Converting Basic D3 Charts into Reusable Style Templates

Jonathan Harper and Maneesh Agrawala

Abstract—We present a technique for converting a basic D3 chart into a reusable style template. Then, given a new data source we can apply the style template to generate a chart that depicts the new data, but in the style of the template. To construct the style template we first deconstruct the input D3 chart to recover its underlying structure: the data, the marks and the mappings that describe how the marks encode the data. We then rank the perceptual effectiveness of the deconstructed mappings. To apply the resulting style template to a new data source we first obtain importance ranks for each new data field. We then adjust the template mappings to depict the source data by matching the most important data fields to the most perceptually effective mappings. We show how the style templates can be applied to source data in the form of either a data table or another D3 chart. While our implementation focuses on generating templates for basic chart types (e.g., variants of bar charts, line charts, dot plots, scatterplots, etc.), these are the most commonly used chart types today. Users can easily find such basic D3 charts on the Web, turn them into templates, and immediately see how their own data would look in the visual style (e.g., colors, shapes, fonts, etc.) of the templates. We demonstrate the effectiveness of our approach by applying a diverse set of style templates to a variety of source datasets.

Index Terms—Chart restyling, reusable style templates, declarative representation, D3 deconstruction, vega-lite

1 Introduction

Designing visually appealing charts that convey data clearly requires navigating a large space of visual styles. Designers must carefully choose visual attributes (e.g., position, size, shape, color, font) for the data encoding marks (e.g., bars in a bar chart or points in a scatterplot) as well as the non-data encoding elements (e.g., tick marks, gridlines, text labels) in the chart. Although researchers have developed design principles for making these choices [1], [2], [3], the principles are not widely known; poorly designed charts that are visually unappealing and hinder understanding by obscuring the data, are ubiquitous [4].

Existing visualization tools like Excel, Tableau, Spotfire, and R/ggplot2 provide a default visual style for the charts they produce. While these tools usually offer controls for manually tweaking the visual attributes of the resulting charts, altering the default style can be tedious. Most users end up exploring a very small region of the design space centered around the default style and the charts produced by these tools often look homogeneous.

In contrast, the Web contains a large collection of charts in a wide variety of different visual styles. These examples can help designers better understand the space of possible visual styles [5]. Moreover, for novice designers it is often far easier to select the desired style from a set of examples than it is to generate a new style from scratch [6], [7]. But existing visualization tools do not provide any support for re-using the visual style of an example chart.

The dominant tool for constructing Web-based charts is the D3 JavaScript library [8]. A growing community of D3 developers has already published tens of thousands of D3 charts online [9], [10], [11]. But despite its widespread use, D3 remains a tool for skilled developers and replacing data or changing the visual look of an existing D3 chart (e.g., converting a bar chart into dot plot) usually requires significant re-coding.

Vega [12] and Vega-lite [13] have introduced higher-level declarative languages that allow users to declaratively specify a chart as a collection of mappings between data and marks. Each mapping describes how a visual attribute of the mark (e.g., position, size, color etc.) encodes the corresponding data field. Mackinlay [2] has shown that this mapping-based representation significantly reduces the amount of code necessary to specify a chart while remaining expressive enough to generate a wide variety of basic chart types (e.g., bar charts, line charts, dot plots, scatterplots, etc.). Today however, relatively few Vega/Vega-lite charts are available online compared to D3, and Vega-lite currently provides only one default visual style for the charts it generates.

Harper and Agrawala [14] recently developed a technique for deconstructing existing SVG-based D3 charts to recover their underlying structure; the data, the marks and the mappings between them. While they also provide a graphical interface for interactively restyling D3 charts, their tool is primarily aimed at visualization experts. All design decisions are left to the user, who must manually modify the deconstructed mappings to adjust the look of a chart and their interface does not allow users to replace the underlying data.

- *J. Harper is with the University of California, Berkeley, CA 95064. E-mail: jharper@berkeley.edu.*
- *M. Agrawala is with Stanford University, Stanford, CA 94305. E-mail: maneesh@cs.stanford.edu.*

Manuscript received 13 Sept. 2016; revised 16 Dec. 2016; accepted 24 Dec. 2016. Date of publication 7 Feb. 2017; date of current version 26 Jan. 2018. Recommended for acceptance by B. Lee.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2017.2659744

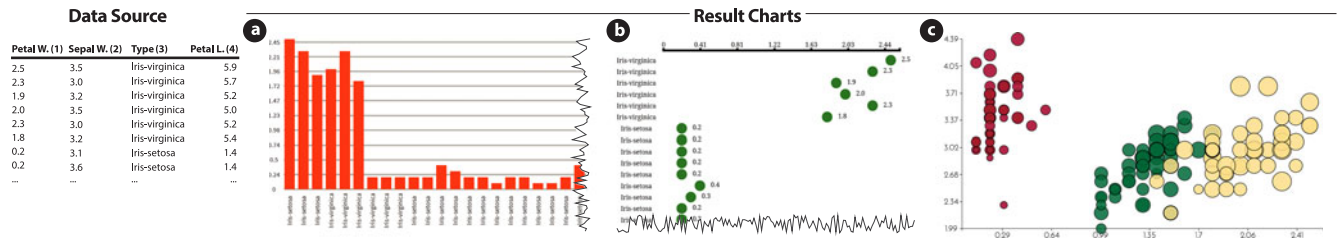


Fig. 1. Converting basic D3 charts (e.g., bar charts, line charts, dot plots, scatterplots, etc.) into reusable style templates lets users explore a variety of visual styles for their data. Once we have constructed the templates (see Fig. 6 for the original template charts) the user can specify a new source data table (left) with the importance of each data field (black number in parentheses) and can then apply the template to immediately produce a chart that maintains the visual style (colors, shapes, fonts, etc.) of the template but depicts the source data. Note that due to space limitations we have clipped the right side of the bar chart (a) and bottom of the dot plot (b). ***(We encourage readers to zoom in to see chart details such as fonts and gridlines.)***

In this paper we introduce an algorithm for converting a basic D3 chart into a reusable style template. Applying the resulting style template to a new data source produces a chart that depicts the new data, but in the visual style of the template chart (Fig. 1). To convert a D3 chart into a style template we first deconstruct the chart using an extension of Harper and Agrawala's approach. We then rank the perceptual effectiveness of the deconstructed mappings based on prior work in graphical perception [1], [2]. To apply the resulting style template to a new data source we first obtain importance ranks for each new data field. We then adjust the template mappings to depict the source data by matching the most important data fields to the most perceptually effective mappings.

We show how our style templates can be applied to source data in the form of either a user-specified data table or another basic D3 chart. Our proof-of-concept implementation focuses on constructing reusable templates for several common chart types: variants of bar charts, line charts, dot plots, and scatterplots. Users can easily find such basic D3 charts on the Web and immediately see how their data would look in the visual style (e.g., colors, shapes, fonts, etc.) of the templates. We demonstrate that our templates enable quick exploration of visual chart styles by applying a diverse set of style templates to a variety of source datasets. Unlike previous chart design tools, our approach lets users focus primarily on their data rather than designing the visual appearance of a chart from scratch or relying on a predefined default chart style.

Our contributions include:

- ***Algorithm for constructing style templates from D3 charts.*** We extend the approach of Harper and Agrawala to recover additional structure from D3 charts, including new types of mappings and relationships between data fields. We demonstrate that the extended representation fully captures the structure of many common chart types and can be directly translated into the mapping-based representation used by Vega-lite [13]. We develop new techniques for ranking the perceptual effectiveness of mappings. We show that the additional structure and the rankings are crucial for converting D3 charts into reusable style templates.
- ***Algorithm for applying style templates to new data sources.*** We provide an algorithm for applying the resulting style templates to any user-specified data table or D3 chart. If the new source data is a table we assume the user has specified the importance of the data fields. If the new source data is another D3 chart we show that we can infer the importance of the deconstructed data.
- ***Evidence for power of mapping-based chart representation.*** Mackinlay [2], Harper and Agrawala [14], and Vega-lite [13] have previously shown that the declarative mapping-based representation of charts is expressive enough for authors to describe a variety of basic chart types. Our work complements these results and shows that the mapping-based representation is high-level enough to allow programmatic manipulation of a chart's visual appearance and structure.

2 Related Work

Constructing a chart requires mapping data to the visual attributes (e.g., position, area, color) of graphical marks [2], [15]. While a number of programmatic chart construction tools such as InfoVis Toolkit [16], ggplot2 [17], and Vega/Vega-lite [12], [13] have been designed to facilitate this mapping process, D3 [8] has become the most popular Javascript library for producing charts for the Web. Our work converts existing basic D3 charts into reusable style templates that can be easily applied to new data sources.

Chart design tools like Excel, Google Spreadsheets, Polaris/Tableau [18], Lyra [19] iVoLVER [36] and Data Driven Guides [37] allow users to specify the mappings between data and mark attributes through a graphical user interface. However, users must rely on their own expertise to choose the appropriate mappings. Mackinlay [2], [3] was the first to develop an algorithm for constructing basic charts by automatically mapping the most important data fields (as specified by the user), to the most perceptually effective mark attributes (as determined via graphical perception studies [1]). Our work inverts this process; given a chart we rank the importance of the recovered data fields based on perceptual effectiveness of the attributes they map to.

Deconstructing a chart involves recovering its data, its marks and the mappings that relate them. Researchers have developed a number of image processing techniques for recovering marks and data from bitmap images of charts [20], [21], [22], [23]. While bitmaps are the most commonly available format for charts, accurate extraction

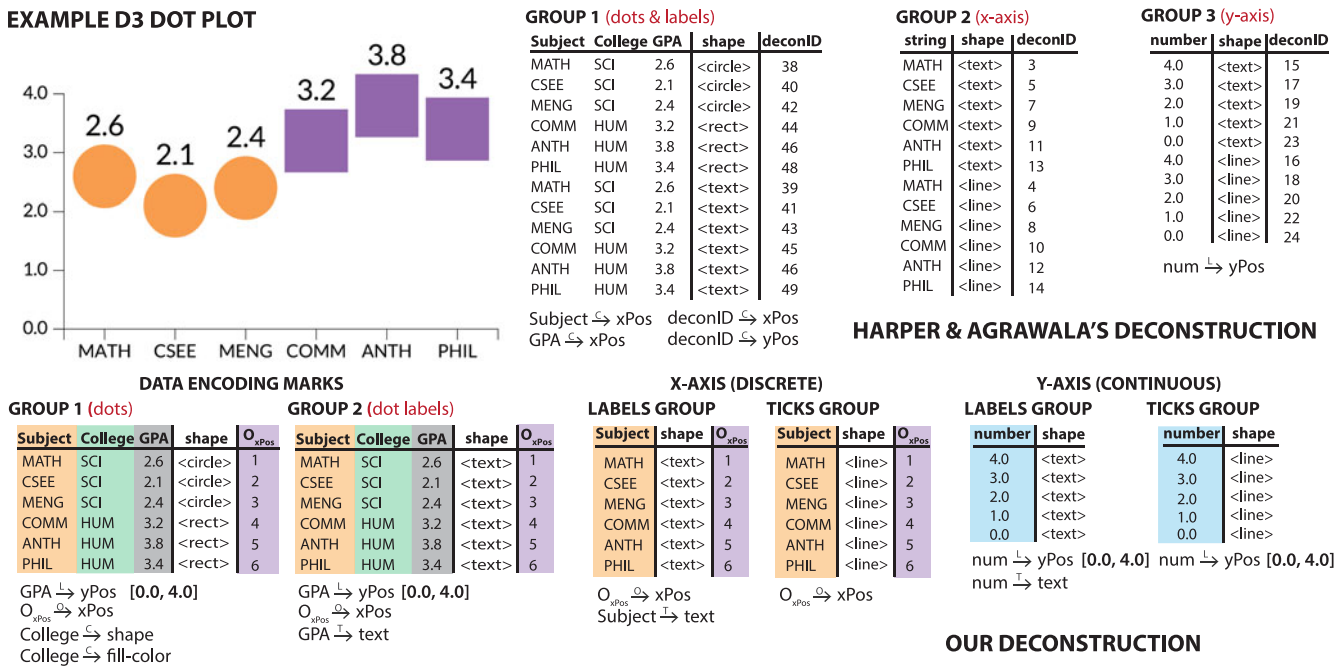


Fig. 2. Harper and Agrawala's deconstruction tool (top) extracts three groups of marks for the example dot plot along with five mappings (four mappings for Group 1 and one mapping for Group 3). We extend their tool to recover much more of the chart's structure (bottom). Our tool identifies two groups of data encoding marks, as well as a discrete *x*-axis and a continuous *y*-axis, each comprised of two groups of marks (labels and ticks). We unify data fields across the mark groups as indicated by the corresponding colored backgrounds. We construct attribute ordering O_{attr} fields for each mark attribute (only O_{xPos} is shown). We recover 13 mappings, including several attribute order mappings (denoted $\xrightarrow{O}$) and text mappings (denoted $\xrightarrow{T}$). We also construct a data domain for each linear mapping (e.g., the data domain for *GPA* $\xrightarrow{L}$ *yPos* is [0.0, 4.0]). Note that we have added the red labels in parenthesis to some of the groups to make it easier for readers to match the groups to the chart. These labels are not recovered by either tool.

remains challenging because of low image resolution, noise and compression artifacts. Despite such inaccuracies recent work has shown that it is possible to use the recovered marks and data to aid chart reading by adding graphical overlays [24] and by connecting the chart to explanatory text in the surrounding document [25].

Harper and Agrawala [14] focus on deconstructing D3 charts. Their approach recovers the marks and data with 100 percent accuracy and also recovers many of the mappings relating the data to the marks. Our work builds on their deconstruction approach. However, we significantly extend their deconstruction tool to recover additional chart structure, including new types of mappings and relationships between data fields. While Harper and Agrawala demonstrate a manual tool for restyling charts using their deconstructions, we show how the the additional structure we recover allows us to create style templates that can be applied directly to new data sources with no additional user effort.

Our work is inspired by recent techniques for manipulating visualizations. Transmogrification [38] lets users apply user-specified warps to images of charts and thereby produce new visual forms. Bigelow et al. [39] develop tools that allow users to easily move visualizations between programmatic construction tools like D3 and drawing tools like Adobe Illustrator, so that they can be edited wherever it is most convenient. Unlike these manual tools however, our work focuses on automating the chart styling process via reusable style templates.

Style transfer is a well studied problem in Computer Graphics. Researchers have developed a number of methods for transferring local characteristics such as texture [26], color [27], [28], non-photorealistic effects [29], [30], and noise [31], [32], from one image to another. These methods rely on non-parametric learning and signal processing techniques to separate the style of the image from its content and then apply the resulting model of the style to a new image. However these techniques cannot capture higher-level aspects of design (e.g., fonts, color palettes, line thickness) or domain-specific semantics (e.g., chart type, axes) and therefore are not well suited to our problem of transferring style between charts.

Our work is similar in spirit to Bricolage [33], an example-based tool for restyling webpages. Given two webpages, a content source and a style target, Bricolage matches page elements that are visually and semantically similar and then transfers the content from each source page elements to best matching target page element. Our style transfer approach similarly considers visual, perceptual and semantic similarity between elements of two input charts to determine how to map data from the data source chart to mark attributes of the style target chart.

3 Deconstruction

Our style templates explicitly represent the visual structure of a chart as a set of mappings between its data and its visual mark attributes. Harper and Agrawala [14] recently developed a tool for deconstructing basic SVG-based D3 charts into this representation. While D3 is general enough to work with other Web-based graphics APIs like Canvas and WebGL, we have found that SVG-based D3 charts are most common, likely because SVG provides a well-known scene graph representation for 2D graphics. To extract the

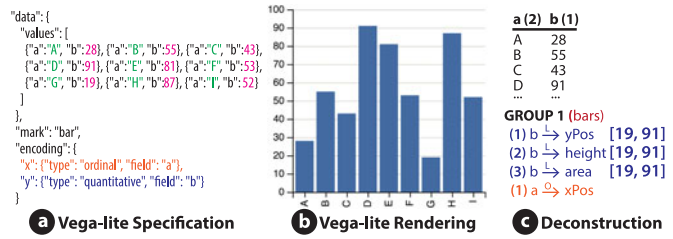


Fig. 3. Vega-lite [13] lets users describe a chart as a collection of mappings from data to mark attributes (a). The Vega-lite compiler renders such specifications using a single default visual style (b). Our parser can directly convert Vega-lite specifications into our deconstructed representation by analyzing the Vega-lite data, marks and encodings (c).

data and associated mark information their tool uses the fact that D3 charts bind input data to the SVG nodes representing marks. Their tool further checks if there are any linear or categorical mappings (denoted *data field* → *mark attribute*) that explain the relationships between the data and the mark attributes. They represent linear mappings as linear functions that take quantitative data values to quantitative attribute values. They represent categorical mappings as a table of correspondences between unique data values and unique attribute values.

We extract additional structure from D3 charts by extending their deconstruction tool in six ways: (1) we explicitly label data-encoding marks and axis marks as well as axis orientation (x- or y-axis) and whether the axis represents continuous or discrete data, (2) we group marks to identify additional mappings, (3) we construct data fields (e.g., O_{xPos}) to represent the ordering of the marks with respect to each mark attribute, (4) we unify data fields which are the same across mark groups, (5) we identify text format mappings which generate the text strings for text marks from the values in a data field, and (6) we compute data domains—the range of meaningful input values – for each linear mapping. Fig. 2 shows the additional structure we recover when we deconstruct a dot plot chart using our extensions. We describe these extensions in detail in Appendix A.

Note that even with these extensions our deconstructor is limited to basic charts—those that can be described as a collection of mappings between the data and mark attributes. Our implementation also inherits a few limitations from Harper and Agrawala and cannot handle certain types of charts including those that contain non-linear functional mappings (e.g., log scales), algorithmic layouts (e.g., treemaps), and non-axis reference marks (e.g., legends). We detail all of the limitations of our implementation in Section 7.1.

Similarity to Vega-Lite [13]. Our deconstructed representation for charts is very similar to the mapping-based representation of Vega-lite. As shown in Fig. 3a Vega-lite specification consists of data, marks and a collection of mappings from the data to mark attributes. To increase brevity of specification, users do not specify reference marks (e.g., axes, tick marks, etc.) and their mappings, and instead rely on Vega-lite to generate them implicitly based on the data.

But because the representations are so similar, we have developed a parser that can directly convert such Vega-lite specifications for data-encoding marks into our deconstructed representation and vice versa. For example, given the Vega-lite specification in Fig. 3a, where the mark type is *bar*, our parser directly translates the *encodings* into three linear mappings $b \xrightarrow{L} yPos$, $b \xrightarrow{L} height$ and $b \xrightarrow{L} area$ as well as an attribute order mapping $a \xrightarrow{O} xPos$. Our parser uses the mark type as well the the encoding information (e.g., type: *quantitative* or *ordinal*, field: *a* or *b*) to generate these mappings. It can similarly convert our mapping-based representation into a Vega-lite specification. Note however that because Vega-lite does not include reference marks and mappings we leave those out of these conversions.

4 Converting a D3 Chart Into a Style Template

Deconstructing a basic D3 chart recovers the mappings from the data to the mark attributes. To convert this deconstructed representation into a style template we rank the perceptual effectiveness of each recovered mapping. As we show in Sections 5 and 6, these rankings are essential for applying the resulting style template to new data sources.

We use Mackinlay's [2] rankings of the perceptual effectiveness of visual attributes to set the ranking of each mapping for the data encoding marks. Mackinlay's rankings differ depending on mapping type (quantitative/linear or categorical). We adapt these rankings to our set of mark attributes as in the inset Figure. Fig. 4a shows a style template with ranked mappings (numbers in green parentheses) for the D3 chart from Fig. 2. Note that while we have chosen Mackinlay's rankings for our examples because of their grounding in prior graphical perception research, the rankings are fully customizable within our system.

Quant./Linear	Categorical
1 xPos, yPos	1 xPos, yPos
2 width, height	2 fill color
3 area	3 stroke color
4 fill color	4 shape, text
5 stroke color	5 opacity
6 opacity	6 width, height
7 shape, text	7 area

5 Applying Style Template to Source Data Table

Given a style template and a source data table as input, our goal is to replace the data in the template chart with data from the source table (Figs. 4a, 4b, and 4e). We also require that an importance value (1 = most important, N = least important) is associated with each field of the input data source. In practice we assume that the user has provided this importance information as part of the source data table. For many chart creators this importance information is easy to provide as they are familiar with the data.

We apply the style template to the source data table using a three-stage algorithm; (1) we first compute additional metadata (e.g., data type) for the source data table, (2) we then use this metadata to adjust the mappings for the data encoding marks of the template chart to depict the source data, and (3) finally we rebuild the axes of the template chart to serve as reference lines for the updated data encoding marks.

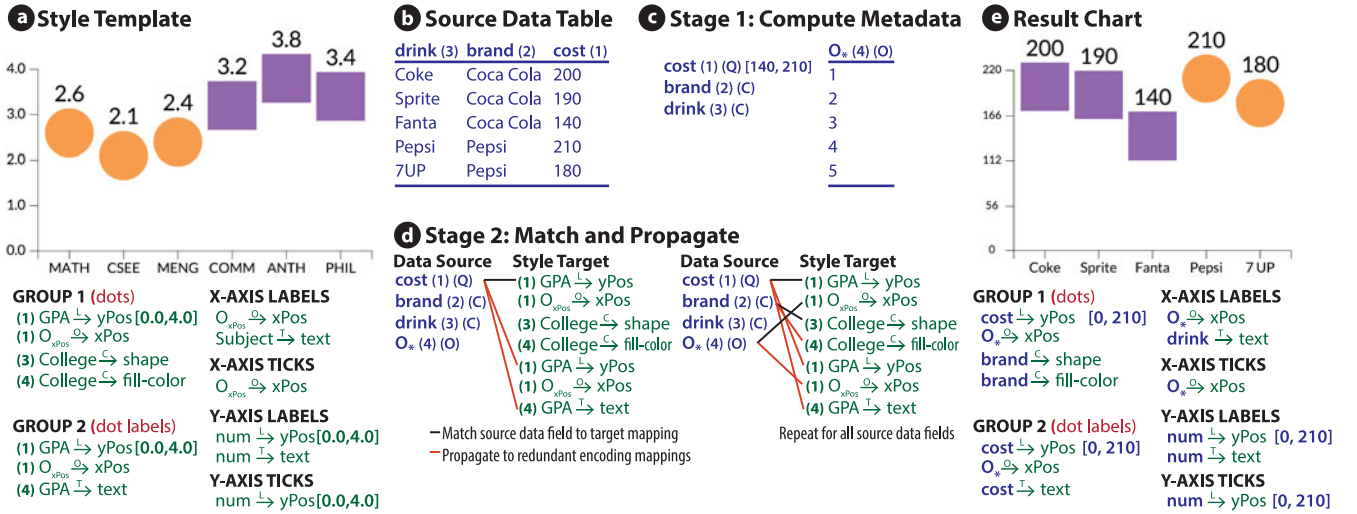


Fig. 4. We apply a style template (a) to a source data table (b) to generate the result chart (e). We convert a D3 chart (depicting average GPA for engineering and humanities colleges) into a style template (a) by deconstructing it and then ranking the perceptual effectiveness of the mappings for the data encoding marks (green numbers in parentheses for mark groups 1 and 2). Given a source data table (containing costs of Coke and Pepsi brand drinks) (b) with importance ranks for each field (blue numbers in parentheses), in stage 1 (c) we compute additional metadata (data type, data domain and ordering field O_*). In stage 2 (d) we match the most important data field to the most perceptually effective template mapping, while ensuring that the type of the data field is compatible with the mapping type. Once we find a match we propagate the replacement data field to any other redundant mapping in the template. For example, after matching the source data field *cost* to the $GPA \xrightarrow{L} yPos$ mapping we propagate *cost* as the replacement for *GPA* in every other template mapping, including axis mappings. Finally we generate the marks and rebuild the axes to produce the result chart (e).

5.1 Stage 1: Compute Metadata for Source Data Table

In the first stage of our algorithm we infer the data type (quantitative or categorical) for each field in the source data table. Specifically, we analyze the data values in each field of the source table. If the field contains only numeric values we set its data type to *quantitative* and if it contains non-numeric values (e.g., text strings) we set its data type to *categorical*. We also set the data domain for each quantitative data field to the min/max range of its data values.

Note that this simple classification heuristic will incorrectly label numerical categorical data (e.g., employee ID numbers, social security numbers, etc). as quantitative. But, because stage two of our algorithm allows quantitative data fields to serve as input to categorical mappings it can generate the proper result chart. We also allow users the option of specifying the data type for any field in the source table.

We also extend the source data table to include an *ordering* field O_* that represents the row index of each tuple in the data table. Since we construct this ordering data field and it does not represent any of the actual data in the table we assign it the least importance of all the source data fields. We use this ordering field in stage two of our algorithm to serve as input to attribute order mappings. Figs. 4b and 4c shows how we analyze and extend an input source data table.

5.2 Stage 2: Update Data-Encoding Marks

Data-encoding marks are the most important marks in a chart because their attributes directly encode the underlying data. To update the data-encoding marks of the template chart so that they reflect the source data, we first match source data fields to attribute mappings in the template. For each such match we then synthesize a new mapping function that maps the source data values to template mark attribute values. Finally we generate the marks for the new chart according to the updated mappings.

5.2.1 Match Source Data Fields to Template Mappings

Algorithm 1 describes our procedure for matching the source data fields to template mappings. Fig. 4d shows our matching process. Our approach is to map the most important data to the most perceptually effective mark attributes.

Our matching algorithm considers each source data field in order by decreasing importance (line 4) and selects a matching template mapping based on the data field type (lines 4-24). Quantitative data can serve as input to any linear mapping in the style template. Moreover, by treating each unique numeric data value as a distinct category, quantitative data can also serve as input to categorical mappings. Thus, if the data field is quantitative we match against the top ranked linear or categorical data mapping. A categorical data field however, may not be numeric and can therefore only serve as input to categorical mappings in the template. Thus, if the data field is categorical we match it to the top ranked categorical mapping. Note that text mappings are treated as categorical mappings in this matching procedure. Finally we match the *ordering* data field O_* to the top ranked attribute order mapping.

If we find a match we replace the template data field with the source data field in the matched mapping. The style template may include redundant encodings in which a single data field maps to several different visual attributes. We consider such redundant encodings to be stylistic constraints and if we replace such a redundantly mapped data field, we propagate the replacement to all of the redundant mappings including axis mappings (lines 25-27 and Fig. 4d). Once we have completed the propagation we continue the matching process with the next highest ranked source data field and the remaining unmatched style template mappings.

Algorithm 1. Match Source Data Fields To Style Target Mapping

```
1  Input: Source data table with importance rank and data
   type for each field. Deconstructed style target chart with
   perceptual effectiveness rankings for each target mapping.
2  U = {target mappings}      //Set of unmatched target mappings
3  M = {}                     //Set of matched target mappings
4  for each source data field s in descending importance order do
5    match = null             //Initialize target mapping match
6    switch DataType(s) do
7      case Quantitative
8          match = Top ranked linear or categorical
9            mapping in U. If tie in rank, pick any one
10           of top ranked linear mappings. If no linear
11           mapping available, pick any top ranked
12           categorical mapping.
13     end
14     case Categorical
15         match = Top ranked categorical mapping in U.
16           If tie in rank, pick any one of top ranked
17           categorical mappings.
18     end
19     case Ordering
20         match = Top ranked attribute order mapping in
21           U. If tie in rank, pick any one of top ranked
22           attribute order mappings.
23     end
24   end
25   t = DataField(match)   //Get original data field for match
26   DataField(match) = s   //Set replacement data field
27   Propagate s as replacement data field to any other
28     mapping in U for which t is the original data field.
29   Move all such modified mappings from U to M.
30 end
```

If we do not find a match for a source data field our data replacement result will not depict the data field. Such unmatched data fields occur when either the data source table includes many more data fields than the style template chart depicts, or when the source data fields are incompatible with the template mappings (i.e., the source contains only categorical data fields, but the style template contains only linear mappings). Alternatively, if the style template chart depicts more data fields than contained in the source data table, some style template mappings may remain unmatched after one complete pass of the matching loop. We can optionally use these extra template mappings to redundantly encode source data by repeating the matching loop over all of the source data fields, but only permitting a match with the remaining unmatched template mappings. If the style template includes any unmatched attribute order mappings this redundant encoding approach matches the ordering field O_* to them.

5.2.2 Synthesize Mapping Functions

After completing the matching process we synthesize new mapping functions for each matched pair of source data field and style template mapping, based on the template mapping's type: linear, categorical, attribute order, or text.

Linear. To synthesize a linear mapping function, we relate the endpoints of the data domain of the source data field to the endpoints of mark attribute range for the style template mapping. Since our matching algorithm ensures that the source data field for a linear mapping is quantitative we directly look up its data domain as computed in stage one of our algorithm. We compute the attribute range of the style template mapping by applying its mapping function to the endpoints of its original data domain. We then fit a linear function which maps the start and end points of the source data domain to the start and end points of the style template attribute range respectively. We use the resulting linear function as the mapping function for our result chart.

Note that for some chart types (e.g., bar charts, dot plots) it is critical that the chart include the origin at zero when depicting a quantitative data field. For other chart types (e.g., scatterplots) including the origin at zero can make it difficult to see the marks. To handle these two cases we check whether the data domain of the original template mapping include zero and if so we extend the source data domain to also include zero. If not, we leave the source data domain as we computed it in stage one (e.g., the min/max range of the data values). In Fig. 4e the linear *yPos* mapping for the dots is synthesized using the source *cost* field with domain [140, 210]. However, the original data domain of the style template was [0.0, 4.0] and so we extend the data domain of the result mapping to [0, 210].

Categorical. To synthesize a categorical mapping function, we create a new correspondence table pairing each unique value of the source data field with a unique value of the style template mapping's mark attribute. If the number of unique data values is less than or equal to the number of unique attribute values our approach generates a one-to-one correspondence table that serves as the new mapping function. For example, in Fig. 4b the source *brand* data field contains 2 values (Coca Cola and Pepsi) while the template contains two unique *fill-color* values and two unique *shape* values. In this case we assign one fill-color and one shape to each brand. If however, there are more unique data values than unique attribute values, our approach will leave some of the data values unmatched. In such cases we report to the user that it is impossible to construct a one-to-one categorical mapping and instead we reuse attribute values in cyclic order so that the same attribute value may be paired with multiple data values. Although the resulting chart does not correctly depict the source data—it visually aggregates distinct data values—it can still provide a visual sense for overall look of the resulting chart.

Attribute order. Attribute order mappings typically capture information about the chart's layout. To maintain the template chart's layout while using the ordering data from the source data table, we update the data field of the template attribute order mapping, but maintain the linear parameters of the mapping function unchanged. If the new source field contains more values (or fewer values) than the template ordering data field this approach extends (or shrinks) the layout to fit the new number of data values. The transfer result in Fig. 4e shows an example of such shrinking as the data source contains only five data elements while the style template contains six data elements.

Text. For text mappings we create a function that simply converts the source data value into a string.

5.2.3 Generate Marks

The final step of stage two is to generate the data encoding marks for the result chart. For each group of data encoding marks and mappings in the style template we retrieve the matched data fields from the source data table. We then join together these data fields into a unified data table and generate a mark for each row in the resulting table. We set the attributes for each mark by applying the newly synthesized mappings. For any unmapped mark attributes, we set the attribute value to the average (for numeric attributes) or mode (for other attributes) of the corresponding attribute values from the style template.

5.3 Stage 3: Rebuild Axes

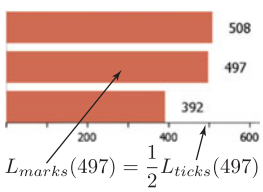
Once we have updated the data encoding marks of the style template, we rebuild its axes to reflect the new data. We use a different rebuilding algorithm depending on whether the axis is continuous or discrete. We describe the algorithms assuming we are rebuilding an *x*-axis; the algorithms for a *y*-axis are similar.

Continuous axis. A continuous *x*-axis commonly appears in a scatterplot or a horizontal bar chart and serves as a reference line relating the positions of the data encoding marks to data values. Maintaining the relationship between the *xPos* of the data encoding marks and the *xPos* of the axis tick marks is critical for such an axis to function properly.

Suppose $d \xrightarrow{L} \mathit{xPos}$ is a linear mapping from data field d to the *xPos* attribute. We can represent the linear mapping function as a matrix L in homogeneous coordinates where

$$L = \begin{bmatrix} a & b \\ 0 & 1 \end{bmatrix}, \tag{1}$$

and $L \cdot d = a \cdot d + b = \mathit{xPos}$. Our deconstruction tool recovers the linear parameters a and b for each such linear mapping. In this notation we refer to the *xPos* mappings for the data encoding marks and the axis ticks marks in the original style template as L_{marks} and L_{ticks} respectively.



Even though the data encoding marks and the *x*-axis tick marks depict the same data domain, the mappings L_{marks} and L_{ticks} may differ. For example, in a horizontal bar chart (inset) the bar positions are based on the center point of the rectangle and the *xPos* mapping for the bars is $\frac{1}{2}$ the value of the *xPos* mapping for the ticks. Therefore, to rebuild a continuous *x*-axis for the result chart we must first recover the relationship between these mappings in the original style template.

We compute this relationship R between the two mappings as

$$R \cdot L_{marks} = L_{ticks} \tag{2}$$

$$R = L_{ticks} \cdot L_{marks}^{-1}. \tag{3}$$

We treat this relationship R between the data-encoding marks and the axis tick marks as part of the style of the original template chart that must be maintained when we rebuild the axis for the new data. After synthesizing a new *xPos* mapping for the data encoding marks in stage two of our algorithm, we denote the new mapping function as L'_{marks}. To build the new x-position mapping for the axis tick marks L'_{ticks} while preserving the relationship to the data encoding marks we compute $L'_{ticks} = R \cdot L'_{marks}$ Although this approach updates the mapping from the data to the x-position of the tick marks, it preserves the positions of the ticks in image space. It simply changes the data value associated with each tick mark. To recover these data values we invert L'_{ticks} and apply it to the tick positions. We then treat these data values as the data for the text mappings of the axis labels. The tick marks of the continuous *y*-axis in the result chart of Fig. 4e match in position with the corresponding tick marks in the style template, but the axis labels are based on the updated source data field.

Discrete axis. A discrete *x*-axis commonly appears in a vertical bar chart or dot plot and is typically used to label the data encoding marks (e.g., bars or dots). For such charts, our style template contains a unified attribute order data field O_{xPos}, and mappings from this field to the the *xPos* of the ticks, labels and data encoding marks. The unification ensures when we update the attribute order mapping for the data encoding marks in stage two, our algorithm will propagate the update to set the *xPos* mappings for the *x*-axis ticks and labels. After updating the tick positions we adjust the axis line to span the new tick marks.

To update the text mappings for the axis labels we search the source data table for a data field containing a unique value for each row in the table. If our search finds more than one such data field we favor using a categorical field over a quantitative field as the input for the text mapping. If we cannot find any such data field we use O_* as the data field for the axis label text mapping. For the example in Fig. 4e, we find the *drink* data field through this search process and create the $\mathit{drink} \xrightarrow{T} \mathit{text}$ mapping for the result chart.

6 Applying Style Template to Source D3 Chart

In some cases users may have access to a basic D3 chart depicting their data, but wish to quickly explore other chart styles by applying alternative style templates. We can apply a style template to a D3 chart using our three-stage algorithm with small modifications to stage one. Moreover, our modified algorithm infers the importance values for the data fields depicted in the source D3 chart and therefore does not require that the user provide them as part of the input. However the user can always supply these importances if their preference differs from the inferred values.

Computing source data importance ranks. We start by deconstructing the source D3 chart to obtain its data, mark attributes and mappings. We focus on the subset of mappings for the data-encoding marks and assign a perceptual effectiveness ranking to each one using the same approach we used to construct the style templates (Section 4). Based on the assumption that charts map the most important data to the most perceptually effective mark attribute, we then directly treat the effectiveness rank as the importance of the mapped data field. If the same data field appears in more than one source mapping we give it the importance of its highest ranked mapping. Fig. 5a and 5b shows a source D3

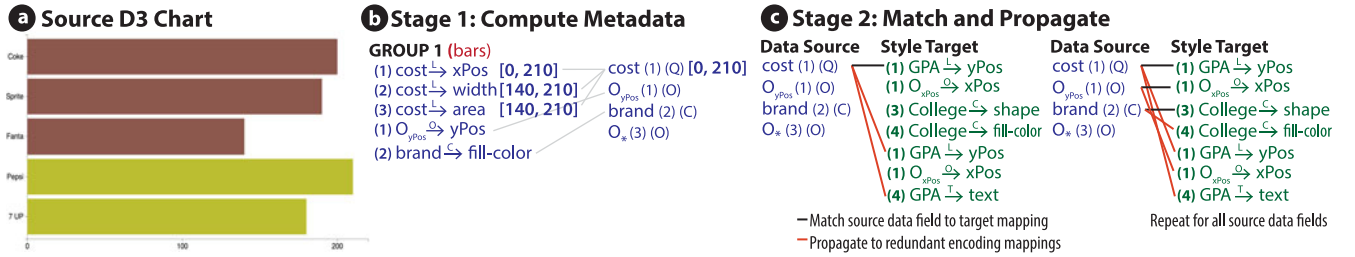


Fig. 5. We can apply style templates to a source D3 chart (a) using the three-stage algorithm of Section 5 but modifications to stage one (b). We deconstruct the source chart and use the perceptual effectiveness of the mappings for the data encoding marks (b left column) to set the importance of the source data fields (b right column). We also infer the data type, data domain and ordering field O_* using the deconstructed mappings. Finally we apply stages two and three of our algorithm as before to generate a result chart that appear exactly the same as the chart in Fig. 4e, but with all attribute order mappings using O_{yPos} from the source D3 chart rather than O_* from the source data table.

bar chart with perceptually ranked mappings for the data encoding bars (left column of Fig. 5b), aggregated into importance ranks (right column of Fig. 5b).

Computing source data types. We also use the source mappings to set the data type for each source data field. If the field is involved in any linear mapping we set its data type to *quantitative* and set its data domain to the domain of any one of the corresponding linear mappings. Our deconstruction process ensures that all linear mappings for the same data field are equivalent. If a source data field is only involved in categorical or text mappings we set its data type to *categorical*.

Working with source attribute ordering mappings. The deconstructed source D3 chart may also contain attribute order mappings. We set the data type to *ordering* for each data field involved in such an attribute order mapping. As in stage one of the original algorithm we extend the deconstructed source data table with an ordering data field O_* that holds the row index of each tuple in the table. Thus, the deconstructed source D3 chart may provide more than one ordering data field with different importance ranks (Fig. 5b). Nevertheless we can apply stage two of our algorithm without any modification to this deconstructed source data table (Fig. 5c). In this case Algorithm 1 matches the the most important ordering data fields of the source chart to the most perceptually effective attribute order mappings of the style template. Any unmatched attribute order mapping in the style template is then matched with O_*. In the Fig. 5 example we first match O_{yPos} and O_* does not need to be used. The result chart for this example appears exactly the same as in Fig. 4e, but with all attribute order mappings using O_{yPos} from the source D3 chart rather than O_* from the source data table.

7 Results

We have implemented a pair of tools for (1) converting a basic D3 chart into a re-usable style template and (2) applying the resulting style template to new source data (either a data table or the data in another D3 chart). Our chart conversion tool extends Harper and Agrawala's D3 Deconstructor Chrome plugin [14] so that users can click on any basic D3 chart from the Web and produce the corresponding style template. Our template application tool is a command-line tool that takes a data source file as input (either a CSV file with the importance of each field specified as metadata, or a deconstructed D3 chart with its corresponding data table as produced by the D3 Deconstructor) and produces an SVG-based chart that matches the style of the input chart as output. Users can further tweak the resulting chart if necessary using the manual re-styling tools included with the original D3 Deconstructor.

As shown in Figs. 1 and 6, our techniques for constructing and applying style templates let users quickly explore a variety of visual styles for any input source dataset. When the source data set is given as a data table (Figs. 1 and 6 cols a,b), the user must also specify the importance of each data field (numbers in parentheses). When the source data is given as a D3 chart (Fig. 6 cols c,d), the importance is inferred by our algorithm. In these cases users can see how a default chart style (e.g., the Excel style of the data source in Fig. 6 col c) might be restyled to produce better looking charts. All result charts were generated automatically without any additional user intervention.

The result charts maintain the visual style of the style template with similar attribute values (colors, fonts, shapes, etc.) for the marks, axes and labels. Yet, the data values and the numbers of marks differ significantly between the template charts and the result charts. Our algorithm is robust to these differences and generates result charts that depict the source data with the look of the style template.

The fonts, colors and gridlines vary considerably between the templates. A few of the templates include text labels on the marks (Fig. 6 rows 2,3,4) that redundantly encode data values to make it easier for viewers to read the exact values. The bar charts (Fig. 6 rows 1,2), dot plots (Fig. 6 rows 3,4), line chart (Fig. 6 row 5) and area chart (Fig. 6 row 6) all include a quantitative axis that starts at zero because zero is included in the template chart data domain. In contrast the scatterplots (Fig. 6 rows 7,8,9) do not always include an origin at zero.

The blue horizontal bar chart (Fig. 6 row 2) and orange purple dot plot (Fig. 6 row 3) style templates use color to depict a two-valued categorical variable. Several of the result charts for these templates (Fig. 6 rows 2,3 cols a,b,d) cycle between the colors because the source data fields that map to color contain more than two unique values (Section 5.2.2). However, the source data in Fig. 6 col c includes a field with only two unique data values and the result charts use color to depict it accurately.

The source data chart in Fig. 6 col c includes only one quantitative data field. Thus, when we apply the line, area, and scatterplot chart templates (Fig. 6 rows 5-9) to this source we produce a 1-dimensional result where the

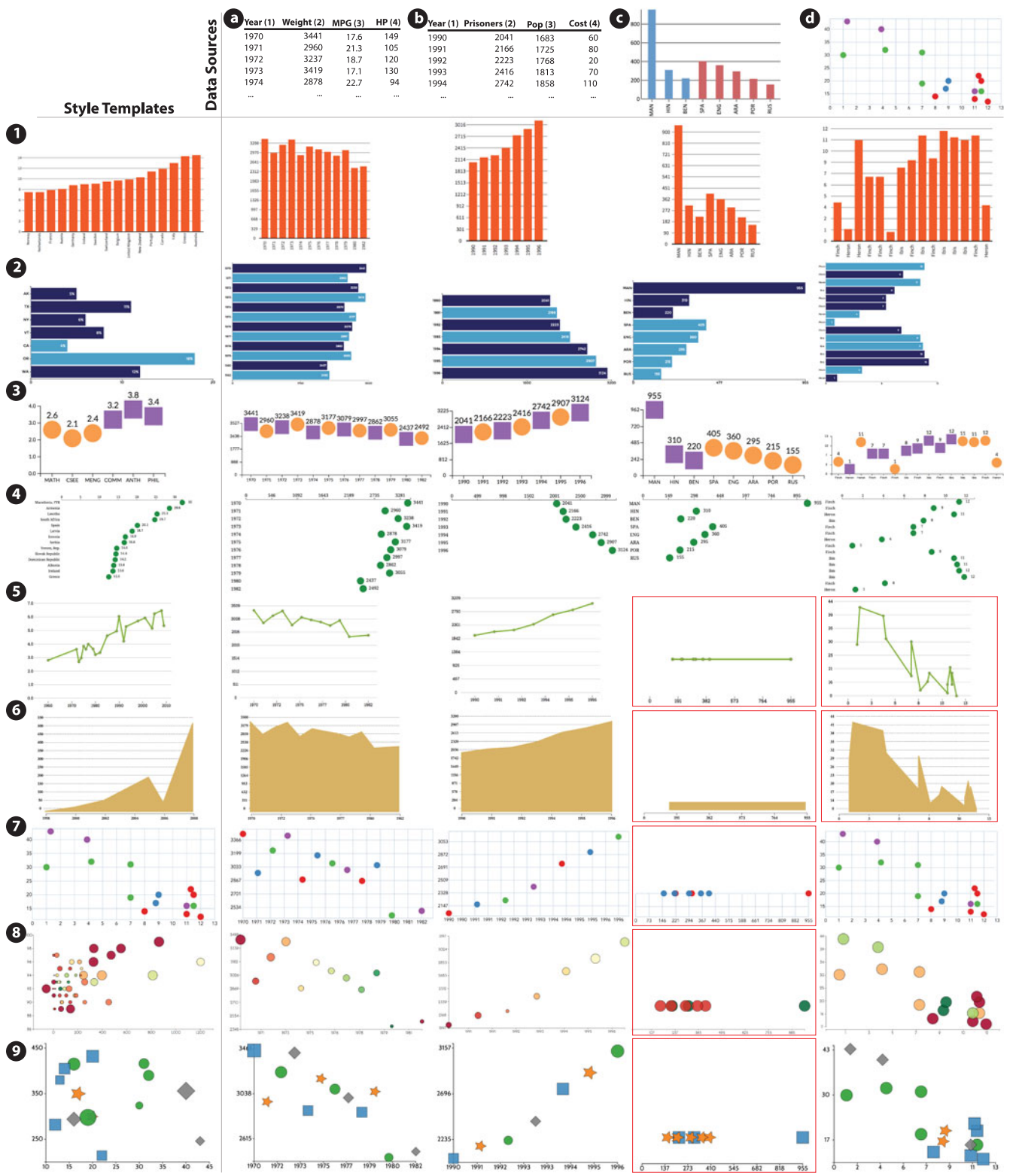


Fig. 6. Once we have converted a D3 chart into a style template we can apply it to new data sources in the form of data tables (a,b) or other D3 charts (c,d). The result charts maintain the visual style of the style template chart with similar attribute values (colors, fonts, shapes, etc.) for the marks, axes and labels. Yet, the data values and the numbers of marks differ significantly between the template charts and the result charts. Our algorithm is robust to these differences and generates result charts that depict the source data in the look of the style template. Red borders indicate transfer results that our algorithm warns users about as noted in Section 7.(Please zoom in to see chart details like fonts and gridlines.)

quantitative data field is mapped to *xPos* and the *yPos* remains unmapped so that all of the marks appear on the same horizontal line. While the line and area charts are difficult to read because of self-occlusions in this case, the 1-dimensional scatterplots can be useful plots for seeing the distributions of the quantitative data field. In all of these cases the resulting charts are correct in the sense that they depict the single quantitative dimension of the data source on a single axis. The resulting charts also remain visually similar to the style templates and thereby convey how the

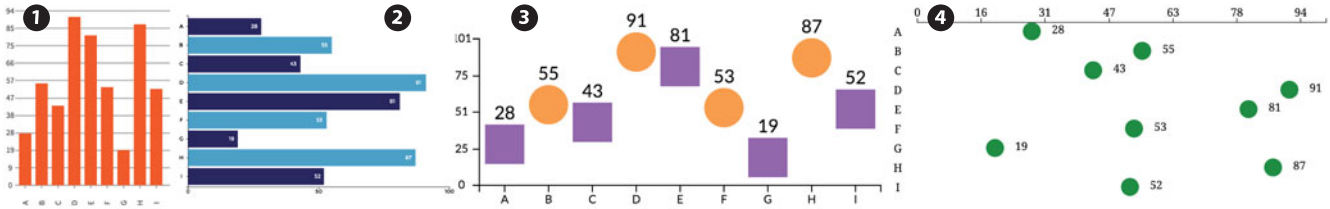


Fig. 7. After deconstructing a Vega-lite [13] chart using our parser (original Vega-lite specification and our deconstruction shown in Fig. 3) we can apply our style templates (original template charts 1-4 shown in Fig. 6 to to this source chart to explore additional chart styles that go beyond the Vega-lite default.

data would look using the template. However the line and area charts in particular are difficult to read and therefore whenever we apply a template that has a mismatch in the number of quantities or categorical mappings from the number of such fields in the data source we report the mismatch to the user and suggest that it would be better to pick an alternative style template that matches the data source. By showing the result rather along with the warning, the user can see what happens when there is a mismatch and potentially learn from it.

The source data chart in Fig. 6 col d contains two quantitative data fields, but neither are monotonic. Thus, when we apply the line and area chart templates (rows 5 and 6) to this data we produce charts that can be difficult to read and interpret. However, in the line chart case we produce a valid connected scatterplot [34] where the ordering of the connections is based on an ordering data field. In all such cases we report the lack of monotonicity whenever a line or area chart template produces such a result. Here again, our warning can help users learn what happens when data points are connected in a non-monotonic order in a line or area chart.

Stylization tool for vega-lite [13]. As noted in Section 3, we have also developed a parser for converting Vega-lite specifications of common chart types into our deconstructed representation and vice versa. Vega-lite currently offers a single default visual style (with pre-selected, fonts, colors, axis thicknesses, absence of gridlines, etc.). With our parser and style transfer approach we can take a Vega-lite specification as input (Fig. 3), and apply any template D3 charts we have generated to try out different appearances for the chart (Fig. 7). In this case since we are applying the templates to a source chart rather than a data table we can use the automatic approaches for computing data importance ranks, source data types and also make use of the source attribute ordering mappings (Section 6). Similarly we can apply our deconstructor to common D3 charts, parse the resulting deconstruction into Vega-lite and apply the Vega-lite toolchain to the D3 visualization [12].

User feedback. To further understand the usefulness of our tools we showed them to seven professional data analysts who visited our lab for a visualization workshop as well as three professional journalists experienced in chart-making. The data analysts were familiar with tools like Excel and R/ggplot2 but did not go much beyond the defaults in generating charts. The journalists regularly used a variety of chart construction software including Excel, Adobe Illustrator, Tableau as well as programmatic tools like R/ggplot2 and D3. Unlike the data analysts they were experts in these construction tools.

After a brief introduction explaining the capabilities of our tools, we showed them how our tool could be used to take any data table, specify the importance of each field and immediately see the data in a variety of styles, using a set of 15 templates we had constructed earlier. We offered to let the visitors try stylization on their own datasets and four of them took us up on this offer. One of the journalists who primarily worked in D3 also asked us to convert one of his own D3 bar charts into a style template and tested the template on several of our datasets.

While the visitors gave us oral feedback throughout the demonstrations we explicitly asked them to provide qualitative feedback about our tools via a written feedback form right before they left. On a 5 point Likert scale ranging from strongly disagree to strongly agree, all of the visitors wrote that they agreed or strongly agreed that "Re-usable style templates make it quick and easy to see data in a variety of styles". They also agreed or strongly agreed that "Choosing a basic D3 chart from the Web as a style template is useful." Note that only one of the visitors tested the ability to use a D3 chart as a template, but he told us that he was satified with the results he obtained when applying the template to new data sets.

The experienced journalists did mention that a few of our results could still use a bit of tweaking (e.g., spacing gridlines, reorienting text labels, etc.) before they would be ready for publication. They were happy to learn that they could use the manual re-styling toools of Harper and Agrawala's D3 Deconstructor [14] or load the resulting chart into an SVG editor like Adobe Illustrator to perform such tweaks. Overall, they thought that the stylized charts produced by applying our templates were excellent starting points and could save them hours of time in the initial chart design stage.

While this qualititative user feedback suggests that our re-usable style templated offer useful functionality to both novice and expert users, we believe that a formal user study is an excellent direction for future work in order to fully evaluate the effectiveness of our tools.

7.1 Limitations

Although our technique for applying style templates successfully handles a variety of input charts it does have some limitations. Our approach requires a structural representation of the data, marks and mappings of a chart. While our deconstruction tool can produce this representation for SVG-based D3 charts, it is currently limited to basic chart types (e.g., variants of bar charts, scatterplots, dot plots, line charts, etc.) with linear, categorical, attribute order and text mappings. Specific limitations include:

Cannot recover non-linear functional mappings. Our deconstructor cannot recover non-linear functional mappings (e.g., logarithmic, polynomial exponential, etc.) between the data and mark attributes. Extending our deconstructor to use function fitting techniques to test whether commonly used mappings functions (e.g., log scales) produce good fits whenever the linear mapping cannot be generated is a direct next step for our work.

Cannot fully manipulate mark shape. Like Harper and Agrawala's [14] deconstructor, we parameterize the geometric attributes of marks using their bounding boxes. While this approach lets us recover mappings from data to many geometric mark attributes including position, x-scale, y-scale, and area, we cannot recover or manipulate mappings to the internal angle of a shape. More specifically, given a D3 pie chart our deconstructor cannot recover or modify the mapping between the data and the pie slice angle. Modifying our deconstructor to appropriately parameterize commonly used mark types such as pie slices is a direct extension of our work. Note that Vega-lite also does not support generation of pie charts, but if it did then we could apply our stylization tool for Vega-lite (see Section 7) to generate and apply pie chart templates.

Cannot recover algorithmic mappings. Some chart types like treemaps, jittered scatterplots and force-directed node-link graphs use complex algorithmic techniques to choose the position of marks. Our deconstructor cannot correctly recover the mapping algorithm between data the mark position attribute for these charts. Perhaps using more sophisticated program slicing and analysis techniques it would be possible to directly extract the code implementing the mapping function from the template D3 chart.

Cannot handle non-axis reference marks. As noted in Section A.1 non-axis reference marks such as legends are treated as data-encoding marks by our deconstructor breaks our style transfer process. Developing techniques for identifying such non-axis reference marks and deconstructing them separately from the main chart and its axes is an open direction for future work.

Cannot handle interaction and animation. Our style templates focus on capturing the visual appearance and structure of basic D3 charts that are static. While some D3 charts include interaction and/or animation our techniques cannot capture these dynamic aspects of the charts and therefore cannot apply them to new data sources. One challenge is to develop a declarative representation for interaction and animation. Recent work by Satyanarayan et al. [35] extends the declarative language of Vega-lite to represent certain kinds of chart interactions. Converting an interactive D3 chart into this Vega-lite interaction specification is an exciting direction for future work.

As we have noted the first two of these limitations require extending the implementation of our deconstructor in relatively direct ways and would not affect our algorithms for capturing and applying style templates. The other limitations are deeper challenges that may require new algorithmic techniques. Nevertheless, despite these limitations, our work shows that our deconstruction tool fully captures the structure of many of the most common types of basic charts.

8 Conclusion

We have presented a technique for converting a basic D3 chart into a style template and a three-stage algorithm for applying the resulting template to new data sources. Our algorithm operates on a high-level structural representation of charts. Our work shows that this representation is sufficient to capture both the style and content of a chart, and that it can be recovered by analyzing only the data and marks in a chart. Our approach let users quickly try new looks for their charts. In practice we have seen that it is much faster to apply our style templates than it is to create a chart or restyle an existing chart using common chart-making tools like Excel, Tableau, Illustrator, or D3.

Appendix: Extended Deconstruction

We detail the six extensions we make to Harper and Agrawala's [14] D3 chart deconstruction technique to recover additional chart structure.

A.1 Label Data-Encoding Marks and Axis Marks

A chart is often composed of two types of marks—*data-encoding marks* that depict the data via their visual attributes, and *reference marks*, such as axes and legends, that allow viewers to associate the visual attributes of the data-encoding marks (e.g., *xPos*, *yPos*, etc.) with specific data values. Maintaining the relationships between these two types of marks is critical for viewers to correctly read the data from a chart. However, Harper and Agrawala's deconstruction tool does not differentiate between these two mark types.

We extend their deconstruction tool to explicitly label *axis marks*, which are the most common reference marks. We use the fact that D3 groups together all of the marks comprising an axis and stores a specialized axis *scale object* with the group. In deconstruction we check whether each SVG group node has an associated scale object and if so we label all of its child SVG nodes as axis marks. We also recover two additional properties from the scale object; the orientation of the axis (*x*-axis or *y*-axis) and whether the axis is a reference for discrete data (e.g., dot plot *x*-axis) or for continuous data (e.g., dot plot *y*-axis). Finally, we examine the geometry of the axis marks to differentiate tick marks, text labels and the axis line. Note that in D3 gridlines are typically created as chart-spanning tick marks, so we label gridlines as tick marks.

After labeling all of the axis marks we label the remaining marks as data-encoding. Note that our approach labels non-axis reference marks, such as legends, as data-encoding marks. This incorrect labeling can break our style transfer process. We leave it to future work to automatically label such non-axis reference marks.

A.2 Regroup Marks to Identify Additional Mappings

In order to construct mappings Harper and Agrawala [14] first group together marks that have the same data schema. For each such group they then identify any linear or categorical functions that relate the data to the mark attributes. But this approach can over-group marks and fail to find some of the mappings. We instead start by only grouping together the marks if they have the same SVG node type (e.g., `<circle>`, `<rect>`, `<text>`). We then construct mappings for each

group independently. Finally, for each pair of groups we check whether the mappings are equivalent; that is, for each mapping in one group we check whether there is a mapping in the second group for which the data field and mark attribute match. If we find a match for all the mappings in both groups, we merge the two groups and test whether we can construct additional mappings.

For our example dot plot (Fig. 2), Harper and Agrawala group together the <circle> and <rect> marks corresponding to the dots with the <text> marks corresponding to their labels because they all share the same data schema. This grouping prevents their technique from recovering the $GPA \xrightarrow{L} yPos$ mapping since the same *GPA* value maps to two different *yPos* values (one for the dot and one for the text) and they instead recover a less useful categorical mapping $GPA \xrightarrow{C} xPos$. In contrast, our procedure only groups together the <circle> and <rect> marks while leaving the <text> marks in a separate group. It can then recover a $GPA \xrightarrow{L} yPos$ mapping for both of these groups as well as a $College \xrightarrow{C} shape$ mapping for the dots.

A.3 Construct Mark Attribute Ordering Data and Mappings

In some charts mark attributes are not related to any data field but instead form a regular ordered sequence in attribute space. For example, the *xPos* of each dot in our dot plot is regularly spaced in the image. Harper and Agrawala attempt to recover such ordering information from the SVG rendering order of the marks in the chart. However, the rendering order does not always correspond to the attribute ordering and in such cases their approach will fail to find the ordering.

Our approach for recovering such ordering information is to construct an *attribute ordering* data field O_{attr} for each mark attribute. We set the data for this field as the sort ordering index of the corresponding attribute values. For an attribute that is regularly spaced, we can then recover a linear mapping between this ordering data and the attribute values to capture the regular spacing relationship. We call such mappings *attribute order mappings*.

In our dot plot example (Fig. 2), Harper and Agrawala only recover a categorical mapping between the SVG rendering order index *deconID* and the *xPos* attribute which does not represent the regular spacing between the marks. In contrast, our approach recovers an attribute order mapping $O_{xPos} \xrightarrow{O} xPos$ that captures the regular spacing as a linear function.

A.4 Unify Replicated Data Fields

Some charts replicate the same data for two different groups of marks. In our dot plot example (Fig. 2), dots and dot labels replicate the *GPA* data field and linearly map these fields to the *yPos* attribute in slightly different ways. We unify such replicated data fields so that any change to the data field propagates to all of the mark attributes mapped by it. So, if the unified *GPA* data is changed the *yPos* for both the corresponding dot and dot label will be updated.

Our unification approach considers each pair of data fields in the chart and checks for two conditions: (1) the data field names match and (2) they contain the same data values, including repeated values (i.e., we sort the data and check that corresponding values match). Some D3 charts do not include the data field name with the data bound to the marks. In these cases Harper and Agrawala generate a data field name based on the the type of the data (number, string, or boolean). Since these data values do not have a semantic field name, we attempt to unify them with all other data fields by only performing the second check and matching the sorted data values. In our example dot plot (Fig. 2) we unify a number of fields across the different groups of data encoding and axis marks.

A.5 Extract Text Format Mappings

Charts commonly use text marks to display specific data values. For example, our dot plot (Fig. 2) labels each dot with a text mark that shows the exact *GPA* value for the dot. Given the group of label marks, Harper and Agrawala would recover a categorical mapping from the *GPA* data field to the *text* attribute of the text mark. However, because a categorical mapping is represented as a table of correspondences between unique data values and unique attribute values, it is not extensible and cannot convert new data values into attribute values. It does not capture the general functional relationship between the data and the text string attribute.

We recover a more general and extensible *text format mapping*. For each categorical mapping between a data field and a text string attribute we further check if the string version of the data value matches the corresponding *text* attribute value. If so we set the text format mapping function to simply convert the data value to a string—, i.e., string(data value). However, in some cases the data value may be related to the text string attribute value by a more complicated formatting function. For example, data representing U.S. dollars may be prefixed with the "$" symbol or postfixed with the string "dollars" when displayed as a text mark. In such cases we apply a regular expression parser on the group of text marks to recover the common prefix and/or postfix and set the text format mapping function to: *prefix + string(data value) + postfix*.

A.6 Recover Data Domain of Linear Mappings

Although a linear mapping function can be applied to any numeric input to produce an attribute value, in the context of the chart, only a limited domain of input data values produce meaningful mark attribute values. In our dot plot (Fig. 2) the $GPA \xrightarrow{L} yPos$ mapping for the dots is only meaningful over the data domain $[0.0, 4.0]$, the limits of the *y*-axis. We construct such data domains for linear mappings as follows.

For each linear mapping we first set its data domain to the min/max range of its data values. However, the resulting domain may only represent a subset of the meaningful data domain for the chart. In the dot plot, the data domain for the $GPA \xrightarrow{L} yPos$ mapping would initially range from $[2.1, 3.8]$. To recover the complete meaningful data domain we consider all linear mappings to same mark attribute (*yPos* in our example) which also have overlapping min/max data ranges. We assume that all such mappings share the same data domain and compute the domain as the union of the overlapping min/max data ranges. In our example, this approach considers the min/max data range for *yPos* mappings of the dots and of the *y*-axis axis tick marks together and thereby recovers the complete $[0.0, 4.0]$ data domain.

ACKNOWLEDGMENTS

This work was supported by an Allen Distinguished Investigator Award.

REFERENCES

[1] W. S. Cleveland, *The Elements of Graphing Data*. Monterey, CA, USA: Wadsworth Advanced Books and Software, 1985.
[2] J. Mackinlay, "Automating the design of graphical presentations of relational information," *ACM Trans. Graph.*, vol. 5, no. 2, pp. 110–141, 1986.
[3] J. Mackinlay, P. Hanrahan, and C. Stolte, "Show me: Automatic presentation for visual analysis," *IEEE Trans. Vis. Comput. Graph.*, vol. 13, no. 6, pp. 1137–1144, Nov./Dec. 2007.
[4] "WTF Visualizations," 2015. [Online]. Available: http://viz.wtf/, retrieved Mar 2016.
[5] S. R. Herring, C.-C. Chang, J. Krantzler, and B. P. Bailey, "Getting inspired!: Understanding how and why examples are used in creative design practice," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2009, pp. 87–96.
[6] B. Lee, S. Srivastava, R. Kumar, R. Brafman, and S. R. Klemmer, "Designing with interactive example galleries," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2010, pp. 2257–2266.
[7] D. Ritchie, A. A. Kejriwal, and S. R. Klemmer, "D. tour: Style-based exploration of design example galleries," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 165–174.
[8] M. Bostock, V. Ogievetsky, and J. Heer, "D^3 data-driven documents," *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 12, pp. 2301–2309, 2011.
[9] M. Bostock, "D3 Gallery," 2015. [Online]. Available: https://github.com/mbostock/d3/wiki/Gallery, retrieved Mar. 2016.
[10] I. Ros, 2015. [Online]. Available: https://bl.ocksplorer.org, retrieved Mar. 2016.
[11] C. Viau, "The Big List of D3.js Examples," 2015. [Online]. Available: http://christopheviau.com/d3list/, retrieved Mar. 2016.
[12] "Vega," 2015. [Online]. Available: https://vega.github.io/, retrieved Mar. 2016.
[13] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer, "Voyager: Exploratory analysis via faceted browsing of visualization recommendations," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 649–658, Jan. 2016.
[14] J. Harper and M. Agrawala, "Deconstructing and restyling D3 visualizations," in *Proc. 27th Annu. ACM Symp. User Interface Softw. Technol.*, 2014, pp. 253–262.
[15] J. Bertin, *Semiology of Graphics: Diagrams, Networks, Maps*. Madison, WI, USA: University of Wisconsin press, 1983.
[16] J. Fekete, "The infovis toolkit," in *Proc. IEEE Symp. Inf. Vis.*, 2004, pp. 167–174.
[17] H. Wickham, *Ggplot2: Elegant Graphics for Data Analysis*. Berlin, Germany: Springer, 2009.
[18] C. Stolte, D. Tang, and P. Hanrahan, "Polaris: A system for query, analysis, and visualization of multidimensional relational databases," *IEEE Trans. Vis. Comput. Graph.*, vol. 8, no. 1, pp. 52–65, 2002.
[19] A. Satyanarayan and J. Heer, "Lyra: An interactive visualization design environment," *Comput. Graph. Forum (Proc. EuroVis)*, 2014. [Online]. Available: http://idl.cs.washington.edu/papers/lyra
[20] Y. P. Zhou and C. L. Tan, "Hough technique for bar charts detection and recognition in document images," in *Proc. Int. Conf. Image Process.*, 2000, pp. 605–608.
[21] W. Huang, R. Liu, and C. L. Tan, "Extraction of vectorized graphical information from scientific chart images," in *Proc. 9th Int. Conf. Document Anal. Recognit.*, 2007, pp. 521–525.
[22] L. Yang, W. Huang, and C. L. Tan, "Semi-automatic ground truth generation for chart image recognition," in *Document Analysis Systems VII*. Berlin, Germany: Springer, 2006, pp. 324–335.
[23] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei , M. Agrawala, and J. Heer, "ReVision: A utomated classification, analysis and redesign of chart images," in *Proc. 24th Annu. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 393–402.
[24] N. Kong and M. Agrawala, "Graphical overlays: Using layered elements to aid chart reading," *IEEE Trans. Vis. Comput. Graph.*, vol. 18, no. 12, pp. 2631–2638, Dec. 2012.
[25] N. Kong, M. A. Hearst, and M. Agrawala, "Extracting references between text and charts via crowdsourcing," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2014, pp. 31–40.
[26] A. A. Efros and W. T. Freeman, "Image quilting for texture synthesis and transfer," in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Techn.*, 2001, pp. 341–346.
[27] E. Reinhard, M. Ashikhmin, B. Gooch, and P. Shirley, "Color transfer between images," *IEEE Comput. Graph. Appl.*, vol. 21, no. 5, pp. 34–41, Sep. 2001.
[28] F. Pitie, A. C. Kokaram, and R. Dahyot, "N-dimensional probability density function transfer and its application to color transfer," in *Proc. 10th IEEE Int. Conf. Comput. Vis.*, 2005, pp. 1434–1439.
[29] A. Hertzmann, C. E. Jacobs, N. Oliver, B. Curless, and D. H. Salesin, "Image analogies," in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Techn.*, 2001, pp. 327–340.
[30] L. Ritter, W. Li, B. Curless, M. Agrawala, and D. Salesin, "Painting with texture," in *Proc. Rendering Techn.*, 2006, pp. 371–376.
[31] J. Chen, C.-K. Tang, and J. Wang, "Noise brush: Interactive high quality image-noise separation," *ACM Trans. Graph.*, vol. 28, no. 5, 2009, Art. no. 146.
[32] K. Sunkavalli, M. K. Johnson, W. Matusik, and H. Pfister, "Multi-scale image harmonization," *ACM Trans. Graph.*, vol. 29, no. 4, 2010, Art. no. 125.
[33] R. Kumar, J. O. Talton, S. Ahmad, and S. R. Klemmer, "Bricolage: Example-based retargeting for web design," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2011, pp. 2197–2206.
[34] S. Haroz, R. Kosara, and S. Franconeri, "The connected scatterplot for presenting paired time series," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 9, pp. 2174–2186, 2016.
[35] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, "Vega-lite: A grammar of interactive graphics," *IEEE Trans. Vis. Comp. Graph. (Proc. InfoVis)*, 2017. [Online]. Available: http://idl.cs.washington.edu/papers/vega-lite
[36] G. G. Méndez, M. A. Nacenta, and S. Vandenheste, "iVoLVER: Interactive Visual Language for Visualization Extraction and Reconstruction," in *Proc. 2016 CHI Conf. Human Factors Comp. Syst.*, 2016, p. 13, doi: 10.1145/2858036.2858435.
[37] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic and H. Pfister, "Data-Driven Guides: Supporting Expressive design for information graphics," *IEEE Trans. Vis. Comput. Graph.*, no. 1, pp. 491–500, 2017.
[38] J. Brosz, M. A. Nacenta, R. Pusch, S. Carpendale, and C. Hurter, "Transmogrification: causal manipulation of visualizations," in *Proc. UIST*, 2013, pp. 97–106.
[39] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer, "Iterating between tools to create and edit visualizations," *IEEE Trans. Vis. Comput. Graph.*, no. 1, pp. 481–490, 2017.

Jonathan Harper received the graduate degree from UC Berkeley building information visualization design tools. He is a software engineer working at Strava to build tools that help athletes better understand and share their activities.

Maneesh Agrawala is a professor of computer science and director of the Brown Institute for Media Innovation at Stanford University. He works on computer graphics, human computer interaction and visualization. His focus is on investigating how cognitive design principles can be used to improve the effectiveness of audio/visual media. The goals of this work are to discover the design principles and then instantiate them in both interactive and automated design tools.