MobileVisFixer: Tailoring Web Visualizations for Mobile Phones Leveraging an Explainable Reinforcement Learning Framework

Aoyu Wu, Wai Tong, Tim Dwyer, Bongshin Lee, Petra Isenberg, and Huamin Qu

Fig. 1. MobileVisFixer automatically transforms SVG-based visualizations from mobile or desktop versions into mobile-friendly designs with several modifications: (A) resize the view and modify axes; and (B) resize the view, reposition the legend, modify the title and axes.

Abstract—We contribute MobileVisFixer, a new method to make visualizations more mobile-friendly. Although mobile devices have become the primary means of accessing information on the web, many existing visualizations are not optimized for small screens and can lead to a frustrating user experience. Currently, practitioners and researchers have to engage in a tedious and time-consuming process to ensure that their designs scale to screens of different sizes, and existing toolkits and libraries provide little support in diagnosing and repairing issues. To address this challenge, MobileVisFixer automates a mobile-friendly visualization re-design process with a novel reinforcement learning framework. To inform the design of MobileVisFixer, we first collected and analyzed SVG-based visualizations on the web, and identified five common mobile-friendly issues. MobileVisFixer addresses four of these issues on single-view Cartesian visualizations with linear or discrete scales by a Markov Decision Process model that is both generalizable across various visualizations and fully explainable. MobileVisFixer deconstructs charts into declarative formats, and uses a greedy heuristic based on Policy Gradient methods to find solutions to this difficult, multi-criteria optimization problem in reasonable time. In addition, MobileVisFixer can be easily extended with the incorporation of optimization algorithms for data visualizations. Quantitative evaluation on two real-world datasets demonstrates the effectiveness and generalizability of our method.

Index Terms—Mobile visualization; Responsive visualization; Machine learning for visualizations; Reinforcement learning.

◆

1 INTRODUCTION

The last decade has seen an explosive growth of smartphone usage: statistics show that mobile devices have been used more than traditional desktops for web access globally since 2016 [20,22]. It is, therefore, becoming increasingly important to develop mobile-friendly websites that are readable and usable on mobile devices. We see efforts to promote mobile-friendly websites from both industry and research communities. For instance, Microsoft [45] and Google [23] have developed tools to test mobile friendliness, and favor mobile-friendly websites for their search results, which contribute to a trend towards mobile-first design. In addition, there are commercial services [47] and research efforts [3, 43] to fix problems with mobile designs that could cause a frustrating experience. These efforts are typically focused on the

design and layout of websites for mobile devices and do not address the specific challenges of mobile visualization design. As a result, a considerable number of visualizations on the web suffer from readability and usability issues, such as visual clutter, overlapping or tiny text, and overflowing content [8, 78].

Despite the increasing acknowledgment of the opportunities and importance of mobile data visualizations [13,39,40,60], little work has attempted to investigate and fix the problems with mobile web-based visualizations. From a theoretical aspect, we lack empirical studies to understand the types of problems that occur in mobile visualizations. Recent work [43] has identified common types of mobile-related problems for general websites, but they are not readily applicable to the visualization context. From an applied perspective, existing approaches are limited in helping practitioners detect and repair problems with visualizations on mobile devices. For example, the mobile friendliness test tools mentioned earlier [23,45] do not properly handle SVG-based visualizations. Practitioners who carefully craft custom designs, therefore, have to manually test and verify their visualizations on different screen sizes, which is tedious and time-consuming. After detecting problems, practitioners often find it difficult to repair them [7,9]. Practitioners typically need to adjust multiple SVG elements and CSS style properties simultaneously, while ensuring that those adjustments do not introduce any new problems [43]. Automated tools that help tailor visualizations for mobile devices are one way to address these challenges.

- *A. Wu, W. Tong, and H. Qu are with Hong Kong University of Science and Technology. Email: {awuac, wtong, huamin}@ust.hk.*
- *T. Dwyer is with Monash University. E-mail: Tim.Dwyer@monash.edu.*
- *B. Lee is with Microsoft Research. E-mail: bongshin@microsoft.com.*
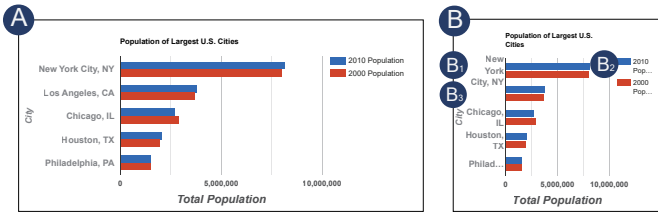- *P. Isenberg is with Inria. Email: petra.isenberg@inria.fr.*

Fig. 2. A motivating example of rule-based techniques for responsive visualizations [34]: (A) The desktop version; (B) To fit visualizations on mobile devices, Google Charts employs rules including word-wrapping if not breaking words (B_1), using ellipses else-wise (B_2), and removing labels if overlapping with others (B_3).

Existing automated solutions [21,34,68] usually adjust visualizations by rule-based methods, *e.g.*, word-wrapping if text overflows. Their decision rules are interpretable and almost operable in their built-in visualization options. However, such rules are often deterministic that could result into sub-optimal results in real-world scenarios. Fig. 2 illustrates a real-world example by Google Charts [34] where it adjusts texts according to a set of rules. Those adjustments lead to readability issues as some text components (*i.e.,* Fig. 2 B_2, B_3) become invisible and unreadable. Although it is possible to add new rules to handle the situation, such rule-based methods face challenges such as large manual efforts and the combinatorial explosion of possible conditions [61]. It remains challenging and time-consuming to design rules for automatic responsive visualizations that scale well to the real-world diversity.

In this paper, we present MobileVisFixer, an interpretable reinforcement-learning-based approach that automatically learns and applies decision rules for generating mobile-friendly visualizations. We focus on SVG-based, single-view Cartesian visualizations with linear or discrete scales, which are found common on the web [4]. To motivate the design of MobileVisFixer, we collected 374 web visualizations and categorized problems we saw when displaying these visualizations on mobile devices into five types of common issues. To optimize visualizations, MobileVisFixer first deconstructs charts into a declarative format which not only captures the underlying data encoding but also efficiently reduces the difficult, multi-criteria optimization problem. MobileVisFixer then utilizes a novel greedy heuristic based on Policy Gradient methods that solves the reduced optimization problem. Quantitative evaluation on a real-world dataset shows that MobileVisFixer successfully solve 89% of visualizations with mobile-friendly problems in reasonable time: Figs. 1 and 3 show several mobile-friendly visualizations that were automatically generated with MobileVisFixer. Further evaluation of MobileVisFixer on a different dataset demonstrates the generalizability of the learned model.

In summary, the primary contributions of this paper are:
- A categorization of five common issues with mobile visualizations derived from 374 web-based visualizations.
- The design and implementation of MobileVisFixer, which automatically converts SVG-based visualizations into mobile-friendly designs. MobileVisFixer takes an explainable machine learning approach to optimize the resolution of four problems across a large set of mobile visualizations.
- A set of quantitative evaluations that demonstrate the effectiveness, generalizability, and explainability of MobileVisFixer.

2 RELATED WORK

This paper draws upon prior work at the intersection of mobile web and mobile visualization, machine understanding of visualization, as well as automated visualization design.

2.1 Mobile and Mobile Web Visualization

There is a growing body of work examining how to adopt desktop web content to mobiles. Typical approaches include *Responsive Web Design* [49] that dynamically responds to size changes of the browser window using fluid grids and CSS media queries, as well as *Adaptive Web Design* [26] that detects screen size and selects an appropriate

design from multiple alternatives. However, both approaches introduce considerable development and testing costs, as developers must verify web-page appearance through trial-and-error. As such, much research in the software community has studied how to automatically detect [3, 74, 75] and repair [43, 44] mobile-friendly issues. In contrast to our work, none of those approaches is targeted at visualizations. In particular, existing techniques do not consider layout constraints in visualizations, and would potentially break the visual encoding and data binding. Moreover, they do not support SVG which is the basis of many web-based visualization. SVG is a difficult target because it has its own set of elements, attributes, and properties that make it more complex than HTML alone.

In order to approach fixing problems with mobile visualizations we considered past design guidelines from the Visualization community. Already 14 years ago, Chittaro [12] argued that the different characteristics of mobile visualizations present new research challenges. Since then, research has proposed and evaluated mobile encodings for a wide range of data-types such as temporal data [5, 11, 37], spatial-temporal data [38], or small multiples [6]. In contrast to this work on the development and study of dedicated mobile encodings is research that looked at how to adapt larger visualizations to smaller screen. Hoffswell *et al.* [31] recently conducted a survey of existing practices for responsive visualization design and subsequently developed a tool to help people manually edit visualizations for different screen sizes. On the commercial side, software like Power BI [21] and Tableau [68] also offered support for responsive layout but they do not proactively detect and diagnose potential problems. Our work adds to this stream of research by proposing novel approaches that automatically detect and repair issues in mobile visualizations.

2.2 Machine Understanding of Visualization

To detect and repair problems present in mobile visualizations, our work takes inspiration from past work on the automatic extraction and manipulation of information in visualizations. In a broader sense, recent research has been devoted to enabling machines to understand data visualizations from different perspectives. A majority of work investigates ways to retrieve data from charts [2, 14, 15, 35, 53, 64], while other research attempts to retrieve color mappings [54] and visual importance [10]. Furthermore, researchers have developed and evaluated deep neural networks that reason about data visualizations including performing graphical perception tasks [27] and visual question answering tasks [36].

The work more closely related to ours is Battle *el al.*'s [4] Beagle system, which analyzes general SVG-based visualizations and automatically classifies them by type. The approach is similar to ours in that it also targets the general SVG-based visualizations we focus. We, however, take a different focus on adjusting SVG attributes related to layouts and visual styles to alleviate mobile-friendly issues while preserving the visual encoding. As such, related to ours are Harper and Agrawala's methods for re-styling visualizations [28] and for generating reusable templates [29] as well as following work [32] on how to infer the visual style and structure from visualization collections. However, they do not specifically address the relationships of visual styles among elements in visualizations, *e.g.*, the layout relationship between text labels and corresponding marks. Our work contributes to this space by studying how to model and deconstruct such relationships from SVG-based visualizations.

2.3 Automated Visualization Design

Generating precise and elegant data visualizations is considered difficult even for experts [56]. Several automated visualization design tools have been proposed to ease this process by **rule-based** or **model-based** systems. Rule-based systems typically introduce a set of heuristic rules to recommend visual encodings [42, 77] or generate layouts [57]. They have proven effective since their rules span a rich range of carefully curated design constraints considering data types and encoding channels. Nevertheless, it requires system designers to apply prior domain knowledge from empirical studies to manually construct rules and curate a rule set [61]. Therefore, recent research starts shifting to model-based
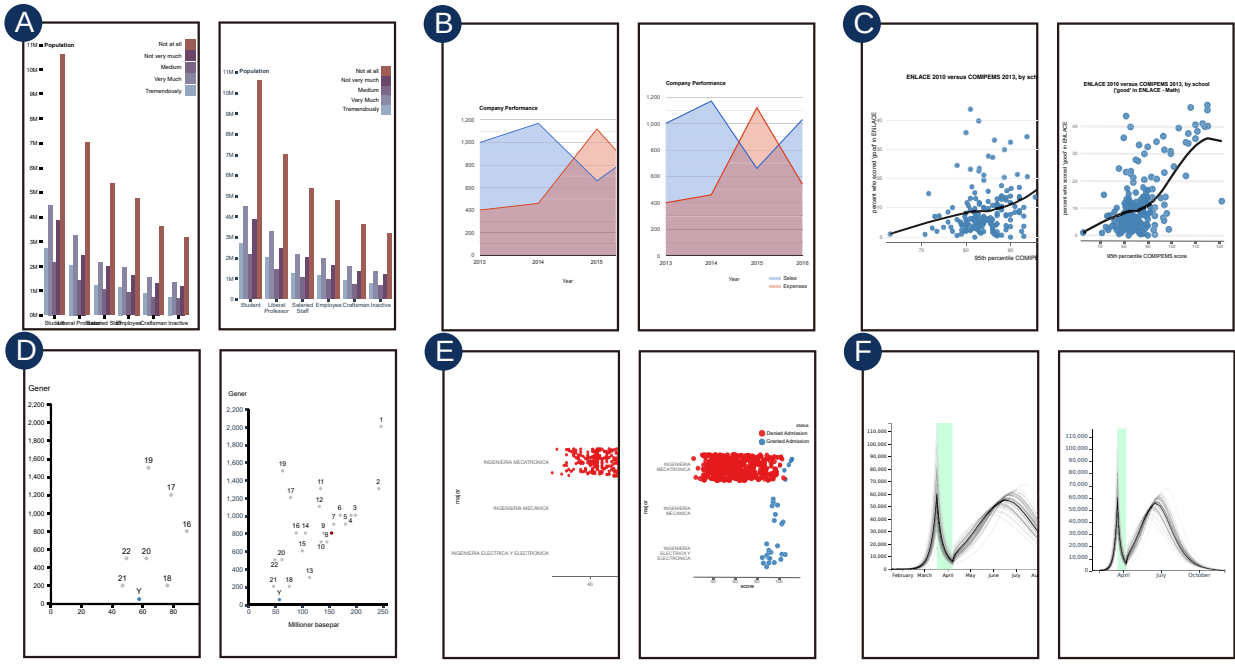
Fig. 3. MobileVisFixer generates more mobile-friendly visualizations of various chart type while preserving original information. Visualization origins: (A) [19]; (B) [24]; (C) [70]; (D) [65]; (E) [69]; and (F) [59]. Modifications: (A) modify axes; (B) resize the view, reposition charts and legends; (C) resize the view and modify the title; (D) resize the view, reposition charts and labels, modify axes; and (E)(F) resize the view and modify axes.

systems such as Data2Vis [18] and VizML [33], which recommend design choices that are learned from a large corpus through machine-learning models. However, despite promising results, their models have not proven superior. In addition, the comprehensive data collection and labelling process could be expensive.

Different from the above work that recommends visual design given data, we study how to automatically adapt the layouts of existing visualizations to mobile screens. Our method is inspired by the recent success of **hybrid** systems that augments rules with machine learning models [48, 61]. In particular, we proposes a novel explainable reinforcement learning framework that automatically learns and executes human-interpretable rules for adjusting layout parameters to improve mobile-friendliness. Compared with existing rule-based responsive visualization techniques (e.g. [21, 68]) that are usually deterministic, our framework could learn stochastic decision rules (policies) that allows generating more flexible solutions to varying real-world scenarios. Besides, our framework embraces algorithmic explainability and transparency which helps model developers debug mistakes in cost functions, and reason about the quality of the learned model.

3 MOBILE-FRIENDLY ISSUES IN WEB VISUALIZATIONS

To gain an insight into mobile-friendly issues in web visualizations, we collected and analyzed SVG-based visualizations on the web. Our focus on web-based visualization is motivated by the fact that visualizations are often consumed on mobile devices, custom-designed, and thus difficult to adjust for all viewing scenarios. We note that we focus on layout-related readability issues of SVG-based visualizations and do not consider interaction problems.

We developed a web crawler to collect SVG-based visualizations following Hoque and Agrawala's approach [32]. As their results are mainly from the *bl.ock.org* domain, we extended their seeding pages with other visualization portals, such as Google Charts. In addition, we randomly visited the hyperlinks in the queue to increase the diversity. We used the Device Mode by Chrome DevTools to crawl visualizations rendered on an iPhone X screen. We also crawled the desktop version to help us reason about if the creators had attempted to adapt visualizations to mobile screens or simply scaled them down. At the end, we obtained 374 visualization examples from 103 domains.

Two authors of this paper manually inspected all mobile visualiza-

tions and coded problems that hurt the appearance and readability of the mobile visualizations. The coding schemes were originally based on existing literature about mobile-friendly problems in general web content [23, 45], such as small font size and wrong viewport. Throughout the coding, we iteratively updated the coding schemes and re-coded samples when necessary. In the following text, we describe the most common sources of problems we found in detail.

3.1 Mobile-friendly issues

We identify five common issues that impair the mobile-friendliness of visualizations (Fig. 4). We discuss them together with the contributing inappropriate changes between desktop and mobile versions.

1) Out of the viewport. Out of 374 visualizations, 122 (32.6%) had problems related to content being placed outside of the screen. This problem can occur when absolute or miscalculated values of SVG properties lead to display coordinates outside of the current viewpoint. This problem forces viewers to scroll horizontally to view the whole content, resulting in a poor user experience [41].

2) Unreadable font size. A large number (118, 31.5%) of visualizations included font sizes that were hardly readable. This problem occurs when programmers only resize visualizations to fit the current screen, making visualizations fully visible but making content less legible.

3) Cluttered text. About 16.0% of visualizations (60) contained overlapping and cluttered text elements. This problem is partly due to the absence of intrinsic mechanisms for preventing overlap in SVG elements or no implemented ways to avoid label overlap.

4) Distorted layout. For 85 (22.7%) web-based visualizations the layout was artificially stretched. This problem occurs because mobile devices are predominantly held in portrait orientation even when browsing multimedia content [55], while desktop web browsing is more typically in landscape mode. To address this difference, web programmers often adjust web content to the screen's width, letting content spread out vertically. However, such practices, when applied in the SVG context, can render visualizations in a distorted aspect ratio that potentially causes unintended bias in visual perception [67].

5) Unwanted white space. The fifth most common (21, 5.6%) cause of problems in the SVG-based visualizations we coded was excess white space, leading to non-optimal space usage and potentially unreadable
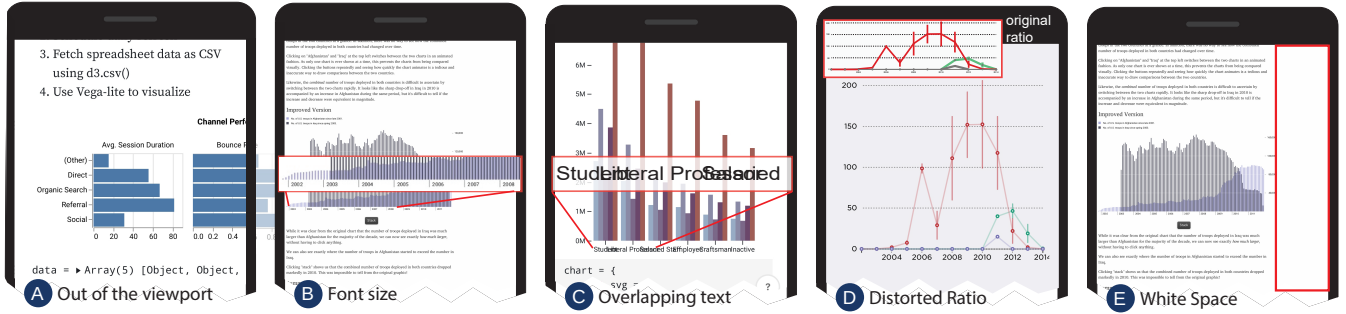
Fig. 4. We identified five common mobile-friendly issues in SVG-based visualizations on mobile devices: (A) A chart is out of the viewport that requires users to scroll horizontally, which can easily get overlooked; (B) The font size is too small and unreadable; (C) The text labels in the axis suffer from visual clutter; (D) The height/width ratio differs from that of the desktop version, which could lead to perceptual bias; and (E) The visualization has much empty space on its left, which could be used more effectively.

content. This problem is often due to the use of fixed-width layout attributes such as padding and margins.

We found that 142 (37.9%) visualizations exhibited no changes between desktop and mobile designs, while only 98 (26.2%) visualizations exhibited none of the above five issues. This indicates that many visualization creators might neglect responsive design. Besides, we observed entanglement among issues. About 36.6% (101) out of the 276 non-mobile-friendly visualizations contained more than one issue.

Beyond these five common issues, we also found several rare cases pertained to mobile-friendliness. A few visualizations embed third-party icons or images that do not automatically scale well to the mobile screen. Besides, touch elements (*e.g.*, buttons) can be too close to each other that users might have difficulties tapping desired ones.

3.2 Discussion

Our analysis shows several visualization-specific problems for mobile content compared to those of general web content [43]. While content sizing and viewport configuration are commonplace for both visualizations and general web content, research by Mahajan *et al.* [43] does not discuss our last three issues (*i. e.*, cluttered text, distorted layout, and unwanted white space). This underscores the potential for developing an automatic, visualization-specific approach.

It is challenging to address all five issues we uncovered simultaneously, since they are highly inter-dependent. For example, increasing the font size to make text legible might lead to overlapping text making it illegible. Therefore, designers must fix related problems in a trial-and-error process, often leaving some problems not optimally solved. Furthermore, designers might fail to anticipate data changes that could distort the layout [73]. For example, Fig. 4 (C) [1] forks a template [2] of a grouped bar chart. However, its labels are considerably longer than that of the template and the mobile version is compromised.

4 MOBILEVISFIXER - OVERVIEW

MobileVisFixer automatically generates mobile-friendly designs for SVG-based visualizations. It currently addresses four common types of problems introduced in Sect. 3: *content sizing for viewport, font sizing, text overlap,* and *white space*. We leave the last one, *distorted ratio*, to future work because it requires perceptual guidelines for a large number of visualizations for which they are not yet clearly defined. Specifically, due to the different screen size between desktops and mobiles, resizing is the most common compromise solution for responsive visualization design [31] that usually distorts the aspect ratio. It, however, remains unclear to what extend that such distortions influence perceptions.

There are some straightforward repairs for the four issues in graphics design [51], *e.g.*, moving text or graphical marks elsewhere to prevent overlap. However, naive movements of elements can easily violate data representations where positions are mapped to data. Therefore,

---

[1]https://observablehq.com/@2shabby/grouped-bar-chart
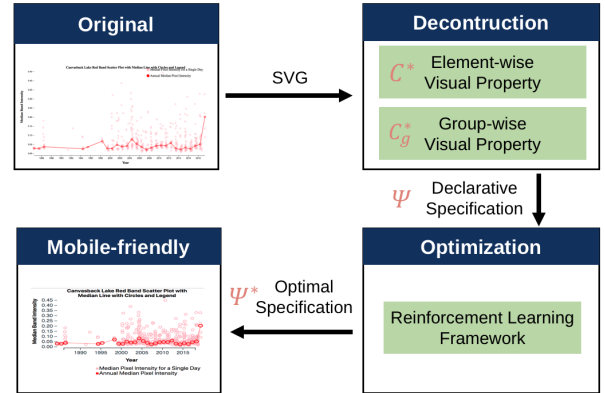[2]https://observablehq.com/@d3/grouped-bar-chart

Fig. 5. Given only an SVG as input, MobileVisFixer generates mobile-friendly designs in two steps: deconstruction and optimization.

the challenge of generating a reasonable repair involves two objectives–addressing multi-criteria mobile-friendly issues and strictly maintaining the underlying visual encoding.

MobileVisFixer transforms visual encoding to a mathematical form. It defines a visualization as a set of visual *elements* ($e \in \mathcal{E}$), including text and graphical marks. Each element is described by a visual *encoding* which specifies *values* ($v_{e,p} \in \mathbb{R}$) for visual *properties* ($p \in \mathcal{P}$), such as positions and sizes in SVG attributes and CSS styles. We use a simplified notation to consider all values in the real domain, assuming that categorical attributes can be expressed by enumerations. Let $\mathcal{C}$ denote all valid element-property pairs, where $\mathcal{C} \subseteq \mathcal{E} \times \mathcal{P}$. The visualization is thereby expressed as a vector containing values for all those pairs, namely $\chi \in \mathbb{R}^{|\mathcal{C}|}$. MobileVisFixer quantifies the mobile-friendly issues with a multi-criteria cost function, denoted (J). The objective is to determine a set of *patches*, denoted χ^*, that minimizes J:

$$\chi^* := \underset{\chi \in \mathbb{R}^{|\mathcal{C}|}}{\arg\min} J(\chi) \qquad (1)$$

Our approach for solving the above multi-criteria, high-dimensional optimization problem consists of two phases, deconstruction and optimization, as shown in Fig. 5. The input to MobileVisFixer is an SVG file containing a visualization to be rendered on mobile devices. The deconstruction phase (Sect. 5) decodes the visualization to extract the data and encoding, which are described in a declarative format. The output is ψ, the parameters used in the declarative descriptions of visualizations, which are reduced from χ in the convenience of solving effectively. The optimization phase (Sect. 6) proposes a novel explainable reinforcement learning framework that solves (Eq. 1) and generates optimal χ^*, as well as the corresponding visualization.

The term "optimal" here refers to a reasonable solution that minimizes built-in costs and thus improves mobile-friendliness for a particular visualization. It does not mean "optimal" for any specific set of requirements (*e.g.,* data, context) of a human designer. A designer,

instead could view different results from our tool and select from or refine multiple generated designs until they find the "best" designs for their own tasks and additional considerations [48].

5 DECONSTRUCTION

The deconstruction phase decodes the SVG to identify $\mathcal{C}^*$ – sets of SVG elements with visual properties that are subject to adjustments to improve the mobile-friendliness, as well as ψ – a declarative description of visualizations that facilitate the computation. The general intuition of this decision is two-fold: (1) most encoding properties (*e.g.*, color, border) should remain faithful to the original chart; and (2) each mobile-friendly issue typically maps to a small set of properties. This reduces the solution space of χ in Eq. 1 from $\mathbb{R}^{|\mathcal{C}|}$ to $\mathbb{R}^{|\mathcal{C}^*|}$. However, this phase needs to consider constraints for the change of properties that are related to the underlying data binding and visual consistency.

To consider those constraints, MobileVisFixer introduces visual *groups* ($g \in \mathcal{G}$) – sets of elements described by the same set of encodings, *e.g.*, the tick-labels in an axis form a visual group. Following the same notation used for element-wise encoding ($\mathcal{C}$), group-wise encodings can be expressed as $\mathcal{C}_g^* \subseteq \mathcal{C}_g \subseteq \mathcal{G} \times \mathcal{P}$, and corresponding values $\chi_g \in \mathbb{R}^{|\mathcal{C}_g^*|}$. Thus, the solution space is further reduced to $\mathbb{R}^{|\mathcal{C}_g^*|}$. MobileVisFixer determines $\mathcal{C}_g^*$ by deconstructing the visual encoding through two steps: (1) generating visual groups and their intra-group encoding and (2) determining *layout dependencies* – the inter-group layout relationships.

The first step builds on Hoque and Agrawala's [32] method for recovering data, marks, and encoding from a D3 chart. Due to the complexity and diversity of data visualizations, MobileVisFixer currently focuses on single Cartesian visualizations with linear or discrete scales. MobileVisFixer extends their method to support non-D3 charts by dismissing D3's specification for the SVG tree structure. For instance, D3[3] utilizes a <g> template for rendering axes. In addition to searching and traversing such <g> nodes, MobileVisFixer also uses a linear scan algorithm to search axis candidates with aligned tick-labels and ticks. The result of this step is a set of visual groups, as well as coordinate scales. Fig. 6 (B) shows each visual group annotated by the same background color, except for the line in the x-Axis and the bars which already has their own background color.

The second step aims to identify the layout dependency of the visual group. Similar to Vega's specification [62], the layout of a group depends on either the coordinate scale or another *anchoring group*. The latter case is referred to as *reactive geometry* which is particularly common for describing the layout of text labels. MobileVisFixer describes reactive geometry by a tuple $\langle p, p_a, o \rangle$, where p and p_a are the *anchoring position* for the group and anchoring group respectively, and o is the offset value in corresponding direction. Possible anchoring positions include *Left, X-Center, Right, Top, Y-Center,* and *Bottom.* For instance, the labels in Fig. 6 (A) are horizontally aligned center to their corresponding bars, which is described as ⟨X-Center, X-Center, 0⟩.

The resulting specification of visualizations is similar with Vega [63], a declarative format based on grammar-based specifications. We choose a subset of Vega specifications related to layouts. As shown in Table 1, MobileVisFixer includes five classes (Title, Axis, Legend, Mark, and Label) and 10 visual groups, which are basic structural elements of visualizations. Different from Vega where labels are included in marks, we consider labels as a separate class since they have unique encoding properties such as font-sizes. For each group, MobileVisFixer identifies a set of encoding properties which are subject to adjustments to improve mobile-friendliness. For instance, adjustable properties for `TitleText` include <fontSize>, <orient>, and layout properties such as <x> and <y>. All those group-property pairs form $\mathcal{C}_g^*$.

MobileVisFixer classifies adjustable properties into independent and dependent attributes. Independent attributes are expressed as `variables`, while dependent attributes are `function-like`. MobileVisFixer currently only considers the dependency relationships of layout-related properties (*e.g.*, x, y coordinates), including Global Scale (GS), Local Scale (LS), and Reactive Geometry (RG). For example,

---

³https://github.com/d3/d3-axis

Class	Group	Adjustable Encoding Properties
Title	TitleText	RG(AxisLine), <fontSize><orient>
Axis	AxisTick	GS() , <number>
	AxisLabel	RG(AxisTick) , <number> <fontSize> <orient>
	AxisLine	GS() , <height> <width>
	AxisTitle	RG(AxisLine), <fontSize><orient>
	Grid	GS() , <number>
Legend	LegendShape	RG(AxisLine) , LS() , <height> <width> <radius>
	LegendText	RG(LegendShape) , <fontSize><orient>
Mark	Shape	GS()
Label	LabelText	RG(Shape), <fontSize><orient>

Notation: Encoding includes `layout dependency` and `independent attributes`. The former includes `Global Scale (GS)`, `Local Scale (LS)`, or `Reactive Geometry (RG)`.

Table 1. MobileVisFixer identifies 5 classes and 10 groups that effectively represents a data visualization. Each group has a set of adjustable encoding properties. All those group-property combinations yield $\mathcal{C}_g^*$.

the positions of marks depend on the global coordinate scales, while titles and legends are located based on the axes positions. In addition, a legend also has its own local scale for placing its constituent shapes and text, and legend texts have a reactive geometry depending on corresponding legend shapes. Fig. 6 (C) shows an example.

There exist alternative specifications for the aforementioned layout dependency, since the mapping $\chi_g^* \rightarrow \psi$ is potentially one-to-many. The layouts of titles and legends could directly map to the global scale rather than using reactive geometry. The advantage of the latter is that it supports quick modifications of inter-group layouts through discrete operations, *e.g.*, switching the anchor position from top to bottom to render titles underneath the chart, whereas the former would require updating the vertical position in a continuous space. In Sect. 6.3 we discuss how such discretization provides conveniences for effectively solving the problem.

We note that MobileVisFixer does not consider potential dependencies of non-layout properties a designer might have chosen. For instance, there might exist a dependency between font-sizes of title and labels. Future work should address describing such dependencies. By adopting the declarative specifications, MobileVisFixer maps the group-wise visual encoding $\chi_g^* \in \mathbb{R}^{|\mathcal{C}_g^*|}$ to a parameter space (ψ). For instance, a group-property pair (`AxisTick`, `GS`) is mapped to declarative parameters (`scaleRangeMin`, `scaleRangeMax`). Eventually, the optimization problem (Eq. 1) is reduced to

$$\psi^* := \underset{\psi}{\operatorname{argmin}} J(\psi) \tag{2}$$

6 OPTIMIZATION

MobileVisFixer proposes an explainable reinforcement learning framework for solving the optimization problem. We present our design goals, the Markov Decision Process (MDP) model, and the heuristic.

6.1 Design Goals

MobileVisFixer is designed to improve common readability and aesthetics issues of web-based visualization while requiring minimal human intervention. We set ourselves the following design goals that aim to increase practical applicability.

G1: Simulate a trial-and-error process for manual repair. Manual creation of mobile-friendly visualizations is known to be an *ad hoc.*, iterative process [31], involving the adjustment of visual encodings while ensuring that these adjustments do not impact other parts of the visualization. MobileVisFixer aims to automate this process by mimicking human behavior.

G2: Ensure the transparency and explainability of the automation. Algorithmic interpretability and transparency are increasingly
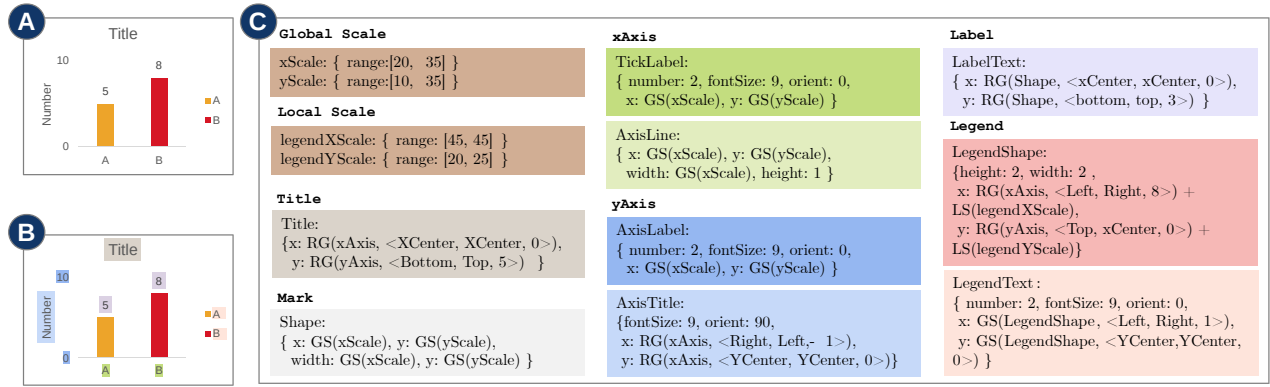
Fig. 6. Decoding phase: (A) The input is the original chart; (B) The first step identifies visual groups - a set of elements with the same encoding; (C) The second step determines the group-wise encoding in a declarative format. All the parameters constitute the output ψ.
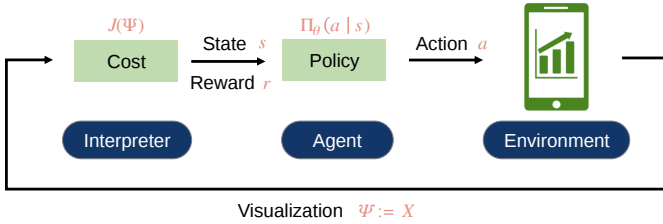

Fig. 7. Illustration of the reinforcement learning framework.

Scope	Mobile-friendly Issues	Notations
Global	Unwanted White Space	LeftMargin, RightMargin, TopMargin
Local	Out-of-viewport	LeftOutOfViewport, RightOutOfViewport, TopOutOfViewport
	Unreadable Fontsize	FontSize
	Overlapping Text	OverlappingText

Table 2. Summary of the notations of mobile-friendly issues.

important for automated systems. By utilizing explainable approaches, MobileVisFixer aims not only to support us in understanding our model, but also to help us gain insights for designing mobile-friendly visualizations by summarizing what machines have learned.

G3: Remain as faithful as possible to the original visualization. Visualizations are usually crafted with deliberate designs, which machines could fail to understand and preserve. As an automated framework, MobileVisFixer strives to maintain the visual encoding and design.

G4: Support compatibility with other algorithms. Mobile-friendly issues of visualizations are complex and even potentially ill-posed – a one-size-fits-all solution may not exist, which does not require modifications by a human. MobileVisFixer's goal is to alleviate such challenges by supporting existing algorithms for optimizing visualizations.

G5: Execute in browser rendering time. The last goal of MobileVisFixer concerns practical applicability – the automatic process should terminate in approximately similar time to the browser rendering process to meet real-world performance needs.

6.2 Explainable MDP Model

MobileVisFixer uses a reinforcement learning framework to solve the optimization problem, because reinforcement learning can theoretically mimic human behavior by learning from rewards (**G1**) [52].

A reinforcement framework is modelled as a Markov decision process (MDP), as illustrated in Fig. 7. The *environment* is the visualization specified by declarative parameter ψ. An *interpreter* calculates the cost $J(\psi)$ in respect to mobile-friendly issues. The *agent* observes the *state* ($s \in \mathcal{S}$) and *reward* ($r \in \mathbb{R}$), thereby taking an *action* ($a \in \mathcal{A}$) to manipulate ψ and consequently the environment. The agent's action selection is based on the policy $\Pi(a|s)$ - the probability that the agent takes action a when in state s. Thus, the goal is to learn the optimal policy that maximizes the rewards and therefore solves Eq. 2 effectively. In the following text, we explain the states, actions, and costs in detail.

6.2.1 State

MobileVisFixer features explicit definitions of states for framework explainability (**G2**). Specifically, states describe mobile-friendly issues that are observable by both humans and computers (*i.e.*, the interpreter). The notations of those issues, denoted $\mathcal{N}$, are summarized in Table

2. MobileVisFixer assumes that a visualization can be scrolled infinitely towards the bottom, and, thus the bottom orientation is not included in the out-of-viewport category. MobileVisFixer classifies notations into global ($\mathcal{N}_G$) and local ($\mathcal{N}_L$). The former applies to the global visualization, while the latter is specific to an individual visual element. Consequently, the total number of possible issues in a visualization is $|\mathcal{N}_G| + |\mathcal{E}| \times |\mathcal{N}_L|$. Consider that those issues could appear simultaneously which means that the total number of states becomes $2^{|\mathcal{N}_G|+|\mathcal{E}|\times|\mathcal{N}_L|}$ which renders the time complexity exponential. In addition, as visualizations can vary from $|\mathcal{E}|$, the model is not generalizable across visualizations. To alleviate those challenges (**G5**), MobileVisFixer proposes two strategies for State Aggregation – a common technique to reduce the number of states [66]:

1) Leveraging domain knowledge. We take advantage of our domain knowledge of visualizations. In particular, we aggregate visual elements into their corresponding classes (Title, Axis, Legend, Mark, and Label), as shown in Table 1. The resulting number of states is therefore reduced to $2^{|\mathcal{N}_G|+5\times|\mathcal{N}_L|}$. Furthermore, such state aggregation allows some generalizability as those classes are universal across visualizations, as demonstrated in Sect. 7.

2) Greedy aggregation. The result after the above step still has exponential complexity, which motivates us to adopt a greedy design – an established method for approximation algorithms [72]. Instead of considering all mobile-friendly issues simultaneously, MobileVisFixer greedily selects only one as the current state and, upon solving it, moves to the next issue as another state. This greedy strategy reduces the total number of states to $|\mathcal{N}_G| + 5 \times |\mathcal{N}_L|$, which becomes polynomial.

It should be noted that the optimization problem (Eq. 2) is indeed a multi-objective reinforcement learning problem – one that could require compromising solutions that balance different objectives (*i. e.*, mobile-friendly issues) [71]. The above state aggregation naturally leads to an approximation algorithm, by greedily solving a single-objective reinforcement learning problem. This inherits disadvantages of greedy algorithms which may make commitments to non-optimal solutions too early [17]. We will discuss our solution to this challenge in Sect. 6.3.

6.2.2 Action

Actions in MobileVisFixer manipulate the data visualization by updating the parameter space ψ. Most actions are defined by incremental or decremental operations due to two considerations: first, such progres-

sive changes are based on the original value (**G3**); and second, it allows us to discretize continuous action space, which could significantly improve the efficiency [50] (**G5**).

To be specific, inheriting from χ, the parameter in ψ (*e.g.*, the scale range) are in continuous space $\mathbb{R}$. Thus, the action space which manipulates those parameters is also continuous and infinite. MobileVisFixer discretizes this infinite space through incremental or decremental operations, *e.g.*, increase the min value of the scale range by a fixed number Δ. Users could specify the value of Δ (5px by default).

As shown in Fig. 11, MobileVisFixer currently has 23 actions. Twenty-one actions are incremental or decremental operations for global scale (8), local scale and reactive geometry (8), font-size (2), tick number of axes (2), and anchoring position of reactive geometry (1). Besides, it includes 2 actions which executes third-party algorithms for optimizing visualizations[4,5]. While it currently only utilizes two third-party algorithms, more can be added in the same manner (**G4**).

6.2.3 Cost Function

MobileVisFixer defines the cost function $J(\psi)$ in respect to each mobile-friendly issue. The cost function is designed iteratively – during development we found that a seemingly reasonable definition could lead to unexpected behavior of machines. Thanks to the explainable nature of our framework, we were able to locate the root cause in cost functions and make corrections. Below, we introduce cost functions in turn.

Out of the viewport. The cost is defined by the length exceeding the viewport on the left, right, and top orientation (Fig. 8 (A)). Initially we defined the cost by the area outside the viewport. However, the machine tended to reduce the visualization height when in the out-of-viewport state (Fig. 8 (B)). This action reduced the costs but yielded no actual improvement. More reasonable costs include the length or the relative area, and we utilize the former which is validate after experiments.
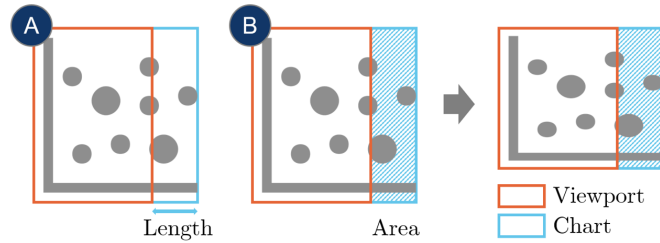

Fig. 8. (A) The cost function of out-of-viewport is determined by the length exceeding it; (B) A seemingly reasonable alternative is the area. However, the agent can learn to compress the height to reduce the cost.

Unwanted white space. Similar to the above, the cost is determined by the length of margins that exceed the threshold. The thresholds for amount of whitespace can be determined by users.

Unreadable font-size. The cost is calculated by:

$$\frac{1}{n} \sum_{i=1}^{n} f^+(s_i - \tau) \qquad (3)$$

where s_i is the font-size of the i-th text, and τ is the minimal font-size (default 12px). A seemingly reasonable alternative is the sum instead of the average. However, we found that the machine learned to delete texts to reduce costs which we did not want to allow for this issue.

Overlapping text. The cost is computed as the sum of the total overlapping area. Using the sum we allow removing texts to solve overlapping, which is a common technique for responsive visualization design [31].

6.3 Heuristic

We propose a greedy heuristic to train the agent effectively (Algorithm 1). The heuristic is based on a policy-based approach that directly learns the optimal policy $\Pi(a|s)$. The advantages of using a policy-based approach is the effectiveness in high-dimensional action spaces

---

[4]https://github.com/vega/vega-label
[5]https://bl.ocks.org/mbostock/7555321

Algorithm 1: Greedy Heuristic

Input : Policy approximation Π_θ, learning rate α, penalty rate β
Result: hidden variable θ
initialize θ with zeros;
Greedily select an existing mobile-friendly issue as current state
  s_0 and compute initial cost J_0;
i $\leftarrow$ 0; $s \leftarrow s_0$; $J \leftarrow J_0$;
while s **do**
    i++;
    Sample an action a_i based by $\Pi_\theta(a|s)$;
    Evaluate the resulting state s_i, cost J_i, and return R;
    $\theta \leftarrow \theta + \alpha R \nabla_\theta log \Pi_\theta(a|s)$;
    if s_i != s **then** // Current issue solved
      if s_i *is previously entered* **then** // Deadlock
        foreach *action a between s and s_i* **do**
          $\theta \leftarrow \theta - \beta \nabla_\theta log \Pi_\theta(a|s)$;
        end
      end
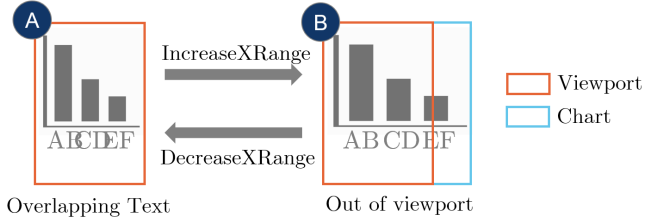 $s \leftarrow s_i$; $J \leftarrow J_i$;
    end
end

Fig. 9. The naive greedy algorithm could take short sighted, selfish actions that cause deadlocks between two states A and B.

and the possibility to learn a stochastic policy [66], which adheres to the intuition that the "optimal" visualization is non-deterministic.

Specifically, MobileVisFixeris based on the well-established policy-gradient algorithm by Williams [76]. This algorithm approximates the optimal policy using a parameterized function $\Pi_\theta(a|s)$, where θ is the hidden variable and Π_θ is usually the softmax function. At each step, it updates the policy gradient through

$$\theta \leftarrow \theta + \alpha R \nabla_\theta log \Pi_\theta(a|s) \qquad (4)$$

where α is the learning rate, and R is the return.

Here we simply use the reward r at each timestamp as the return, that is, $R_t = r_t$, which allows us to update the policy at each timestamp. The motivation is that one action could potentially have a significant influence on the visualization. The reward r_t is determined by the difference between the current and previous cost, normalized by the initial cost upon entering the state, since the cost function for each mobile-friendly issue varies from scales:

$$r_t = \frac{J_t - J_{t-1}}{J_0} \qquad (5)$$

MobileVisFixer utilizes a greedy algorithm to approximate the optimal solution. It greedily selects the state based on a predefined order, as shown in the leftmost columns of Fig. 11 (from top to bottom). We experimentally define this order: global parameters (*e.g.* margins, axes) are adjusted first, while font-related issues (*e.g.*, font-size, text overlapping) are solved last. Thanks to the greedy nature, MobileVisFixer computes the cost and updates the hidden variable θ in respect to only the current state at each timestamp, thereby reducing the time complexity from $\mathcal{O}(|S|)$ to $\mathcal{O}(1)$ (**R5**). However, such a greedy algorithm is "short sighted" – it only focuses on solving the current mobile-friendly issue, potentially causing other issues. Fig. 9 demonstrates an example where a deadlock occurs.
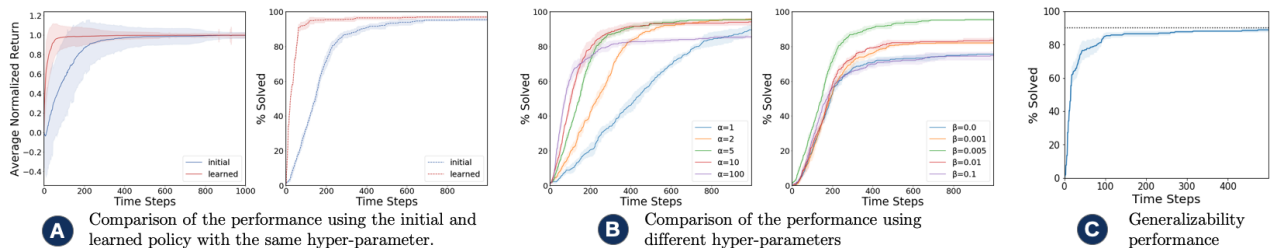
Fig. 10. The solid line and shaded regions represent the mean and standard deviation respectively across 5 runs.

MobileVisFixer addresses this challenge by imposing a long-term penalty for deadlocks, which takes advantages of the fact that reinforcement learning is particularly well-suited to problems that include a long-term versus short-term reward trade-off [66]. Upon entering a state s_{i+1} that is previously visited, MobileVisFixer penalizes all actions from s_i to s_{i+1} by

$$\theta \leftarrow \theta - \beta \nabla_\theta log \Pi_\theta(a|s) \qquad (6)$$

where β is the penalty rate. In other words, MobileVisFixer imposes a long-term penalty to counterbalance the actions toward short-term payoffs. As suggested by Mnih *el al.* [46], we use a fixed value which makes it easier to use the same rate across multiple visualizations.

7 EVALUATION

This section presents a series of quantitative and qualitative studies that aim to evaluate the performance, generalizability, and quality of the learned policy. The core of MobileVisFixer is implemented in Typescript. We tested MobileVisFixer on a MacBook Pro 2015 with a 2.7GHz Intel Core i5 processor and 8GB memory.

7.1 Training

We trained the agent on a small dataset including 81 visualizations. The dataset was manually selected from our corpus to alleviate bias caused by an unbalanced data problem. We kept charts that exhibits mobile-friendly issues and satisfies our prerequisites and removed charts with similar types, designs, and mobile-friendly issues.

In supervised learning, it is usually straightforward to track the model's performance by evaluating the performance on the training and testing dataset. However, accurately evaluating the progress of an agent in reinforcement learning is challenging [46]. Thus, we analyze the training performance using two metrics. The first is the average cumulative reward (returns), which is the most commonly used of such metrics [30]. However, since the returns vary across different visualizations, we normalize the returns so that 100% corresponds to the final score. The second metric is the percentage of solved problems [16]. Here we define that a visualization is solved if its cost is zero, *i.e.*, no mobile-friendly issue is detected. It should be noted that our cost functions exclude the distorted ratio issue, which usually results into mobile versions with different aspect ratios from the desktop version.

We perform 5 experiment runs with 1,000 time steps using the same hyper-parameter ($\alpha = 5, \beta = 0.005$). As suggested by Riedmiller *el al.* [58], we compare the training performance of the initial policy and the learned policy. As shown in Fig. 10 (A), the learned policy achieves rewards faster and with less variance. Also, the learned policy speeds up problem solving and eventually has a slightly better solving rate. Finally, MobileVisFixer successfully solves around 90% of visualizations within 100 steps, and 95% of visualizations within 1,000 steps.

We also investigate the impact of hyper-parameters, *i.e.*, the learning rate α and penalty rate β, as demonstrated in Fig. 10 (B). The results of the learning rate are consistent with existing knowledge [25] – small values (*e.g.*, 1 and 2) require more training steps, while larger values (*e.g.*, 100) cause the model to converge too quickly to a sub-optimal solution. It should be noted that our values are considerably larger than typical values $(0, 1]$ to compensate for the percentage scaling caused by the normalization of the return in Eq. 4.

We observe high parameter sensitivity for the penalty rate, which controls the long-term penalty to counterbalance the greedy actions towards short-term rewards. It tends to converge towards sub-optimal solutions when without penalty (*i.e.*, $\beta = 0$), which demonstrates the effectiveness of our penalty strategy. MobileVisFixer turns out to be very robust when choosing a medium value $\beta = 0.005$, while larger values lead to less successful learning behavior. We also find that the convergence rate is not sensitive to the penalty rate. This might be because problem solving is mainly related to choosing "good" actions (*i.e.*, get rewards) instead of avoiding "bad" ones (*i.e.*, get penalized). Specifically, penalizing "bad" actions does not necessarily contribute to increasing chances of "good" ones.

7.2 Evaluating Generalizability

We further evaluate the performance of MobileVisFixer on another dataset to understand its generalizability across different visualizations. Since there is no benchmark dataset, we collected our test dataset by searching and crawling visualizations with the keyword "Covid-19" on Observable (https://observablehq.com) published in March 2020, with the intention of capturing recent practice in web visualization. This results in 51 visualizations under the current prerequisite of MobileVisFixer. We adopt the desktop-version chart as the input to demonstrate MobileVisFixer's generalizability across input specifications.

We follow the methods by Cobbe *et al.* [16] to quantify generalization. Based on the results in the training, we perform 5 runs with 500 time steps on the testing dataset. Fig. 10 (C) shows the performance. In general, MobileVisFixer succeeds in finding good solutions in 89% cases, which is slightly worse than that in the training dataset.

We identified two common causes for failing cases. First, the optimization problem can be ill-posed, *i.e.*, there does not exist a solution given the actions defined by MobileVisFixer. For instance, considering a vertical bar chart with more than 30 bars, possible solutions include removing some bars or changing the orientation to horizontal. Future work should address how machines can reasonably take such actions. Second, the training is insufficient since several states are seldom visited. In particular, MobileVisFixer performed poorly when marks were out of the viewport, which was not observed during training. More training could alleviate this issue.

7.3 Interpreting the Model

We interpret the learned model to qualitatively evaluate the performance. The learned policies can be interpreted as probabilistic decision rules. As visualized in Fig. 11, each row (S0-S28) corresponds to a state, and each column (A0-A22) represents an action. The cell encodes the probability that the agent takes a decision (*i.e.*, action) under a condition (*i.e.*, state). For instance, the first row indicates that when the top margin exceeds the threshold (S0), the agent has a near 100% probability to decrease the min range of the y-axis (A4), which conforms to human choices. In general, we identified four patterns, which provide implications for future improvements of automated approaches, as well as challenging issues to which human designers should attach importance when designing visualizations for mobiles.

First, the agent converges to a single action under most states. For instance, when the margin exceeds the threshold (S0-2), the agent has learned to adjust the scale in corresponding orientations, which matches our intuition. This shows that the agent has found a confident successful solution to this mobile-friendly issue. Second, the agent could also converge to multiple actions. S24 illustrates such an example, where the agent has learned that both reducing the tick number (A11) and
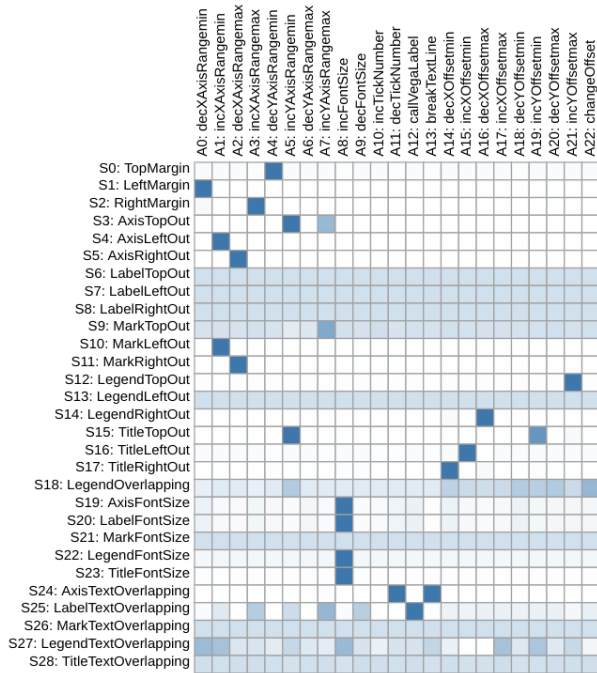
Fig. 11. The learned policy - the probability that the agent takes an action (column) when in a state (row). The color encodes the probability (truncated to 0.3).

breaking lines (A13) can alleviate the text overlapping problem at axes. However, the latter has less chance of improvement. Third, the agent does not get well trained on several states (*e.g.* due to insufficient observations, S6-9, S21, S26), as it exhibits a near equal distribution of probabilities. The former is due to the fact that labels were usually included by axes, that is, labels will stay in the viewport if the axes do. The latter two seemingly indicate invalid states – marks are unlikely to include text. This suggests that states in MobileVisFixer can be further reduced. Finally, the agent cannot make an action with high probability under fewer states (*e.g.*, S18, S27), that is, the machine has difficulties solving those difficult issues. This implies that designers should attach importance to certain challenging issues when creating web visualizations.

7.4 Gallery

Fig. 3 shows successful outputs of MobileVisFixer: given only the SVG as input, MobileVisFixer successfully generated visualizations with improved mobile-friendliness while remaining faithful to the original information and style. Note that there are subtle changes, e.g., the text labels in Fig. 3 (D) are re-positioned. More examples can be found in the supplemental material along with illustrating videos.

8 DISCUSSION AND FUTURE WORK

We now reflect on MobileVisFixer and discuss areas for future work.

8.1 Interpretable Automated Visualization Design

MobileVisFixer embraces algorithmic explainability and transparency that makes it easier for both model developers and end users to understand how the automatic system works. MobileVisFixer supports play-backing the optimization process step-by-step to help users understand the automation process. As described in Sect. 6.2.3, explainability helps us reason about unexpected system behavior and distinguish seemingly reasonable cost functions during development. More importantly, explainability allows interpreting our reinforcement learning model in the format of human-readable decision rules, which helps evaluate the quality of trained models. Compared with existing rule-based approaches that are often deterministic, the learned rules are stochastic and thus able to generate more flexible solutions to diverse visualizations, while removing the heavy manual effort of writing and polishing rules that scale well to real-world diversity.

With this in mind, MobileVisFixer adds to the recent discussion on rule-based and model-based systems for automatic visualization design [61]. While model-based methods (*e.g.*, [18,33]) have demonstrated promising results, they have not proven definitely superior to carefully crafted rules derived from human knowledge. However, rule-based approaches face limitations such as expensive rule creation and the combinatorial explosion of possible conditions [1]. As such, MobileVisFixer demonstrates a hybrid perspective that augment rules with models for their combined power. MobileVisFixer automatically learn reasonable decision rules from a reinforcement learning model, given that human-crafted rules in existing systems have not scaled to diverse real-world conditions. In the future, we are excited to explore how to embed rules in models to leverage the advantages that rules allow flexible and continuous extension. One possible solution is to enable users to adjust the policy and add customized states or actions.

8.2 Improving MobileVisFixer

MobileVisFixer has several limitations and further work is warranted to improve its usability.

Overcoming simplifying assumptions. MobileVisFixer does not address interaction problems since most web visualizations are static [31]. Future work should study how to automatically model and deconstruct interactive visualizations. Besides, MobileVisFixer omits the perception problems, since existing perceptual studies (e.g., [5,6]) on mobile visualizations only focus on a limited set of chart types. Future research should address a wider range of visualization designs.

Balancing agency and automation. We are excited to extend MobileVisFixer to include humans in the loop. For example, MobileVisFixer does not currently enable designers to input dependency relationships between non-layout properties among visual elements, such as the sizes of titles and axis labels, or to choose a desired aspect ratio to avoid distortions. It would be interesting and valuable to design and develop an interactive, semi-automated tool by taking a human-in-the-loop approach (*e.g.*, [35]), *i.e.*, supporting manual adjustment based on automatically generated results.

Improving greedy heuristics. MobileVisFixer utilizes problem reduction techniques and a greedy heuristic to solve a complex, multi-criteria optimization problem which may not find a "best" solution. For instance, our parameter specification might not accurately represent some visualizations, especially those with deliberate data- or context-dependent human design choices. Like similar greedy algorithms, MobileVisFixer speeds up computation at the cost of not always converging to a global optimum. Future research should propose approaches that better address this trade-off. Moreover, the greedy heuristic utilizes a pre-defined order to solve the multi-objective optimization problem, which could result into sub-optimal results. We hope to improve the performance by applying more advanced techniques such as adaptive reinforcement learning to dynamically update the greedy state.

Quantifying the mobile-friendliness through empirical studies. We evaluate MobileVisFixer through quantitative studies on two datasets containing 132 real-world visualizations. However, due to the scarcity of benchmarks, we cannot conclusively determine that our datasets are fully representative. Besides, we do not consider the distort ratio issue since resizing visualizations is a common technique, which, however, might cause perceptual bias that warrants future empirical studies. Our approach also inherits limitations common to reinforcement learning that the evaluation metric is based on the training objectives. Our results show that MobileVisFixer could efficiently solve 89% cases of the optimization problem with respect to the defined cost functions. However, it does not directly reflect on the overall quality of generated results because there lacks a metric for measuring mobile-friendliness for visualizations, which is currently not supported by Google and Bing mobile-friendly test tools. In the future, we plan to study such metrics through user studies and evaluate MobileVisFixer with those metrics on more data.

REFERENCES

[1] C. C. Aggarwal et al. *Recommender systems*, vol. 1. Springer, 2016.

[2] R. A. Al-Zaidy and C. L. Giles. Automatic extraction of data from bar charts. In *Proc. of the Conference on Knowledge Capture (K-CAP)*, number 30, pp. 1–4. ACM, 2015. doi: 10.1145/2815833.2816956

[3] I. Althomali, G. M. Kapfhammer, and P. McMinn. Automatic visual verification of layout failures in responsively designed web pages. In *Proc. Conference on Software Testing, Validation and Verification (ICST)*, pp. 183–193. IEEE, 2019. doi: 10.1109/ICST.2019.00027

[4] L. Battle, P. Duan, Z. Miranda, D. Mukusheva, R. Chang, and M. Stonebraker. Beagle: Automated extraction and interpretation of visualizations from the web. In *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, pp. 1–8. ACM, 2018. doi: 10.1145/3173574.3174168

[5] 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, 2018. doi: 10.1109/TVCG.2018.2865234

[6] 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, 2019.

[7] N. Bremer. Techniques for data visualization on both mobile & desktop. https://www.visualcinnamon.com/2019/04/mobile-vs-desktop-dataviz, Apr. 2019. Last visited: Apr. 2020.

[8] M. Brych. Data visualization in web and mobile apps. https://espeo.eu/blog/making-data-visualization-a-major-feature-of-your-next-app/, Apr. 2018. Last visited: Apr. 2020.

[9] M. Brych. Want to make data visualization a major feature of your next app? https://dzone.com/articles/want-to-make-data-visualization-a-major-feature-of, May 2018. Last visited: Apr. 2020.

[10] Z. Bylinskii, N. W. Kim, P. O'Donovan, S. Alsheikh, S. Madan, H. Pfister, F. Durand, B. Russell, and A. Hertzmann. Learning visual importance for graphic designs and data visualizations. In *Proc. of the Symposium on User Interface Software and Technology (UIST)*, pp. 57–69. ACM, 2017. doi: 10.1145/3126594.3126653

[11] Y. Chen. Visualizing large time-series data on very small screens. In *Proc. of the European Conference on Visualization (EuroVis) – Short Papers*, pp. 37–41. Eurographics Association, 2017. doi: 10.2312/eurovisshort.20171130

[12] L. Chittaro. Visualizing information on mobile devices. *IEEE Computer*, 39(3):40–45, 2006. doi: 10.1109/MC.2006.109

[13] E. K. Choe, R. Dachselt, P. Isenberg, and B. Lee. Mobile Data Visualization (Dagstuhl Seminar 19292). *Dagstuhl Reports*, 9(7):78–93, 2019. doi: 10.4230/DagRep.9.7.78

[14] J. Choi, S. Jung, D. G. Park, J. Choo, and N. Elmqvist. Visualizing for the Non-Visual: Enabling the Visually Impaired to Use Visualization. *Computer Graphics Forum*, 38(3):249–260, 2019. doi: 10.1111/cgf.13686

[15] M. Cliche, D. Rosenberg, D. Madeka, and C. Yee. Scatteract: Automated extraction of data from scatter plots. In *Proc. of the Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pp. 135–150. Springer, 2017. doi: 10.1007/978-3-319-71249-9_9

[16] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning. In *Proc. of Machine Learning Research*, vol. 97, pp. 1282–1289. PMLR, 2018.

[17] R. A. DeVore and V. N. Temlyakov. Some remarks on greedy algorithms. *Advances in Computational Mathematics*, 5(1):173–187, 1996. doi: 10.1007/BF02124742

[18] 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

[19] Emper0r. Grouped bar chart. https://observablehq.com/@2shabby/grouped-bar-chart, Apr. 2020. Last visited: Apr. 2020.

[20] E. Enge. Mobile vs. desktop usage in 2019. https://www.perficientdigital.com/insights/our-research/mobile-vs-desktop-usage-study, Apr. 2019. Last visited: Apr. 2020.

[21] R. Gal. Responsive visualizations coming to Power BI. https://powerbi.microsoft.com/fr-fr/blog/responsive-visualizations-coming-to-power-bi/, July 2017. Last visited: Apr. 2020.

[22] S. Gibbs. Mobile web browsing overtakes desktop for the first time. https://www.theguardian.com/technology/2016/nov/02/mobile-web-browsing-desktop-smartphones-tablets, Nov 2016.

[23] Google Inc. Mobile-friendly test – google search console. https://search.google.com/test/mobile-friendly. Last visited: Apr. 2019.

[24] Google Inc. Visualization: Area chart. https://developers.google.com/chart/interactive/docs/gallery/areachart. Last visited: Apr. 2020.

[25] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012. doi: 10.1109/TSMCC.2012.2218595

[26] A. Gustafson. *Adaptive web design: Crafting rich experiences with progressive enhancement*. New Riders, 2015.

[27] D. Haehn, J. Tompkin, and H. Pfister. Evaluating 'graphical perception' with cnns. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):641–650, 2018. doi: 10.1109/TVCG.2018.2865138

[28] J. Harper and M. Agrawala. Deconstructing and restyling D3 visualizations. In *Proc. of the Symposium on User Interface Software and Technology (UIST)*, pp. 253–262. ACM, 2014. doi: 10.1145/2642918.2647411

[29] J. Harper and M. Agrawala. Converting basic D3 charts into reusable style templates. *IEEE Transactions on Visualization and Computer Graphics*, 24(3):1274–1286, 2017. doi: 10.1109/TVCG.2017.2659744

[30] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Proc. AAAI Conference on Artificial Intelligence*, 2018.

[31] J. Hoffswell, W. Li, and Z. Liu. Techniques for flexible responsive visualization design. In *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, p. 1–13. ACM, 2020. doi: 10.1145/3313831.3376777

[32] E. Hoque and M. Agrawala. Searching the visual style and structure of D3 visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):1236–1245, 2019. doi: 10.1109/TVCG.2019.2934431

[33] K. Hu, M. A. Bakker, S. Li, T. Kraska, and C. Hidalgo. VizML: A machine learning approach to visualization recommendation. In *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, pp. 1–12. ACM, 2019. doi: 10.1145/3290605.3300358

[34] G. Inc. Bar charts — google developers. https://developers-dot-devsite-v2-prod.appspot.com/chart/interactive/docs/gallery/barchart, Apr. 2020. Last visited: May. 2020.

[35] D. Jung, W. Kim, H. Song, J.-i. Hwang, B. Lee, B. Kim, and J. Seo. Chartsense: Interactive data extraction from chart images. In *Proceedings of the 2017 chi conference on human factors in computing systems*, pp. 6706–6717, 2017.

[36] K. Kafle, B. Price, S. Cohen, and C. Kanan. DVQA: Understanding data visualizations via question answering. In *Proc. of the Conference on Computer Vision and Pattern Recognition*, pp. 5648–5656, 2018. doi: 10.1109/CVPR.2018.00592

[37] 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 *Proc. of the Conference on Human Factors in Computing Systems (CHI)*, pp. 5092–5103. ACM, 2016. doi: 10.1145/2858036.2858558

[38] S. Y. Kim, Y. Jang, A. Mellema, D. S. Ebert, and T. Collinss. Visual analytics on mobile devices for emergency response. In *Proc. of the Symposium on Visual Analytics Science and Technology (VAST)*, pp. 35–42. IEEE, 2007. doi: 10.1109/VAST.2007.4388994

[39] B. Lee, M. Brehmer, P. Isenberg, E. K. Choe, R. Langner, and R. Dachselt. Data visualization on mobile devices. In *Extended Abstracts of the Conference on Human Factors in Computing Systems (CHI)*. ACM, 2018.

[40] B. Lee, E. K. Choe, P. Isenberg, K. Marriott, and J. Stasko. Reaching broader audiences with data visualization. *IEEE Computer Graphics and Applications*, 40(2):82–90, 2020.

[41] P. LePage. Responsive web design basics. https://developers.google.com/web/fundamentals/design-and-ux/responsive. Last Accessed: Dec. 2019.

[42] 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

[43] S. Mahajan, N. Abolhassani, P. McMinn, and W. G. Halfond. Automated repair of mobile friendly problems in web pages. In *Proc. of the Confer-*

ence on Software Engineering (ICSE), pp. 140–150. ACM, 2018. doi: 10. 1145/3180155.3180262

[44] S. Mahajan, A. Alameer, P. McMinn, and W. G. Halfond. Automated repair of layout cross browser issues using search-based techniques. In *Proc. of the Symposium on Software Testing and Analysis (ISSTA)*, pp. 249–260. ACM, 2017. doi: 10.1145/3092703.3092726

[45] Microsoft Corp. Bing – mobile friendliness test tool. `https://www.bing.com/webmaster/tools/mobile-friendliness`. Last visited: Dec. 2019.

[46] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. In *Deep Learning Workshop NIPS*, 2013.

[47] Mobify. Mobify – the modern front-end platform as a service. `https://www.mobify.com/`. Last visited: Dec. 2019.

[48] D. Moritz, C. Wang, G. L. 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, 2018. doi: 10.1109/TVCG. 2018.2865240

[49] C. Mullins. Responsive, mobile app, mobile first: untangling the UX design web in practical experience. In *Proc. of the International Conference on the Design of Communication (SIGDOC)*, pp. 1–6. ACM, 2015. doi: 10.1145/2775441.2775478

[50] R. Munos and A. Moore. Variable resolution discretization in optimal control. *Machine learning*, 49:291–323, 2002. doi: 10.1023/A: 1017992615625

[51] P. O'Donovan, A. Agarwala, and A. Hertzmann. Learning layouts for single-pagegraphic designs. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1200–1213, 2014. doi: 10.1109/TVCG.2014. 48

[52] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics*, 37(4), 2018. doi: 10.1145/3197517 .3201311

[53] J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. In *Computer Graphics Forum*, vol. 36, pp. 353–363. The Eurographics Association & John Wiley & Sons, Ltd., 2017. doi: 10.1111/cgf.13193

[54] J. Poco, A. Mayhua, and J. Heer. Extracting and retargeting color mappings from bitmap images of visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 24(1):637–646, 2017. doi: 10.1109/TVCG.2017. 2744320

[55] C. Podger. How do mobile video viewers hold their phone? `https://www.scientiamobile.com/how-do-mobile-video-viewers-hold-their-phone/`, Feb. 2019. Last accessed: Apr. 2020.

[56] X. Qin, Y. Luo, N. Tang, and G. Li. Making data visualization more efficient and effective: a survey. *The VLDB Journal*, 29:93–117, 2020. doi: 10.1007/s00778-019-00588-3

[57] 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, 2018. doi: 10.1109/TVCG.2018.2865158

[58] M. Riedmiller, J. Peters, and S. Schaal. Evaluation of policy gradient methods and variants on the cart-pole benchmark. In *Proc. of the Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pp. 254–261. IEEE, 2007. doi: 10.1109/ADPRL.2007.368196

[59] J. Ritchie. Covid-19 charts. `https://observablehq.com/@jritch/covid-19-charts`, Mar. 2020. Last visited: Apr. 2020.

[60] 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. doi: 10.1109/ MCG.2014.82

[61] B. Saket, D. Moritz, H. Lin, V. Dibia, C. Demiralp, and J. Heer. Beyond heuristics: Learning visualization design. *arXiv preprint arXiv:1807.06641*, 2018.

[62] 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, 2016. doi: 10.1109/TVCG.2016. 2599030

[63] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE transactions on visualization and computer graphics*, 22(1):659–668, 2015. doi: 10.1109/TVCG.2015.2467091

[64] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer.

Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 393–402, 2011.

[65] stiansjoli. Scatterplot av humane-gener mot totalt antall basepar for de 23 kromosomene. dataene er hentet fra første tabell i. `https://observablehq.com/@stiansjoli/scatterplot-av-humane-gener-mot-totalt-antall-basepar-for-d`, Sept. 2019. Last visited: Apr. 2020.

[66] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. The MIT Press, 2018.

[67] J. Talbot, J. Gerth, and P. Hanrahan. An empirical model of slope ratio comparisons. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2613–2620, 2012. doi: 10.1109/TVCG.2012.196

[68] S. Teal. Making mobile-friendly data visualizations with tableau 2019.1. `https://public.tableau.com/en-us/s/blog/2019/01/making-mobile-friendly-data-visualizations-tableau-20191`, Feb 2019.

[69] D. Valle-Jones. Wasted talent. `http://www.diegovalle.net/charts/unam/wasted.html`. Last visited: Apr. 2020.

[70] D. Valle-Jones. The best high schools in mexico city (COMIPEMS analysis). `https://blog.diegovalle.net/2013/07/the-best-high-schools-in-mexico-city.html`, July 2013. Last visited: Apr. 2020.

[71] K. Van Moffaert and A. Nowé. Multi-objective reinforcement learning using sets of pareto dominating policies. *The Journal of Machine Learning Research*, 15(1):3483–3512, 2014.

[72] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[73] J. Walny, C. Frisson, M. West, D. Kosminsky, S. Knudsen, S. Carpendale, and W. Willett. Data changes everything: Challenges and opportunities in data visualization design handoff. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):12–22, 2019. doi: 10.1109/TVCG.2019. 2934538

[74] T. A. Walsh, G. M. Kapfhammer, and P. McMinn. Automated layout failure detection for responsive web pages without an explicit oracle. In *Proc. of the International Symposium on Software Testing and Analysis (ISSTA)*, pp. 192–202. ACM, 2017. doi: 10.1145/3092703.3092712

[75] T. A. Walsh, P. McMinn, and G. M. Kapfhammer. Automatic detection of potential layout faults following changes to responsive web pages (n). In *Proc. of the Conference on Automated Software Engineering (ASE)*, pp. 709–714. IEEE, 2015. doi: 10.1109/ASE.2015.31

[76] R. J. Williams. *Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning*, pp. 5–32. Springer US, Boston, MA, 1992. doi: 10.1007/978-1-4615-3618-5_2

[77] 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. of the Conference on Human Factors in Computing Systems (CHI)*, pp. 2648–2659. ACM, 2017. doi: 10.1145/3025453.3025768

[78] J. Young. Mobile data-visualization has a y axis problem. `https://jydesign.com/mobile-data-visualization-has-a-y-axis-problem`, Dec. 2019.