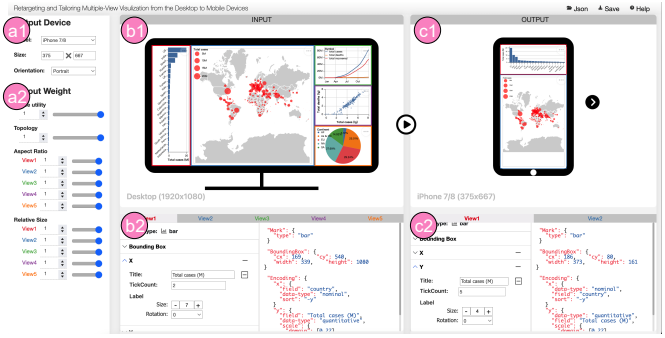


Fig. 9. The interactive user interface consists of a control panel (a) for specifying target display and layout weights, an input panel (b) for displaying the input MV visualization, and an output panel (c) for displaying the adapted result.

5 Evaluation and User Feedback

We demonstrate the effectiveness of the proposed layout retargeting algorithm in four ways: 1) We demonstrate the feasibility and expressivity of our approach in supporting a series of desktop MV visualizations adapted to various displays (Sect. 5.1); 2) We conduct a quantitative user study that compares results by our approach with those generated by existing methods (Sect. 5.2); and 3) We present a runtime analysis of the layout retargeting method (Sect. 5.3). 4) Last, we conduct a qualitative user study and present the participants' feedback on the tailoring interface (Sect. 5.4).

5.1 Feasibility and Expressivity

Figure 10 presents four real-world examples of top baby names in the US (A), the global burden of disease (B), daily number of COVID-19 cases and deaths by country (C), and ratings of TV shows and Movies (D). The target mobile devices include a smartphone (A1−D1) and a tablet (A2−D2). All results are generated by the layout retargeting algorithm and refined using the rule-based configuration, without user tailoring using the interface.

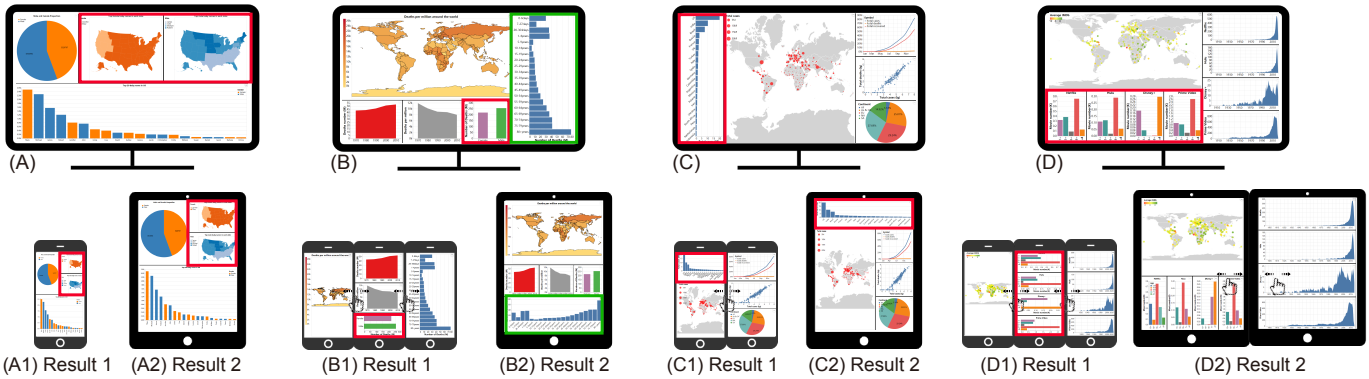


Fig. 10. Examples of MV visualizations adapted for a smartphone (A1−D1) and a tablet (A2−D2). The input MV visualizations A-D consist of diverse mark types. Extra viewports are added in B1, C1, D1, and D2.

- In Figure 10(A), the input contains only 4 views, which can be fitted into one single viewport on both the smartphone (A1) and the tablet (A2). The results slightly change the view topology to ensure that the aspect ratio of the two map views (highlighted with red boxes in the figures) remain consistent between the original and the target visualizations.
- In Figure 10(B), our method alters view topology for both the smartphone and tablet displays. On the smartphone (B1), two extra viewports are added and arranged from left to right side-by-side. Specifically, the three views below the map are put in one viewport vertically, and the bar chart framed by the red box changes from the vertical to the horizontal orientation to fit the viewport. On the tablet (B2), the bar chart marked in the green box is placed at the bottom, and its orientation is transposed to fit the viewport.
- In Figure 10(C), our method identifies that the smartphone (C1) does not have enough display space and adds an extra viewport, while the tablet (C2) can pack all the views into a single viewport. In both cases, the bar chart marked in the red box is stacked on top of the map and the fields for x- and y-axis are swapped.
- Last, Figure 10(D) consists of a map and 2 sets of small multiples. Additional viewports are added for both the smartphone (D1) and the tablet (D2), and both sets of small multiples are always kept in the same viewport. Specifically on the smartphone, orientations of the bar charts in the red box are adjusted to fit the aspect ratio of the display.

More examples can be found in Supplementary Table S4.

5.2 User Study on Retargeting Algorithm

We conducted a user study to compare the visualizations generated using our approach (denoted as *AdaMV*) with three existing techniques: scale-and-stretch (*SS*, see Figure 2(a)), vertical stacking (*VS*, see Figure 2(b)), and horizontal scrolling (*HS*, see Figure 2(c)). For fairness, view encodings including font size produced by all methods were automatically refined using the rule-based configuration described in Sect. 4.3.

Participants. We recruited 15 participants between the ages of 23-25 from a data visualization course at a university. The participants were graduate students majoring in diverse disciplines including computer science, electronic engineering, and biology. All participants had basic knowledge about design principles of MV visualization and experience using MV visualizations. Further, all participants reported that they had at least one mobile phone that they use daily.

Fig. 11. Results on iPhone 12 by existing techniques used in the user study: *SS* for column 1, *VS* for column 2, and *HS* for column 3.

TABLE 1
The study asks participants to rate consistency of geometry (Q1−Q2) and topology (Q3), and feasibility of completing analytic tasks (Q4).

Q1	The aspect ratios between views on the source and target displays are consistent
Q2	The relative sizes between views on the source and target displays are consistent
Q3	The topology between views on the source and target displays is consistent
Q4	Completing the analytic tasks on the target display is feasible

Apparatus and Implementation. We selected three exemplar cases shown in Figure 10 (A, C & D). The MV visualizations were adapted to and tested on an iPhone 12 of display 390×844 and an iPad (7th gen.) of display 810×1080. Both devices were equipped with touch interactions as input modality. The results on iPhone 12 by *SS*, *VS*, and *HS* are presented in Figure 11 columns 1–3, respectively. All visualizations running on iPhone 12 and iPad can be found in Supplementary Table S2.

Procedure. The studies were performed in the order of introduction, training, experiment, and questionnaire. First, we gave a 5-min introduction about the motivation of our work and described the concepts of white space, aspect ratios, relative sizes, and view topology. Then, we asked the participants to do one training round using an exemplar 3-view visualization with the same questionnaire used in the formal study (see below). We started the experiment after the participants confirmed that they understood the tasks and the concept of MV visualization retargetting.

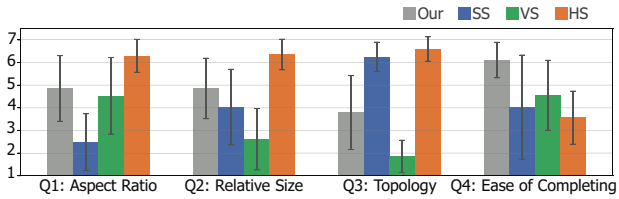


Fig. 12. User ratings of the layouts by *AdaMV*, *SS*, *VS* and *HS* on Q1−Q4.

For the main task, the participants were first presented a MV visualization on a desktop, and asked to freely explore the visualization until they felt familiar with it. Next, the participants were asked to explore the results generated by *AdaMV*, *SS*, *VS* and *HS* for each display smartphone and tablet. The orders of the visualizations shown to the participants were randomized, and the participants were not aware of which result is by our method. After exploring each of the visualizations, the participants were asked to complete a questionnaire that included five questions on the quality of results, as listed in Table 1. The participants were asked to rate their degree of agreement with each of the questions in a scale of 1-7 where 1 represents completely disagree and 7 represents completely agree. Here, Q1−Q2 reflects the consistency of the geometrical properties of the layouts. Q3 concerns topological differences between the source and target layouts. Q4 is about the feasibility of completing analytic tasks that were different for each tested MV visualization. For example, the exploration tasks for Figure 10(C) were *"Which country, Argentina or South Africa, has more COVID-19 cases?"* and *"Identify the positions of Argentina and South Africa on the map."* In the end, the participants were asked to give an overall rating for each layout.

In total, each participant completed 24 trials in each experiment (4 methods × 3 MV visualizations × 2 displays). On average, each participant spent 1.5–2 hours for the study, and received a compensation of CNY ¥100 (∼$15.5) for their participation.

Result Analysis. We collected a total of 2160 answers (24 trials × 6 questions × 15 participants). Figure 12 presents the average ratings of *AdaMV*, *SS*, *VS* and *HS* for questions Q1 − Q4. Below we summarize our findings.

- *Consistency of geometric properties (Q1 & Q2)*. The participants rated *HS* to best preserve consistency of aspect ratio (Q1) and relative size (Q2) between the source and target layouts, with mean=6.30, SD=0.73 for Q1 and mean=6.36, SD=0.68 for Q2. *AdaMV* receives the second highest scores, with mean=4.86, SD=1.45 for Q1 and mean=4.86, SD=1.34 for Q2.
- *Consistency of view topology (Q3)*. Participants rated *HS* (mean = 6.60, SD = 0.54) and *Scale-and-Stretch (SS)* (mean = 6.26, SD = 0.66) as the top two performers in preserving consistency of view topology. However, some participants reported that '*the layouts by SS are unreadable*' and '*those by HS requires scrolling to see all views*'. Interestingly, for the tested visualizations neither methods altered the view topology. The fact that these visualizations still received ratings below 7 possibly reflected the participants' inability to read the visualizations effectively. *AdaMV* method received an average rating, with mean = 3.78, SD = 1.63, but still significantly ($p < 0.01$) better than *Vertical Stacking (VS)* (mean=1.86, SD=0.71).
- *Feasibility of completing analytic tasks (Q4)*. *AdaMV* was rated highest in supporting the participants to complete their analytic tasks, with mean=5.93, SD=1.16. The result is significantly better than *SS* ($t_{(14)} = 8.58$, $p < 0.01$), *VS* ($t_{(14)} = 9.24$, $p < 0.01$), and *SS* ($t_{(14)} = 16.46$, $p < 0.01$). Specifically, participants reported that layouts by *SS* were hardly usable because of aspect ratio distortion, and those by *VS* and *HS* required the users to continuously scroll the screen to find the answers.

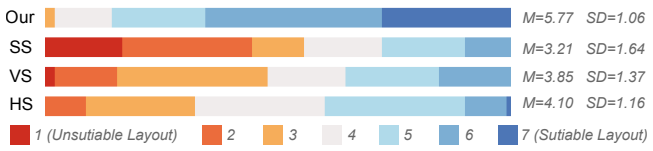


Fig. 13. Overall preference of layouts generated by *AdaMV*, *SS*, *VS* and *HS*. The rightmost column denotes Mean and SD.

Figure 13 further presents the preference of layouts by *AdaMV* and the other baseline methods. The stacked bar charts show detailed rating proportions, and the right side presents means and standard deviations. Overall, *AdaMV* is the most preferred method, followed by *HS* and *VS*, and *SS*.

Feedback. We collected feedback from the participants with open-ended questions regarding the reasons for their ratings and solicited their suggestions for future improvements. All participants agreed that the layouts by *SS* were unsuitable because aspect ratios of the views were heavily distorted, even though the method fully preserved view topology. In contrast, *VS* could better preserve the aspect ratio, but altered view topology. The participants were familiar with vertically stacked layouts, since most webpages and applications allow users to scroll up/down the screen to view content. However, in terms of MV visualization, the layout can separate neighboring views in the desktop MV visualization and thus required frequent screen scrolling for analytic tasks that required coordination between views. For example, the four bar charts in Figure 10(D) were separated into two viewports by *VS*. When answering the question *'Compare the number of people under the categories of Netflix and Prime Video'* (the third case as shown in Figure 10(D)), the participants needed to scroll the screen frequently in order to find the answer.

The *HS* method managed to preserve aspect ratios, relative sizes, and view topology, which was the main reason for the relatively high ratings for overall preference. However, the participants complained that the layouts generated by *HS 'required too much horizontal scrolling'*, and *'could not guarantee all content of a single view in one viewport'*, especially on mobile phones. Some participants also noted that the scrolling interaction can be confused with view navigation interactions such as panning in a map view.

The participants agreed that *AdaMV* achieved a good balance between the preservation of geometric and topological consistency, and best supported completing the analytic task. The participants preferred the way AdamMV packed views in one viewport and arranged multiple viewports in a pagination mechanism, rather than requiring continuous scrolling such as the case with *VS* and *HS*. The participants reported that the compact layouts generated by *AdaMV* allowed them to focus on exploring the content of the views. For future work, the participants suggested that some content details could be removed, and to use tabs instead of page flipping to switch between extra viewports.

5.3 Computation Efficiency

We conducted runtime experiments to measure the computation efficiency of our approach, to analyze what factors affect the computation efficiency, and identify possible solutions for future improvements. The experiments were run on a desktop PC with

2.4GHz 8-Core Intel Xeon E5-4640 CPU. The tree traversal and simulated annealing methods are implemented in Python, while the view tailoring interface is implemented using HTML5 and JavaScript. Communication between the frontend interface and backend server is implemented using jQuery Ajax.

TABLE 2
Running time (in seconds) of our method on different conditions.

Num. Views	View Grouping		Layout Retargeting		Total	
	SA	GA	SA	GA	SA	GA
3	0.01	0.01	8.44	7.68	8.45	7.69
5	0.01	0.01	27.56	24.77	27.57	27.58
7	0.01	0.01	34.97	30.64	34.98	30.65
9	0.01	0.02	60.02	52.58	60.03	52.60

Table 2 presents the running times (in seconds) of the view grouping, and layout retargeting methods using simulated annealing (SA) and genetic algorithm (GA) for a different number of views. Three running examples are used, and each example has been tested for two trails, for each number of views. Overall, the total runtime of our method increases as the number of views increases, and the main cost is the layout retargeting stage. The results are within our expectation since the layout retargeting problem is related to bin-packing – a problem that is known to be NP-Complete. The layout retargeting is currently realized using SA, which can be feasibly changed to other methods like GA. Both methods share the same running parameters, including the termination condition. The experimental results show that SA and GA methods have comparable running times, while the resulting layouts are the same.

Specific to our approach, the number of view grouping options is a factor that affects the efficiency of layout retargeting. We have implemented a number of heuristics to reduce the overall computational complexity. For example, we always keep 'small multiples' in one view group, and ignore infeasible grouping options such as putting every view in an individual viewport. To further improve the retargeting, one possible solution is to pre-compute the number of extra viewports needed, by measuring display resolutions of the desktop and target mobile devices. Another is to utilize multi-core CPUs and parallelize the computation. We leave other retargeting strategies for future work.

5.4 User Feedback on Tailoring Interface

We interviewed five graduate students specializing in data visualization (denoted as $E1-E5$) regarding the tailoring interface. All participants had experience in developing MV visualization systems for visual analytics in practice. We started with a system demonstration using a 3-view example and explained the four design criteria we considered: *space utility, relative size, aspect ratio*, and *topology*. We also described the SA algorithm and the reason behind the stochasticity of the results. After the demonstration, the participants were allowed to freely explore the interface with 3-view, 5-view, and 9-view examples, and freely adjust the weights of design criteria. The participants were encouraged to share their thoughts and ask questions at any time during the exploration, which were recorded for subsequent analysis. Their feedback is summarized below.

Auto-generated results are inspiring. In general, the participants expressed positive feedback on the system. The participants noted that auto-generated layouts acted as recommendations and inspired them to achieve the final layout. *E5* commented "*The adapted layouts inspire me a lot. I can make improvements based on the recommended layouts, especially for complex multi-view designs.*"

Adjusting weights cause unpredictable results. The participants agreed that layouts generated by the SA algorithm with default weights were generally reasonable, while those generated with different weights were less satisfying. Sometimes the participants adjusted the weights and expected a different result from the previous one, but the same result was generated. "*There were times when I looked at a large number of parameters and felt overwhelmed*," E3 noted. Most of the participants suggested keeping the weight adjustment controllers only for debugging.

Manually fine-tuning views is necessary. The participants appreciated the functionalities of fine-tuning view configurations, including dragging to adjust view sizes and changing font sizes and orientations. Currently, the interface only allows resizing views but not their positions. *E2* suggested adding the flipping method to the interface. *E4* suggested adding functionality that fixes the most important view, such as the map in Figure 10(D2), when adjusting view position and size. "*The other views are all some abstraction of the data presented in the map, so I would expect to always see the most important view (the map view)*", he mentioned. The suggestion is similar to adjusting *boundary* layout considered in [8], and we plan to realize it in future work.

6 Discussion, Limitations, and Future Work

We contribute a semi-automated approach to layout adaptation for responsive MV visualization design. The work is motivated by the increasing use of mobile devices with small displays to render interactive MV visualizations for visual communication, data exploration and analysis. The layout retargeting method is fully automated. It is built on design rules crafted from prior work and our own experience, such as to consider properties of MV visualizations and those of the target displays [27]. We also adopt heuristics from existing visualization designs, including the mark-oriented aspect ratio range learned from the MV visualization dataset by Chen et al. [14]. Once the automated algorithm has identified a "good" MV layout, a user can manually fine-tune and adjust the visualization to achieve the desired outcome.

Semi-Automated Approach. The reason for the hybrid automated and manual approach is the design considerations are considered "soft constraints" by the simulated annealing algorithm and can therefore conflict with each other. For instance, satisfying the two goals of maintaining aspect ratio (C3) and preserving view topology (C4) may not always be possible (see Figure 10 (B1 & B2) for example). As such, providing a user an interface to manually adjust the layout based on their preference and fine-tune the choices of encoding for each view is necessary. Further, following the key principle in responsive visualization design [25], this interface should provide immediate visual feedback on user input in a WYSISWG fashion (see Figure 9).

Evaluation Metrics. Our automated layout retargeting algorithm optimizes on a number of metrics, including space utility, aspect ratio, relative size, and view topology. In our user study, we find that out *AdaMV* approach is not optimal on any single evaluation metric, yet achieves the best readability, ease for completing analytic tasks, and overall user preference. This finding suggests that the metrics work in a unified manner and should not be considered independently but holistically.

Generalization. We observe that our technique for generating MV visualization layouts is flexible and generalizable. Manipulating the weights to the cost function (see Equation 5) can produce a range of layouts. Simple changes to the weights can result in replicating the three existing layout algorithms used in our user study. Sometimes the resulting layout can become unstable as the search process can become stuck in local optima. Below we report of the effect of removing each of the metrics from the cost function.

- *Space utility*. Omitting space utility from the cost function (by setting its weight to zero in Equation 5) will generate layouts as the same with those by uniform scaling (see Figure 2(a)). These layouts fully maintain aspect ratio, relative size, and view topology but result in much wasted space and poor readability.
- *Aspect ratio*. Omitting aspect ratio can lead to the same layouts generated by *SS* which maximizes space utility, relative size, and view topology while sacrificing consistency in aspect ratio. However, in some cases the resulting layout becomes unstable because the stretch may generate heavily distorted aspect ratios beyond the valid ranges for different view types.
- *View topology* & *relative size*. Lastly, omitting the view topology and relative size terms will most likely result in unstable layouts. At times the result would be the same as that by *VS* that breaks view topology and changes relative size, and arrange the views from top to bottom.

View topology also plays an important role in view group assignment and layout adjustment. If no topology is considered, the views can be randomly organized together, causing an increasing number of grouping options. As discussed in Sect. 5.3, more view groups will dramatically reduce the computational efficiency. Moreover, the steps for moving bounding boxes of the views (see Figure 7) are not arbitrary but dependent on view topology. In this sense, considering view topology actually reduces the search space.

Limitations. There are several limitations in our work. First, the automated layout adaptation method focuses on the layout of MV visualizations by analyzing their geometric and topological properties while neglecting user interaction that is regarded as a key consideration for visualization design on small displays [17], [30]. In fact, the layout by *horizontal scrolling* is the best option for persevering the layout as it proportionately rescales the MV visualization, which is demonstrated by users ratings on the ability to preserve aspect ratios (Q1), relative sizes (Q2), and view topology (Q3). However, the option is not suitable for visual analytics (Q4), as users need to continuously scroll the display to explore the views. The situation becomes worse when similar interactions are enabled for individual views, for example, panning operation on the map view in the COVID-19 dashboard. There is a necessity for a new conceptual framework that incorporates interactions, similar to that in VisTiles [37], for responsive MV visualization design.

Second, the view tailoring approach allows a user to fine-tune formatting configurations such as label and title sizes to improve view readability. Some recent studies have suggested that data and mark type also need adjustments for effective visualization. For example, Kim et al. [35] proposed an automatic approach to approximate the loss of support for task-oriented visualization insights, based on which one can optimize a series of encoding channels for various chart types. The aim is to preserve a user's ability to arrive at certain insights, by balancing the graphical density that is affected by the display's DPI, data density, and visual marks [34]. The current framework treats view tailoring as a subsequent task after layout retargeting. A superior solution is to optimize both the layout and view configurations (including data density and visual marks) simultaneously. Nevertheless, it is challenging to determine what are the possible adaptions for single views when retargeting view layout. We leave it as a future work to integrate the method with our approach.

Future Work. Our work opens several directions for future research. First, the work currently considers only six popular mark types, while neglecting many other charts such as table, text, and trees & networks. Design considerations for these charts can be very different. For example, trees & networks have many layout options including linear and radial layouts and can be robust to rotation and orientation (*e.g.*, using a force-directed layout for graphs). Second, we have designed several computational metrics for assessing the similarity between MV visualization layouts. Though the metrics are used for layout adaptation in this work, they can be feasibly converted to cost functions and employed to guide the learning process for automatic layout design. Given the recent advances on this topic (*e.g.*, [14], [55], [60], [61]), we are excited to explore how other learning techniques, such as deep-learning, can be used to automate layout design for MV visualizations.

7 Conclusion

We present a new approach to adapt the layout of a desktop MV visualization design for small displays. This two-staged approach consists of an automatic layout retargeting algorithm and a semi-automatic view tailoring method. The layout retargeting algorithm requires minimum guidance from the developer, as we cast the requirements and considerations as energy terms that can be solved using a tree traversal method and a simulated annealing algorithm. Once a layout is found, the user can adjust layout weights and fine-tune visual encodings of each view using our interactive visual editing tool. A gallery of MV visualizations adapted for diverse mobile devices demonstrates the feasibility and expressivity of our approach. User feedback collected from a qualitative user study confirms the effectiveness of our approach over existing methods.

Acknowledgment

The authors wish to thank the anonymous reviewers for their valuable comments. This work was supported by the National Natural Science Foundation of China (No. 62172398) and the National Science Foundation (OAC-1940175, OAC-1939945, OAC-2118201, IIS-1452977).

References

[1] ArcGIS Dashboards. https://www.esri.com/en-us/arcgis/products/arcgis-dashboards, last accessed on 31/01/2022.
[2] Power BI. https://powerbi.microsoft.com, last accessed on 02/09/2021.
[3] Spotfire. https://www.tibco.com/products/tibco-spotfire/, last accessed on 02/09/2021.
[4] Tableau. https://www.tableau.com/, last accessed on 02/09/2021.
[5] M. Agrawala and C. Stolte. Rendering effective route maps: Improving usability through generalization. In *Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques*, pages 241–249, 2001.
[6] K. Andrews. Responsive Visualisation. In *Proceedings of MobileVis Workshop at CHI*, 2018.
[7] K. Andrews and A. Smrdel. Responsive Data Visualisation. In *Proceedings of EuroVis (Posters)*. The Eurographics Association, 2017.

[8] S. K. Badam and N. Elmqvist. Effects of screen-responsive visualization on data comprehension. *Information Visualization*, 20(4):229–244, 2021.

[9] M. Q. W. Baldonado, A. Woodruff, and A. Kuchinsky. Guidelines for using multiple views in information visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 110–119, 2000.

[10] T. Blascheck, L. Besanon, A. Bezerianos, B. Lee, and P. Isenberg. Glanceable visualization: Studies of data comparison performance on smartwatches. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):630–640, 2019.

[11] M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe. Visualizing ranges over time on mobile phones: A task-based crowdsourced evaluation. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):619–629, 2019.

[12] M. Brehmer, B. Lee, P. Isenberg, and E. K. Choe. A comparative evaluation of animation and small multiples for trend visualization on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):364–374, 2020.

[13] R. Chen, X. Shu, J. Chen, D. Weng, J. Tang, S. Fu, and Y. Wu. Nebula: A coordinating grammar of graphics. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2021.

[14] 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.

[15] Y. Chen. Visualizing Large Time-series Data on Very Small Screens. In B. Kozlikova, T. Schreck, and T. Wischgoll, editors, *Proceedings of the IEEE VGTC/Eurographics Conference on Visualization (Short Papers)*. The Eurographics Association, 2017.

[16] H. Chung, C. North, S. Joshi, and C. Jian. Four considerations for supporting visual analysis in display ecologies. In *Proceedings of IEEE Conference on Visual Analytics Science and Technology*, pages 33–40, 2015.

[17] S. Drucker, D. Fisher, R. Sadana, J. Herron, and M. C. Schraefel. TouchViz: A case study comparing two interfaces for data analytics on tablets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2301–2310, 2013.

[18] J. Eisenstein, J. Vanderdonckt, and A. Puerta. Applying model-based techniques to the development of UIs for mobile computers. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 69–76, 2001.

[19] N. Elmqvist and P. Tsigas. A taxonomy of 3d occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14(5):1095–1109, 2008.

[20] D. Filonik, R. Medland, M. Foth, and M. Rittenbruch. A customisable dashboard display for environmental performance visualisations. pages 51–62, 2013.

[21] K. Gajos and D. S. Weld. SUPPLE: automatically generating user interfaces. In *Proceedings of the International Conference on Intelligent User Interfaces*, pages 93–100, 2004.

[22] M. Gleicher. Considerations for visualizing comparison. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):413–423, 2018.

[23] J. Heer, N. Kong, and M. Agrawala. Sizing the horizon: The effects of chart size and layering on the graphical perception of time series visualizations. In *Proceedings of SIGCHI Conference on Human Factors in Computing Systems*, pages 1303–1312, Boston, MA, 2009.

[24] B. Hinderman. *Building Responsive Data Visualization for the Web*. Wiley, 2015.

[25] J. Hoffswell, W. Li, and Z. Liu. Techniques for flexible responsive visualization design. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.

[26] T. Horak, S. K. Badam, N. Elmqvist, and R. Dachselt. When David meets Goliath: Combining smartwatches with a large vertical display for visual data exploration. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2018.

[27] T. Horak, A. Mathisen, C. N. Klokmose, R. Dachselt, and N. Elmqvist. Vistribute: Distributing interactive visualizations in dynamic multi-device setups. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2019.

[28] C. Jacobs, W. Li, E. Schrier, D. Bargeron, and D. Salesin. Adaptive grid-based document layout. *ACM Transactions on Graphics*, 22(3):838–847, 2003.

[29] W. Javed and N. Elmqvist. Exploring the design space of composite visualization. In *Proceedings of the IEEE Pacific Visualization Symposium*, pages 1–8, 2012.

[30] J. Jo, S. L'Yi, B. Lee, and J. Seo. TouchPivot: Blending wimp & post-wimp interfaces for data exploration on tablet devices. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 2660–2671, 2017.

[31] Johns Hopkins Coronavirus Resource Center. COVID-19 dashboard. https://coronavirus.jhu.edu/map, last accessed on 02/09/2021.

[32] M. Kay, T. Kola, J. R. Hullman, and S. A. Munson. When (ish) is my bus? user-centered visualizations of uncertainty in everyday, mobile predictive systems. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 5092–5103, 2016.

[33] S. Kieffer, T. Dwyer, K. Marriott, and M. Wybrow. HOLA: Human-like orthogonal network layout. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):349 – 358, 2016.

[34] 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.

[35] H. Kim, R. Rossi, A. Sarma, D. Moritz, and J. Hullman. An automated approach to reasoning about task-oriented insights in responsive visualization. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2021.

[36] C. E. Kulkarni and S. R. Klemmer. Automatically adapting web pages to heterogeneous devices. In *Proceedings of Extended Abstracts of the CHI Conference on Human Factors in Computing System*, pages 1573–1578, 2011.

[37] R. Langner, T. Horak, and R. Dachselt. VisTiles: Coordinating and combining co-located mobile devices for visual data exploration. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):626–636, 2018.

[38] R. Langner, U. Kister, and R. Dachselt. Multiple coordinated views at large displays for multiple users: Empirical findings on user behavior, movements, and distances. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):608–618, 2019.

[39] M. Lu, C. Wang, J. Lanir, N. Zhao, H. Pfister, D. Cohen-Or, and H. Huang. Exploring visual information flows in infographics. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2020.

[40] R. Ma, H. Mei, H. Guan, W. Huang, F. Zhang, C. Xin, W. Dai, X. Wen, and W. Chen. LADV: Deep learning assisted authoring of dashboard visualizations from images and sketches. *IEEE Transactions on Visualization and Computer Graphics*, 27(9):3717 – 3732, 2021.

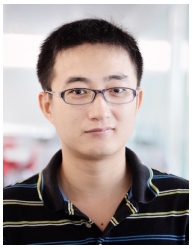
[41] K. Matkovic, W. Freiler, D. Gracanin, and H. Hauser. ComVis: A coordinated multiple views system for prototyping new visualization technology. In *Proceedings of the International Conference Information Visualisation*, pages 215–220, 2008.

[42] C. North and B. Shneiderman. Snap-together visualization: a user interface for coordinating visualizations via relational schemata. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 128–135, 2000.

[43] P. ODonovan, A. Agarwala, and A. Hertzmann. Learning layouts for single-page graphic designs. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1200 – 1213, 2014.

[44] S. Park, C. Gebhardt, R. Rdle, A. M. Feit, H. Vrzakova, N. R. Dayama, H.-S. Yeo, C. N. Klokmose, A. Quigley, A. Oulasvirta, and O. Hilliges. Adam: Adapting multi-user interfaces for collaborative environments in

real-time. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2018.
[45] C. Qian, S. Sun, W. Cui, J. G. Lou, H. Zhang, and D. Zhang. Retrieve-Then-Adapt: Example-based automatic generation for proportion-related infographics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):443–452, 2021.
[46] 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.
[47] J. C. Roberts. State of the art: Coordinated & multiple views in exploratory visualization. In *Proceedings of the International Conference on Coordinated and Multiple Views in Exploratory Visualization*, pages 61–71, 2007.
[48] J. C. Roberts, P. D. Ritsos, S. K. Badam, D. Brodbeck, J. Kennedy, and N. Elmqvist. Visualization beyond the desktop–the next big thing. *IEEE Computer Graphics and Applications*, 34(6):26–34, 2014.
[49] R. Sadana and J. Stasko. Designing and implementing an interactive scatterplot visualization for a tablet computer. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 265–272, 2014.
[50] R. Sadana and J. Stasko. Designing multiple coordinated visualizations for tablets. *Computer Graphics Forum*, 35(3):261–270, 2016.
[51] 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, 2019.
[52] L. Shao, Z. Chu, X. Chen, Y. Lin, and W. Zeng. Modeling layout design for multiple-view visualization via bayesian inference. *Journal of Visualization*, pages 1–16, 2021.
[53] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(77):2539–2561, 2011.
[54] E. R. Tufte. *The Visual Display of Quantitative Information*. 2001.
[55] S. Wang, W. Zeng, X. Chen, Y. Ye, Y. Qiao, and C. W. Fu. ActFloor-GAN: Activity-guided adversarial networks for human-centric floorplan design. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2021.
[56] C. Weaver. Building highly-coordinated visualizations in improvise. In *Proceedings of the IEEE Symposium on Information Visualization*, pages 159–166, 2004.
[57] C. Weaver. Cross-filtered views for multidimensional visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 16(2):192–204, 2010.
[58] B. Wong. Negative space. *Nature Methods*, 8(1):5–5, 2011.
[59] A. Wu, W. Tong, T. Dwyer, B. Lee, P. Isenberg, and H. Qu. MobileVisFixer: Tailoring web visualizations for mobile phones leveraging an explainable reinforcement learning framework. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):464–474, 2021.
[60] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu. AI4VIS: Survey on artificial intelligence approaches for data visualization. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2021.
[61] A. Wu, L. Xie, B. Lee, Y. Wang, W. Cui, and H. Qu. Learning to automate chart layout configurations using crowdsourced paired comparison. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 14:1–13, 2021.
[62] Y. Wu, X. Liu, S. Liu, and K. Ma. ViSizer: A visualization resizing framework. *IEEE Transactions on Visualization and Computer Graphics*, 19(2):278–290, 2013.
[63] O. M. Yigitbasioglu and O. Velcu. A review of dashboards in performance management: Implications for design and research. *International Journal of Accounting Information Systems*, 13(1):41–59, 2012.
[64] L.-F. Yu, S.-K. Yeung, C.-K. Tang, D. Terzopoulos, T. F. Chan, and S. J. Osher. Make it home: Automatic optimization of furniture arrangement. *ACM Transactions on Graphics*, 30(4):1–12, 2011.



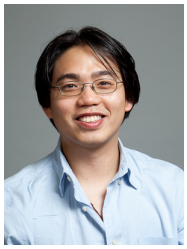
Wei Zeng is an assistant professor in the Hong Kong University of Science and Technology (Guangzhou). He received the PhD degree in computer science from Nanyang Technological University, in 2015. He received ICIV'15 and VINCI'19 Best Paper Award, and ChinaVis'21 Best Paper Honorable Mention Award. He serves as Program Chair for VINCI'23, program committee for venues including IEEE VIS, EuroVis STARs, and ChinaVis. His recent research interests include visualization and visual analytics, computer graphics, AR/VR, and HCI.

Xi Chen got the master degree from Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, and also with University of Chinese Academy of Sciences. Her research interests include data visualization and human computer interaction.

Yihan Hou is currently a PhD student at the Hong Kong University of Science and Technology (Guangzhou). She received her bachelor degree in information computer and science from Xi'an Jiaotong-liverpool University. Her research interests include visual analytics and information visualization.

Lingdan Shao is currently a master student at Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, and also with University of Chinese Academy of Sciences. Her research interests include data visualization and human computer interaction.

Zhe Chu is currently a master student at Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, and also with University of Chinese Academy of Sciences. He received his bachelor degree in computer science and technology from Northwestern Polytechnical University. His research interests include data visualization and human computer interaction.



Remco Chang is currently is an associate professor in Computer Science at Tufts University. He received his PhD in Computer Science from the University of North Carolina Charlotte. His research interests include visual analytics, information visualization, human computer interaction, and databases.