Charticulator: Interactive Construction of Bespoke Chart Layouts

Donghao Ren, Bongshin Lee, and Matthew Brehmer

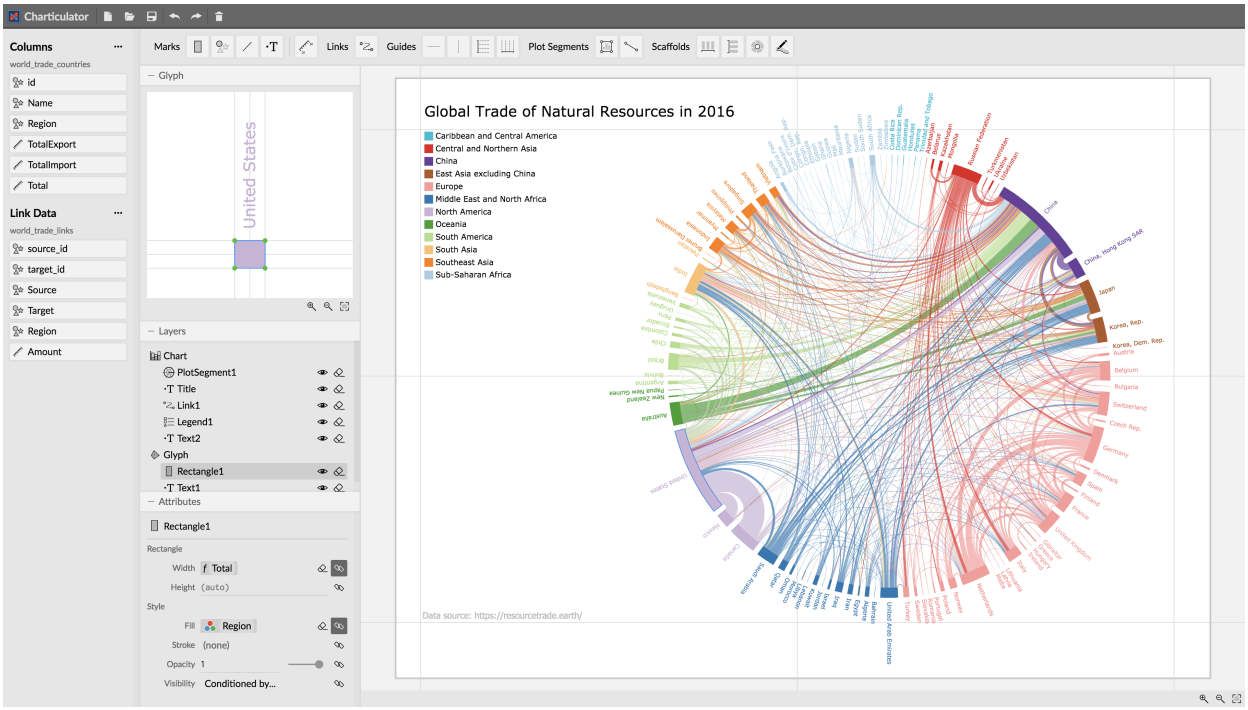


Fig. 1. Charticulator enables a chart designer to interactively create a custom chart layout. It transforms a chart specification into mathematical layout constraints and automatically computes a set of layout attributes that satisfy the constraints using a constraint-solving algorithm. (Data from `https://resourcetrade.earth`)

Abstract—We present *Charticulator*, an interactive authoring tool that enables the creation of bespoke and reusable chart layouts. Charticulator is our response to most existing chart construction interfaces that require authors to choose from predefined chart layouts, thereby precluding the construction of novel charts. In contrast, Charticulator transforms a chart specification into mathematical layout constraints and automatically computes a set of layout attributes using a constraint-solving algorithm to realize the chart. It allows for the articulation of compound marks or glyphs as well as links between these glyphs, all without requiring any coding or knowledge of constraint satisfaction. Furthermore, thanks to the constraint-based layout approach, Charticulator can export chart designs into reusable templates that can be imported into other visualization tools. In addition to describing Charticulator's conceptual framework and design, we present three forms of evaluation: a gallery to illustrate its expressiveness, a user study to verify its usability, and a click-count comparison between Charticulator and three existing tools. Finally, we discuss the limitations and potentials of Charticulator as well as directions for future research. Charticulator is available with its source code at `https://charticulator.com`.

Index Terms—Interactive visualization authoring, Chart layout design, Glyph design, Constraint-based design, Reusable chart layout.

1 Introduction

The ability to create a highly customized visual representation of data, one tailored to the specificities of the insights to be conveyed, increases the likelihood that these insights will be noticed, understood, and remembered by its audience [4]. This expressiveness also gives the author of this visual representation a competitive advantage in a landscape awash in conventional charts and graphs.

However, most interactive charting tools ask chart authors to choose from a collection of standard chart types or templates, such as bar, line, or pie charts, and they provide limited customization options beyond the choice of chart type. Besides interactive charting tools, people also create charts via manual illustration or programming. Illustration tools such as Adobe Illustrator are insufficient for authoring bespoke charts because they cannot bind multiple attributes of data to graphical elements. Meanwhile, programming a bespoke chart using a library such as D3.js [6] or a declarative language such as Vega [32] provides considerable control over the encoding of data to graphical marks and their layout. This approach, however, is accessible only to a small group of people who have advanced programming knowledge.

In recent years, researchers have revisited the prospect of creating bespoke charts via interactive authoring, with tools such as Lyra [30],

- *Donghao Ren is with the University of California, Santa Barbara, and started this work during an internship at Microsoft Research. E-mail: donghaoren@cs.ucsb.edu.*
- *Bongshin Lee and Matthew Brehmer are with Microsoft Research. E-mail: {bongshin, mabrehme}@microsoft.com.*

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

iVisDesigner [27], iVoLVER [22], and Data Illustrator [16]. Lyra and iVisDesigner use data-flow models to represent charts. Complex layouts in these models are modeled as data transforms that compute layout parameters. This makes layout construction a matter of choosing the optimal series of layout transforms and applying them to the data. These layout transforms are represented as configuration panels in the user interface. It is tedious to switch between layouts and exceedingly difficult to compose layouts from scratch. iVoLVER follows a bottom-up constructive approach where charts are constructed by creating graphical elements and data-calculation elements individually. However, for a chart with more than a dozen elements, this becomes very tedious. Furthermore, iVoLVER currently lacks the ability to lay out elements except for simple alignment and positioning according to (x, y) coordinates. Data Illustrator introduces the concept of repetition and partition, and supports basic layout configurations in the repetition or partition groups, combining bottom-up and top-down approaches. However, Data Illustrator can neither produce non-Cartesian charts nor charts featuring visual links including node-link graphs.

In this paper, we present Charticulator (Fig. 1), an interactive chart authoring tool that addresses the limitations of existing approaches by prioritizing the articulation of chart layouts as well as the visual linking between glyphs. Charticulator allows authors to specify chart layouts interactively in lieu of programmatically specifying data transformations. It then converts author-specified layouts into mathematical constraints, and employs a constraint solver to realize the chart. Furthermore, Charticulator supports exporting chart designs into reusable templates, which can be imported into other systems to visualize other data. As Charticulator leverages a constraint-based layout specification, exported templates correctly respond to canvas size changes and different distributions of data. The key research contributions of this work are as follows:

- The design framework of Charticulator, which can be applied to generate a variety of reusable chart layouts.
- The implementation of Charticulator, which realizes the design framework by transforming the chart specification into layout constraints and incorporating a constraint-based layout algorithm, with a user interface that enables interactive chart layout specification.
- Results from three forms of evaluation: a gallery of charts to illustrate Charticulator's expressivity (Fig. 2), a chart reproduction study, and a click-count comparison against three existing tools.

2 Related Work

As discussed in surveys by Grammel et al. [13] and Mei et al. [21], there are many approaches to visualization authoring. We first discuss related work in visualization authoring based on the classification presented in Grammel et al.'s survey. We then briefly discuss the constraint-based authoring of layouts.

2.1 Imperative and Declarative Programming

Programmatic approaches to visualizing data include imperative languages and libraries such as Processing [25], ProtoVis [5], and D3 [6], as well as declarative grammars such as ggplot2 [37], Vega [32], and Vega-Lite [31]. Imperative approaches allow for considerable expressiveness at the cost of complexity, and thus involve a steep learning curve. On the other hand, declarative approaches provide varying degrees of expressiveness while obfuscating the programming concepts that are required when using imperative approaches. For example, Vega uses a low-level reactive data-flow model which makes it highly flexible and expressive, while Vega-Lite is a high-level visualization grammar, which is easier to write but less expressive. Declarative approaches still require chart authors to write code, which remains to be a hurdle for those lacking programming skills.

Charticulator's framework extends these approaches by adding the concepts of layout constraints and layout composition. Using constraints instead of transformations for layout specification allows for independent and partial layout specifications that can be combined freely. Separating glyph-level and chart-level specifications eliminates their dependencies, and thus enhances flexibility. For example, revising a glyph layout will not break the chart layout, and vice versa.

Recently, to leverage the strength of programming and interactive illustration with vector graphics software, Hanpuku [3] provides the ability to create bespoke charts by working fluidly between Adobe Illustrator and D3. However, the chart author is nevertheless required to begin the process with D3 and thus understand programming concepts such as loops and the binding of data to attributes of graphical elements. In contrast, Charticulator allows for the articulation of novel layouts along with links and expressive marks, without requiring any coding.

2.2 Template Editing

There are many existing charting tools that employ a template editing approach [28]. Examples include Datamatic [44], RAWGraphs [19], Infogram [49], Easelly [45], Plot.ly [52], Piktochart [51], ChartBlocks [43], iCharts [48], and Quadrigram [53]. Chart templates are also widely available in general-purpose authoring tools such as Microsoft Office (Excel and PowerPoint), Google Docs, and Apple iWork. These tools provide templates for standard chart types (e.g., bar, line, and pie charts). The basic workflow is to select (or upload) a dataset, select a chart template, and then configure template options. These tools vary in terms of the number of templates, styling capabilities, interactivity, and extensibility. For example, Plot.ly has the ability to individually specify the mark type for each categorical series in a dataset. RAWGraphs and Flourish [46] provide APIs to develop new templates, allowing developers to add new chart types to the tool. Quadrigram provides a dashboard mechanism in which multiple charts or widgets can be presented. It also allows chart authors to add and configure basic widgets for filter and selection.

However, with the template editing approach, novel chart designs are not achievable because templates are not typically modifiable by chart authors. In addition, charts created with a particular tool look similar to each other as templates inherently limit the aesthetic style of their associated charts.

2.3 Shelf Configuration

Interactive tools to facilitate exploratory data analysis such as Tableau (formerly Polaris [34]), Polestar [38], Voyager [38], and Voyager 2 [39] are not prescriptive regarding chart types and instead opt for a shelf configuration approach. Chart authors specify the bindings from data points to corresponding graphical elements by dragging data dimensions as well as derived measures to shelves assigned to mark type, x and y position, facets, and mark attributes such as color, shape, and size. The shelf configuration generates a chart and automatically performs necessary data transformations such as computing quantitative ranges.

However, these systems do not allow both a specification of glyphs comprised of multiple marks and a fine-grained specification of mark style, while their underlying chart layouts are specified by templates that are not configurable by chart authors. Yet, as their drag-and-drop interactions are easy to learn and use, Charticulator applies similar interactions for layout specification.

2.4 Visual Building

Recently, researchers have investigated methods for interactively exposing lower-level aspects of chart design that are typically obfuscated in template- and shelf-based approaches, developing tools that we refer to as *visual builders*. They provide interactions for binding data to graphical mark primitives like rectangles, symbols, and lines. For example, Lyra [30] uses *dropzones* and *connectors* to specify the binding of data dimensions to mark properties and the relative positioning of marks, respectively. iVisDesigner [27] incorporates familiar drop-down configuration panels and menus to specify data bindings, and InfoNice (a.k.a. Infographic Designer) [36] supports expressive mark design using icons, images, and texts. However, these tools still do not treat layout as a first-class citizen; a layout is typically inferred from the data binding operation, particularly when directly applying layout transformations to mark position and size (e.g., the stacking layout in Lyra). Another recent tool is DataInk [41], which uses pen-and-touch interaction to support object-oriented drawing [40]. However, similar to InfoNice, DataInk primarily focuses on expressive glyphs instead of layouts, and while it allows chart authors to create hand-drawn layouts,

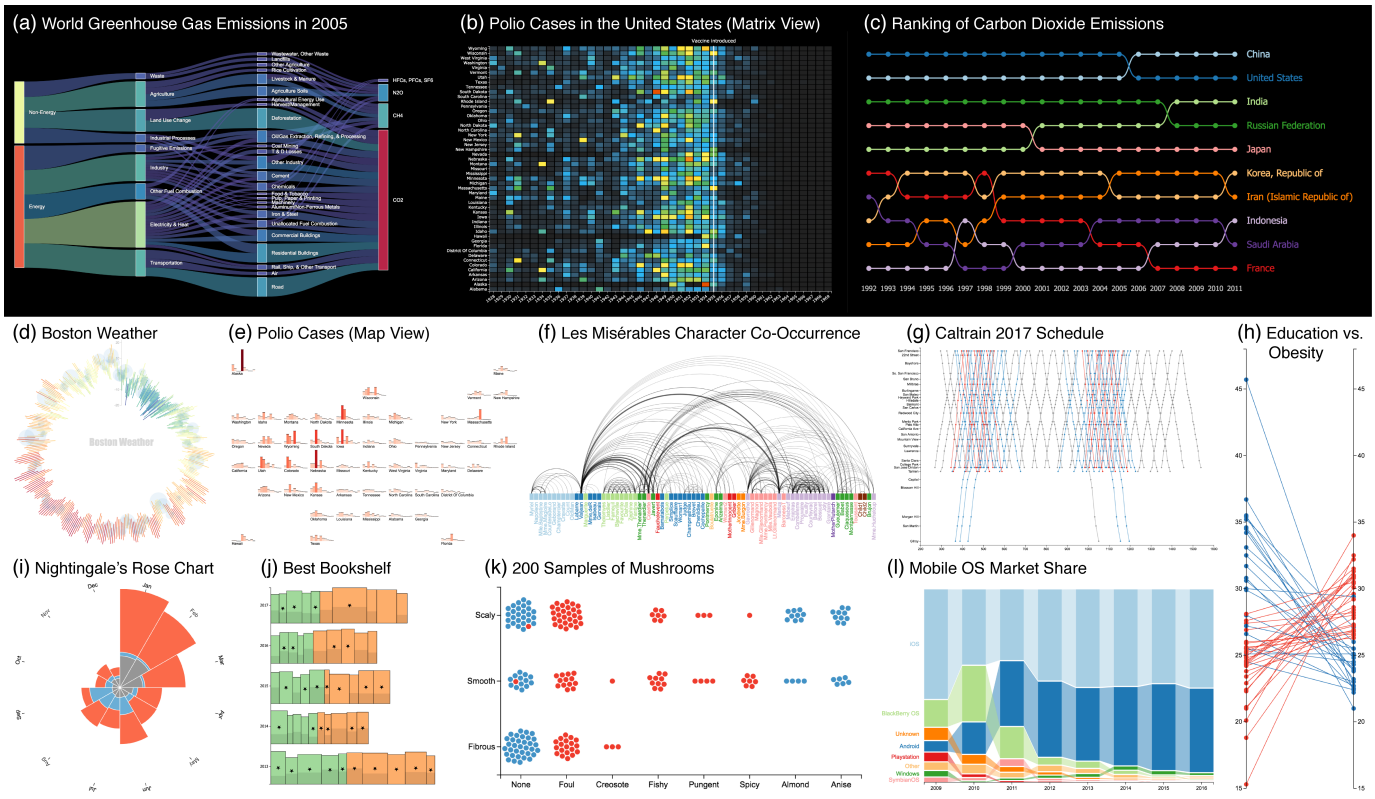


Fig. 2. A gallery of 12 visualization examples showing the expressiveness of Charticulator. Here we hide legends and chart titles for compact presentation. More examples along with high resolution images, detailed descriptions, and videos illustrating the creation processes can be found in the website (`https://charticulator.com`). Data Source: (a) [59]; (b) [66]; (c) [63]; (d) [64]; (e) [66]; (f) [62]; (g) [57]; (h) [8]; (i) [65]; (j) [61]; (k) [68]; (l) [67]. Visual Design Source: (b) [58]; (d) [60]; (g) [18].

it is still difficult to create complex layouts such as a stacked bar chart. Data Illustrator [16] uses a repetition and partition operator for multiplying marks. However, the repetition and partition grid can handle only simple layouts such as grid and horizontal and vertical stacking, and these layouts only work with primitive shapes. To create layouts with a complex glyph, chart creators have to manually overlay multiple groups of marks that are individually laid out. VisComposer [20] combines visual programming, textual programming, direct manipulation, and surrogate objects for chart creation. Different visual encoding design choices can be arranged in a scene graph editor to create complex charts. However, VisComposer supports only a fixed set of compositions and requires programming to create additional visual encodings.

In contrast, Charticulator prioritizes layout as a deliberate design choice. With multiple levels of partial layout specifications, chart creators can construct a wide range of chart layouts. Charticulator also supports layouts incorporating several alternative coordinate systems. Furthermore, Charticulator allows chart creators to export their designs into reusable chart templates which can then be imported into other visualization systems such as Microsoft Power BI (Fig. 8).

2.5 Constraint-based Authoring

Prior work has incorporated mathematical constraints for designing user interfaces, diagrams, and some forms of graphs. An early example is ThingLab [24], which involves connecting geometrical objects (e.g., lines, points) using constraints via mouse and keyboard interactions. Later, Fogarty and Hudson created the GADGET toolkit [12] to incorporate numerical optimization for user interface systems. Beyond research prototypes, the Android development ecosystem recently introduced ConstraintLayout [42] as an option to specify widget positioning in user interfaces: an interactive constraint editor in Android Studio facilitates constraint authoring by allowing developers to specify constraints via drag-and-drop interactions. However, these existing interactive constraint-based authoring approaches only support a fixed set of objects or widgets.

Constraint-based approaches have also been applied to data visualization. GLIDE [29] is a constraint-based graph visualization system that allows people to specify *visual organization features* such as symmetry, alignment, or clustering, whereupon these features are translated to geometrical constraints. Delaunay [10] supports similar constraints for visualizing hierarchical data. The TRIP system [35] employs a constraint-based approach for visualization and animation, where the constraints are specified as a *visual structure representation* in Prolog. These existing approaches either focus on graph or tree visualizations, or they use programming languages for constraint specification. With Charticulator, we set out to enable the creation of bespoke charts without programming, so we investigated ways to efficiently specify common layout constraints via a set of well-designed user interactions and by incorporating data binding into these constraints.

3 Charticulator

In this section, we first present our design principles for Charticulator and its conceptual framework. We then describe its user interface and interactions along with two usage scenarios, and explain our constraint-based layout approach.

3.1 Design Principles

Charticulator is designed for people who lack programming skills, which may include designers, journalists, and analysts. To support a wide variety of chart layout designs, we identified the following three guiding design principles.

Promote layout as a deliberate design choice. In existing visualization authoring tools, layouts are specified either via a set of predefined templates or by binding data to the attributes that affect layouts (e.g., x and y position, width and height of a rectangle mark). To facilitate the creation and manipulation of diverse layouts without programming, Charticulator treats layout as a first-class citizen and exposes layout in the user interface.

Compose a layout using a set of partial specifications. To enable a flexible way to specify layouts, we break layout specifications into composable parts. For example, the chart in Fig. 1 incorporates a polar coordinate layout where wedge shapes are laid out along the circle. The spans of the wedges are bound to a data value. The text labels are positioned at the outer-middle point of the wedges. To accomplish this, we allow for a small set of partial layout specifications to be combined to produce a variety of complex layouts.

In addition, to support expressive glyph design and custom layout of these glyphs, we need to be able to express layout relationships within and between glyphs. To achieve this, we divide layout specification into two levels: chart-level and glyph-level. Charticulator presents partial specifications as objects that can be individually manipulated, and changing one part does not require chart authors to re-specify other parts. To reduce clutter, Charticulator provides two separate editing canvases: one for glyph-level editing and second for chart-level editing.

Balance direct manipulation and configuration panels. Most existing vector graphics authoring tools incorporate direct manipulation of objects on a canvas, and those familiar with these tools have come to expect interaction mechanisms such as click-to-select and objects with draggable anchors. In addition, Metoyer et al. found that people often treat white space as a manipulable element [23]. Charticulator, as mentioned above, enables chart authors to directly manipulate layout parameters such as anchors, margins, and gaps. On the other hand, many layout aspects cannot be easily expressed as manipulable objects. For example, glyphs can be stacked horizontally or vertically, or they can be laid out in a grid. Since a layout is specified as a combination of partial specifications, we design a concise set of menus and panels to represent such options that cannot be directly manipulated.

3.2 Framework

Conceptual frameworks play a central role in visualization authoring tools. For example, the underlying framework of Lyra is the Vega specification [32], and thus Lyra is largely designed around the structure of Vega. Data Illustrator uses a framework based on partition and repetition, and its user interface design is deeply tied to these concepts. Charticulator's framework is designed to support easy creation and composition of novel layouts. Below is the formal specification of the framework, and Fig. 3 illustrates the use of the framework.

```
Mark := Rectangle | Symbol | Line | Text
GlyphElement := Mark | Guide | GuideCoordinator |
    DataDrivenGuide
Glyph := GlyphElement*, LayoutConstraint<GlyphElement>*
ChartElement := PlotSegment | Link | Mark | Legend | Guide |
    GuideCoordinator
PlotSegment := Glyph, (Scaffold | Axis){0..2}, Sublayout,
    CoordinateSystem
Attribute := "x1" | "y1" | "x2" | "y2" | ...
ElementAttribute<ElementType> := ElementType, Attribute
ParentAttribute := Attribute
ConstraintType := "equals"
LayoutConstraint<ElementType> := (ParentAttribute |
    ElementAttribute<ElementType>){2}, ConstraintType
Chart := ChartElement*, Scale*, LayoutConstraint<ChartElement>*
```

Notation `"*": zero or more; "{0..2}": zero to two; "|": or; "X<Type>": template with parameter "Type"`

3.2.1 Layout Elements

A set of elements can be used either at the glyph level or at the chart level. *Marks* are primitive graphical elements (e.g., rectangle, text) whose attributes such as width, height, and color can be manually specified or bound to data. *Guides* are indicators that are visible only during the design phase and are used for aligning objects (e.g., a horizontal guide for aligning the top of marks). *Guide Coordinators* add multiple guides with a layout relationship (e.g., five guides with an equal distribution) to a glyph or a chart.

The *Glyph-level* specification includes *glyph elements* and *layout constraints*, which specify position relationships between glyph elements. Charticulator currently supports only equality constraints. For example, a star symbol can be positioned at the center of a rectangle (i.e., $star.x = rect.cx, star.y = rect.cy$). To determine the mark position by data, Charticulator incorporates the *Data-Driven Guides* technique [15]. A data-driven guide provides data-driven anchor points from data columns with the same unit (e.g., min and max values of temperature). Glyph elements can be snapped to these anchor points by adding layout constraints (e.g., put the star symbol at the anchor point of the "min" data column). In addition, data-driven guides can be displayed as axes; to avoid duplicated axes, Charticulator shows only the first instance. This is useful for incorporating familiar graphical elements like error bars (e.g., the X-axis in Fig. 9).

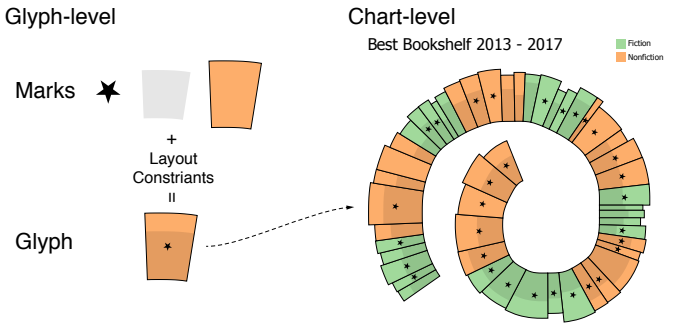


Fig. 3. An example illustrating Charticulator's framework. Inspired by the *Best Bookshelf* visualization [61], this example uses three marks to compose the glyph: a star symbol and two rectangles. The glyphs are laid out by the plot segment with a horizontal scaffold and a custom curve coordinate system, which morphs the glyphs into wedge-like shapes. A text mark is used for the chart title and a legend shows the color scale.

The *Chart-level* specification includes *chart elements*, layout constraints between them, and *scales*. The most important chart element is a *plot segment*, which lays out glyphs according to its *scaffolds* and/or *axes*, and transforms them according to its *coordinate system*, which we discuss in Section 3.2.2. Scales specify how data is mapped to attributes such as width, height, and color, and they can be shared among several marks. *Legends* visualize the scales used in the chart. Charticulator currently uses a predefined legend for each scale type. Links draw between-glyph connections as described in Section 3.2.3.

3.2.2 Layout Composition

The layout of a plot segment is determined by its *scaffolds* and *axes*, and further determined by *sub-layout* options, if applicable. A plot segment has two independent layout directions. The terms we use for these directions depend on the plot segment's coordinate system: "X" (or "Horizontal") and "Y" (or "Vertical") for Cartesian coordinates, "Angular" and "Radial" for Polar coordinates, and "Tangent" and "Normal" for coordinates along custom curves. Similar to Transmogrifier [7], Charticulator morphs mark shapes in non-Cartesian systems. However, while Transmogrifier's morphing is image-based, Charticulator's morphing is performed in vector graphics and thus maintains a precise data binding. For example, a rectangle becomes a wedge shape in a custom curve coordinate system (Fig. 3).

The X and Y direction of a plot segment can be assigned as a *scaffold*, a categorical *axis*, or a numerical *axis*. A scaffold stacks the glyphs sequentially within the plot segment; a categorical axis groups the glyphs according to their categories to place them with even spacing along the axis; and a numerical axis positions the glyphs based on their data values. Fig. 4 shows the layouts produced from the combinations of scaffolds and axes.

For categorical axes, Charticulator employs *sub-layouts* to determine a within-group layout. Charticulator currently supports four such options: horizontal stacking, vertical stacking, grid, and circle-packing. However, when a categorical axis is combined with a scaffold or a numerical axis, the sub-layout does not apply because a scaffold or a numerical axis positions glyphs based on their data values (e.g., Fig. 2-c). When no scaffold or axis is specified, we treat all glyphs as a single group, and thus the selected sub-layout option constitutes the primary layout (e.g., Fig. 2-f).

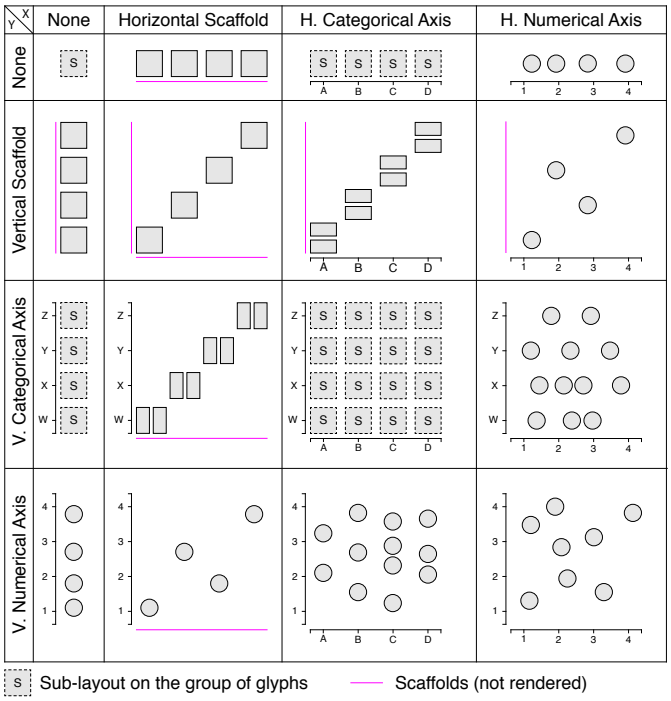


Fig. 4. Possible combinations of scaffolds and axes.

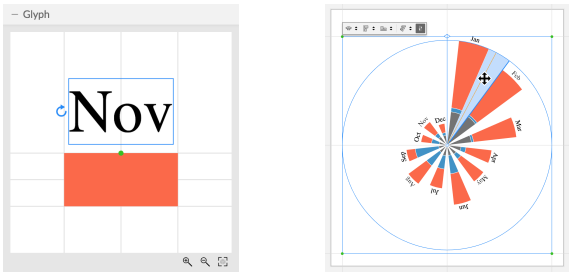


Fig. 5. Anchors and handles in the glyph editor (left) and in the chart canvas (right). The green (dot) represents that the anchor is snapped to an underlying guide.

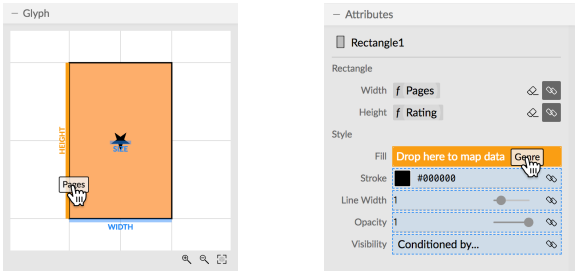


Fig. 6. Charticulator uses Dropzones [30] for drag-and-drop style data mapping in the glyph editor (left) and the attributes panel (right).

3.2.3 Link Construction

Charticulator supports three types of *links* for displaying connections between data items: (1) links through a data series (e.g., the link through each Operating System in Fig. 2-l); (2) links connecting data items on multiple plot segments (e.g., the link between two axes in Fig. 2-h); and (3) links from a separate data table (e.g., the links correspond to the edges in *Les Misérables*'s character co-occurrence dataset in Fig. 2-f).

Links must be anchored on glyphs, and can take the form of lines or bands. The shape of links can be straight or curved (Bézier curve, arc). For example, the straight bands are anchored from the right side of the starting rectangle to the left side of the ending rectangle in Fig. 2-l, while the line arcs are anchored at the top of each rectangle in Fig. 2-f.

3.3 User Interface and Interaction

Charticulator's user interface consists of a dataset panel, a glyph editor, a layers panel, an attributes panel, and a chart canvas (Fig. 1). We explain Charticulator's interaction with two example scenarios. To see how Charticulator works in action, refer to gallery videos on the website (`https://charticulator.com`).

3.3.1 Basic Interaction Mechanisms

Using Charticulator, the chart creator can compose a glyph in the glyph editor, adding marks and specifying relationships between them. They can specify the layout relationships between the glyphs either within the chart canvas or from the attributes panel, which displays attributes for the currently selected item.

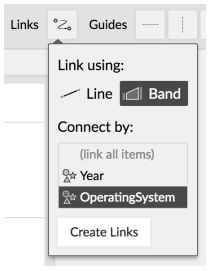
To add a mark to a glyph, the chart creator can simply drag the desired mark and drop it into the glyph editor. Charticulator places the mark at its default position, the center of the glyph, and adds it to the layers panel. To place the mark at a specific location, the creator can click the mark to activate it and click or drag on the canvas depending on the mark type. Guides and plot segments share the same mechanism.

In addition to selecting it from the layers panel, the chart creator can select a mark from the glyph editor or the chart canvas. For the selected item, Charticulator shows anchors and/or handles, which can be used to specify the layout relationship between two objects in the corresponding canvas. For example, the chart creator can ensure the same gap between a text mark and a rectangle by dragging the anchor of the text to the top of the rectangle (Fig. 5-left). Similarly, they can drag a gap handle to adjust the gaps between glyphs (Fig. 5-right).

The chart creator can perform data binding in multiple ways. They can drag a data column and drop it into dropzones [30] in the glyph editor (Fig. 6-left), the chart canvas, or the attributes panel (Fig. 6-right). They can also select the data column from a popup panel. When the data is bound to mark attributes, Charticulator automatically infers the appropriate scales (categorical and numerical), which can be manually adjusted if necessary in the scale editor (Fig. 7-left).

The layout of a plot segment can be specified in multiple ways. A plot segment provides dropzones for binding data to its principal directions such as X and Y for Cartesian coordinates in the chart canvas and in the attributes panel. Once a data attribute is dropped, an axis is created for the direction. To add a scaffold to a plot segment or to change the plot segment's coordinate system, the chart creator can drag a scaffold button from the toolbar to the plot segment's dropzones.

For creating links, the chart creator can specify the shape (i.e., line or band) and type of links from a popup panel (see inset figure), and adjust anchor positions directly from the chart canvas. Charticulator also supports *data-driven visibility*; as the name implies, the visibility of marks is determined by data values. Clicking on the "Conditioned by" button for the Visibility attribute invokes a popup panel, where the chart creator can set a filter via a set of checkboxes (Fig. 7-right).



As constraints are independently specified, Charticulator supports a flexible order of operations. For example, after specifying the layout of a plot segment, the chart creator can alter the glyph design by adding and removing marks, or by changing layout constraints in the glyph editor; it is also possible to change the glyph layout after adding links.

3.3.2 Scenario 1: Mobile Operating System Market Share

We will create a chart depicting the yearly market share trends of mobile operating systems from 2009 to 2016, as shown in Fig. 2-l. The dataset contains three columns (`Year`, `OperatingSystem`, and `Share`) and 64 rows. In this chart, the values for each year are shown as a stacked column, ordered by the `Share` values. Each operating system is connected by a band; a crossing of the bands shows a rank change.

We begin by adding a rectangle mark to the glyph by dragging it from the toolbar into the glyph editor. To indicate the operating system, we drag the `OperatingSystem` data column to the fill attribute of

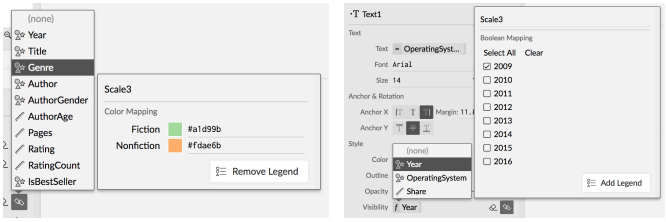


Fig. 7. Specifying data mapping in the attributes panel. (left) Color-coding marks with `Genre` and editing the color palette. (right) Conditional visibility of marks, showing only the first `Year`.

the rectangle. Then, we bind `Share` to the height of the rectangle by dragging the data column to the height dropzone.

Next, we drag the `Year` column to the X-axis in the chart canvas to group glyphs by year. To stack the glyphs vertically, we toggle the "Stack Y" sub-layout option of the plot segment. To order the glyphs by market share value, we select the `Share` column among the order option for the plot segment. We then adjust the gaps between stacked glyphs and between years by dragging horizontal and vertical gap handles. To create bands between the glyphs, we invoke the "Link" options, select a "Band" link style, group by `OperatingSystem`, and click "Create Links."

To add the text labels, we add a text mark to the glyph by dragging it into the glyph editor, and then we move the text anchor to the middle left of the rectangle and move the text to the left side of the anchor. We then drag the `OperatingSystem` to the Color attribute of the text to make it consistent with the rectangle fill color, and we also drag it to the text dropzone so it is obvious which operating system the glyph refers to. To show the text marks for the first year only (i.e., 2009), we toggle the "Conditioned by" options for the text marks' "Visibility" attribute. From this menu, we select the `Year` column, clear the default selection, and select 2009 only. Finally, we adjust the left margin of the chart by dragging the guide and then type the chart title.

3.3.3 Scenario 2: Reproducing the Rose Chart

We will reproduce Florence Nightingale's Rose Chart, one depicting the monthly death toll and causes of death by disease, wounds, and other causes among the Army of the East during the Crimean War in 1855. In this chart (Fig. 2-i), each wedge of the circle corresponds to a month, with the three causes of death stacked radially and the area of each being proportional to the number of deaths.

We start by dragging a rectangle mark into the glyph editor. We drag the `Death` column to its Height attribute and the `Type` column to its Fill attribute. To convert Cartesian coordinates to Polar coordinates, we drag the circle scaffold from the toolbar into the chart canvas. This converts the chart into a circular layout of rectangles tangential to the circle. We then drag the `Month` column to the angular axis.

Selecting the plot segment, we toggle a radial stacking sub-layout. We reduce the inner radius of the plot segment by dragging the inner circle to the center (a radius of 0). To bind data (the number of deaths) to the area, we check the "Height to Area" checkbox. To remove angular separation between the wedges, we set the angular gap to 0%, but we give each a white stroke in the rectangle's attributes panel. Next, we select the rectangle mark to assign a custom palette of colors to the causes of death, and we toggle a legend. Finally, we adjust the chart margins, anchor the legend to left guide, and edit the chart title.

Since this is a chart layout that we would like to reuse, we export it as a Power BI custom visual [50]. We open the File panel, toggle the Export tab, and select "Power BI Custom Visual." In the following panel, we replace the data column names with more generic data slot names: `Month`, `Type`, and `Death` are replaced by "Angular Axis," "Category," and "Area," respectively. We name this template "Rose Chart" and click the "Export" button. Finally, we import the resulting ".pbiviz" file in Power BI and use it to visualize other data (e.g., Fig. 8).

3.4 Constraint-based Layout

At the system level, partial layout specifications are not independent of one another. For example, binding data to the width of a rectangle and stacking all rectangles horizontally to a total width are two inter-dependent partial specifications: the scaling parameter of the width binding depends on the total width of the stacking layout. To support the combination of partial layout specifications, we express partial specifications as mathematical constraints and employ a constraint solver to determine the final layout.

3.4.1 Constraints at Multiple Levels

Since there are chart-level elements, glyph-level elements, and scales to be considered, Charticulator generates constraints at multiple levels.

Chart-level constraints specify how plot segments, legends, chart-level marks, and guides are related to one another. For example, a constraint that legend A should be on the right of the plot segment B translates to $B.x_2 = A.x_1$ mathematically. Guides and guide coordinators can be used at this level to position elements.

Each plot segment has constraints specifying how its glyphs are positioned, and these constraints depend on the plot segment's scaffolds. For example, a single horizontal scaffold enforces the glyphs to be horizontally adjacent to one another (with an optional gap):

$$\forall G_1, G_2 : \text{Glyph} \wedge \text{adjacent}(G_1, G_2), G_1.x_2 + gap = G_2.x_1$$
$$G_{\text{first}}.x_1 = PlotSegment.x_1$$
$$G_{\text{last}}.x_2 = PlotSegment.x_2$$

The two preceding statements are boundary conditions, while the following vertically aligns glyphs by their anchors:

$$\forall G : \text{Glyph}, G.anchor.y = y$$

Glyph-level constraints are similar to chart-level constraints except that their scopes are individual glyphs. Guides, guide coordinators, and data-driven guides [15] can be used at this level.

A mark or chart element can have intrinsic relationships among its attributes. For example, consider the attributes of a rectangle mark: x_1, x_2, x_{center}, and $width$. Increasing x_2 will also increase $width$ and x_{center}. Intrinsic constraints enforce these relations. In this example they are:

$$x_1 + x_2 = 2 \times x_{\text{center}}$$
$$x_2 - x_1 = width$$

Intrinsic constraints allow us to expose multiple related attributes simultaneously and maintain the consistency of partial specifications. Chart authors do not have to fully specify all of the attributes. In the above example, they can specify any two of the four attributes, and the constraint solver computes the remaining attributes.

3.4.2 Data Binding Constraints

Constraints are also involved in the data binding process when layout attributes are bound to data. For example, when a data dimension is bound to the width of a glyph, the scale constraint will be:

$$\forall i, G_i.width = Scale.factor \times d_i$$

Assuming a horizontal layout which enforces the rectangles to be adjacent to each other, we will have the following constraints:

$$\forall i, G_i.width = Scale.factor \times d_i$$
$$\forall i, G_i.x_2 - G_i.x_1 = G_i.width$$
$$\forall j = i+1, G_i.x_2 + gap = G_j.x_1$$
$$G_1.x_1 = PlotSegment.x_1$$
$$G_N.x_2 = PlotSegment.x_2$$

The first statement is the scale constraint, the second statement is the glyph's intrinsic constraint, and the third to fifth statements refer to the plot segment's layout constraint. Together, the constraint solver can determine the correct scale factor:

$$Scale.factor = \frac{PlotSegment.x_2 - PlotSegment.x_1 - (N-1)gap}{\sum_i d_i}$$

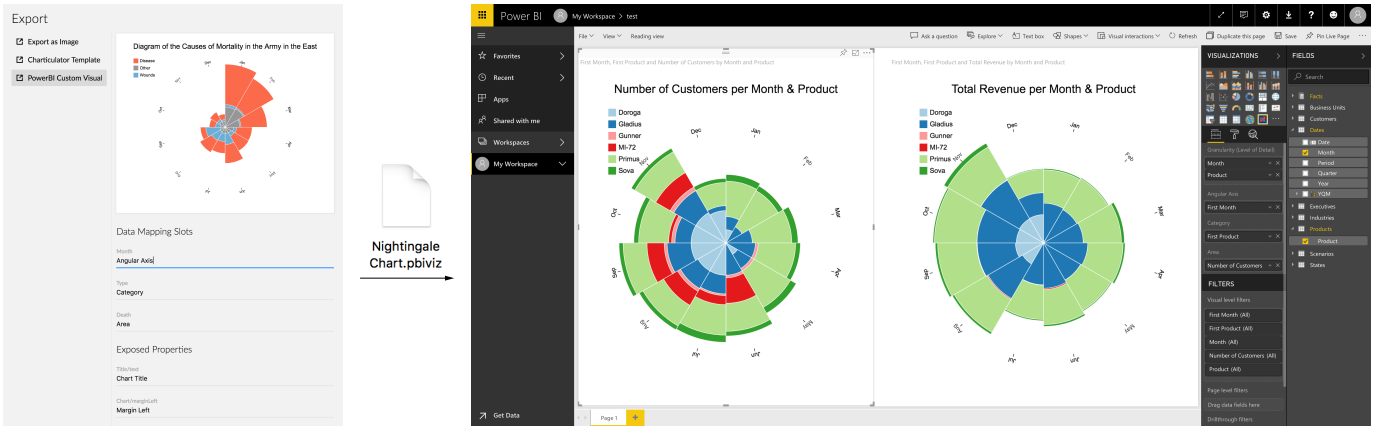


Fig. 8. Charticulator allows the chart creator to export chart layouts as reusable templates in the form of custom visuals for Microsoft Power BI, which can be imported into Power BI's web or desktop client to visualize new data.

For the sake of simplicity, we are assuming that the plot segment's attributes are given. In Charticulator, these attributes are subject to chart-level constraints: plot segments can be snapped to other chart-level elements, such as plot segments and guides.

3.4.3 Two-stage Constraint Solving

The mathematical constraint solver we are using is a linear solver, which is fast and has guaranteed convergence. However, many possible charts, such as those incorporating a packing layout or a force-directed layout, can only be expressed via nonlinear constraints. Charticulator therefore incorporates a two-stage constraint solving mechanism. In the first stage, Charticulator invokes the linear constraint solver to determine the layout of the chart elements and a basic binding for stacking-based layouts, which can be expressed via linear constraints. The first stage can generate additional variables that can be used in the second stage. Then, Charticulator invokes additional layout algorithms (e.g., circle packing) to address nonlinear layouts.

3.5 Reusable Chart Templates

A notable strength of Charticulator is that the chart design is not tightly coupled with the input data because automatic layout computation allows the chart to adapt to new data as long as the types of data columns are the same. This makes it possible to export the chart design as a template to be used in other charting systems. Charticulator extracts a list of data mapping slots (e.g., "X-Axis," "Group By") and properties (e.g., "Chart Title") from the chart specification, and allows the chart creator to choose names for the slots and properties (e.g., Fig. 8-left). Once the custom names are specified, Charticulator bundles the named list with the chart specification and essential Charticulator components including the constraint solver and the chart renderer into a single JavaScript package. This package provides an API to render the chart design with given data columns and properties. For each target visualization tool, we implement an adapter that exposes the API in the format required by the visualization tool. For example, to export as a Microsoft Power BI custom visual, we produce a ".pbiviz" file with appropriate metadata (Power BI refers to these as "capabilities") and glue code. The metadata tells Power BI what the input data should be like, and the glue code takes the input data from Power BI and produces the chart using the exported JavaScript package.

3.6 Implementation

Charticulator is implemented as an HTML5 application and uses technologies including TypeScript [55], React [54], and WebAssembly [56]. It follows the basic Flux application architecture [47], with the addition of a constraint solver. The application maintains a *chart specification* and a *chart state*. The chart specification describes the chart in a JSON object that mirrors the framework discussed in Section 3.2. The chart state stores all of chart elements' attributes. Together, they form the "Store" part of the Flux architecture. For each chart editing interaction, an "Action" is emitted and dispatched through the global "Dispatcher" to the "Store." The store then modifies the chart specification and invokes the *constraint solver* component to update the chart state. Once the state is successfully updated, the "Store" emits an update event which causes the user interface components to update.

Many algorithms in the literature solve linear constraints. Notably, the Cassowary algorithm [2] extends the simplex method to allow for prioritizing constraints, and the Conjugate Gradient algorithm [33] is very efficient for solving sparse linear systems. We experimented with both algorithms and settled on a sparse least-squares conjugate gradient algorithm. We selected a sparse solver because most layout constraints involve only a few variables and thus produce a very sparse matrix; we use the least squares minimization technique to deal with weighted constraints, if any. However, this algorithm only supports equality constraints, and thus Charticulator only allows for the specification of such constraints.

More details about implementation as well as algorithm benchmarks can be found in the supplemental material.

4 Evaluation

We evaluate Charticulator in three forms: (1) a gallery to demonstrate its expressiveness; (2) a user study to assess its usability; and (3) a click-count comparison with three existing chart creation tools.

4.1 Gallery

To demonstrate the expressive power of Charticulator, we produced a variety of charts with a diverse collection of datasets; Fig. 2 shows 12 examples from our gallery. Many of these charts are reproductions or variations of charts created by news graphics teams. The full gallery can be found in the website, along with high-resolution images, detailed descriptions, and videos illustrating the creation processes.

4.2 User Study: Chart Reproduction

To determine if people can produce bespoke charts with Charticulator, we followed a procedure similar to those used to evaluate Data Illustrator [16] and ChartAccent [26], which focuses on chart reproduction.

4.2.1 Study Design

We recruited 11 participants (5 female) who had normal or corrected-to-normal vision from the Puget Sound area. All participants had at least three years of experience using graphics editors such as Adobe Illustrator, Adobe Experience Design, Sketch, and Inkscape. They also had created infographics and used spreadsheets (e.g., Microsoft Excel, Google sheets, Apple Numbers) within the past three months. In addition to five designers, we had participants with six other occupations: a print operator, an architect, a digital marketing manager, an editor, a sales & marketing professional, and a major gift officer. Their ages ranged from 22 to 48, with an average age of 33. We compensated them with a $150 eBay gift card.

We prepared four chart reproduction tasks, covering the basic concepts of Charticulator. Each subsequent task increased in terms of complexity. Task 1's chart depicts monthly activity by intensity in a stacked radial layout (similar to Nightingale's rose chart) using the activity tracking data. Task 2's chart (Fig. 2-k) depicts the 200 types of mushrooms grouped by odor and surface quality, using a circle packing sub-layout. Task 3's chart depicts the co-occurrence of characters in *Les Misérables* using a radial layout similar to the chart in Fig. 1. Task 4's chart (Fig. 2-l) is described in Section 3.3.2. Videos illustrating the step-by-step construction of these charts are provided on the website. We also prepared six additional charts, three for a tutorial and three for practice tasks.

We used a 3.47 GHz Windows 10 desktop machine with 12 GB RAM, using two side-by-side 24-inch LCD displays running at 1920 × 1200 resolution. For each task, we showed the target chart image on the left monitor, and asked participants to reproduce the same chart in Charticulator on the right monitor. We logged participants' interactions with Charticulator and recorded chart images at each construction step. We also captured screen recordings along with concurrent video and audio recordings of the participants as they proceeded.

After providing a brief explanation of the study goals and procedure, we asked participants to complete a pre-study background questionnaire. We then provided a tutorial on the basic features of Charticulator using three charts. After completing three practice tasks to familiarize themselves with Charticulator, participants performed the four tasks described above on their own. Before starting each task, we described the target chart and the underlying dataset. When they are ready, participants pressed a "Start Task" button to load the dataset and begin the task. After completing the task, they pressed a "Complete Task" button. We encouraged participants to think aloud, especially when any aspect of the task was unclear, or if they were unsure of how to use Charticulator's features. We provided hints to participants when they asked for help or when their progress stalled, noting the cause in such cases. At the end of the study session, participants filled out a questionnaire about their experience with Charticulator. On average, the training (tutorial + practice) lasted about 50 minutes, and the entire session lasted about 80 minutes.

4.2.2 Results

Ten out of 11 participants successfully reproduced the target charts for all four tasks with only a few hints, and six participants successfully completed all tasks without any hints. The one remaining participant (P3) completed the first three tasks, but had trouble understanding that, in Task 4, the bars were ordered by the share values in each year.

We conducted two pilot sessions before this study, and in doing so we identified three usability issues: (1) text manipulation (i.e., alignment and rotation), (2) conditional visibility (of text), and (3) legend creation. To address these issues, we incorporated direct manipulation for the text mark and revised the attributes panel. Our results indicated that we fully addressed the first two issues. None of the participants had trouble manipulating text marks. All participants managed to control the visibility of text marks. However, two participants still forgot how to add a legend. Considering a legend as a property of the chart itself, they tried to look for a button to add a legend at the chart level. P9, who forgot how to add a legend during the practice session, mentioned that *"I'm still thinking in a regular [Microsoft] Excel format."*

With regards to task completion time, the average task completion time was less than four minutes, ranging from 130 to 235 seconds (Fig. 9). Note that we did not explicitly ask the participants to complete the task as quickly as possible. When rating Charticulator on three satisfaction criteria using a 7-point Likert scale (1: "Strongly Disagree" to 7: "Strongly Agree"), participants indicated that Charticulator is easy to learn (Avg = 6.55) and that it was easy (6.27) and enjoyable (6.73) to create charts with it. Participants appreciated various aspects of Charticulator. Seven participants mentioned that the drag-and-drop interaction was what they liked most. Participants also appreciated the immediate visual feedback, its power and flexibility, its similarity to Adobe InDesign, and its easy access to the underlying data.

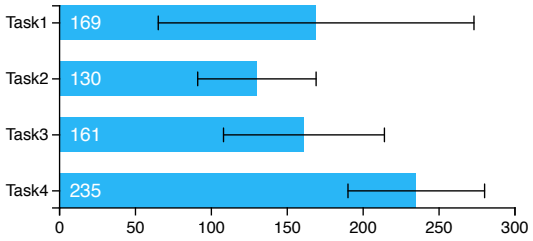


Fig. 9. Task completion time in seconds. Error bars represent the standard deviations. (This chart is created with Charticulator.)

4.3 Comparison to Existing Tools

We also sought a way to compare Charticulator with three existing chart creation tools: Data Illustrator [16], Lyra [30], and iVisDesigner [27]. While a comparative user study would have been ideal, these tools differ in terms of conceptual framework, feature set (e.g., z-ordering, undo/redo), the availability of tutorials, low-level interface design choices, and the stability of their implementation. We thus decided to compare the number of user interactions as a proxy assessment of their respective complexities, inspired by the keystroke-level model [9] approach. However, we concede that a fewer number of interactions does not necessarily mean that a tool is easier to learn to use the tool.

We proceeded to generate seven charts (Table 1) with each of the aforementioned tools. We selected charts that are representative of a variety of designs and layouts, those that can be created with other tools. We considered the chart creation to be successful if it was possible to realize the chart design with reasonable fidelity, focusing on the correctness of the data binding, layout, and linking because exact chart size, legend, label positioning, and colors can vary given a tool's available features. In addition, we did not count repetitive interactions to fine-tune a design (e.g., trying out different colors or gap sizes), and assumed that one such interaction yielded the desired result.

We report the number of clicks, drag-and-drops, and text inputs separately (Table 1). Double-clicks count as two clicks; one text input interaction includes a click to activate the textbox. The actual counts may vary depending on the exact order of creation. We again note that these numbers should only be treated as a general impression of the complexity of interaction. Based on this comparison, we found that Charticulator has a comparable complexity to other tools in terms of the number of low-level interactions.

5 Discussion and Future Work

5.1 Limitations

Framework. Charticulator currently cannot create three charts from Data Illustrator's gallery. While some of the limitations can be addressed by simply implementing more features (e.g., adding a triangle mark type), other limitations are inherent to the framework design. The most notable limitation is that Charticulator supports only one level of repetition in plot segments, and thus the data granularity must be predetermined by the chart creator. In contrast, the desired granularity can be achieved by multiple repetition and partition operations in Data Illustrator's framework. To address this limitation, we would like to extend our framework in two ways. First, a data transformation pipeline can be added to the plot segment level; by specifying a set of data transformations, the chart creator will be able to aggregate the data to a desired granularity. Second, and more importantly, we can allow a glyph design to be a nested chart (e.g., Fig. 2-e can be seen as a scatterplot of bar chart glyphs): this can be done recursively to support multiple levels of granularity.

Scalability. The main drawback of incorporating a constraint solving algorithm is speed (and thus scalability), as most of the constraint solvers have $O(n^2)$ time complexity. For example, positioning a few hundred glyphs with chaining constraints (e.g., stacking them horizontally with varying widths) may take 1–2 seconds to complete. We have improved the solving speed around 1.5–2× by using WebAssembly [56] and running the solver in a background worker to avoid blocking the

	Charticulator			Data Illustrator [16]			Lyra [30]			iVisDesigner [27]		
Chart	CL	DD	TI	CL	DD	TI	CL	DD	TI	CL	DD	TI
Nightingale chart (Fig. 2-i)	31	13	2	Not supported			24	16	7	Not supported		
Color-coded matrix [a] (Fig. 2-b)	15	13	2	20	5	2	11	12	2	27	1	2
Parallel coordinates [b] (Fig. 2-h)	18	10	5	20	5	1	29	9	9	26	0	5
Red and Blue America [17]	20	8	2	43	9	1	14	9	2 [c]	57	4	4
Ranking of CO_2 Emissions (Fig. 2-c)	29	6	3	44	3	2	45	11	3	60	0	11
Caltrain's schedule [d] (Fig. 2-g)	13	8	4	39	9	2	22	16	5	47	1	4
Best bookshelf (Fig. 2-j)	26	11	2	50	8	1 [e]	36	19	7 [e]	Not supported		

Table 1. Number of clicks (CL), drag-and-drops (DD), and text inputs (TI) to create each chart using Charticulator and three existing tools. Notes: (a) While the original chart has a multi-stop custom gradient, we opted to use a two-stop gradient for the sake of simplicity; (b) Axis labels are omitted and circles are optional; (c) We were unable to add state name text labels, but this appears to be a bug and not a shortcoming of the tool; (d) The Y axis represents the distance between the stations; and (e) We were unable to determine how to add the stars to indicate bestsellers. Lyra supports connectors, but it does not work as desired with a stacked layout.

user interface. In the future, we will investigate ways to further improve the performance of the constraint solving algorithm, for example, by using a preconditioner such as incomplete Cholesky factorization [1] in the conjugate gradient algorithm to address chaining constraints.

User Study. In our reproduction study, we intended to assess if people could produce chart layouts using Charticulator when provided with a reference chart, data, and about an hour of training. However, just as it is difficult to ascertain if a particular text editor will result in highly expressive prose, we cannot conclusively determine if the use of Charticulator will result in the production of expressive and novel charts. We invested a significant amount of time and effort to design bespoke charts for our gallery and for our reproduction study, and thus it would be logistically difficult to study the design of truly bespoke chart layouts in 60–90 minute laboratory study sessions. In the future, we plan to evaluate the expressiveness of Charticulator with chart creators in a workshop or hackathon setting. To promote engagement with the tool, we will ask participants to bring their own data. We also intend to conduct a long-term evaluation through engagement with students taking a data journalism course taught by our academic collaborators.

5.2 Manual Manipulation vs. Layout Specification

People familiar with vector graphics editing tools such as Adobe Illustrator are familiar with the "everything is manipulable" idea and have come to expect that objects will respond to direct manipulation. In a constraint-based layout approach, this may not always be feasible because constraints are inherently interconnected, and chart creators can try to set conflicting constraints. To alleviate this issue, Charticulator controlled the complexity of constraints that can be specified, in an attempt to minimize the level of unexpectedness. It would be important to investigate ways to provide appropriate feedback to chart creators when their desired layout is not possible due to conflicting constraints.

Visualization design generally involves specifying a large number of attributes. For example, a rectangle mark has width, height, color, stroke, opacity, and visibility. Furthermore, people tend to iteratively design elements and revisit earlier design choices. Even though our study participants were familiar with graphics editing tools, some participants were overwhelmed by the many options Charticulator offered. For example, P8 stated that *"It was sometimes difficult determining what I needed to click to reveal other properties/options."* In several occasions, knowing that they needed to apply a packing sub-layout, participants tried to remember where the option for the sub-layout was. Thus, an important direction for future research would involve envisioning new forms of interactive tutorials to facilitate the self-teaching of highly configurable user interfaces such as Charticulator's.

5.3 Additional Expressivity

Charticulator currently includes only four basic mark types. We can support more complex glyph designs beyond the composition of rectangles, symbols, lines, and text. For instance, incorporating the "Pen" tool from Adobe Illustrator would allow chart creators to draw arbitrary polylines or curves. The control points of these elements can be anchored on other marks or positioned by data-driven guides [15]. Similarly, it would be useful to investigate the use of pen and touch interactions to create custom marks such as in DataInk [41]. In addition, we can further enhance the expressive power of Charticulator by incorporating the annotation capabilities of ChartAccent [26], which provides a rich palette of data-driven annotation options.

Charticulator primarily deals with layouts of glyphs. Links are connected to glyphs by user-specified anchors, and no further layout of links is supported. In the future, we will investigate how to interactively specify the layout of links by incorporating edge bundling (e.g., hierarchical edge bundling [14]) and routing techniques (e.g., force-directed edge routing [11]), which may involve the insertion of magnetic elements that interact with a force-directed edge routing algorithm to aid the interactive placement of links. Besides, we can further extend Charticulator's expressive range by supporting additional specialized layout algorithms (e.g., treemap and force-directed graph layout).

As demonstrated in Fig. 8, Charticulator can export charts as reusable templates. Our ultimate goal is to allow Charticulator to *import* these templates and use them as glyph-level elements that can be bound to data. This will facilitate the creation of faceted charts and small multiples. For example, we can import a line chart of monthly market share values and display it at multiple geographical locations.

5.4 Deployment: Reaching a Worldwide User Community

Microsoft Power BI is an established business intelligence tool with a large worldwide user community, which includes a marketplace for sharing custom visuals. We are presently collaborating with Power BI to deploy Charticulator as a publicly available custom visual designer. In conjunction with this deployment, they also intend to host user community events such as a custom visual design competition. We are excited about the opportunity to reach Power BI's user community, which will help us better understand how people use Charticulator with their own data.

6 Conclusion

In this paper, we presented Charticulator, an authoring tool for specifying bespoke and reusable chart layouts. Unlike other interactive charting tools, Charticulator prioritizes the configuration of layout between marks or glyphs, and leverages a constraint solver to achieve a wide variety of chart designs. We explained the design principles behind Charticulator's framework, which generalizes to the creation of a wide range of charts. We described how Charticulator transfers the chart specification into layout constraints and incorporates a constraint-based layout algorithm, which enables the export of chart designs as reusable chart templates. We also illustrated how its user interface enables interactive chart layout specification. We demonstrated the expressive potential of Charticulator through a gallery of charts, assessed its usability via a chart reproduction study, and counted the number of interactions required to create charts, comparing against three existing tools. Finally, we discussed several ways to further evaluate and enhance the expressive power of Charticulator.

Charticulator is available at `https://charticulator.com`. The source code is released under an MIT open source license at `https://github.com/Microsoft/charticulator`.

REFERENCES

[1] O. Axelsson. *Iterative solution methods*. Cambridge University Press, 1996.

[2] G. J. Badros, A. Borning, and P. J. Stuckey. The Cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 8(4):267–306, 2001. doi: 10.1145/504704.504705

[3] A. Bigelow, S. Drucker, D. Fisher, and M. Meyer. Iterating between tools to create and edit visualizations. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 23(1):481–490, 2017. doi: 10.1109/TVCG.2016.2598609

[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 (Proceedings of InfoVis)*, 22(1):519–528, 2016. doi: 10.1109/TVCG.2015.2467732

[5] M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 15(6):1121–1128, 2009. doi: 10.1109/TVCG.2009.174

[6] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-driven documents. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 17(12):2301–2309, 2011. doi: 10.1109/TVCG.2011.185

[7] J. Brosz, M. A. Nacenta, R. Pusch, S. Carpendale, and C. Hurter. Transmogrification: Causal manipulation of visualizations. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 97–106, 2013. doi: 10.1145/2501988.2502046

[8] A. Cairo. *The Functional Art: An Introduction to Information Graphics and Visualization*. New Riders, 2012.

[9] S. K. Card, T. P. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Communications of the ACM*, 23(7):396–410, 1980. doi: 10.1145/358886.358895

[10] I. F. Cruz and P. S. Leveille. Implementation of a constraint-based visualization system. In *Proceedings of the IEEE Symposium on Visual Languages*, pp. 13–20, 2000. doi: 10.1109/VL.2000.874345

[11] T. Dwyer, K. Marriott, and M. Wybrow. Integrating edge routing into force-directed layout. In *International Symposium on Graph Drawing*, pp. 8–19, 2006. doi: 10.1007/978-3-540-70904-6_3

[12] J. Fogarty and S. E. Hudson. GADGET: A toolkit for optimization-based approaches to interface and display generation. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 125–134, 2003. doi: 10.1145/964696.964710

[13] L. Grammel, C. Bennett, M. Tory, and M.-A. Storey. A survey of visualization construction user interfaces. In *Short Paper Proceedings of EuroVis*, 2013. doi: 10.2312/PE.EuroVisShort.EuroVisShort2013.019-023

[14] D. Holten. Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 12(5):741–748, 2006. doi: 10.1109/TVCG.2006.147

[15] 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 (Proceedings of InfoVis)*, 23(1):491–500, 2017. doi: 10.1109/TVCG.2016.2598620

[16] 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 ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 123:1–123:13, 2018. doi: 10.1145/3173574.3173697

[17] Z. Liu, J. Thompson, A. Wilson, M. Dontcheva, J. Delorey, S. Grigg, B. Kerr, and J. Stasko. Data Illustrator: Gallery, 2018. `http://data-illustrator.com/gallery.php`.

[18] E.-J. Marey. *La méthode graphique dans les sciences expérimentales et principalement en physiologie et en médecine*. G. Masson, 1878.

[19] M. Mauri, T. Elli, G. Caviglia, G. Uboldi, and M. Azzi. RAWGraphs: A visualisation platform to create open outputs. In *Proceedings of the ACM Italian CHI Conference*, pp. 28:1–28:5, 2017. doi: 10.1145/3125571.3125585

[20] H. Mei, W. Chen, Y. Ma, H. Guan, and W. Hu. Viscomposer: A visual programmable composition environment for information visualization. *Visual Informatics*, 2(1):71–81, 2018. doi: 10.1016/j.visinf.2018.04.008

[21] H. Mei, Y. Ma, Y. Wei, and W. Chen. The design space of construction tools for information visualization: A survey. *Journal of Visual Languages & Computing*, 2017. doi: 10.1016/j.jvlc.2017.10.001

[22] G. G. Méndez, M. A. Nacenta, and S. Vandenheste. iVoLVER: Interactive visual language for visualization extraction and reconstruction. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 4073–4085, 2016. doi: 10.1145/2858036.2858435

[23] R. Metoyer, B. Lee, N. Henry Riche, and M. Czerwinski. Understanding the verbal language and structure of end-user descriptions of data visualizations. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 1659–1662, 2012. doi: 10.1145/2207676.2208292

[24] J. Moloney, A. Borning, and B. Freeman-Benson. Constraint technology for user-interface construction in ThingLab II. *ACM SIGPLAN Notices*, 24(10), 1989. doi: 10.1145/74878.74917

[25] C. Reas and B. Fry. Processing: Programming for the media arts. *AI & Society*, 20(4):526–538, 2006.

[26] D. Ren, M. Brehmer, B. Lee, T. Höllerer, and E. K. Choe. ChartAccent: Annotation for data-driven storytelling. In *Proceedings of the IEEE Pacific Visualization Symposium (PacificVis)*, pp. 230–239, 2017. doi: 10.1109/PACIFICVIS.2017.8031599

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

[28] L. C. Rost. What I learned recreating one chart using 24 tools. *Source*, 2016. `https://goo.gl/uGE5dc`.

[29] K. Ryall, J. Marks, and S. Shieber. An interactive constraint-based system for drawing graphs. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*, pp. 97–104, 1997. doi: 10.1145/263407.263521

[30] A. Satyanarayan and J. Heer. Lyra: An interactive visualization design environment. *Computer Graphics Forum (Proceedings of EuroVis)*, 33(3), 2014. doi: 10.1111/cgf.12391

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

[32] A. Satyanarayan, R. Russell, J. Hoffswell, and J. Heer. Reactive Vega: A streaming dataflow architecture for declarative interactive visualization. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 22(1):659–668, 2016. doi: 10.1109/TVCG.2015.2467091

[33] J. R. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

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

[35] S. Takahashi, S. Matsuoka, K. Miyashita, H. Hosobe, and T. Kamada. A constraint-based approach for visualization and animation. *Constraints*, 3(1):61–86, 1998. doi: 10.1023/A:1009708715411

[36] 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 ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 335:1–335:12, 2018. doi: 10.1145/3173574.3173909

[37] H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer, 2009.

[38] K. Wongsuphasawat, D. Moritz, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager: Exploratory analysis via faceted browsing of visualization recommendations. *IEEE Transactions on Visualization and Computer Graphics (Proceedings of InfoVis)*, 22(1):649–658, 2016. doi: 10.1109/TVCG.2015.2467191

[39] K. Wongsuphasawat, Z. Qu, D. Moritz, R. Chang, F. Ouk, A. Anand, J. Mackinlay, B. Howe, and J. Heer. Voyager 2: Augmenting visual analysis with partial view specifications. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp. 2648–2659, 2017. doi: 10.1145/3025453.3025768

[40] H. Xia, B. Araujo, T. Grossman, and D. Wigdor. Object-oriented drawing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pp. 4610–4621. ACM, New York, NY, USA, 2016. doi: 10.1145/2858036.2858075

[41] H. Xia, N. Riche, F. Chevalier, B. D. Araujo, and D. Wigdor. DataInk: Enabling direct and creative data-oriented drawing. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI)*, pp.

223:1–223:13, 2018. doi: 10.1145/3173574.3173797

[42] Android developers constraint library. https://developer.android.com/reference/android/support/constraint/ConstraintLayout.html.

[43] ChartBlocks. https://chartblocks.com.

[44] Datamatic. http://datamatic.io.

[45] Easelly. https://easel.ly.

[46] Flourish. https://flourish.studio.

[47] Flux. https://facebook.github.io/flux/docs/overview.html.

[48] iCharts. https://icharts.net.

[49] Infogram. https://infogram.com.

[50] Microsoft Power BI: Custom visuals. https://powerbi.microsoft.com/en-us/custom-visuals.

[51] Piktochart. https://piktochart.com.

[52] Plotly. https://plot.ly.

[53] Quadrigram. http://quadrigram.com.

[54] React. https://reactjs.org.

[55] TypeScript. http://typescriptlang.org.

[56] WebAssembly. https://webassembly.org.

[57] Caltrain's schedule. http://caltrain.com/schedules/weekdaytimetable.html.

[58] T. DeBold and D. Friedman. Battling infectious diseases in the 20th century: the impact of vaccines. *The Wall Street Journal*. http://graphics.wsj.com/infectious-diseases-and-vaccines.

[59] T. Herzog, World Greenhouse Gas Emissions: 2005, World Resources Institute, http://wri.org/resources/charts-graphs/world-greenhouse-gas-emissions-2005.

[60] T. Kekeritz. Weather Radials: An infographic on heat waves and snow storms in 35 cities around the globe. http://weather-radials.com.

[61] T. Kim. Best Bookshelf. http://tany.kim/best-bookshelf.

[62] D. Knuth. The Stanford GraphBase: A Platform for Combinatorial Computing. https://www-cs-faculty.stanford.edu/~knuth/sgb.html.

[63] Millennium Indicators, United Nations Statistics Division. http://mdgs.un.org/unsd/mdg/SeriesDetail.aspx?srid=749.

[64] National Centers for Environmental Information. https://www.ncdc.noaa.gov/cdo-web/search?datasetid=GHCND.

[65] F. Nightingale and W. Farr and A. Smith. *A contribution to the sanitary history of the British army during the late war with Russia*. John W. Parker and Son, 1859.

[66] Project Tycho. University of Pittsburgh, www.tycho.pitt.edu.

[67] StatCounter: Mobile operating system market share worldwide. http://gs.statcounter.com/os-market-share/mobile/worldwide.

[68] UCI Machine Learning Repository, 2017. http://archive.ics.uci.edu/ml.