A Mixed-Initiative Approach to Reusing Infographic Charts

Weiwei Cui, Jinpeng Wang, He Huang, Yun Wang, Chin-Yew Lin, Haidong Zhang, and Dongmei Zhang

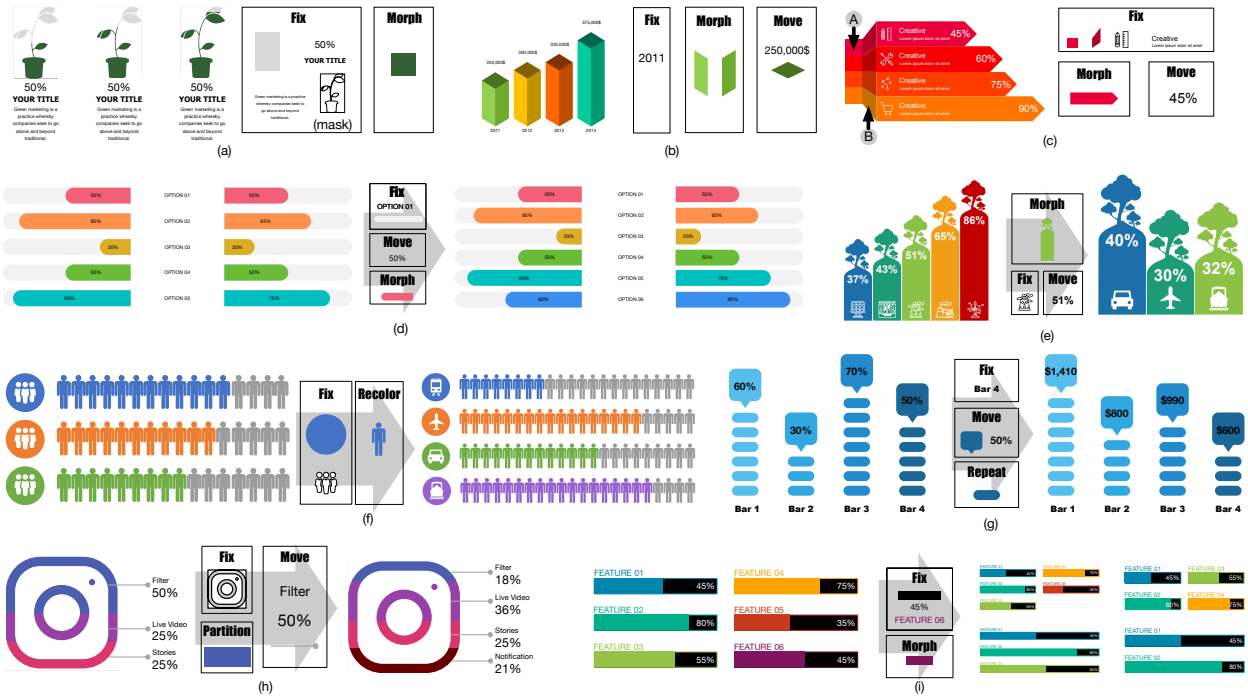

Figure 1. Examples created by our PowerPoint add-in. The left ones are examples, and the right ones are the updated results (The new colors and icons in the generated results are manually edited). (a)-(c) Three example infographics and their decomposition result. (d) The intervals of the tornado chart are shrunk to maintain the same aspect ratio. (e) The value-height mappings are altered to maintain the aspect ratio. (f) The percentages are inferred from the filled icons, and more icons are added to maintain the aspect ratio. (g) The input values are different type, and the unit value is also changed accordingly. (h) An example of stacked bars. (i) The grid layout is altered when the number of data entries is changed to maintain the same aspect ratio.

Abstract— Infographic bar charts have been widely adopted for communicating numerical information because of their attractiveness and memorability. However, these infographics are often created manually with general tools, such as PowerPoint and Adobe Illustrator, and merely composed of primitive visual elements, such as text blocks and shapes. With the absence of chart models, updating or reusing these infographics requires tedious and error-prone manual edits. In this paper, we propose a mixed-initiative approach to mitigate this pain point. On one hand, machines are adopted to perform precise and trivial operations, such as mapping numerical values to shape attributes and aligning shapes. On the other hand, we rely on humans to perform subjective and creative tasks, such as changing embellishments or approving the edits made by machines. We encapsulate our technique in a PowerPoint add-in prototype and demonstrate the effectiveness by applying our technique on a diverse set of infographic bar chart examples.

Index Terms—Infographics, Reusable templates, Graphic design, Automatic visualization.

1 Introduction

Infographics are a type of visualization that utilizes embellished versions of common charts and serves a presentation purpose [15], as they are memorable and engaging. However, unlike traditional charts that can be effortlessly created by many existing tools (e.g., Excel and Tableau), infographics are usually crafted in generic vector editors (e.g., PowerPoint and Illustrators). During an authoring process, designers need to not only consider effective visual encodings (e.g., length) of numerical information, but also the non-data encoding embellishments (e.g., icons) in the design, requiring professional skills and tremendous time, which often poses challenges to average users.

On the other hand, professional infographics are accumulated quickly in our daily life, covering many designs and information types. These examples can help us better understand the design space of infographics [9, 32]. More importantly, average users often rely on them for inspirations, which mitigate the design barrier to some extent. Users can reuse or refine a desired infographic design for their own dataset. However, these exemplars are generally stored as bitmaps or vectorized images, lacking proper abstractions to support reusing. For example, PowerPoint has many artist-designed template files that contain attractive infographic examples, which are intended for reuse. However, they are merely composed of primitive visual elements, such as text blocks and shape geometries. If users like to adapt an example to a specific dataset, they have to manually edit semantic details (e.g., shapes, labels, icons, and colors), which is still a time-consuming process.

- *W. Cui, H. Huang, Y. Wang, C. Lin, H. Zhang, D. Zhang are with Microsoft Research Asia. Emails: {weiweicu, rayhuang, wangyun, cyl, haizhang, and dongmeiz}@microsoft.com.*
- *J. Wang is with Meituan. Email: wjppku@gmail.com.*

Manuscript received 21 Mar. 2021; revised 13 June 2021; accepted 8 Aug. 2021. Date of publication 26 Oct. 2021; date of current version 22 Dec. 2021. Digital Object Identifier no. 10.1109/TVCG.2021.3114856

In this work, we introduce *Chartreuse*, a mixed-initiative approach to help users efficiently reuse infographic bar charts. Based on our observations and existing work [21, 25], there are basically two types of edits during the process. One is strict and mechanical, in which users do not have much freedom to improvise. For example, users must precisely and strictly adjust certain visual attributes, such as rectangle widths, to reflect new data values. The other type is more subjective and creative. For example, users may change labels, colors, or icons based on their personal preferences or the semantic of data. Clearly, computers are more suitable for the first type of edits, while humans are more suitable for the latter one. Thus, it is natural to combine the power of computers and humans to ensure an enjoyable and efficient user experience. Compared with directly drawing an infographic from scratch [21, 34], our approach has two advantages. First, it does not require users to have good design skills, which lowers the authoring barrier for average users. Second, by automatically converting an example into a reusable template, it allows users to quickly adapt the example to their own data, saving a lot of manual editing effort.

There are two key steps in *Chartreuse*, namely deconstruction and reconstruction. Specifically, in the first step, we focus on extracting data-binding strategies and element-placement strategies [21]. Given a primitive element in the example, we determine whether or how its visual attributes (e.g., width, height, and color) are related to underlying data, and its spatial relationships with its neighbors. Since infographic exemplars do not directly contain necessary blueprints to instruct how they are constructed, we need to analyze their visual structures to extract such information. To achieve this goal, we collect a representative dataset, analyze the design space, and build a model to predict the update strategy for each element in an infographic example. In the second step, when users provide their own dataset, we use the predicted results to manipulate visual elements in the example accordingly. When the updated infographic is less satisfying or faulty in some cases, users then can further manually refine it based on their personal opinions.

Considering the wide variety of different designs and information types, we focus and demonstrate our method on infographic bar charts in this work, since it is the most widely used in practice [3]. In addition, we further narrow down our target to vector images based on two considerations. First, although many examples are stored as bitmaps, they are practically all first created using vector graphic editors, and then converted to bitmaps for easy sharing. Therefore, even though they may not be publicly available, vector images are actually the majority in the entire infographic dataset. Second, to analyze bitmap infographics, the first step is to use computer vision techniques to identify visual objects [6, 7]. After this pre-processing step, bitmap infographics can be roughly reduced to vector-based infographics. Therefore, to isolate and exclude unnecessary noises from our method design, such as object detection errors, we decide to focus on vector representations first.

To demonstrate how *Chartreuse* can be integrated into an information worker's typical workflow, we build a proof-of-concept system by encapsulating the techniques into a PowerPoint add-in. Since PowerPoint is a widely used tool for preparing presentations and has a large collection of infographic examples, it is an ideal scenario for *Chartreuse*. Based on the user input, *Chartreuse* directly performs in-place changes to the visual elements in the slide accordingly. Since the updated result is still presented using standard PowerPoint visual elements, users can naturally refine the result using all the built-in and familiar PowerPoint operations. Therefore, our add-in can work transparently and provide an unintrusive user experience for information workers. Finally, we showcase the effectiveness by applying our add-in to a variety of infographic bar chart examples. The contributions of this work include: 1) a mixed initiated approach to reuse infographic bar charts, 2) a dataset and a study of design space of infographic bar charts, and 3) an algorithm for deconstructing and reusing examples.

2 Related Work

2.1 Infographics

Dedicated to presentation scenarios, infographics are a combination of data visualization and graphic design. Research works on infographics can be placed into two categories: understanding and generation.

Many works [45] focus on understanding visual embellishments, and try to study their pros and cons in terms of aesthetics [29], memorability [2], recognition [4], attention [16, 26], accuracy [39], layout [46], etc. Recently, some research [6, 27] has also been conducted to directly extract the semantic meanings of infographics. For example, Bylinskii et al. [6] adopt OCR techniques to assign hashtags to infographics for information retrieval purposes. Madan et al. [27] use computer vision techniques to detect icons in infographics and proposed a solution to automatically summarize infographics. Our solution is also related to parsing infographics. However, we focus on vector images and aim to help users reuse them, while theirs focus on bitmap images and try to extract high-level semantic meanings from them. Therefore, our approach is different in terms of target datasets and goals, requiring a different set of techniques.

Some other works focus on how to create infographics efficiently or even automatically. For example, numerous interactive and dedicated tools [21, 25, 33, 34, 35, 43, 47, 48] have been developed to ease the authoring process. These tools often incorporate special concepts, such as data-binding, and intelligent interactions to balance high-level visual configurations and low-level graphic element manipulations. For example, InfoNice [43] allows general users to convert standard bar charts into infographics by manipulating visual marks intuitively. Based on this work, we proposed a more detailed design space that enables successful deconstructions and reconstructions. Data-Driven-Guide [21] and Data Illustrator [25] adopt late data-bindings that help users link data values to visual attributes of graphical elements. Although these tools significantly improve the authoring experience, they are still not friendly to average users and retain several unique challenges during the entire process from conception to execution. First, these tools generally involve complicated and specialized interactions and concepts (e.g., data binding), which require a learning curve for casual users who only have occasional demands for creating such infographics. Second, users still need to manually build the designs from an empty canvas, which remains difficult and time-consuming, especially for average people who lack of design experiences. In contrast to these authoring tools, our approach more focuses on the opposite aspect. By deconstructing and reusing an existing infographic, our approach can lower the design barrier and help users build a desired infographic result efficiently.

Another category of tools targets novice users and tries to provide end-to-end solutions with the capability to automatically create infographics. For example, Cui et al. [9] propose a framework to automatically translate a statement to its infographic equivalent. Qian et al. [32] further solve the scalability issue by directly leveraging existing examples. DataShot [42] and Calliope [37] enable users to directly create a fact sheet from a raw tabular dataset. Basically, these tools rely on a predefined set of templates to generate final infographic results. Therefore, they do not scale well in terms of design diversity. Although related to our target, these approaches do not require deconstructing infographic examples, which is a core task in our work. Therefore, their techniques are not applicable in our approach.

2.2 Chart Interpretation and Deconstruction

As charts accumulate rapidly on the Internet, computational interpretation has become an important research topic. Targeting different granularities, various algorithms [8, 19, 36] have been designed to extract information including chart type, chart data, visual element, etc.

For example, ReVision [36] uses SVM models to detect chart type, then applies image processing techniques to locate marks and recover data from bar/pie chart images. Later, FigureSeer [38] extends ReVision by leveraging a CNN network and a heuristic rule to process legend information for more accurate data extraction. With a similar goal, ChartSense [19] adopts a mixed-initiative approach involving a CNN model and human interactions. In contrast to these approaches that work for a variety of chart types, some other solutions [1, 10] specifically target bar charts. Although these techniques can accurately extract data from bar chart images, they do not extract visual configurations. Therefore, they do not support the purpose of design reuse.

Compared with the above chart interpretation techniques that mainly extract data from chart images, chart deconstruction needs to further re-

cover the mappings between data and visual elements, which can enable many interesting applications, such as search [17], transformation [5], annotation [23], animation [12, 22] and restyling [13, 31].

For example, iVoLVER [28] contains a rich set of interactions that help users easily extract information data from charts in a variety of formats, such as bitmap and SVG, then allows users to edit the chart through additional widgets. Poco and Heer [30] build a multi-stage pipeline, which combines ML and heuristics techniques, to automatically extract a visual encoding specification from bitmap chart images. Chen et al. [7] propose a mask R-CNN based approach to convert a bitmap timeline infographic into an extensible template. Although bitmaps are commonly used for charts, these approaches often suffer from accuracy issues due to the complexity in chart structures or poor image quality. Therefore, in their approach, the key challenge is to successfully segment a bitmap image into a set of labeled visual elements, while our key challenge is to automatically identify various data binding strategies from a clear set of visual elements.

On the other hand, SVG-based charts are easier to process because of their programmatic representations. For example, Harper and Agrawala [13, 14] focus on the deconstruction and reuse of SVG-based charts. MobileVisFixer [44] leverages a reinforcement model to interpret and transform SVG-based charts into mobile-friendly designs. Their work enables people to intuitively restyle visualizations [13] and extract reusable templates [14]. Our work is inspired by and similar to these approaches. However, there are two key differences between their approaches and ours. First, their techniques require that the SVG files are generated via D3, since they need to leverage the `__data__` attributes programmatically attached to SVG elements, which do not exist in general SVG files. In contrast, we do not require the attached information, and aim to extract data from the visual elements directly. Second, their reusable templates are extracted from basic D3 chart types, such as column charts, line charts, and scatter plots, while our technique only focuses on bar charts but in infographic styles.

3 Dataset and Design Space

According to the dictionary, a bar chart is a diagram in which numerical values are represented by the height or length of lines or rectangles of equal width. However, there is no clear definition of infographic bar charts in previous work, so we first need to find out how much an infographic can deviate from this strict definition while it is still generally considered as a bar chart. To achieve this goal, three of the coauthors (two researchers and one software developer) independently collect distinctive examples that are qualified as infographic bar charts in their personal opinions. After that, these examples are placed together and discussed. The authors end up reaching a common understanding that there are two defining characteristics for infographic bar charts: 1) encoding values with lengths and 2) parallelizing repetitive visual units.

The first characteristic filters out charts that map values to other visual attributes, such as size (e.g., pie charts) or position (e.g., scatter plots). However, the length may not necessarily mean the height/width of a visual element, as artists may use the total height of multiple icons (Figure 2(a)). The second characteristic indicates the spatial relationship between repetitive units. Without this characteristic, charts are considered too deviated from standard bar charts, such as Nightingale rose diagrams. We do not require all units to align on the same axis, as we see some cases where designers place units in a grid or on a stair to better utilize the space (Figure 3). Please note that only single-row/column pictographs are considered valid infographic bar charts (Figure 2(a)). Those pictographs that arrange icons in grids are filtered out (Figure 2(b)), since the values are encoded using areas instead of lengths, violating the aforementioned defining characteristics.

Based on these two characteristics, we collect examples from two sources. One is a PowerPoint template archive that contains 210 files, which are randomly sampled from a variety of reputable websites [11, 40, 41]. Every file contains multiple slides with a consistent style for different types of content, such as portfolio and chart. The other source is the Visually63K [27] dataset. Since it is impractically to manually screen through all 63,738 images in the dataset, we randomly sample 1,000 examples. Based on the aforementioned criteria, we manually go through these files to extract all infographic bar charts for further study. Altogether, we obtain a collection of 438 examples [1] (325 and 113 from these two datasets, respectively) with a variety of visual designs.

Figure 2. Two pictograph layouts: (a) single-row/column layouts are considered as infographic bar charts; (b) grid layouts are not considered as bar charts, since the value is delivered using areas instead of lengths.

3.1 Methodology

We iteratively identify the main features that categorize the designs in these examples from the following aspects:

- Visual element: What types of visual elements are commonly used in infographic bar charts.
- Visual structure: What common layouts are adopted by designers for the underlying data.
- Visual encoding: How designers choose to convey the numerical information via visual attributes.

Considering the obvious repetitive patterns in bar charts, we divide the study into two steps, accordingly. In the first step, we focus on chart-level information to understand the structures of repetitive units and the roles of those visual elements that are not part of repetitive units. In the second step, we drill down into the repetitive units. Specifically, we try to anatomize the visual structure in a unit and identify the commonly used visualizations to represent numerical information. Two coauthors conduct these analyses independently. Ambiguities and disagreements are resolved through discussions.

3.2 Chart-Level Design

At the chart level, there are three key aspects that contribute to overall chart appearances, namely, *unit*, *embellishment*, and *layout*.

Unit. The dominant part of an infographic bar chart is repeated units. Each unit contains multiple visual elements, which together represent a single data item. All units generally follow a uniform visual structure. In many cases, all units have the same number of visual elements. Based on our analysis, there is only one scenario in which units may have different numbers of element. Figure 1(g) shows such an example, where quantity values are represented using pictographs.

Embellishment. In addition to the repetitive units, designers may also add embellishments for various purposes, such as highlighting and annotating. They are considered parts of charts, instead of units. During our analysis, we find that there are only 19 (4.34%) examples. Overall, the use of embellishments is uncommon based on our observations, which may be due to visual complexity management.

Unit Layout. The spatial arrangement of units is also a salient characteristic of a bar chart, which is determined by how the zero lines of all units are spatially related. In standard bar charts, rectangles can only be aligned vertically or horizontally, which are also dominant in the examples, since it is most convenient for comparison. However, we also identify several examples using special layouts, including 6 (1.37%) grid layouts (Figure 3(a)), 10 (2.28%) stacked layouts (Figure 3(b)), 11 (2.51%) tornado layouts (Figure 3(c)), and 5 (1.14%) other layouts that place units on a slope, a stair, or on a curve (Figure 3(d)).

Most of the examples have units evenly distributed, either vertically or horizontally. Only 4 (0.91%) examples have un-even distribution of units, mainly used to highlight certain units.

3.3 Unit-Level Design

Each unit corresponds to a data item in the underlying data. One item is a tuple of values that is composed of a number (e.g., a quantity or a percentage) and one or multiple texts. How to graphically encode the numerical information is the most salient feature of unit designs.

[1] The examples are included as a supplemental material of this paper.

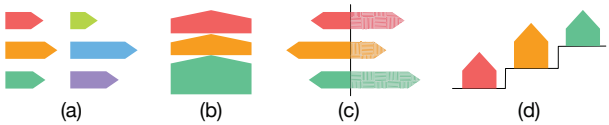

Figure 3. Examples of typical layout strategies: (a) grid, (b) stack, (c) tornado, (d) other strategies.

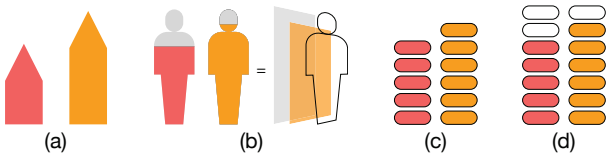

Figure 4. Encoding strategies for numerical information: (a)-(b) element length, (c) pictograph, (d) filled pictograph.

Understanding the data-binding strategies is also critical to determine how elements should be updated according to new data. As indicated in Figure 4, there are three common strategies.

Element Length. The height/width of a visual element is still the most common visual attribute encoding numerical values (Figure 4(a)). 355 (81.05%) examples adopt this strategy. Please note that we put all the filled-icon style bar charts (Figure 4(b)) into this category, because many of the filled-icon bar charts are constructed by applying clipping mask to rectangular bar charts (Figure 4(b)).

Pictograph. 20 (4.57%) examples use pictographs (Figure 4(c)). Each icon represents a unit of value, and the numerical information is conveyed by the count of icons in each chart unit.

Filled Pictograph. 63 (14.38%) examples use filled pictographs to encode numerical information (Figure 4(d)). The numerical information is delivered by the ratio of the count of icons that are filled with the foreground color to the count of total icons. Filled pictographs are similar to pictographs in terms of each icon representing a unit of value. However, they have different update operations. For the *pictograph* strategy, we need to change the count of icons when the underlying value is changed. On the other hand, for the *filled pictograph* strategy, we only need to change the colors.

3.3.1 Text Block

For visual clarity, units are often very reticent in terms of using text. Based on our observations, 434 (99.09%) examples have at most three text blocks in a unit, which include: 1) numerical information, such as 30% and $12,000, 2) a short label that only contains a couple of words, and 3) a long descriptive sentence to provide more details. The distribution of all combinations are shown in Table 1.

Text block #	Zero	One		Two			Three
Example #	8	69	34	224	32	8	55
Number		✓		✓	✓		✓
Label			✓	✓		✓	✓
Description					✓	✓	✓

Table 1. The frequencies of different combinations of text blocks. Tick marks in the same column indicate the combination of text types.

Most examples explicitly show the numerical information for clarity. For those examples without explicit numerical information, they often adopt the *filled pictograph* strategy, so that the numerical information is also easy to perceive. The label and description text blocks are key to identify data items. However, 77 (17.58%) examples have neither labels nor descriptions. For these examples, we find that the identity information is either delivered through surrounding context, which cannot be classified as part of units, or delivered using graphical elements, such as icons, in each unit.

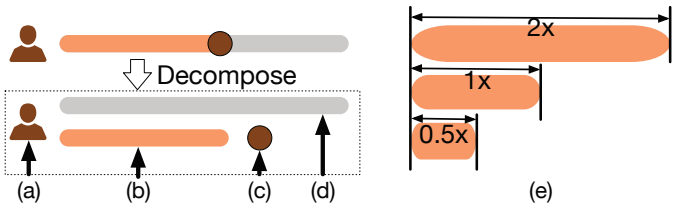

Figure 5. Update types for different elements. (a) *Fix*: invariant to underlying value (but needs to be replaced if the data semantic is changed). (b) *Morph*: need to morph if the underlying value is changed. (c)*Move*: need to move if the underlying value is changed. (d) *Fix*: invariant to underlying data. (e) A rounded rectangle scaled by different factors.

3.3.2 Graphical Element

Since a unit may visualize the number using different strategies, graphical elements also have different update operations based on user input data. We identify the following update types:

Morph. This is the most common type. The shape, such as height and width, of visual elements should reflect the data value (Figure 5(b)). For simple rectangles, we may achieve the correct effect by applying scale transformations on a square. However, in many cases of infographic bar charts, simply scaling a standard shape does not work. For example, if we scale a rounded rectangle by different factors (Figure 5(e)), the rounded corners become inconsistent. Therefore, for complicated shapes, we need a fine-grained morphing mechanism.

Move. This update type is often related to annotations in units (Figure 5(c)). Elements of this type is spatially related to the underlying value. However, in our implementation, we treat them as they have a fixed offset from another element, which is more natural and consistent with the authoring process.

Fix. Some graphical elements are invariant to the underlying numerical information (Figure 5(a)&(d)). Although many icons belong to this type and do not change their locations based on the numerical information, they provide semantic information for the underlying data (Figure 5(a)). Therefore, unlike those general shapes (Figure 5(d)), they should be replaced if the underlaying data is semantically changed. In Chartreuse implementation, we do not differentiate these two types, and rely on users to manually replace icons if necessary.

Recolor. This update type is related to the *filled pictograph* strategy. Elements of this type may flip between background and foreground colors based on underlying values (Figure 4(d)).

Repeat. This update type is related to the *pictograph* strategy (Figure 4(c)). Elements of this type are repeated in a unit to correctly represent the underlying value.

Partition. This update type is related to the stack layout (Figure 3(b)). Please note this type is similar to *morph*. However, elements of this type are stacked together in a fixed space, so they cannot be updated individually, which is different from *morph* elements.

4 Deconstruction

Given an infographic bar chart, we aim to analyze and extract its blueprint based on the aforementioned design space. Specifically, we first identify the basic repetitive units. Based on the units, we further identify various chart and unit level information, such as unit layout, data binding strategy, and update type for each element in a unit.

The input of the deconstruction step is a collection of visual elements, with all visual attributes available, such as size, location, text, font, fill color, stroke color, etc. Inspired by Beagle [3], we extract features from these elements, then train several decision tree models to automatically identify corresponding information for follow-up processing.

4.1 Unit and Sequence Detection

In this step, we aim to detect two pieces of information critical to the repetitive structure: namely *unit* and *sequence*. A unit is a set of elements that work together to illustrate a data item (Figure 6 Units 1-3). On the other hand, a sequence is a set of equivalent elements in different units (Figure 6 Sequences 1-4).

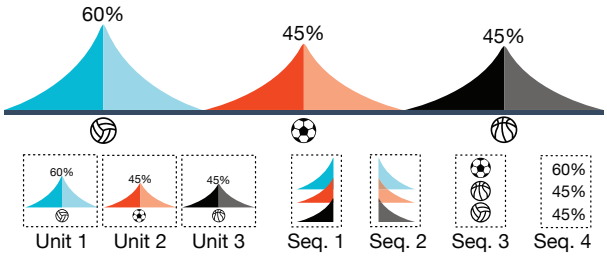

Figure 6. Example of unit and sequence detection. From the top example, Three units and four sequences are identified. Please note that the horizontal line is a chart-level embellishment and excluded.

A two-step algorithm is designed for the detection task. In the first step *clustering*, we aim to identify elements that have the same semantic role in the chart. Please note that elements of the same role do not necessarily form a sequence. For example, all icons in a pictograph (e.g., all rectangles in Figure 4(c) or (d)) have the same role, but they form multiple sequences based on their indexes in the stacks. Since elements of the same role may vary in any attributes (Figure 6), such as size, color, shapes, or text, it is impractical and inefficient to exhaust all combinations and write heuristic rules to complete this task. On the other hand, humans can easily make correct role predictions based on their observations, which make it feasible to design a machine learning algorithm. Therefore, we decide to label the examples and build a regression model on top of it to predict the pairwise similarity of elements and guide the clustering process. In the second step *assignment*, we construct units and sequences from the elements in clusters using Hungarian algorithm [24].

4.1.1 Clustering Elements

Hierarchical agglomerative clustering (HAC) [18] is adopted in this step. Initially, each element is an individual cluster. Then, clusters are iteratively merged together, until the smallest pairwise similarity is bigger than a predefined threshold of 0.65, which is determined by tuning on a validation set. Based on our experiments, we adopt the minimum strategy to measure the similarity of two clusters, which is the minimum similarity between any pair of elements from the clusters. The pairwise similarity of elements are determined based on our labeled dataset. Specifically, for a pair of elements in a training example, we manually assign a label to indicate whether they have the same role or not. Then, we use the labels to train a regression model based on Gradient Boosting Decision Tree [20] with the following features:

- Shape consistency: three `yes` or `no` labels indicating if two elements have the same width, height, and shape type, respectively.
- Color consistency: two `yes` or `no` labels indicating if two elements have the same fill and stroke colors, respectively.
- Alignment: four `yes` or `no` indicating if two elements have the same value on the left, right, top, and bottom, respectively.
- Text consistency: four `yes` or `no` labels indicating if two elements have the same text-related attributes. The first three indicate if they have the same font size, font name, and count of characters, respectively. The last one indicates if both text contents contain numerical related information, such as *n*% and $*n*.

4.1.2 Assigning Units and Sequences

By default, we assume that the most frequent cluster size indicates the number of units. Among the clusters of that size, we pick the one with the highest confidence score and take elements in it as anchors for each unit. Then, we gradually assign elements of other clusters to each anchor to grow these units. Based on our assumption about clusters, a cluster may contain multiple sequences, but a sequence can only be contained by one cluster. Therefore, by assigning elements to anchors, units can grow gradually to include more and more elements, and sequences can also be obtained at the same time.

The assignment is based on an intuition that elements within a unit should be closer to one another than elements across units. However, element-wise assignment which directly assigning an element to the nearest anchor does not work in many cases, since units in an example may overlap one another. Therefore, we model the problem as an cluster-wise assignment optimization problem, and solve it using the Hungarian algorithm [24]. Basically, we build a bipartite graphs with edges connecting the n anchor elements and m ($m \geq n$) elements of another cluster. The weight on each edge ($m \times n$ in total) is the distance between the adjacent elements. The goal is to find n edges with minimum sum of weights. Once the n edges are identified, we then verify their weights and remove the edges with outlier weights, which is to correct some error assignments caused by pictograph representations, in which some sequences may have elements fewer than n. Then, elements in the remaining edges are removed from the cluster and form a sequence accordingly. This process is repeated until all elements in all non-anchor clusters are assigned to exactly one unit and one sequence. In this algorithm, the relationships between elements are considered within a pair of clusters, but relaxing this constrain to consider more clusters at the same time may further improve performance. We leave this for exploration in future work.

4.2 Data-Binding Detection

The goal of this step is to classify all graphic elements into six update types described in the aforementioned design space: *morph*, *move*, *fix*, *recolor*, *repeat*, and *partition*. Similarly, for text elements, we have two strategies: *move* and *fix*, since the other types are all only applicable to graphical elements. Although humans can easily make such a prediction for a given element and its context, we need to summarize our intuitions and construct features over the various visual attributes of the chart, in order to train a prediction model.

Inspired by Beagle [3], we require features from three levels: element level, within-unit level, and across-unit level. Element features track visual attributes, such as fill/stroke colors, which are associated to individual visual elements. Within-unit features aim to reveal the style and attribute relationships between an element and other elements in the same unit. Across-unit features indicate relationships between an element and its equivalents in the same sequence. These features are determined based on our observations in the formative design space study, and are concatenated to help predicate the data binding strategies.

Element Features. Three features are extracted from elements:

- Element type: a `text` or `path` label indicating the element is text or graphical element.
- Fill color: a `yes` or `no` label indicating the element is filled with colors or not.
- Stroke color: a `yes` or `no` label indicating the element is outlined with colors or not.

Please note that we do not directly adopt attribute values as features, such as the actual fill or stroke colors, which we believe are irrelevant to determining the binding strategies.

Within-unit Features. Ten features are extracted to describe the structural feature of an element within a unit:

- Repetitive count: an integer indicating if the graphical element is repeated or how many times it is repeated within the same unit.
- Color consistency: two `yes` or `no` labels indicating if the element and its replicas in the same unit, if any, have the same fill and stroke colors, respectively.
- Alignment: four `yes`, `no`, or `partial` labels indicating if the element and its replicas are aligned on the left, right, top, and bottom, respectively. Please note that `partial` means not all, but only a part of the elements aligned.
- Even distribution: two `yes` or `no` labels indicating if the element and its replicas are evenly distributed horizontally or vertically, respectively.
- Overlap: a `cover`, `within`, `intersect`, `touch`, or `disjoint` label that indicates if another element in the same unit has certain spatial relationships with the element. In our implementation, the types of overlap relationships are prioritized. Thus, if multiple types are detected, the one with the highest priority is used.

Across-unit features. Eleven features describe the relationships between a graphical element and its equivalents in the same sequence.

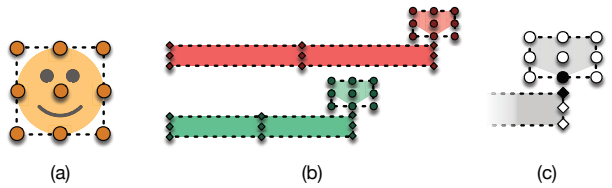

Figure 7. Identifying relative positions for *move* elements. (a) Nine reference points for a given element. (b) Compute the offsets between a *morph* element and a *move* element (81 combinations of all diamonds and circles of the same color). (c) From the pairs that are consistent in all units, choose the one with the shortest distance (the black pair).

- Geometry consistency: a `yes` or `no` label indicating if the element and its equivalents are identical in shape.
- Size consistency: two *yes* or *no* labels indicating if the element and its equivalents have the same height and width, respectively.
- Color consistency: two `yes` or `no` labels indicating if the element and its equivalents have the same fill and stroke colors.
- Alignment: four `yes`,`no`, or `partial` labels indicating if the element and its equivalents are aligned on the left, right, top, and bottom, respectively. Please note that `partial` means not all, but only part of them are aligned.
- Presence consistency: a `yes` or `no` label indicating if the element and its equivalents exist in all units.
- Uniformity: a `yes` or `no` label indicating if all units have the same number of elements. This is a chart level feature that is designed to help disambiguate type *recolor* and type *repeat*.

In total, we collect 24 features, which are used to train a multi-label classification model based on the Gradient Boosting Decision Tree [20].

4.3 Spatial Layout Detection

Layout strategies are detected after the aforementioned automatic predictions. For example, we obtain the canvas size by merging the areas of all units and the constant space between neighboring units by calculating the distance between unit borders.

4.3.1 Relative Position

As mentioned in the formal design space, for a text/graphical element that is predicted as *move*, although the movement can be formulated as a function of the input data value, we believe it is more natural to consider it having a constant offset from a *morph* element in the same unit. Therefore, we need to find the most appropriate element and reference point to define the offset. Specifically, we enumerate to all the other elements in the same unit. Given a visual element, we choose the nine most commonly used reference points (Fig 7(a)). For each candidate element, we calculate all combinations (81 in total) of reference points in the element and the *move* element (Figure 7(b)). Then, we repeat the process in all units, and choose all reference pairs that have consistent offsets in all units. Finally, among all valid reference pairs, we select the pair with shortest distance, then use it to guide the movement when the underlying data are changed (Figure 7(c)).

4.3.2 Zero Line

It is critical to identify the zero line of a unit, since it affects how *morph*, *repeat*, *partition*, and *recolor* elements update according to underlying data. First, we determine the layout of the chart (Figure 3), such as *basic*, *grid*, *stack*, *tornado*, and *other*, based on the spatial arrangement of units. Then we determine zero lines using heuristic rules.

Specifically, for *morph* and *partition* elements, we infer their zero lines based on their "growth" direction. For *repeat* elements, we infer their zero lines from the counts of sequences that are composed of *repeat* elements and the spatial relationships between the sequences. To determine the zero lines for *recolor* elements, it is critical to differentiate the foreground and background colors. Therefore, we first try to identify two distinctive colors within individual units. Then if one color is shared by all units (e.g., the gray color in Figure 1(f)) while the other colors are different from unit to unit, we identify the shared color as the background color. In cases when the other colors are also identical in all units, we then try to find if any text elements contain numerical information, and use such information to infer the foreground color. However, if the numerical information is absent, we stop processing the case, since the foreground and background are indistinguishable.

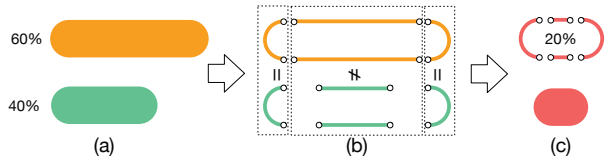

Figure 8. Example of morphing rounded rectangles. (a) The *morph* elements and their numerical information in an example. (b) The path segments extracted from the morphing examples and the variant and invariant segments. (c) The variant segments are adjusted to reflect the new numerical information.

5 Reconstruction

After the deconstruction, we have all the information required to manipulate the visual elements in the infographic bar charts. In a typical reconstruction process, we first recover data from a given chart. Users can operate the data, such as changing values, adding, or removing values, then our algorithm can detect the changes and update visual elements in the chart accordingly.

5.1 Data Recovery

The underlying data in an infographic bar chart is formulated as a list of tuples in the form of `(Text`$_1$`, Text`$_2$`, ⋯, Number)`. The number of texts is determined by the visual structure in the chart, which can be zero, one, or more. First, we enumerate all the text elements in each unit, and extract the corresponding texts. Then we use regular expression to identify numerical contents, such as `##%` and `$#.##`. Currently, we assume that each unit has exactly one number to be graphically represented. If numerical texts are identified, they are placed in the number segment of the tuple, otherwise, they are placed in the text segment of the tuple. However, if the numerical information is not located, we then check whether the chart uses *filled pictograph* strategy and try to infer the information from the ratio of the count of filled icons to the count of total icons.

5.2 Value Update

As described in our formative design study, there are essentially three types of data-binding strategy: *element length*, *pictograph*, and *filled pictograph*, that are related to six different update types.

Element length. In this strategy, we update the geometry of a *morph* element to make its height or width proportional to the underlying value. We assume there is a linear mapping between the value and length, which follows the form of $l = a \cdot v + b$, where l and d denote lengths and values, respectively. We use the minimum and maximum values in the recovered data to identify the parameter a and b. Then, given a new value v, we can use the function to determine the corresponding length l, which is further used to create a new *morph* element.

Directly applying scale transformation to an existing element may cause strange distortions (Figure 5(e)). Therefore, we adopt a segment-based approach (Figure 8). First, we collect the *morph* elements from the same sequence, and break each of them into a list of segments, such as straight lines, cubic curves, and arcs. Then, we compare all the segments at the same index of the list to identify the different ones, which are responsible for the length differences between different shapes (Figure 8(b)). Finally, we only alter these segments to achieve the desired length. This fine-grained approach can help us maintain a consistent visual style among all *morph* elements (Figure 8(c)).

Pictograph. For this binding strategy, we use the maximum value and the count of *repeat* elements in the corresponding unit to estimate the unit of value. Then, we use the unit of value to compute the number of *repeat* elements required by the given value, and then stacked them together to represent the value.

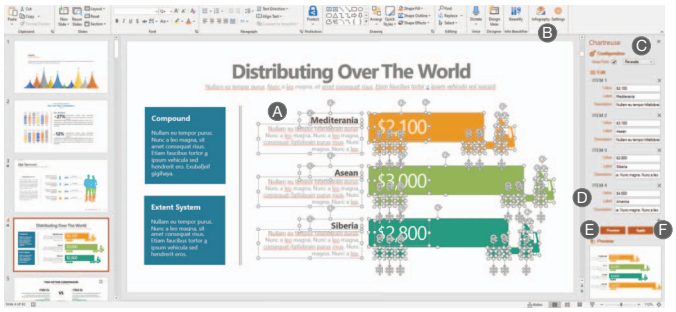

Figure 9. Screen shot of PowerPoint add-in. A: Select the elements of a chart. B: Click the button to analyze the chart. C: Side panel shows the recovered result for further editing. D: Adding a new data item. E: Click the *preview* button to preview the updated chart below. F: Click the *apply* button to replace the selection with the updated chart.

Filled pictograph. Since the color encoding is always applied to percentage value between 0% and 100%. Therefore, given the total number of *recolor* elements in a unit, we can easily calculate how many of them need to be filled with the foreground color.

For other types, such as *fix*, *stack*, and *move*, elements are straightforward and easy to update given the information collected from the deconstruction step.

5.3 Unit Addition and Deletion

Another key operation for reconstruction is to add or delete data entries. For example, when users encounter a good example and like to reuse it for another dataset, sometimes the new data has a slightly different number of items. Therefore, it is necessary to adjust the layout to accommodate the changes in item count. One straight way is to maintain the alignment and spacing between units when adding/deleting units. The advantage is that it can best keep the original style. However, in some cases, such as in a PowerPoint slide where contents are often elaboratively arranged, charts may have a fixed space to occupy. Although we can uniformly scale the new chart to fit it into the space, the aspect ratio of the example chart is unfortunately not maintained. Therefore, we also design alternative methods to rearrange units to maintain the aspect ratio of the chart when adding or removing units.

Basic Layout. When units are sequentially placed, one straightforward way is to expand/shrink the unit intervals to maintain the overall aspect ratio (Figure 1(d)). In addition, for some cases where *morph* elements are used, we can alter the length-value mapping function to eventually maintain the overall aspect ratio. Specifically, we first align and space units based on the extracted information from the example chart. Once the length required along the zero line is determined, we extend or shrink the canvas space along the unit-growth direction to make the new aspect ratio equals to the one of the example chart. Similarly, for *pictograph* and *filled pictograph* charts, we can change the unit value to make more or fewer icons to maintain the aspect ratio (Figure 1(f)). Then, we can uniformly scale the new chart to fit it into the designated space while maintaining the original aspect ratio.

Grid Layout. We first split the chart into multiple sub-charts that follow the basic layout. Then we can handle the sub-charts as the ones with basic layouts. However, when adding or removing entries, we need to consider the balance between sub-charts. For example, it is preferred to add a unit to each sub-chart or add an entire sub-chart when adding multiple units. On the other hand, when the units cannot be evenly distributed among sub-charts, the alignments between units from different sub-charts should be maintained (Figure 1(i)).

However, these strategies all may result in bad layouts if inappropriate data are added/removed. Therefore, in our prototype, we offer users with options to preview the result before actually apply them.

6 PowerPoint Add-in Prototype

To demonstrate our mixed-initiative approach, we create a PowerPoint add-in to help users adapt exemplar infographics to their data easily, as the PowerPoint template dataset is the largest available one that contains vectorized infographic bar charts. Although our mixed-initiative approach involves users in the reusing process, we strive to avoid unfamiliar interactions or context switch to ensure a friendly and uninterrupted user experience.

Figure 9 shows a screen shot of our add-in in use. In a typical workflow, users need to select all the elements that belong to an infographic bar chart (Figure 9A), then click a button (Figure 9B). Our back-end service analyzes the elements, and shows the extracted numbers and texts in a side panel (Figure 9C) in a list of text boxes. Then, users can directly modify the values, add new data entries, or remove old data entries in the panel. For example, users can add one more data items to the chart (Figure 9D). When users finish editing, they can optionally click the *preview* button to preview the result (Figure 9E) in the side panel, or directly click the *apply* to replace the selection with the updated chart (Figure 9F).

One advantage of our prototype is that the new chart is still composed of the built-in shapes or text blocks. Users can easily follow up to directly refine these elements afterward, such as adding highlights or annotations, to make the chart more suitable for their scenarios. In addition, our update is non-destructive, which means that, if users do not like the new chart, they can directly click the global *undo* button to revert the slide back to its original state.

7 Evaluation

In this section, we briefly report experimental results for the methods used in the deconstruction stage, then demonstrate the performance of our PowerPoint add-in via a variety of examples.

7.1 Model Performance

There are three related algorithms used in our system. First, a clustering algorithm is used to group elements based on a regression model. Then, based on the clustering results, an assignment algorithm is designed to detect units and sequences. Finally, a multi-label prediction model is trained to determine the update type for each visual element. Since our models requires detailed element information, we only use the PowerPoint examples (325 in total) to conduct the experiments.

We first evaluate the regression model which predicts the similarity between two elements. We randomly shuffle and split the dataset into training (60%), development (10%) and test (30%) subsets. The regression model is trained on the training set, tuned on the development set and achieved a good discriminability with 0.967 in terms of Area Under the ROC Curve (AUC) on the testing set. Our post-experiment analysis shows that errors are generally caused by indistinguishable shapes. For example, in Figure 1(i), the background and foreground rectangle tend to be predicted to have the same role, because they share a large proportion of common features. Element pairs in Figure 6 Sequence 3 got low confidences as they has different shape id sequences. Fortunately, our strategy in the assignment step can alleviate these problems by assigning elements to correct anchors.

To measure the overall performance of unit detection, Intersection over Union (IoU) is adopted, which is the count of element intersection divided by the count of element union between the ground truth unit and predicted unit. Overall, our end-to-end method (i.e., concatenating our clustering and assignment steps) reaches a score of 0.926 in terms of IoU. To better understand the contributions of individual steps to the detection results, we perform an ablated setting by substituting them with alternative methods. Specifically, if we directly assign elements to the nearest anchor element, instead of leveraging the Hungarian algorithm, the IoU score drop significantly to 0.784. In addition, if the ground truth of clustering results are directly fed to the assignment algorithm, the IoU score just increases slightly to 0.960. These experiment scores show that, our clustering method has achieved a relatively good performance, while the the Hungarian algorithm based method has a significant improvement over the nearest-anchor method.

Finally, we train our multi-label model on the same 325 examples. Specifically, we also randomly shuffle and split the dataset into 70% of training and 30% of test subsets. In addition, for the within-unit and across-unit features, the ground truth of units and sequences is adopted to eliminate noise for the training. Overall, we achieve a score of 0.977

in terms of multi-label AUC. Our post-experiment analysis shows that errors are generally caused by shape alignments. For example, in some cases, elements are not precisely aligned or distributed due to mistakes made by designers. To mitigate this situation, we allow a certain tolerance (6 in our implementation) when checking two elements are aligned. In addition, our model also fails when some charts do not meet our assumptions by design. For example, designer may deliberately misalign coin icons vertically to deliver realistic appearances, which cannot be corrected processed by our models.

7.2 Example Results

Figure 1 presents a variety of infographics created with our add-in. Specifically, Figures 1(a)-(c) show example infographics and their deconstruction results. For example, Figure 1(a) shows how the filled icon effects are uniformly achieved using clipping masks. However, we also find an example where designers directly used clipped shapes to achieve the same effect, which Chartreuse fails to detect correctly due to shape inconsistency. In Figure 1(b), Chartreuse correctly decomposes the cubes into three shapes of different types. Figure 1(c) shows an example that Chartreuse only supports value updates. Since this chart has two special sequences (A and B in Figure 1) that do not have consistent offsets, Chartreuse cannot correctly update the charts when data items are added or removed. Figures 1(d)-(i) show results when changing underlying data, where the left one is the example and the right is the updated result. These examples demonstrate different strategies to maintain the aspect ratio after adding/removing data entries: shrinking/expanding intervals (Figure 1(d)), adding/removing pictographs (Figure 1(f)) changing the value-data mapping function (Figure 1(e)), changing the grid layout (Figure 1(i)).

During our experiments, we find that, in many cases where the charts show percentage information, there are often background bars used to represent the 100% values to provide context information (Figure 1(a), (d), and (i)). However, designers may not create foreground bars precisely based on the percentage values. Sometimes, even arbitrary values are used just for illustration purposes (Figure 1(a)). For these obvious errors, we directly calculate the correct length for foreground elements based on the percentage and the background element, instead of relying on the mapping between percentages and foreground elements.

7.3 User Feedback

To evaluate the pros and cons of our mixed-initiative approach, we conducted an in-lab user study with 10 users. Given that there are no other works that allow to easily reuse existing infographics, we did not quantitatively compare our system with other systems or vanilla PowerPoint. Rather, we asked our participants to edit examples with and without our add-in and evaluate it from the following aspects:

- whether our approach can accelerate the reusing process;
- whether the add-in can improve the reusing experience;
- the pain points in our current implementation.

Participants and Procedure. We recruited 10 participants (aged between 21 to 43, mean = 29, three females) through recruitment messages posted on social media platforms and word-of-mouth. Six have experiences of working with infographics. Specifically, two are professional designers in a technology company, having around ten years of design experience in user interface, graphics, and video. All participants are regular PowerPoint users, familiar with common PowerPoint interactions. And three claim to be heavy users of PowerPoint.

For each participant, we first spent 10 minutes to introduce the background and user interface. Then we provided 12 infographic examples for him or her to freely explore. During the exploration, we adopted the think-aloud protocol to keep track of their thoughts and attentions. After the exploration, we asked the participants for their personal opinions on the aforementioned and general feedback. Each session lasted about half an hour.

Findings. Overall, the feedback from participants confirmed the legitimacy of our approach. However, they also identified some inconveniences and provided valuable suggestions for future improvement.

All participants admitted that the fully manual editing experience is laborious and painful. Specifically, they all commented that the biggest pain point is to specify the data-encoding attributes of visual elements. Our observation of their operations also echos their comments. The time used to simply update the numerical information in an example is almost negligible using our add-in, as it only requires users to provide a value in an input field. On the other hand, to precisely set an element's width or height based on a numerical value without using our add-in, users typically need to first compute the attribute value, open the property dialog of that element in PowerPoint, and then input the value, which took most of the time of their editings. Therefore, all participants highly praise the ability of our add-in to detect data-binding rules and adjust attribute values automatically. Since PowerPoint natively supports reference lines and alignment features to assist users in placing elements, the participants can manage to adjust layout relatively easily. However, they still appreciated that our add-in can help them with these tasks instantly. For those edits left for themselves to perform, the participants expressed their understandings of the challenges after their free explorations. One commented that "now I understand why it is designed this way. Although I like it to be fully automatic, but its features along already improved the efficiency by orders of magnitude."

The smooth user experience is also well recognized by the participants. Their feedback indicates that the learning curve is low due to the straightforward user interface and familiar interactions. In particular, they highlight the integrated experience and the smooth transition between automatic edits and manual edits. One designer commented that "the experience is seamlessly integrated into people's normal workflow, so users can easily switch between different tasks." Another advantage is that our add-in performs direct edits on visual elements in a slide, which is critical according to the designer, who also claimed "the tool is clearly time-saving. But if I am not allowed to edit the result, I am probably not going to use the tool." However, he also suggest that the entrance point of our approach is less obvious. Currently, we require users to manually select all visual elements in the same infographic and then press a button to explicitly invoke our editing panel. He suggested that we consider to promptly popup the panel when users manually editing a visual element in the infographic, which is likely a clear signal that they like to update the infographic.

We also receive suggestions that implied further necessary improvements to our approach. One biggest request is to make more intelligent and robust edits to further save users' effort. During the free explorations, the participants tested many extreme cases that produce strange results, such as too long rectangles or overlapping texts. Although they can be further fixed during the manual editing step, it still requires unnecessary workload from users. Therefore, it is suggested to infer more rigorously constraints from the example to avoid such strange layouts or visual encodings. In addition, our current system does not consider the semantic of user input. Therefore, many edits related to semantics, such as changing icons, still rely on humans. The participants would love to see our system can make related suggestions. Finally, they suggested that we make a centralized infographic library, so that they can search or browse to find desired exemplars in that library, instead of being restricted to the ones that can be found by themselves.

8 Discussion

8.1 Democratizing Infographics

Infographic design has drawn much attention recently, due to their increasing popularity in practice. Many ideas and prototype systems, such as DDG [21] and InfoNice [43], have been developed to alleviate the pain of infographic authoring with intelligent features, such as data-visual binding. However, almost all existing infographic designs are "unintelligent", and these designs will keep accumulating in the foreseeable future. With the absence of chart models, these designs are hard to reuse and adopt by general users, hence hindering the democratization of infographics.

Chartreuse is proposed to bridge this gap. Specifically, it has two unique benefits. First, Chartreuse is not intrusive to designers. Designers can still use any tools they are familiar with to create infographic designs. Chartreuse works transparently and tries to recover the chart model after charts are created. Therefore, there is no extra workload or learning curve to designers. Second, Chartreuse can easily blend in

with mainstream software, such as PowerPoint. To democratize infographics, it is a key to make infographics more accessible in people's daily life. Unlike standard charts, infographics are still not innately supported by mainstream software, hence it is hard to attract users to take the first step to try infographics. By turning existing examples into reusable templates without forcing users to leave their familiar software, Chartreuse can serve as a temporary solution to provide a seamless experience for general users to leverage infographic effortlessly, and eventually grow the user base of infographics.

However, our reuse approach is not proposed to replace the authoring approach. On the contrary, we believe they will co-exist/evolve and benefit one another. On one hand, the examples for reuse still heavily rely on professionals and authoring tools, which are clearly more powerful and capable. On the other hand, our reuse model can also be integrated with existing authoring tools to help designers quickly test and generate different variations of the same design. For example, a designer may first manually create a couple of repetitive units, our model then will detect the structure and help him/her quickly create more units in different layouts.

8.2 Mixed-Initiative Approach

Our PowerPoint add-in is designed to be a mixed-initiative approach, where human judgments play an vital role in the reuse process. First, users need to make the decision of which exemplar to reuse based on their judgments. Some charts have specific semantics. For example, Figure 1(e) is more related to environment than to business. Charts may be suitable for different numbers of data items or different lengths of text description. Users need to put all these factors into consideration and pick the most appropriate one for their own dataset. Otherwise, the adaptation result is likely to be strange or erroneous. Therefore, our add-in relies on users to control the overall visual quality, and a good adaption result is a joint effort of human and machine intelligences. However, we think this is rather reasonable and easy for users, since humans are good at making subjective judgments. On the other hand, Chartreuse is efficient at extracting the data-visual binding strategy and adapting the exemplar to a new dataset, which is often time-consuming and error-prone to users. When users provide their own data in the side panel, we also expect them to manage the output quality. For example, some text blocks in a chart are designed to hold short descriptions, while others are designed to hold long descriptions (Figure 1).

The exemplar reuse scenario is inevitably a human-in-the-loop process, as users need to pick the exemplar, provide new data, and approve the result. Therefore, we decide to automate the labor-intensive parts with algorithms and leave the subjective parts to human decisions. However, we also try not to interrupt users with unfamiliar operations. Therefore, the interactions adopted by the add-in are all basic ones, such as selecting and filling a form. In addition, the updated chart can be reverted by clicking the universal undo button, which also places users in a comfort zone with our tool. Furthermore, the updated chart is still composed of shapes and text blocks that users are familiar with in PowerPoint. So that users can easily perform refinement afterwards, such as changing colors or font size.

8.3 Failure Cases, Limitations, and Reflections

Currently, our algorithms work best for units that have consistent visual structures. Although the majority of the collected examples fall into this category, we also find some special and artistic designs that complicate unit structures and lead to element updates. For example, in a case where man and woman icons are interleaved in pictographs, Chartreuse fails to recognize they have the same meaning. In addition, we assume that elements in a series have the same shape. However, we find some examples with elements tilted differently to achieve a 3D effect (Figure 1(c)). Chartreuse can correctly identify the unit structures and help users update their content. However, it is difficult to mimic the effect when adding or removing data entries. In some other examples, the zero lines are skewed to achieve a 3D effect, which will lead to incorrect value-length mapping detection. Unfortunately, for these certain failure cases, users cannot benefit from our current implementation. However, since the modifications made by our add-in are nondestructive and pushed to the undo stack of PowerPoint, users can simply click the undo button to revert the design to its original state.

In other cases, defects may also happen by applying extreme values or adding/removing inappropriate numbers of data items. For example, bars in Figure 1(d) will overlap if we add more items. In addition to inappropriate inputs, internal mistakes, such as incorrect predictions, may also lead to incorrect modifications. To recover from these partially failed cases, users can directly edit the result to correct minor defects. In addition, we may also expose in the future more configurations on the user interface, so that users can easily override internal settings to help our model recover from failures that are caused by internal errors.

Currently, chart-level elements are not managed by Chartreuse. For example, the y-axis tick labels or peak highlights will not be updated based on new input data. By default, Chartreuse simply leaves these elements as they are after the adaption. This will result in erroneous results and require users to manually correct them after the adaption. We believe these chart-level embellishments are essential to a good chart reuse experience, but they are anecdotal in our examples and we could not find a systematic solution for them. We plan to collect more examples and address this issue in the future.

Last but not least, our current proof-of-concept implementation only works for infographic bar charts. However, our approach contains multiple components, and it is possible to extend or generalize some of them to a broader range of infographics. For example, design spaces of different types of infographic may overlap, as they often share similar visual elements. Therefore, our design space may serve as a starting point to be expanded for more infographic types. In addition, our label-prediction models can also be directly used for other infographic types if data are labeled properly. However, the main challenge for generalization is the huge amount of human effect involved, such as design space analysis and data labeling, making our approach hard to scale out. Thus, we believe it is desired and critical to work out a more generic learning-based model to handle the variety of infographic types, which will be explored in our future work.

8.4 Copyright Issue

Copyright is a concern for our approach, as it is illegal to duplicate an example without owning proper copyrights. There are several solutions to deploy our technique in practice. First, the simplest way is for users to directly provide an example with a correct copyright for adaption, which is common in the PowerPoint scenario, as there are many dedicated websites [41] that host free or licensed example files for people to explore or purchase. Another promising solution is to provide more editing features to allow users to customize the final appearances. According to our design experts, it is common for them to reuse a part of the design of an example, and revise it with personalized touches, such as colors, shading, textures [32]. Therefore, this approach is more aligned with the common practice. For users who do not have sufficient skills to customize the final design, we may further explore techniques to combine designs from multiple examples.

9 Conclusion and Future Work

In this paper, we introduce *Chartreuse*, a mixed-initiative approach to reusing infographics. Chartreuse takes a vectorized infographic chart as an input and extracts a chart model from it to help users quickly adapt it to a new dataset. On top of this technique, we build a PowerPoint add-in to help users seamlessly reuse existing PowerPoint templates of infographic bar chart, and demonstrate the expressiveness of Chartreuse with a variety of examples.

There are several avenues that are promising for future work. First of all, Chartreuse only works on infographic bar charts, but we would like to explore and support more types of infographic types, making our tool more versatile and general. In addition, we believe there are some steps in this paradigm that can be further automated. For example, since we have the collection of exemplars, we can try to recommend example candidates to users to save users time scouting ones blindly. Finally, we are also interested in the semantic and visual structure at a bigger scope, such as slide level or even poster level, which can certainly provide more intelligent assistance to the content creation process.

REFERENCES

[1] R. A. Al-Zaidy and C. L. Giles. Automatic extraction of data from bar charts. In *Proceedings of the 8th international conference on knowledge capture*, pages 1–4, 2015.

[2] S. Bateman, R. L. Mandryk, C. Gutwin, A. Genest, D. McDine, and C. Brooks. Useful junk?: the effects of visual embellishment on comprehension and memorability of charts. In *In Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 2573–2582. ACM, 2010.

[3] L. Battle, P. Duan, Z. Miranda, D. Mukusheva, R. Chang, and M. Stonebraker. Beagle: Automated extraction and interpretation of visualizations from the web. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2018.

[4] M. A. Borkin, Z. Bylinskii, N. W. Kim, C. M. Bainbridge, C. S. Yeh, D. Borkin, H. Pfister, and A. Oliva. Beyond memorability: Visualization recognition and recall. *IEEE transactions on visualization and computer graphics*, 22(1):519–528, 2016.

[5] J. Brosz, M. A. Nacenta, R. Pusch, S. Carpendale, and C. Hurter. Transmogrification: causal manipulation of visualizations. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 97–106, 2013.

[6] Z. Bylinskii, S. Alsheikh, S. Madan, A. Recasens, K. Zhong, H. Pfister, F. Durand, and A. Oliva. Understanding infographics through textual and visual tag prediction. *arXiv preprint arXiv:1709.09215*, 2017.

[7] Z. Chen, Y. Wang, Q. Wang, Y. Wang, and H. Qu. Towards automated infographic design: Deep learning-based auto-extraction of extensible timeline. *IEEE transactions on visualization and computer graphics*, 26(1):917–926, 2019.

[8] J. Choi, S. Jung, D. G. Park, J. Choo, and N. Elmqvist. Visualizing for the non-visual: Enabling the visually impaired to use visualization. In *Computer Graphics Forum*, volume 38, pages 249–260. Wiley Online Library, 2019.

[9] W. Cui, X. Zhang, Y. Wang, H. Huang, B. Chen, L. Fang, H. Zhang, J.-G. Lou, and D. Zhang. Text-to-Viz: Automatic generation of infographics from proportion-related natural language statements. *IEEE transactions on visualization and computer graphics*, 26(1):906–916, 2019.

[10] S. Elzer, S. Carberry, and I. Zukerman. The automated understanding of simple bar charts. *Artificial Intelligence*, 175(2):526–555, 2011.

[11] Free power point templates. `https://www.free-power-point-templates.com/`. Accessed: 2021-06-20.

[12] T. Ge, Y. Zhao, B. Lee, D. Ren, B. Chen, and Y. Wang. Canis: A high-level language for data-driven chart animations. In *Computer Graphics Forum*, volume 39, pages 607–617. Wiley Online Library, 2020.

[13] J. Harper and M. Agrawala. Deconstructing and restyling d3 visualizations. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*, pages 253–262, 2014.

[14] 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.

[15] R. L. Harris. *Information graphics: A comprehensive illustrated reference*. Oxford University Press, 2000.

[16] L. Harrison, K. Reinecke, and R. Chang. Infographic aesthetics: Designing for the first impression. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 1187–1190. ACM, 2015.

[17] 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.

[18] S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[19] 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 CHI Conference on Human Factors in Computing Systems*, pages 6706–6717, 2017.

[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in neural information processing systems*, pages 3146–3154, 2017.

[21] N. W. Kim, E. Schweickart, Z. Liu, M. Dontcheva, W. Li, J. Popovic, and H. Pfister. Data-driven guides: Supporting expressive design for information graphics. *IEEE transactions on visualization and computer graphics*, 23(1):491–500, 2017.

[22] Y. Kim and J. Heer. Gemini: A grammar and recommender system for animated transitions in statistical graphics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):485–494, 2020.

[23] N. Kong and M. Agrawala. Graphical overlays: Using layered elements to aid chart reading. *IEEE transactions on visualization and computer graphics*, 18(12):2631–2638, 2012.

[24] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[25] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data illustrator: Augmenting vector design tools with lazy data binding for expressive visualization authoring. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, page 123. ACM, 2018.

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

[27] S. Madan, Z. Bylinskii, M. Tancik, A. Recasens, K. Zhong, S. Alsheikh, H. Pfister, A. Oliva, and F. Durand. Synthetically trained icon proposals for parsing and summarizing infographics. *arXiv preprint arXiv:1807.10441*, 2018.

[28] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 4073–4085. ACM, 2016.

[29] A. V. Moere and H. Purchase. On the role of design in information visualization. *Information Visualization*, 10(4):356–371, 2011.

[30] J. Poco and J. Heer. Reverse-engineering visualizations: Recovering visual encodings from chart images. In *Computer Graphics Forum*, volume 36, pages 353–363. Wiley Online Library, 2017.

[31] 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.

[32] C. Qian, S. Sun, W. Cui, J.-G. Lou, H. Zhang, and D. Zhang. Retrieve-Then-Adapt: Example-based automatic generation for proportion-related infographics. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):443–452, 2020.

[33] D. Ren, T. Höllerer, and X. Yuan. iVisDesigner: Expressive interactive design of information visualizations. *IEEE transactions on visualization and computer graphics*, 20(12):2092–2101, 2014.

[34] D. Ren, B. Lee, and M. Brehmer. Charticulator: Interactive construction of bespoke chart layouts. *IEEE transactions on visualization and computer graphics*, 25(1):789–799, 2019.

[35] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. In *Computer Graphics Forum*, volume 33, pages 351–360. Wiley Online Library, 2014.

[36] 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*, pages 393–402, 2011.

[37] D. Shi, X. Xu, F. Sun, Y. Shi, and N. Cao. Calliope: Automatic visual data story generation from a spreadsheet. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):453–463, 2020.

[38] N. Siegel, Z. Horvitz, R. Levin, S. Divvala, and A. Farhadi. FigureSeer: Parsing result-figures in research papers. In *European Conference on Computer Vision*, pages 664–680. Springer, 2016.

[39] D. Skau and R. Kosara. Readability and precision in pictorial bar

charts. In *Proceedings of the Eurographics/IEEE VGTC Conference on Visualization: Short Papers*, pages 91–95. Eurographics Association, 2017.

[40] Slides carnival. `https://www.slidescarnival.com/`. Accessed: 2021-06-20.

[41] Slidesgo. `https://slidesgo.com/`. Accessed: 2021-06-20.

[42] Y. Wang, Z. Sun, H. Zhang, W. Cui, K. Xu, X. Ma, and D. Zhang. DataShot: Automatic generation of fact sheets from tabular data. *IEEE Transactions on Visualization and Computer Graphics*, 26(1):895–905, 2019.

[43] Y. Wang, H. Zhang, H. Huang, X. Chen, Q. Yin, Z. Hou, D. Zhang, Q. Luo, and H. Qu. Infonice: Easy creation of information graphics. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, page 335. ACM, 2018.

[44] A. Wu, W. Tong, T. Dwyer, B. Lee, P. Isenberg, and H. Qu. Mobilevisfixer: Tailoring web visualizations for mobile phones leveraging an explainable reinforcement learning framework. *IEEE Transactions on Visualization and Computer Graphics*, 27(2):464–474, 2020.

[45] A. Wu, Y. Wang, X. Shu, D. Moritz, W. Cui, H. Zhang, D. Zhang, and H. Qu. Survey on artificial intelligence approaches for visualization data. *arXiv preprint arXiv:2102.01330*, 2021.

[46] A. Wu, L. Xie, B. Lee, Y. Wang, W. Cui, and H. Qu. Learning to automate chart layout configurations using crowdsourced paired comparison. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, New York, NY, USA, 2021. Association for Computing Machinery.

[47] H. Xia, N. Henry Riche, F. Chevalier, B. De Araujo, and D. Wigdor. Dataink: Direct and creative data-oriented drawing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, page 223. ACM, 2018.

[48] J. E. Zhang, N. Sultanum, A. Bezerianos, and F. Chevalier. Dataquilt: Extracting visual elements from images to craft pictorial visualizations. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2020.