



ChartDetective: Easy and Accurate Interactive Data Extraction from Complex Vector Charts

Damien Masson, Sylvain Malacria, Daniel Vogel, Edward Lank, Géry Casiez

► To cite this version:

Damien Masson, Sylvain Malacria, Daniel Vogel, Edward Lank, Géry Casiez. ChartDetective: Easy and Accurate Interactive Data Extraction from Complex Vector Charts. CHI 2023 - ACM Conference on Human Factors in Computing Systems, Apr 2023, Hamburg, Germany. 10.1145/3544548.3581113 . hal-04017638

HAL Id: hal-04017638

<https://hal.science/hal-04017638>

Submitted on 7 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

ChartDetective: Easy and Accurate Interactive Data Extraction from Complex Vector Charts

Damien Masson
Cheriton School of Computer Science,
University of Waterloo
Waterloo, Canada
dmasson@uwaterloo.ca

Sylvain Malacria*
Univ. Lille, Inria, CNRS, Centrale Lille,
UMR 9189 CRISTAL
Lille, France
sylvain.malacria@inria.fr

Daniel Vogel
Cheriton School of Computer Science,
University of Waterloo
Waterloo, Canada
dvogel@uwaterloo.ca

Edward Lank
Cheriton School of Computer Science,
University of Waterloo
Waterloo, Canada
lank@uwaterloo.ca

Géry Casiez^{†‡}
Univ. Lille, CNRS, Inria, Centrale Lille,
UMR 9189 CRISTAL
Lille, France
gergy.casiez@univ-lille.fr

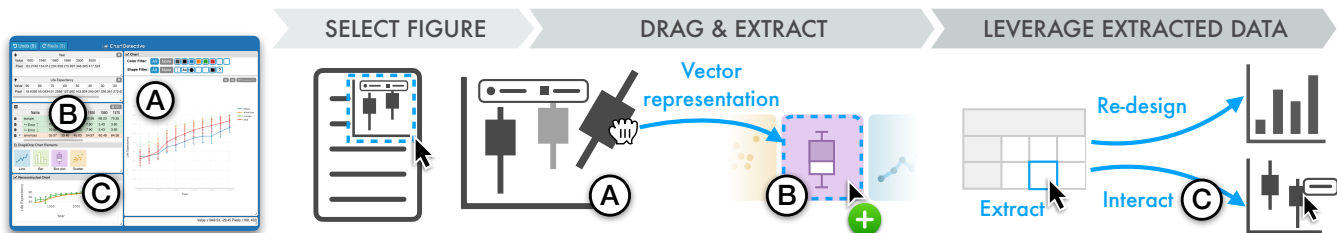


Figure 1: ChartDetective is a system capable of recovering a chart’s underlying data by leveraging its vector representation. Users select a vector chart and then (A, B) drag-and-drop elements that they wish to extract. (C) The extracted data can be leveraged for downstream tasks such as redesigning or interacting with the figure.

ABSTRACT

Extracting underlying data from rasterized charts is tedious and inaccurate; values might be partially occluded or hard to distinguish, and the quality of the image limits the precision of the data being recovered. To address these issues, we introduce a semi-automatic system leveraging vector charts to extract the underlying data easily and accurately. The system is designed to make the most of vector information by relying on a drag-and-drop interface combined with selection, filtering, and previsualization features. A user study showed that participants spent less than 4 minutes to accurately recover data from charts published at CHI with diverse styles, thousands of data points, a combination of different encodings, and

elements partially or completely occluded. Compared to other approaches relying on raster images, our tool successfully recovered all data, even when hidden, with a 78% lower relative error.

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools.**

KEYWORDS

data extraction, chart reverse-engineering, vector graphics

ACM Reference Format:

Damien Masson, Sylvain Malacria, Daniel Vogel, Edward Lank, and Géry Casiez. 2023. ChartDetective: Easy and Accurate Interactive Data Extraction from Complex Vector Charts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*, April 23–28, 2023, Hamburg, Germany. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3544548.3581113>

1 INTRODUCTION

Charts are often the preferred way of presenting data because they offload cognitive work to the visual system [45, 58]. For readers, accessing the numerical data of charts unlocks a broad range of applications: they can explore the data to better understand it [8, 20], generate new visualizations [57, 67, 71], redesign existing charts [29, 64, 73], answer questions [39, 40, 62], and generate

*Also with Cheriton School of Computer Science, University of Waterloo.

[†]Also with Institut Universitaire de France.

[‡]Also with Cheriton School of Computer Science, University of Waterloo.

textual summaries [13, 19, 60], as well as make existing visualizations accessible [11, 24], interactive [43, 76], and more informative [42, 44]. Further, researchers need this data to replicate analyses and compare results. However, despite the push for Open Science, not all scientists publish their data. This has been particularly true in the field of Human-Computer Interaction; an analysis of 509 CHI papers published before 2018 found that data was provided in less than 1% of cases [1]. Much of authors' hesitation comes from privacy concerns and little incentive or perceived benefits [80], suggesting they are unlikely to change their practises. Even if providing data became commonplace, the issue remains for previously published papers for which the data has long been lost.

One solution to recover data is to analyze charts, a practice called “*chart reverse-engineering*” [63]. By carefully locating series and inferring their position in the axis coordinates, the underlying data can be estimated. Of course, the more complex and dense the chart, the more tedious and error-prone the process becomes. Consequently, tools such as ChartSense [37] and WebPlotDigitizer [68] offer semi-automatic features to make the process easier. The core idea is to automate the recovery of the chart's structure, identifying every line, rectangle, and text in the image, and then inferring the element's role such as axis, series or legend.

However, most chart reverse-engineering approaches rely on *what can be seen* using pixels, and all previous tools operate only on raster images. Yet, vector graphics are commonly shared on the web [6] or in scientific publications [12, 13]. In fact, publishers often recommend, or even require the use of vector graphics for their scalability without loss of resolution [33, 75]. As a result, vector charts are ubiquitous, but their potential benefits are lost when existing systems rasterize them [2, 18, 37, 65].

This focus on raster images is a missed opportunity to improve chart reverse-engineering tools in terms of accuracy, usability and performance. With raster graphics, image resolution limits the quality of the data. Even assuming perfect accuracy from the recognition system, information is irreversibly lost, either because pixels do not capture the full resolution of the original data or because an element such as the legend hides part of the information. A pixel in a raster chart may represent a fraction of a unit or millions of units, even if the original data provided much finer resolution. The problem is aggravated when the raster image has compression artifacts or when the chart is dense with overlapping elements. In contrast, charts embedded as vector graphics are ideal for reverse-engineering because they encode the complete image structure and reference all components, even if hidden or overlapped, with an exact position and size.

Leveraging the extra benefits provided by the vector format is challenging because it requires understanding the specifics of the file format, knowledge of how the chart was generated, and the ability to access and operate vector graphics editors. In fact, little is known about how to recover the data from vector charts, how accurate the extracted data is compared to using raster images, and if the extra information encoded by the format can help the reverse-engineering process. To the best of our knowledge, only Choudhury et al. [12] describe an approach for separating curves from vector line charts. All other approaches focus exclusively on raster images.

In this paper, we introduce ChartDetective, a tool to extract underlying data from charts by leveraging their vector specification (Figure 1). The approach creates an interactive pipeline to extract data from a chart: a chart in a vector format, such as SVG and PDF, is processed and presented in a user interface where its underlying data is extracted using an integrated set of interactive selection, filtering, and previsualization mechanisms. Leveraging vector information has several advantages, enabling: novel features (e.g., filtering mechanisms); support for a wide variety of charts such as bar, line, scatter, and box plots; data recovery with greater accuracy and precision than other approaches; and extraction of charts exhibiting challenging characteristics such as diverse styles, thousands of data points, multiple encodings, and occluded elements.

Our work makes the following contributions: (1) Highlight of the advantages of vector graphics over raster images from a theoretical perspective. (2) Design and implementation of a tool to extract underlying data from vector charts demonstrating how using a vector representation enables new features and results in high-quality underlying data. (3) Experiment results showing the system is usable when extracting charts with challenging properties from real scientific publications. (4) A technical evaluation using a dataset of synthetic and in-the-wild charts validating superior accuracy of extracted data compared to existing approaches for raster images.

2 RELATED WORK

2.1 Involving Users to Improve Accuracy of Chart Data Extraction

A large body of work looked at fully-automatic pipelines for chart extraction, see Davila et al. [16] for a recent survey. While a fully automatic approach might be desirable, Davila et al. found that most approaches struggle when faced with charts in-the-wild. They list common characteristics of charts from PubMedCentral papers¹ noting “*despite being common, none of the works covered here dealt explicitly with these and other chart complexities*”. In fact, this motivated a chart mining competition held annually since 2019 [15]. Yet, as of 2022, the best approach (relying on large deep-learning models [50]) could recover only 69% of the data from charts in-the-wild, and the accuracy of the recovered data is not reported [17]. These poor performances may be attributed to the great diversity of charts [37] and the difficulty to obtain large annotated datasets, forcing automatic approaches to use artificial datasets and limit their scope to specific chart-styles and encodings. As a result, these approaches can fail when charts deviate even slightly from the training dataset [16].

When fully automatic approaches fall short, a common solution is to resort to manual or semi-automatic approaches to chart extraction [70]. Manual approaches to chart extraction such as Digitize [65] and Ycasd [26] rely on human annotations: after a calibration step to define the axes, the user needs to click on every data point in the chart. Semi-automatic approaches provide tools relying on computer-vision to facilitate and speed up manual extraction. For example, WebPlotDigitizer [68], Engauge Digitizer [25], and DataThief III [2] include automatic selection tools based on masking and colour filtering. The parameters of the underlying algorithms

¹<https://www.ncbi.nlm.nih.gov/pmc/>

(e.g., curve fitting, blob detector, line tracing) can be tweaked for better results. ChartSense [37] goes one step further by automatically extracting marks from a chart and requiring users to only specify critical features like the y-axis, and to check and correct the automatic selection. By involving users, these tools support a larger variety of charts.

In this work, we also use a semi-automatic approach to support a greater diversity of charts. However, our algorithms leverage structural information in vector graphics for more accurate extraction.

2.2 Leveraging Structured File Formats

Charts can be embedded using formats with more structure than raster images (e.g., HTML, D3.js). Perhaps because of their ubiquity, most of the work on chart data extraction revolves around rasterized charts. Indeed, any format displayable on-screen can be trivially converted to a raster image. However, for the task of chart extraction, the structure will need to be recovered through an often imperfect vectorization step.

Instead, others have circumvented this vectorization step by using formats which preserve structure and semantics. For example, D3 [7] and Vega-Lite [72] directly embed chart data and specifications, making it possible to redesign an existing visualization [29], create re-usable styles [30], search visualizations based on structure and style [31], answer questions [40] and generate visual explanations [40]. However, these approaches are limited to charts embedded in these specialized formats and do not support the broader spectrum of charts in formats such as PDF or SVG.

Although the semantic role of each shape is lost, the vector format constitutes a middle ground as it provides precise information about chart geometries and vector charts are widely used [6, 12]. Previous work leveraged the vector formats to classify visualizations [6, 74], create visualizations [57], retrieve visualizations based on their structure [47], and generate chart animations [25]. While some of these work try to recover information about visualizations including charts, they are not concerned with obtaining the precise underlying data. Instead, they use simplifications, for example, by assuming the data is already available [25] or by recovering only high-level characteristics [6, 74], often sufficient to accomplish their goals. Closest to our work, Choudhury et al. [12, 13] proposed a fully automatic pipeline that extracts information from line graphs in a PDF to generate natural language summary descriptions. However, as is common with automatic pipelines [16], their solution relies on strict assumptions. For example, the approach assumes each line series has a unique colour, axes lines are close to the image boundary, tick marks intersect with axes lines, and legends are close to curve paths.

In this work, we also focus on vector charts. However, we avoid making strict assumptions about chart layout and design. Instead, we adopt a semi-automatic approach to recover data from diverse charts using different styles (line, bar, scatter, and box plots).

3 BACKGROUND

Charts can be represented in two formats: raster images or vector graphics. Below, we review how data can be recovered from both formats and what are the theoretical advantages of vector graphics.

3.1 How Can Data be Recovered From Charts?

If not readily available, data can often be partially recovered from charts as cleaned and aggregated data subsets. Consider how an author creates a chart: first, a chart is generated using visualization tools such as matplotlib, ggplot, excel, or tableau in order to turn tabular data into a visualization like a bar chart that readers can quickly comprehend. The visualization is then exported either as a rendered image in a raster file format (e.g., PNG, JPG, BMP) or re-encoded into a vector file format (SVG, EPS, PDF), and shared by being included in a document or a web page. Recovering the data visualized by a chart is later accomplished by identifying each data point as a shape with a location and size, and transforming those into the local coordinate system defined by the chart axes. With vector charts, the position and size of each shape are recovered from the definition of vector graphics. In contrast, for raster images, the information has to be measured.

3.2 Advantages of Vector Graphics

There are several characteristics of vector graphics that make them advantageous for extracting chart data.

3.2.1 Higher Theoretical Precision. Because of how raster and vector graphics encode information, the precision of the data should be higher for vector graphics.

Raster images are composed of pixels. Given a raster image with one linear axis representing n units displayed on p pixels, one pixel represents n/p units. For example, a linear axis ranging from 0 to 1000 displayed over 100 pixels means that each pixel represents $1000/100 = 10$ units. Thus, a value of 0 is indistinguishable from a value of 9 because they are the same pixel. In other words, achieving a high accuracy when recovering the position of an element in a raster image requires a comparatively high resolution, inevitably increasing the size of the image file.

Vector graphics define shapes in real number coordinates. Thus, precision is limited by the number of decimals used to define coordinate positions and rounding errors due to coordinate transformations. To accommodate 32-bit processors, the PDF format uses the “single-precision floating-point format” and limits floating point numbers to approximately five decimals (ISO 32000-1:2008§C.2). The SVG file format encodes coordinates decimal numbers in strings and does not limit the number of decimals². As for transformations, both PDF and SVG specifications recommend using double-precision floating-point numbers (ISO 32000-1:2008§7.10.5.1) to reduce rounding errors when rendering. However, document viewers perform these operations and could use a higher precision format if needed. Thus, data values encoded in PDF charts can be theoretically recovered with up to five decimals with no impact on file size (the floating value will occupy 32 bits regardless of the number of decimals). The precision of values extracted from SVG charts is theoretically not bounded. Returning to our previous example, obtaining the same five-decimal level of precision with a raster chart ranging from 0 to 1000 would require 100 million pixels.

3.2.2 Recovery of Occluded Data. The vector graphic can include all geometry in a visualization, regardless of what is viewable in a final rendering. In particular, by default occluded shapes are

²<https://www.w3.org/TR/SVG/>

included, even if completely hidden when rendered. This enables the recovery of occluded data, a common issue with many charts [16, 37]. For example, a legend often hides part of a series, line series might overlap or cross each other, or a dense scatterplot might have clusters of indistinguishable points due to stacking.

3.2.3 Reduced Ambiguity. Classifying the role of elements in a chart is challenging in general [17, 37, 63] and raster image compression makes this even more difficult. For example, artifacts like blurry edges and irregular fill colours make elements hard to automatically separate. Additionally, rasterized text such as alphanumeric labels and annotations has to be located and recognized using Optical Character Recognition (OCR). With vector graphics, shapes are clearly identified and text is often directly accessible.

3.2.4 Ubiquitous. Most charting tools offer to export in vector formats, and this format is commonly used to share charts. On the web, the SVG format is natively supported and often preferred in a context in which pages are rendered on different screens of different sizes. Specifically, charts are commonly shared online in the SVG format [6]. Because of their scalability properties, vector graphics also represent a substantial proportion of all charts included in documents. For example, the popular PDF format allows the inclusion of different types of content such as text, fonts, raster images, and, specifically vector graphics [54]. In fact, major publishers such as IEEE [33] and Springer [75] recommend the use of vector graphics, because “*Creating and saving your graphics in vector format will ensure that your graphics appear as clearly as possible in your final published article*”, and “*Vector graphics (rather than rasterized images) should be used for diagrams and schemas whenever possible*”.

Quantifying the proportion of vector charts shared on different platforms and medias is difficult. As an example relevant to the HCI community, we counted³ that vector charts represented 38% of the 5,855 charts published in the last six years (2015-2021) of proceedings at the Conference on Human Factors in Computing Systems (CHI). In a similar analysis, Choudhury et al. found up to 70% of vector charts across the top-50 computer science conferences spanning all fields [12].

4 CHALLENGES AND DESIGN GOALS

We first summarize the main challenges when extracting data from vector graphics, then propose a set of system design goals. We use these to drive the design of ChartDetective, a new system leveraging vector graphics specifications to extract data from charts.

4.1 Challenges of Vector Chart Extraction

When Jung et al. [37] designed ChartSense to extract data from rasterized charts, they faced three main challenges: 1) chart styles are diverse; 2) visual entities can overlap; and 3) there is no off-the-shelf solution for text-region-detection. While reverse-engineering vector charts help with some of these challenges (see Section 3.2), some remain and new ones arise.

C1: Chart Diversity – Charts vary in the way data is represented as graphical shapes and style. To better understand this diversity,

³We manually annotated which figures were charts after extracting all figures in the six years of CHI papers. We calculated the proportion of those that did not contain a single raster element.

we manually reviewed and annotated the 5,855 charts we extracted from the proceedings of CHI from 2015 to 2021. The majority could be classified into 12 categories: bar chart (43.3%); line chart (25%); scatter plot (9.6%); box plot (9.4%); stacked bar (9.3%); heat map (0.9%); pie chart (0.8%); violin chart (0.7%); density plot (0.6%); radar chart (0.4%); and stacked density plot (0.1%). Some combined different encodings (2%), for example a bar chart combined with line series, or a box plot using scatter points. While difficult to quantify, we observed many variations in style, such as embellished charts [5], diverse colour palettes, annotations, and overlays. Tools to extract data from charts require flexibility to adapt to this diversity.

C2: Inconsistent Vector Specifications – A raster image is the result of exactly one configuration of pixels but vector graphics can be generated from a theoretically infinite number of shape arrangements. We define *shape* as a single geometric shape defined in a vector graphics language and *chart element* as a semantic element in a chart (e.g., a series, an axis, a legend). In our exploration, we found three relationships between shapes and chart elements (Figure 2).

- **ONE-TO-ONE** is when each shape maps to a unique chart element (e.g., a line maps to a series).
- **MANY-TO-ONE** is when multiple shapes represent a single chart element. For example, there were two common ways of representing dashed-lines: applying a “dashed-line” style to a line or using several smaller lines, one per “dash”. Even contiguous chart elements like a line series are not necessarily encoded as one polyline. For example, matplotlib has a tendency to split a single line series into smaller connected lines.
- **ONE-TO-MANY** is when a single shape represents multiple chart elements. The vector format is flexible enough to allow the definition of disconnected shapes (by using a *moveto* primitive when defining the path). This behaviour is often exploited to draw all the bars from a bar chart using the same shape, or drawing the legend and the series at the same time.

The challenge thus becomes how to divide or group shapes to get a one-to-one mapping in order to match humans’ perception of a single shape and make the extraction of data possible.

C3: Hidden Shapes – Vector graphics may contain shapes that are invisible in the rendered image such as shapes occluded by other shapes. However, some *hidden shapes* are meaningless and introduced by mistake using vector editing tools such as Inkscape. For example, we found text and annotations completely occluded by other shapes. They serve no purpose because they are invisible when rendered, and most likely result from mistakes. We also observed several examples of shapes hidden by modifying their colours instead of being removed. For instance, a user might hide axes or grid lines by setting the stroke and fill colour to match the background. However, these shapes remain in the vector specification. Of course, these hidden shapes should be ignored, but identifying them systematically is difficult because they take various forms.

C4: Rendered Text – Text in vector graphics can be specified using text-specific vector graphics command to position a string of characters using attribute like font and size or by forming letters using geometric shapes. For the latter, the text cannot be directly recovered as each letter is represented visually not semantically.

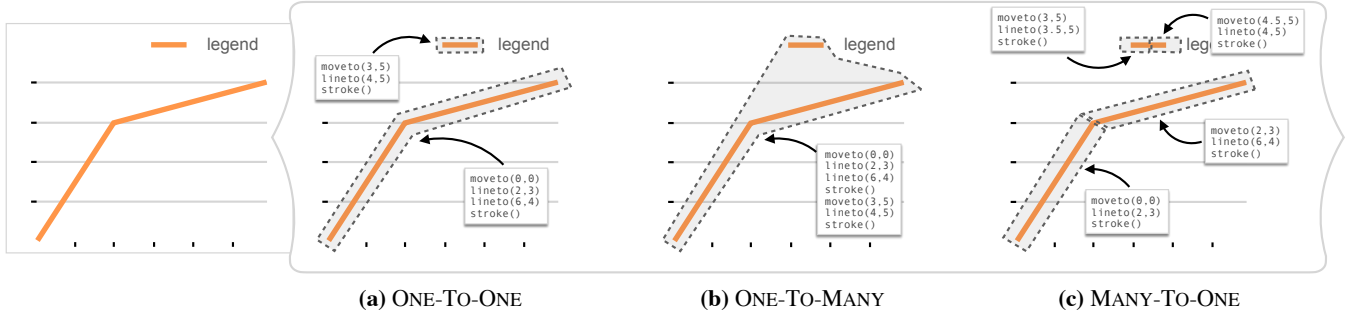


Figure 2: The same chart can be formed with different vector shape arrangements: (a) ONE-TO-ONE when shapes match semantic elements; (b) ONE-TO-MANY when one shape maps to multiple semantic elements; and (c) MANY-TO-ONE when many shapes map to a single semantic element.

In that case, identifying text regions and extracting the text then becomes as challenging as with raster images.

4.2 Design Goals

Using those main challenges, we formulate a set of design goals to guide the development of our system. They consider previous limitations regarding the lack of flexibility and the poor performance when faced with charts *in-the-wild* [15, 16], leverage the advantages of semi-automatic rather than fully-automatic solutions [37, 70], and follow recommendations for mixed-initiative systems [32, 59].

D1: Maximize Data Accuracy – For reliability and repeatability, high accuracy means a low relative error between the extracted values and ground truth values in the original data. Previous work seldom reports accuracy, yet accuracy was necessarily limited by the resolution of the raster images [16]. We consider accuracy as the utmost priority and aim to leverage vector graphics to obtain high-fidelity data.

D2: Support Diversity – Across various forms of charts (see Challenge 1), flexibility is required to support different ways of encoding data (e.g., line, bar, scatter, box) and variations in style (e.g., colour, size, shape, organization). In practice, this means making few assumptions [16], and likely incorporating user interaction in the extraction process to disambiguate alternative extraction outcomes.

D3: Minimize User Interaction – While a fully automatic approach would be ideal, in practice, the user has to be involved—if only to check that the result is correct. Previous work can be placed on a continuum from fully manual [26, 65] to semi-automatic [2, 18, 36, 37, 52, 68] to fully automatic [16]. Our goal is to minimize user involvement by automating tedious and long tasks.

D4: Simplify Verification – Checking data extracted from a large, dense chart could entail verifying thousands of data cells. Users should be able to quickly check that the chart was accurately extracted, identify mistakes (if any), and correct them.

5 CHARTDETECTIVE

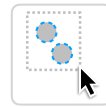
ChartDetective is a system to extract underlying data from vector charts by leveraging the vector information. A live version of ChartDetective is accessible online: <http://ns.inria.fr/loki/chartdetective>. Below, we detail ChartDetective’s interface and functionalities. The functionalities try to tackle each challenge and design goal identified in 4. As such, direct references are added in parenthesis whenever a functionality responds to a challenge or design goal.

5.1 Interface

ChartDetective has two interfaces: one to upload a file or document and select a chart to extract, and one to extract data from a chart. The data extraction interface (Figure 3) consists of three main views: 1) The **Data Table** displaying the data extracted from the charts so far; 2) The **Chart View** showing the chart undergoing data extraction; and 3) the **Reconstructed Chart** which recreates portion of the original chart using data extracted so far. The interface deliberately presents the information in multiple views [81]; all views are showing at all times, side-by-side, and the interface can be re-arranged by dragging and resizing the three views to adapt to different screen resolutions.

5.2 Selection of Chart Elements

The selection of an element in the chart initiates the extraction process. ChartDetective proposes several ways to perform a selection even when the targets are small or occluded (C2, C3).



Simple Selection – Using the **Chart View**, a user selects shapes composing the chart by either clicking on them *one-by-one* or by using a marquee selection through a mouse-dragging motion for multiple selection and small hard-to-select objects. As is common with vector software, users can also add and remove elements from their selection by holding the **shift** key.

Shapes under the cursor or included in the marquee selection, are highlighted to preview the selection. A blue animated dashed outline highlights shapes because it is salient for shapes of different sizes and colours. Once selected, shapes are grouped, surrounded by a blue rectangle, and become draggable.

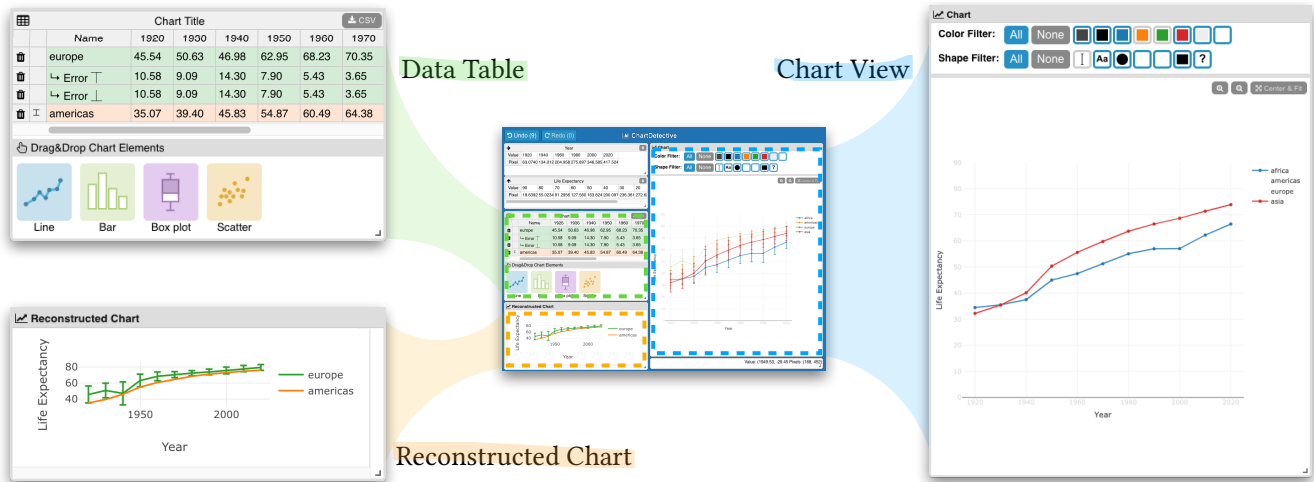
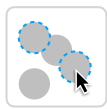


Figure 3: ChartDetective is composed of three views: the **Chart View** showing the chart being extracted; the **Data Table** with the tables for X and Y axes and the extracted values; the interactive **Reconstructed Chart**.



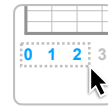
Fine-grained Selection – While simple selections work well for basic charts, selecting elements becomes tedious and slow as the number of data points increases or their size decreases. In a worst-case scenario, selection may be impossible for overlapping elements (C3). There are two mechanisms to help in these situations.

- Users can zoom-in using the mouse wheel and pan by dragging while holding the **space bar**. This helps when small elements are hard to select or distinguish when the full chart is viewed.
- There are also two view filtering mechanisms: a colour filter (Figure 4b) and a shape filter (Figure 4c). When a chart is loaded, all unique colours and shapes used by the chart are identified and displayed as filter buttons. The user can toggle these colours and shapes to remove or add associated shapes from the rendered chart. This helps particularly with dense charts. For example, users can isolate a specific series in a scatter plot by filtering per shape (e.g., only keeping circle-shaped markers or green-coloured dots). Filters can be combined like a logical “AND” (e.g.,

to select only red circle-shaped markers, see Figure 4d). Once only the elements of interest are left, selection is easier and can be done with a quick marquee selection (D3).

5.3 Extraction of Data

To extract data, ChartDetective relies on drag-and-drop interactions where elements selected in the **Chart View** are dropped in the appropriate area of the **Data Table**. Depending on the drop zone, different algorithms are used to extract and analyze the shapes (D3).



Extract Axes – Extracting an axis is accomplished by selecting at least two tick marks in the chart, then dropping them on the corresponding horizontal or vertical axis of the **Data Table**. Typically, extracting both the X and Y axis requires two drag-and-drop interactions. Extracting the axis title requires another drag-and-drop by first selecting the title in the chart view, then dropping it on the title of the data table.

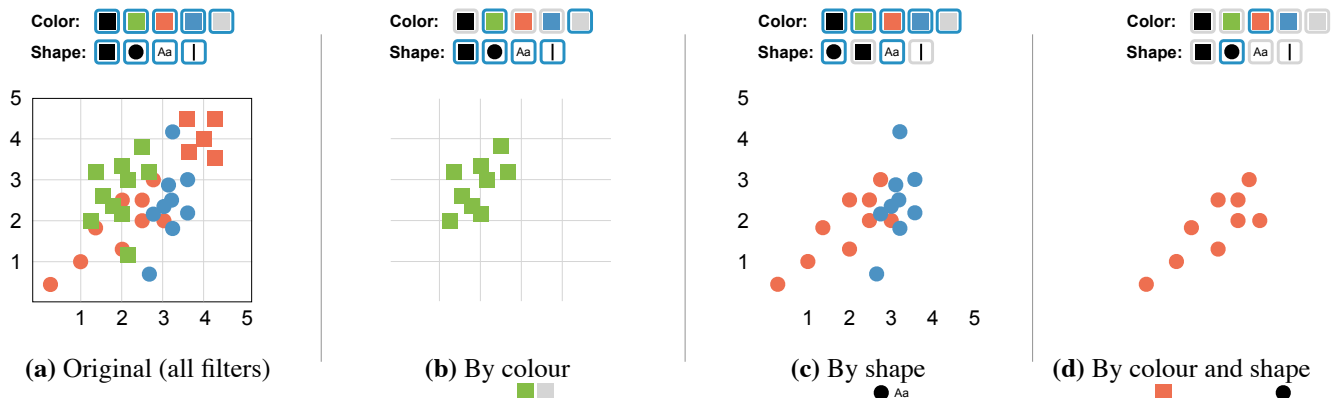
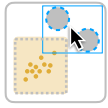







Figure 4: (a) Shapes in the can be filtered (b) by colour or (c) by shape. (d) Filters can also be combined.



Extract Data Points – To add a new series shapes are dropped on one of four zones indicated by a title, an icon, and a colour: line , bar , box plot , scatter 

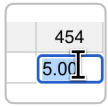
 Each drop zone corresponds to a unique encoding of the data. ChartDetective supports four types of data visualization used by the four most common chart types in CHI publications: bar charts, line charts, box plots, and scatter plots (C1). Supporting other visualizations is a matter of writing the corresponding algorithm which takes a vector shape as input and outputs data points.

Once the data is extracted, a new row is added to the table using the colour of the series to be easily identifiable (D4). The corresponding shapes in the **Chart View** become translucent and unselectable to allow the selection of shapes potentially hidden behind. These translucent shapes also act as a visual guide to immediately see what remains to be selected.

Whenever possible, ChartDetective automatically mines the name of the extracted series by searching for a legend in the figure. The algorithm works in two steps: 1) find another shape with the same colour as the one extracted; and 2) extract the text at the right of the shape and use it as the name of the series. While this algorithm has obvious failure cases (e.g., black-and-white charts), in practice this high-level assumption is more often correct. Errors can be corrected by dropping the legend directly in the cell indicating the series name.



Extract Error Bars – ChartDetective supports the extraction of error values represented as bars or “whiskers”. In ChartDetective, error bars are always linked to an existing series. As such, after extracting a series and adding a new row to the **Data Table**, the user can select error bars in the chart and drop them on a zone at the left of the corresponding series row. The error bars are matched to series data points based on order and the upper and lower bounds are calculated.

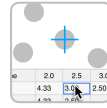


Modify Extracted Values – Extracted series and axes in the **Data Table** can be modified or removed. The **Data Table** is editable like a spreadsheet: clicking a cell edits a value. Data points can also be added to an existing series by first selecting the shapes on the **Chart View** and then dropping them directly on the row of an existing series. The added shapes and extracted data points are incorporated into the series.

Similarly, the title of all tables and the name of all rows can be edited manually (C4) or updated by dropping text selected from the **Chart View**. The selected letters are merged to form words and sentences when dropped into the **Data Table**. If no text glyphs are found in the selection, the text is recovered using an Optical Character Recognizer (C4). This is done by first rendering the selected shapes on a blank canvas before passing it to the recognizer.

5.4 Verify Results

Mistakes can happen when extracting the data from charts, for example: data points can be missed, elements may be incorrectly interpreted as data, and legends might be mismatching. In addition to providing a view of the data table and an option to export it as a CSV file, ChartDetective provides passive and active mechanisms to verify the success of the extraction (D1, D4).



Reconstructed Overlay – Users can actively check that the data cells match their expectation by examining a data point overlay updated when positioning the mouse cursor above cells in the **Data Table**. Two different overlays are shown: 1) when hovering over axis values, a vertical or horizontal bar shows the extent of the axis where ticks were extracted; 2) when hovering over a data point from a series, a blue cross is rendered at the corresponding position in the chart. This allows the user to verify that a data point is correctly extracted and inspect the mapping between series shown in the chart and series in the data table (D1, D4). For example, to find and fix a potential mismatch in the legend.



Reconstructed Interactive Chart – As data is extracted, a second chart is progressively reconstructed in the **Reconstructed Chart** view. To make verification easier, the reconstructed chart shares the same visualization and style such as colours and marker shapes. This allows the user to glance at the **Reconstructed Chart** and compare it with the **Chart View**: a perfect extraction creates a perfect match between the two views (D1, D4). The reconstructed chart is interactive; users can get information on hover (e.g., exact values), hide series, and zoom in on a particular area of the chart. Additionally, the chart can be exported to an HTML file, allowing the generation of interactive charts directly after extracting a static chart.

5.5 Getting Started and Interacting

ChartDetective supports traditional and advanced interaction mechanisms in terms of signifiers, feedback, and feed-forward to support exploratory behaviours and help users get started.

5.5.1 Discoverability. ChartDetective follows common guidelines to promote discovery such as limiting the number of commands available at any given time, making commands distinguishable, and providing continuous feedback [55]. Because interactions relying on drag-and-drop can be hard to discover [51], we took special care to inform users *when* they could initiate a drag-and-drop interaction and *where* they could drop their selection. New users unaware of the drag-and-drop interface are likely to click one of the icons below the **Data Table**. Doing so opens a ToolTip [27] showing a brief explanation and animation of the drag-and-drop interaction to extract new series. Additionally, when a drag-and-drop interaction is initiated, possible drop zones are highlighted based on the shapes being dragged (Figure 5). For example, a drag selection of text elements causes an overlay over all zones accepting text like the table title and series' names (Figure 5B). Conversely, a drag selection containing shapes will only highlight zones accepting shapes, and hovering over a zone accepting only text turns the zone red and the pointer becomes a “prohibition sign” (Figure 5C) to mark the zone as invalid.

5.5.2 Safe Exploration. The interface is designed so that users understand the consequence of their actions and that all actions can be undone. All commands provide detailed feedback after being executed through notifications at the bottom of the screen. For example, when the data extraction fails, a message is shown to indicate what might be the reason (e.g., “too few shapes in the

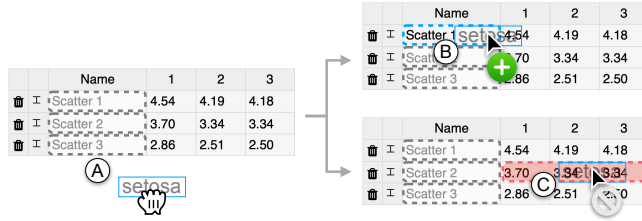


Figure 5: During drag-and-drop interactions, (A) drop zones compatible with the selection are highlighted to indicate where elements can be dropped. (B) The drop zone turns blue if hovered with a selection of the proper type, and (C) red otherwise.

selection.”). Additionally, there is a complete undo/redo mechanism to recover from any action.

5.6 Implementation

ChartDetective is implemented in TypeScript using React⁴ for the interface, Plotly.js⁵ to reconstruct an interactive version of the chart, Tesseract.js⁶ to recover rendered text, and PDF.js⁷ to parse and render PDFs. While there are multiple vector formats, internally, we use the PDF representation as it is the most low-level and any vector format can be trivially converted to PDF. As such, ChartDetective supports all PDF documents conforming to the ISO-32000 (PDF) specification, and also natively support the SVG format using svg2pdf⁸. All modern web browsers are supported. The full source code is available online: <http://ns.inria.fr/loki/chartdetective>.

5.6.1 Access to Vector Specifications. We modified PDF.js to store and provide low-level vector graphics commands after parsing the PDF. This includes retrieving the full list of shapes forming each page of a PDF and determining their final location and size after processing all transformations, group positioning, clipping, and buffered rendering. This allows the selection of a sub-part of the PDF by only keeping shapes completely within a defined area.

5.6.2 Pixel-Perfect Selection. Because shapes can take complex forms, we implemented selection using a “hit-test buffer” for pixel-perfect selection with little computational cost. This means charts are rendered twice: once to show a preview and once in an off-screen buffer in which each shape is assigned a unique colour. Shape selection is achieved by retrieving the colour of the pixel underneath the pointer. The hit-test buffer is only redrawn when absolutely necessary such as a change of zoom or when a shape filter changes.

5.6.3 Shape Filtering. The colour filter is relatively straightforward to implement: create a list of colours to filter then hide shapes with any of those colours. The shape filter requires computing a form descriptor: a vector of numbers describing a shape. To create effective shape filters, the form descriptor must not be too specific

while also not too general that all shapes would match. We use a normalized Freeman chain-code with 8 connectivity [22]; this descriptor is invariant in translation, scale, and rotation, and is robust against slight variations of aspect ratios.

5.6.4 Shape Alignment and Attributes. While ChartDetective makes no assumptions on the style of the charts, it relies on attributes which are fundamental to the way information is visualized. The pseudo-code is provided in Appendix A.1. All shape selections are first sub-divided to recover a consistent specification (C2) before passed to the extractors.

- (1) **Alignment:** The centroid of shapes such as axis ticks, line series, scatter plot markers is used to recover their position. For example, it is assumed that lines go over the centre of the data points. In a vertical bar chart, the top of a bar is used to get the associated value.
- (2) **Grid line:** If a grid line is found close to the tick, its position is used instead, because we found it to be slightly more accurate.
- (3) **Box plot:** It is assumed that box plots use the original and widely used representation first introduced by Tukey [78]: The inner quartiles are represented as a rectangle including any stroke outline when calculating values. The median is a line inside this rectangle, and any stroke outline is ignored when calculating the value.

6 USABILITY STUDY

We conducted a user study to see if the current implementation of ChartDetective fulfills our design goals in terms of supporting diversity (D1) and minimizing user interaction (D3). This study focuses on *usability*, answering the question: can participants use ChartDetective? A follow-up study measures the quality of the extracted data when compared to other tools (Section 7).

6.1 Participants

We recruited 13 participants (22 to 34 age range, mean = 27.8, 7 identified as male and 6 identified as female)⁹. We screened participants for basic knowledge of charts: all participants were familiar with bar charts, line charts, box plots, scatter plots and error bars (self-assessed on a 5-point scale). Remuneration was \$15 CAD.

6.2 Dataset of Charts to Extract

We extracted 12 charts from the proceedings of CHI from 2015 to 2021. We consider the four most popular chart types at CHI: line charts, bar charts, scatter plots, and box plots. Further, we collected three charts per type, according to complexity:

- **SIMPLE:** Few series and few data points that all use the same encoding.
- **COMPOUND:** Two or more encodings are combined to represent data points. For example, a bar chart with lines, or a box plot with scatter points.
- **DENSE:** Large number of series and data points, but all data points use the same encoding.

COMPOUND and DENSE charts have been notoriously difficult to extract using existing systems [16, 37] and thus pose a real challenge.

⁴<https://reactjs.org/>

⁵<https://plotly.com/javascript/>

⁶<https://tesseract.projectnaptha.com/>

⁷<https://mozilla.github.io/pdf.js/>

⁸<https://github.com/yWorks/svg2pdf.js/>

⁹Our study was reviewed and approved by our institutional research ethics board. Consent was collected from all participants.

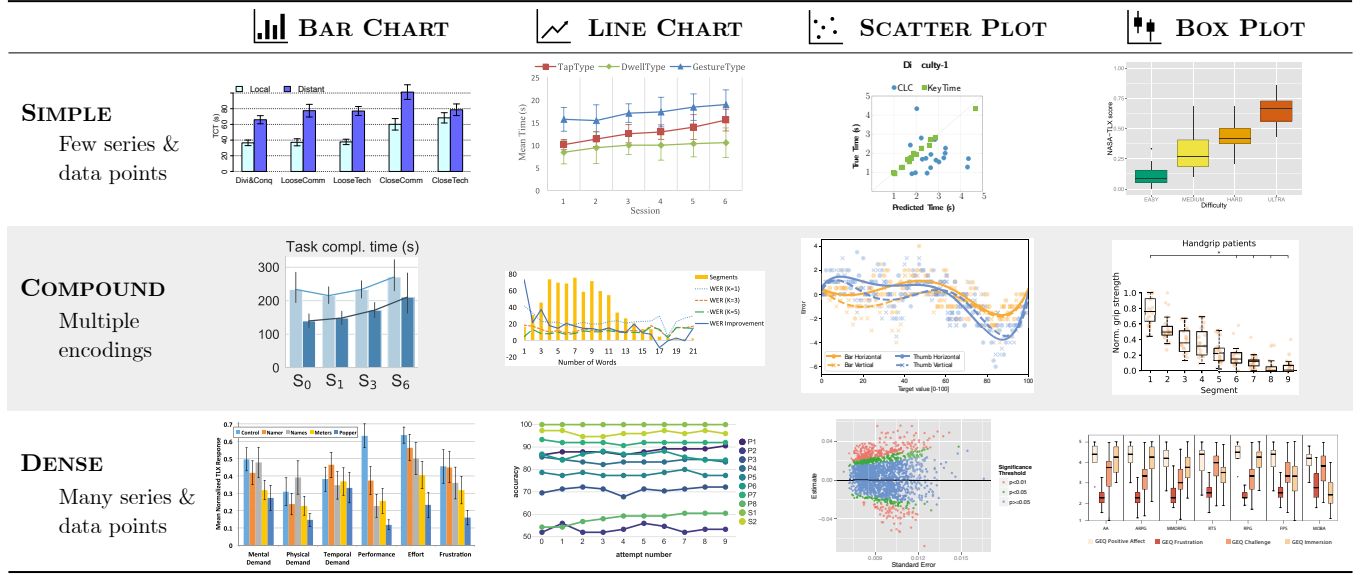


Figure 6: Charts that participants had to extract during the usability study. Charts were extracted from the following CHI proceedings (left-to-right, top-to-bottom): [49] [83] [46] [23] [9] [79] [14] [4] [21] [38] [56] [34]

Note that using our categories above, previous work has exclusively been tested on SIMPLE charts. We chose charts randomly amongst those fitting these criteria to maximize diversity while remaining ecologically valid (see selected charts in Figure 6). All charts were used by participants unaltered from the original paper; as a result, some are missing titles or legends, have overlapping or hidden elements, and some have grouped or separated shapes.

6.3 Procedure and Design

Participants took part remotely. After watching a two-minute video tutorial demonstrating the use of ChartDetective, participants were asked to extract the underlying data of 12 charts as accurately as possible and to think-aloud while doing so. The experimenter only intervened during the first four SIMPLE charts to answer questions and guide participants if necessary; participants worked independently for the remaining eight COMPOUND and DENSE charts. Participants advanced to the next chart by pressing a “Done” button or after five minutes, whichever came first. After each SIMPLE chart, the experimenter asked participants to identify 1) what was difficult; 2) what was easy; 3) what was tedious; 4) what was fast; and 5) what was slow. For each answer, participants also rated its importance on a 5-point scale.

We recorded the participant’s screen and microphone, as well as a log of interactions with ChartDetective and the final extracted data. After the session, participants completed a questionnaire including a System Usability Scale (SUS) [35]. Finally, the experimenter conducted a semi-structured interview.

The order of the charts varied across participants: The four SIMPLE charts were always first, followed by the eight remaining charts. The charts order was counter-balanced within these two groups.

Overall, each participant extracted data from 4 (CHART TYPE) \times 3 (CHART COMPLEXITY) = 12.

6.4 Results

6.4.1 Success Rate. To test the success rate in terms of usability, we compare participants’ data to data extracted by one author before the experiment. The reasoning is twofold: first, we want to isolate the usability aspect and are not concerned by the fidelity of the data extracted by our tool at this stage, only by how well can participants use ChartDetective; second, the success rate can be directly interpreted as a measure of how close participants were to using the tool like an expert user, represented by the author who extracted the data. As such a series from the participant data is matched with a series from the author data (using a best-fit approach). We then classify each data point (i.e., cell in the data table) of each series in one of the following four categories.

- **CORRECT** (\checkmark), for a data point that is expected (i.e., present in ground-truth data) and that is strictly equal to the ground-truth value.
- **INCORRECT** (\times), for a data point that is expected but is not equal to the ground-truth value.
- **MISSING** ($-$), for a data point that is expected but was not extracted (i.e., present in ground-truth but not in the participants’).
- **UNWANTED** ($+$), for a data point that was not expected (i.e., present in participant data but not in ground-truth).

We measure success rate by calculating the rates of these four categories. For the CORRECT and INCORRECT rate, we divide the count by the minimum between the number of data points in ground-truth and the number of data points in the participant data. For the MISSING rate, we divide the count by the number of points in ground-truth. For the UNWANTED rate, we divide the count by the number of points in the participant data.

Overall Success Rate – Overall, participants extracted charts with high success: 99% (SD=5.9) of the extracted data were CORRECT, with only 0.2% (SD=1.5) INCORRECT data points (D1).

Complexity	Bar Chart				Line Chart				Scatter Plot				Box Plot				Total			
	✓	x	-	+	✓	x	-	+	✓	x	-	+	✓	x	-	+	✓	x	-	+
SIMPLE	100	0	0	1.3	100	0	0	0	100	0	0	1	100	0	0.4	0	100	0	0.1	0.6
COMPOUND	100	0	0	1.5	100	0	0	0.1	89.6	1.5	12.3	9.2	99.8	0.1	0.2	0	97.4	0.4	3.1	2.7
DENSE	100	0	0	0	100	0	0.7	0	100	0	0	0	98.6	0.4	1.8	3.9	99.7	0.1	0.6	1
Total	100	0	0	0.9	100	0	0.2	0	96.5	0.5	4.1	3.4	99.5	0.2	0.8	1.3	99	0.2	1.3	1.4

Table 1: Breakdown of the success rate when comparing the series extracted by participants to the series of the ground-truth data. All values are percentages. CORRECT (✓), INCORRECT (x), MISSING (-), and UNWANTED (+).

Table 1 presents the breakdown of the results. In fact, participants achieved perfect success rate in terms of CORRECT data for all bar charts and line charts, and above 98% for all other charts. The only exception being the COMPOUND scatter plot with only 89.6% (SD=17.7) of CORRECT data. Below, we further investigate the cause of some of these results.

Confusion for COMPOUND Scatter Plot – We found that the lower scores for the COMPOUND scatter plot were due to participants misunderstanding the chart. In fact, the data points extracted by participants were CORRECT, but not separated in series as it should have been. Both P1 and P5 interpreted different series as one single series (e.g., grouping all yellow dots as one single series, instead of distinguishing between crosses and circles). Because we calculate success rate by matching one series to another, if a series is missing, its data points count as INCORRECT. Similarly, the extra data points merged within the same series count as UNWANTED data. If, instead, we look for data points independent of series, the percentage of CORRECT data for the COMPOUND scatter plot reaches 99%.






Filtering Causing UNWANTED Data – Participants sometimes selected the legend as part of a series. For example, with the SIMPLE scatter plot, all participants made the selection of series easier using colour-filtering. However, doing so isolates the data points making the legend appear as part of the series. As a result, three participants selected circles from the legend, creating UNWANTED data.




Selection Difficulties with Box Plots – Box plots required a precise selection of the whole element at once. However, when the boxes were close to other elements (e.g., the axis in the COMPOUND box plot), some participants inadvertently included other elements as part of the box, creating an INCORRECT data point.

6.4.2 Time. On average, participants extracted charts in 3min 6s (SD=1min 18s) (D3) Only 2 participants were not able to complete extractions within 5 minutes: P4 could not extract all error bars for the DENSE bar chart and P10 did not have time to select the last two blue lines in the COMPOUND scatter plot. The most dense charts were not necessarily the slowest to extract. For instance, extracting the DENSE scatter plot and its 2,000 data points took only 1min 58s (SD=23s). But overall, DENSE charts were the slowest (M=3min 17s, SD=1min 22s), followed by SIMPLE charts (M=3min 12s, SD=1min 10s) and COMPOUND charts (M=2min 50s, SD=1min 19s). Still, all average times were well under 4 minutes, confirming that ChartDetective minimizes user interaction enough to allow the extraction of charts in reasonable time (D3).

6.4.3 Error Bars and Series’ Names. Participants correctly extracted 98.4% (SD=11) of the error bars and 82.8% (SD=34.1) of series names. These results were calculated on a subset of charts considering that not all charts had error bars or legends. It is unclear why the series names score is lower; some participants did not extract the series names for no apparent reasons, even though they were aware of the feature as they all did it for SIMPLE charts.

6.4.4 Usability. On average, the System Usability Score was 90 (Mdn=90, SD=4.2). For reference, a System Usability score above 85 is considered excellent [3].

On a 5-point scale, participants rated all features of Chart-Detective as useful (4 or above): participants “strongly agree” on the usefulness of the colour filter (Mdn=5, SD=0, ) , the shape filter (Mdn=5, SD=0.7, ) , the selection system (Mdn=5, SD=0.3, ) and the reconstructed chart (Mdn=5, SD=0.7, ) . Additionally, participants “agree” that the overlay was useful (Mdn=4, SD=1.5, ) .

Regarding participants self-assessed performance with Chart-Detective, they all agreed that they could extract and reconstruct charts accurately (Mdn=5, SD=0.5, ) and that they were in control of what they wanted to extract (Mdn=5, SD=0.6, ) . Finally, they all agreed that they would like to use the system again (Mdn=5, SD=0.4, ) .

6.4.5 Strategies. While the tasks were identical across participants, they sometimes adopted different strategies to extract the data.

Filter to Isolate, to Declutter, or to Guide? – All 13 participants used the filters but we observed three distinct strategies: filtering to *isolate* only the element to select (i.e., only one active filter); filtering to *declutter* the image by removing the few elements that were preventing a selection; filtering to *guide* the selection by going through each colour one-by-one (sometimes multiple times) to be sure not to miss any series. Participants using the *declutter* strategy had the advantage of preserving visual context. For example, it made it easier to distinguish marks that are part of the legend and would have looked like data had the *isolate* strategy been used.

Step Order – The order in which to perform the extraction was often a trade-off between speed and cognitive load. Some participants extracted all series (e.g., bars, lines) before moving on to error bars or legends, making the selection process easier by repeating the same task and reusing the same filters. Others preferred to extract the error bars and the legend right after extracting the corresponding series, making it easier to match a series with its

meta-data. Participants would often decide on a strategy based on chart complexity.

Selection Strategy – Participants either selected elements precisely, often one-by-one, using the zoom-in function if necessary, or they made a first rough selection then refining it by using SHIFT to add to or remove elements. Some participants also relied on the ghost shape mechanism to speed up the selection of the last series: once a series is extracted, its shapes become unselectable, meaning that when only one series is left, the shapes that are selectable will necessarily belong to the last series.

6.4.6 Comments. Overall, participants were positive about ChartDetective and its functionality. Below, we group participant comments from the interview and during the study around a set of themes that were frequently mentioned. Due to the use of semi-structured interviews, some of these themes were only mentioned by a subset of participants.

Learnability – A few participants commented on learnability. All agreed that the tool was quick to learn.

P6 – "I liked that the cognitive load was pretty low, like, it was super fast to learn... I got markedly better after like 2, 3 tries. I really liked that it had a lot of the traditional settings and feel to it."

P12 – "I've been using it for less than an hour and I already feel at ease"

Specifically, P7, P11, and P13 commented about the drag and drop interface saying it made the interaction easy due to the visual feedback of what is being dragged and where it can be dropped.

P7 – "That is actually quite useful that it shows you the... sort of see-through thing you're dragging."

P11 – "Drag'n'drop is truly useful. It's super clean, like you can easily select and then you immediately see where you can drop."

P13 – "I liked how it was organized, how you could... like it was easy to have that one navigation bar on the side and pull everything over and see it appear on the [Reconstructed] chart below."




Most appreciated features – An overwhelming majority of participants commented about the colour and shape filters, most cited them as one of their favourite features.

P8 – "Filters are super useful. Really facilitate the task. Some [charts] would even be impossible [to extract] without."

P12 – "The filters, I really thought it was a killer-feature. Your chart is super crowded, you ask yourself: «Wow, how am I gonna do that, it's too difficult», I do two clicks, then it becomes super easy."



Other features were less often mentioned by participants as something they liked: the reconstructed chart (P2, P11 P13), the ghost shapes (P3, P7, P8), the overlay (P8), how the legend is sometimes automatically matched with series (P2), and the coloured table (P12).

What was tedious, difficult or slow? – After each SIMPLE chart, participants were asked what they found tedious, difficult or slow and how much using a 5-point scale. The most frequently mentioned

difficulty was the selection of elements (P5, P6, P7, P10, P12, P13), although they still rated it as relatively easy on average (from very hard to very easy: Mdn=4, SD=.83, ). Specifically, the selection of error bars were most often mentioned as moderately tedious (from extremely to not at all: Mdn=4, SD=.71, ) and slow (from very fast to very slow: Mdn=3, SD=.75, .

P1 – "Selecting error bars. That was difficult, just because they were overlapping."

P8 – "Selecting error bars [was slow] because you have to select them a bit like... one-by-one to [...] distinguish them between series."

What was easy or fast? – Similarly, we asked participants what they found easy and fast. The drag-and-drop interface was most commonly cited as being easy (from very hard to very easy: Mdn=5, SD=0, ). The selection after applying filters was most often mentioned as fast (from very fast to very slow: Mdn=1, SD=.37, .

P3 – "The drag and drop... that tool is easy to understand and use. And it is easy to isolate the data you want to collect [using the filters]."

P2 – "Selecting the points, thanks to the filtering, it was really fast".

Accuracy perception – While participants were highly accurate overall, P7 mentioned that the artificiality of the task might have had an impact on the quality of the data extracted.

P7 – "I'm not sure how accurately I actually covered... copied the charts, I have to say. Because the image [Reconstructed chart] was relatively small, and I did not spend a lot of time looking at data points if they were correct or not. It was more like a «meh» roughly looks the same, fine, cool. [...] Obviously because it is not data that I am invested in so I don't care if it is accurate or not."

7 DATA QUALITY STUDY

The goal of this study is to measure the quality of the data obtained from vector charts and ChartDetective relative to what could be obtained using rasterized images and existing tools (D1). Quality is defined as how similar the extracted data is compared to the original data that was used to create the chart.

7.1 Dataset

We create a new dataset of charts for which we know the *exact* underlying data. To cover a wide diversity of chart styles and chart generators, we mix generated charts with charts from CHI papers:

- **Generated Charts:** We generated four different charts (1 bar chart, 1 line chart, 1 scatterplot and 1 boxplot) with four different chart generators: Microsoft Excel for Mac version 16.61, Python matplotlib version 3.5.1, Javascript plotly version 4.10.0 and R ggplot2 version 3.3.6. All 16 generated charts use a dataset on life expectancy and GDP per country obtained from GapMinder¹⁰. Each chart visualization presented different information: line charts show the evolution of the life expectancy over the years for four regions of the world and with error bars; scatter plots

¹⁰<https://www.gapminder.org/data/>

show the life expectancy depending on a country’s GDP; and box plots show the life expectancy per region and for male and female). Generators used the default style parameters.

- **Extracted Charts:** Using our dataset of papers with charts published at CHI between 2015-2020, we used a script to find those with vector charts and with data available on Open Science Framework¹¹ (OSF). Only 23 papers fulfilled this criteria (74 papers had an OSF link, but 44 of those did not contain qualifying vector charts, and 7 had no data in their OSF repository). Using reasonable effort, we cleaned and recreated the data used by charts in 14 different papers and extracted between 1 and 3 different charts per paper. The final study dataset counted 26 charts (13 bar charts, 6 scatter plots, 5 line charts, and 2 box plots).

7.2 Baseline

Like ChartSense [37], we use WebPlotDigitizer as our baseline. Other tools either do not provide their source code¹² or a working implementation [37], do not provide a full pipeline to obtain the data from charts [12, 13, 63], or are limited in the styles and types of charts that they support [13]. Moreover, our comparison here focuses on the best achievable results using vector graphics compared to raster images. In that regard, the result obtained with raster images should be comparable across tools. Thus, in the rest of this section we use “rasterized charts” to refer to charts extracted using WebPlotDigitizer.

7.3 Procedure

One author with hours of experience with both ChartDetective and WebPlotDigitizer extracted all charts from our dataset as accurately as possible using both tools. The author had no time limitation and ensured the data was as accurate as possible. To use WebPlotDigitizer, extracted charts were rasterized at 300 dot-per-inch (DPI) which is considered high-resolution and recommended by IEEE [33]. Generated charts were obtained from chart generators and directly outputted as PNGs for WebPlotDigitizer (300DPI), and PDFs for ChartDetective.

7.4 Results

Dataset	n	Vector Charts	Raster Charts
Generated			
excel	4	0.24% (0.19)	0.37% (0.21)
matplotlib	4	0% (0)	0.16% (0.08)
ggplot2	4	0% (0)	0.13% (0.12)
plotly	4	0% (0)	0.13% (0.09)
Extracted	26	0.13% (0.27)	0.68% (0.69)
Total	42	0.11% (0.23)	0.50% (0.60)

Table 2: Average relative error of the values obtained from vector or raster charts. Standard deviation shown between parenthesis.

¹¹<https://osf.io/>

¹²On request, authors of ChartSense could not provide their source code due to proprietary reasons.

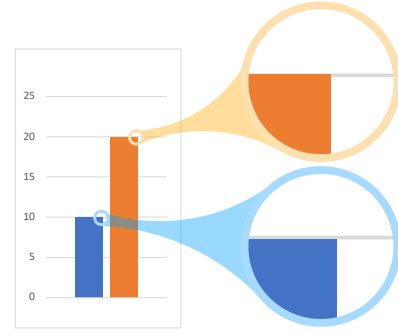


Figure 7: Minimal example of a chart generated by Microsoft Excel and for which the data is imprecisely depicted. Despite the bars representing exact values (10 and 20), they do not consistently line up with corresponding grid lines: the grid line is either below (blue bar) or above (orange bar).

Like ChartSense [37], relative error measures how close the extracted data is to ground-truth:

$$Relative\ Error = \left| \frac{v_{groundtruth} - v_{extracted}}{v_{groundtruth}} \right|$$

We assume the chart represents the data precisely and accurately, so relative error is solely attributed to the extraction tool. Note that the expert always made sure no data points were missing. Thus, the ground truth data and extracted data had the same number of points and in the same order. We calculated the relative error of each pair of data points from the ground truth and extracted data, and then aggregated them using the mean of all the relative errors. This measure corresponds to how close the extracted data is to the ground truth.

Overall, data extracted from vector charts using ChartDetective had a significantly lower relative error than data extracted from rasterized charts using WebPlotDigitizer (0.11% vs 0.5%, Student’s t-test $p < .05$). This is more than 4 times lower, corresponding to a factor of 78%. Table 2 breaks down the relative error for tool and dataset.

Effect of Generator – The data obtained with ChartDetective from generated vector charts was identical to the ground-truth with the exception of Excel charts (relative error matplotlib: 0% SD=0, ggplot2: 0% SD=0, plotly: 0% SD=0, excel: 0.24% SD=0.19). After further investigation, it appears that charts generated with Excel are using some approximations and do not perfectly represent data. While the problem exists with all types of charts such as line, box plots and scatter, the issue becomes obvious by generating bar charts with exact real values. Examining the SVG description of bar charts generated by Excel, positions of bar tops is inconsistent even when data is a series of real and regularly spaced values (e.g., 50, 60, 70). This is demonstrated by zooming into a bar chart to see how bars do not line up consistently with corresponding grid lines (Figure 7). We verified this behaviour with macOS Excel (version 16.61) and Windows Excel (version 2205).

8 DISCUSSION

Despite the importance of extracting high-fidelity data, approaches to chart reverse-engineering predominantly focus on rasterized formats, limiting the accuracy of the data obtained. We provide theoretical and empirical evidence showing that extracting charts using their vector representation has advantages that can lead to an improved quality of the extracted data. We also detail the design and implementation of the ChartDetective system demonstrating how vector information can be used to provide new features, and how it can be processed to obtain underlying data. Through a usability study, participants found the system highly usable and were capable of extracting even the most challenging charts. A second study demonstrated that extracting a chart using its vector representation lead to higher accuracy of data than when extracting the same chart in raster format and using existing tools.

Comparison to ChartSense. Jung et al. also use the relative error to compare their ChartSense system to WebPlotDigitizer [37]: their system achieved 0.7% whereas WebPlotDigitizer achieved 0.81%. For comparison, we found a relative error of 0.11% with ChartDetective and of 0.5% with WebPlotDigitizer. Differences in our methodology likely explain different results for WebPlotDigitizer: Jung et al. obtained their result from a user study with 16 participants whereas our results were obtained by an expert user. Furthermore, our dataset was different: Jung et al. used line and bar charts found on Google Images with at most two series and nine marks per series. In contrast, we used a dataset of generated charts and charts published at CHI, including charts with hundreds of marks. Regardless, both studies suggest that relative error below 0.5% may be out of reach for raster chart data extraction using existing approaches and that ChartDetective fulfills its goal of maximizing accuracy (D1).

Control of Anchor Points. ChartDetective differs from other manual and semi-automatic tools in that users select whole shapes and let the system decide how to handle them to extract the data (D3). In contrast, other systems often rely on users directly specifying *anchor points* to define the exact point depicted by a marker, even if this marker takes various forms and sizes. One advantage of our system generated anchor points is reduced standard deviation (D1): the same shape selection always results in the same value. In contrast, giving users control over anchor points inevitably results in lower precision due to selection errors or simply because users have to “guess” the centroid of shapes. This can vary greatly depending on the style (e.g., thick lines or large markers) and the forms of the shapes [28, 69]. We advocate for a shape-selection approach because users can reliably select shapes (see Study 1) and that our assumptions regarding anchor points were valid across a diverse set of charts (see Study 2).

Open Science at CHI. Corroborating the findings of previous work [1, 80], and further motivating the need for chart extraction methods, we experienced first-hand the difficulty of obtaining data related to CHI papers. Of the 3,673 papers published at CHI from 2015 to 2020, only 74 papers contained an OSF link (2%). Our automatic mining approach likely missed data published using other methods like custom webpages. But more importantly, even within these papers, we could not always reproduce the charts. This was sometimes due to missing data (the OSF link contained other material) or because

only raw data was provided without guidance to reproduce processed data used in charts. For example, the cleaning procedure, aggregation method, and formulas applied were missing. Additionally, there was often a mismatch between the data names in the chart and labels in the raw data.

8.1 Limitations and Future Work

8.1.1 Support For Raster Charts. A large portion of charts remain embedded as raster graphics and cannot leverage the benefits provided by ChartDetective. A tempting alternative to using tools such as WebPlotDigitizer [68] could be to vectorize raster charts so that they can be used with ChartDetective. New state-of-the-art vectorization algorithms [48] might provide the best approximation for the location of shapes representing chart elements and possibly help disentangle overlapping shapes. However, many benefits provided by “original” vector charts would be lost and the quality of the input raster image will limit the vectorization process. While a vectorization approach can extend our system to rasterized charts, it seems unlikely to provide substantial benefits over using raster-based extraction tools.

8.1.2 Optical Character Recognition of Rendered Text. When the user study was conducted, participants had an earlier version of ChartDetective where rendered text could not be automatically retrieved, and required participants to enter it manually. While we found rendered text in vector charts to be relatively rare in practise, we added OCR support in ChartDetective (C4). Preliminary tests suggest excellent performance: we rendered all text in the charts used in the usability study, and the OCR engine was able to recover 97.5% of all characters correctly. A more extensive evaluation is needed to make definitive conclusions.

8.1.3 Diversity of Chart Styles. The many ways in which charts represent and encode data is one of the main difficulties faced by reverse-engineering approaches [15–17, 37]. We choose to evaluate our tool on real charts published at CHI that exhibited challenging properties like high density, overlapping shapes and mixing encoding (C1, D2). We encourage other work to do so as well, considering such charts are abundant in practice. Of course, our dataset is not universally representative. First, we only examine charts in the HCI research community, but others communities might have different practices regarding charts. Second, the HCI community is arguably more aware of good data visualization practices. This is both a strength of our dataset because HCI charts may be more creative in their use of marks and visual channels, but also a weakness because charts may be clearer and exhibit fewer flaws [10].

Further, we cannot guarantee that our tool is general enough to handle all charts. ChartDetective relies on fundamental attributes of charts and on the structure of the vector representation. We verified these were reasonable and applied to major chart generators, but charts could use different encoding structures. Moreover, our system focused on the four most common data visualizations (bar, line, scatter, and box plots), but more work is needed to implement extractors for other types such as stacked bar, violin, and pie charts.

8.1.4 Automatic Selections Through Suggestions. Considering the limitations of previous work, a goal of ChartDetective was to preserve some user-control to allow the selection of specific data, the

support of complex and diverse charts (D2), and the verification of the results (D4). While the time it took participants to extract charts was under 4 minutes, this could be further shortened by automating some tasks (D3). For example, participants often mention the selection of error bars as the most tedious. We believe this could be improved through suggestions generated by continuously learning from user actions: after selecting the error bars for one series, the system could learn to recognize the characteristic shapes composing error bars and suggest repeating the action for other series. General selection suggestions could also be learned from the community and be offered prior to the user first selection, only based on the shapes identified in the chart. We believe this active learning approach with suggestions is the best compromise between incorporating automation while preserving high controllability [70]

8.2 Applications of ChartDetective

ChartDetective can power several downstream tasks that require access to accurate data when only the charts are readily available. Specifically, readers interested in re-analyzing the results presented in a chart can use ChartDetective to extract underlying data and then use it as input to their analysis, or to compare their results against. Other applications include the use of ChartDetective as an intermediate step to re-design existing charts: a chart found online or in a document might benefit from being redesigned if it is poorly structured or deceptive [10, 53, 61], uses a representation ineffective to support users' task [58, 66], is overblown and shows too many data points [77], or is not accessible because of its colour palette and style [41, 82]. For all these scenarios, the chart can be loaded in ChartDetective to let users select only the data of interest. Users can then export the underlying data to be visualized in an authoring tool, or, they can use the automatically reconstructed interactive chart and tweak its specification such as changing its aspect ratio to avoid deceptive charts that exaggerate or undermine slopes [61], its scale to remove truncated axes [10], its colour to make it print- and colorblind-safe [41], and its encodings to make it align with the user's task [58, 66].

8.3 Takeaway for Chart Authors

Through this work, we hope to encourage authors to share their figures as vector graphics. Besides facilitating data extraction, vector graphics have numerous advantages: high quality at any resolution; more accessible; easily modifiable; and typically smaller in size. All major chart generators have an option to export charts as vector graphics which can then be directly imported into documents such as \LaTeX , MS Word documents, or web pages. We also recommend authors carefully choose chart generators because they can differ in how well they represent data. For example, we found that MS Excel generated less accurate charts than either matplotlib, ggplot2, or plotly. Although these differences are invisible to the naked-eye (Figure 7) they are a concern in the context of chart extraction.

9 CONCLUSION

We presented ChartDetective, a tool within the pipeline to extract data from charts using their vector representation. Through theoretical and experimental evidence, we showed the benefits of using vector graphics to extract data compared to using raster images. We

identified the challenges associated with building such a system, demonstrated opportunities for novel features, and evaluated its usability and quality of the extracted data. Recovering complete and accurate data is the first step to tackle downstream tasks such as redesigning existing charts or making them dynamic, interactive, and accessible. Besides helping users recover this data, we hope our system serves as a building block to leverage the wealth of information currently locked inside static visualizations.

ACKNOWLEDGMENTS

This research received ethics clearance from the Office of Research Ethics, University of Waterloo. This work was made possible by NSERC Discovery Grant 2018-05187, the LAI R app, and the Agence Nationale de la Recherche (Discovery, ANR-19-CE33-0006).

REFERENCES

- [1] Jacob Abbott, Haley MacLeod, Novia Nurain, Gustave Ekobe, and Sameer Patil. 2019. *Local Standards for Anonymization Practices in Health, Wellness, Accessibility, and Aging Research at CHI*. Association for Computing Machinery, New York, NY, USA, 1–14. <https://doi.org/10.1145/3290605.3300692>
- [2] B. Tummers. 2006. DataThief III. <https://www.datathief.org/>.
- [3] Aaron Bangor, Philip T. Kortum, and James T. Miller. 2008. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction* 24, 6 (2008), 574–594. <https://doi.org/10.1080/10447310802205776>
- [4] Lilianna Barrios, Pietro Oldrati, David Lindlbauer, Marc Hilty, Helen Hayward-Koennecke, Christian Holz, and Andreas Lutterotti. 2020. A Rapid Tapping Task on Commodity Smartphones to Assess Motor Fatigability. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–10.
- [5] Scott Bateman, Regan L. Mandryk, Carl Gutwin, Aaron Genest, David McDine, and Christopher Brooks. 2010. Useful Junk? The Effects of Visual Embellishment on Comprehension and Memorability of Charts. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2573–2582. <https://doi.org/10.1145/1753326.1753716>
- [6] Leilani Battle, Peitong Duan, Zachery Miranda, Dana Mukusheva, Remco Chang, and Michael Stonebraker. 2018. Beagle: Automated Extraction and Interpretation of Visualizations from the Web. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–8. <https://doi.org/10.1145/3173574.3174168>
- [7] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D³ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics* 17, 12 (Dec. 2011), 2301–2309. <https://doi.org/10.1109/TVCG.2011.185>
- [8] Bret Victor. 2011. Explorable Explanations. <http://worrydream.com/ExplorableExplanations/>.
- [9] Daniel Buschek, Martin Z rn, and Malin Eiband. 2021. The Impact of Multiple Parallel Phrase Suggestions on Email Input and Composition Behaviour of Native and Non-Native English Writers. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3411764.3445372>
- [10] Alberto Cairo. 2019. *How Charts Lie: Getting Smarter about Visual Information* (illustrated edition ed.). WW Norton, New York.
- [11] Jinho Choi, Sanghun Jung, Deok Gun Park, Jaegul Choo, and Niklas Elmqvist. 2019. Visualizing for the Non-Visual: Enabling the Visually Impaired to Use Visualization. *Computer Graphics Forum* 38, 3 (2019), 249–260. <https://doi.org/10.1111/cgf.13686>
- [12] Sagnik Ray Choudhury, Shuting Wang, and C. Lee. Giles. 2016. Curve Separation for Line Graphs in Scholarly Documents. In *2016 IEEE/ACM Joint Conference on Digital Libraries (JCDL)*. IEEE, Newark, NJ, USA, 277–278.
- [13] Sagnik Ray Choudhury, Shuting Wang, and C. Lee. Giles. 2016. Scalable Algorithms for Scholarly Figure Mining and Semantics. In *Proceedings of the International Workshop on Semantic Big Data - SBD '16*. ACM Press, San Francisco, California, 1–6. <https://doi.org/10.1145/2928294.2928305>
- [14] Ashley Colley, Sven Mayer, and Niels Henze. 2019. Investigating the Effect of Orientation and Visual Style on Touchscreen Slider Performance. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–9. <https://doi.org/10.1145/3290605.3300419>
- [15] Kenny Davila, Bhargava Urala Kota, Srirangaraj Setlur, Venu Govindaraju, Christopher Tensmeyer, Sumit Shekhar, and Ritwick Chaudhry. 2019. ICDAR 2019 Competition on Harvesting Raw Tables from Infographics (CHART-Infographics).

- In *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, Sydney, NSW, Australia, 1594–1599. <https://doi.org/10.1109/ICDAR.2019.00203>
- [16] Kenny Davila, Srirangaraj Setlur, David Doermann, Bhargava Urala Kota, and Venu Govindaraju. 2021. Chart Mining: A Survey of Methods for Automated Chart Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43, 11 (Nov. 2021), 3799–3819. <https://doi.org/10.1109/TPAMI.2020.2992028>
- [17] Kenny Davila, Chris Tensmeyer, Sumit Shekhar, Hrituraj Singh, Srirangaraj Setlur, and Venu Govindaraju. 2021. ICPR 2020 - Competition on Harvesting Raw Tables from Infographics. In *Pattern Recognition. ICPR International Workshops and Challenges (Lecture Notes in Computer Science)*, Alberto Del Bimbo, Rita Cucchiara, Stan Sclaroff, Giovanni Maria Farinella, Tao Mei, Marco Bertini, Hugo Jair Escalante, and Roberto Vezzani (Eds.). Springer International Publishing, Cham, 361–380. https://doi.org/10.1007/978-3-030-68793-9_27
- [18] Markus Demleitner. 2010. Dexter.
- [19] Longxu Dou, Guanghui Qin, Jinpeng Wang, Jin-Ge Yao, and Chin-Yew Lin. 2018. Data2Text Studio: Automated Text Generation from Structured Data. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, Brussels, Belgium, 13–18. <https://doi.org/10.18653/v1/D18-2003>
- [20] Pierre Dragicevic, Yvonne Jansen, Abhira Sarma, Matthew Kay, and Fanny Chevalier. 2019. Increasing the Transparency of Research Papers with Explorable Multiverse Analyses. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–15. <https://doi.org/10.1145/3290605.3300295>
- [21] David R. Flatla, Alan R. Andrade, Ross D. Teviotdale, Dylan L. Knowles, and Craig Stewart. 2015. ColourID: Improving Colour Identification for People with Impaired Colour Vision. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 3543–3552. <https://doi.org/10.1145/2702123.2702578>
- [22] Herbert Freeman. 1961. On the Encoding of Arbitrary Geometric Configurations. *IRE Transactions on Electronic Computers* EC-10, 2 (1961), 260–268. <https://doi.org/10.1109/TEC.1961.5219197>
- [23] Jérémy Frey, Maxime Daniel, Julien Castet, Martin Hachet, and Fabien Lotte. 2016. Framework for Electroencephalography-based Evaluation of User Experience. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 2283–2294. <https://doi.org/10.1145/2858036.2858525>
- [24] Jinglun Gao, Yin Zhou, and Kenneth E. Barner. 2012. View: Visual Information Extraction Widget for Improving Chart Images Accessibility. In *2012 19th IEEE International Conference on Image Processing. IEEE*, Orlando, FL, USA, 2865–2868. <https://doi.org/10.1109/ICIP.2012.6467497>
- [25] Tong Ge, Yue Zhao, Bongshin Lee, Donghao Ren, Baoquan Chen, and Yunhai Wang. 2020. Canis: A High-Level Language for Data-Driven Chart Animations. *Computer Graphics Forum* 39, 3 (June 2020), 607–617. <https://doi.org/10.1111/cgf.14005>
- [26] Arnd Gross, Sibylle Schirm, and Markus Scholz. 2014. Ycasd – a Tool for Capturing and Scaling Data from Graphical Representations. *BMC Bioinformatics* 15, 1 (June 2014), 219. <https://doi.org/10.1186/1471-2105-15-219>
- [27] Tovi Grossman and George Fitzmaurice. 2010. ToolClips: An Investigation of Contextual Video Assistance for Functionality Understanding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. Association for Computing Machinery, New York, NY, USA, 1515–1524. <https://doi.org/10.1145/1753326.1753552>
- [28] Tovi Grossman, Nicholas Kong, and Ravin Balakrishnan. 2007. Modeling Pointing at Targets of Arbitrary Shapes. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 463–472. <https://doi.org/10.1145/1240624.1240700>
- [29] Jonathan Harper and Maneesh Agrawala. 2014. Deconstructing and Restyling D3 Visualizations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology (UIST '14)*. Association for Computing Machinery, New York, NY, USA, 253–262. <https://doi.org/10.1145/2642918.2647411>
- [30] Jonathan Harper and Maneesh Agrawala. 2018. Converting Basic D3 Charts into Reusable Style Templates. *IEEE Transactions on Visualization and Computer Graphics* 24, 3 (March 2018), 1274–1286. <https://doi.org/10.1109/TVCG.2017.2659744>
- [31] Enamul Hoque and Maneesh Agrawala. 2020. Searching the Visual Style and Structure of D3 Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 26, 1 (Jan. 2020), 1236–1245. <https://doi.org/10.1109/TVCG.2019.2934431>
- [32] Eric Horvitz. 1999. Principles of Mixed-Initiative User Interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems the CHI Is the Limit - CHI '99*. ACM Press, Pittsburgh, Pennsylvania, United States, 159–166. <https://doi.org/10.1145/302979.303030>
- [33] IEEE. 2022. Resolution and Size. <https://journals.ieeeauthorcenter.ieee.org/create-your-ieee-journal-article/create-graphics-for-your-article/resolution-and-size/>
- [34] Daniel Johnson, Lennart E. Nacke, and Peta Wyeth. 2015. All about That Base: Differing Player Experiences in Video Game Genres and the Unique Case of MOBA Games. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 2265–2274. <https://doi.org/10.1145/2702123.2702447>
- [35] Patrick W. Jordan, B. Thomas, Ian Lyall McClelland, and Bernard Weerdmeester (Eds.). 1996. *Usability Evaluation In Industry* (1st edition ed.). CRC Press, London ; Bristol, Pa.
- [36] Joseph A. Huwaldt. 2001. Plot Digitizer. <http://plotdigitizer.sourceforge.net/>
- [37] Daekyoung Jung, Wonjae Kim, Hyunjo Song, Jeong-in Hwang, Bongshin Lee, Bohyoung Kim, and Jinwook Seo. 2017. ChartSense: Interactive Data Extraction from Chart Images. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6706–6717. <https://doi.org/10.1145/3025453.3025957>
- [38] Hernisa Kacorri, Kris M. Kitani, Jeffrey P. Bigham, and Chieko Asakawa. 2017. People with Visual Impairment Training Personal Object Recognizers: Feasibility and Challenges. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 5839–5849. <https://doi.org/10.1145/3025453.3025899>
- [39] Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. 2018. DVQA: Understanding Data Visualizations via Question Answering. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE*, Salt Lake City, UT, 5648–5656. <https://doi.org/10.1109/CVPR.2018.00592>
- [40] Dae Hyun Kim, Enamul Hoque, and Maneesh Agrawala. 2020. Answering Questions about Charts and Generating Visual Explanations. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376467>
- [41] N. W. Kim, S. C. Joyner, A. Riegelhuth, and Y. Kim. 2021. Accessible Visualization: Design Space, Opportunities, and Challenges. *Computer Graphics Forum* 40, 3 (2021), 173–188. <https://doi.org/10.1111/cgf.14298>
- [42] Nicholas Kong and Maneesh Agrawala. 2012. Graphical Overlays: Using Layered Elements to Aid Chart Reading. *IEEE Transactions on Visualization and Computer Graphics* 18, 12 (Dec. 2012), 2631–2638. <https://doi.org/10.1109/TVCG.2012.229>
- [43] Nicholas Kong, Marti A. Hearst, and Maneesh Agrawala. 2014. Extracting References between Text and Charts via Crowdsourcing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. Association for Computing Machinery, New York, NY, USA, 31–40. <https://doi.org/10.1145/2556288.2557241>
- [44] Chufan Lai, Zhixian Lin, Ruikang Jiang, Yun Han, Can Liu, and Xiaoru Yuan. 2020. Automatic Annotation Synchronizing with Textual Description for Visualization. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376443>
- [45] Jill H. Larkin and Herbert A. Simon. 1987. Why a Diagram Is (Sometimes) Worth Ten Thousand Words. *Cognitive Science* 11, 1 (1987), 65–100. <https://doi.org/10.1111/j.1551-6708.1987.tb00863.x>
- [46] Luis A. Leiva, Daniel Martín-Albo, Réjean Plamondon, and Radu-Daniel Vatavu. 2018. KeyTime: Super-Accurate Prediction of Stroke Gesture Production Times. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Montreal QC Canada, 1–12. <https://doi.org/10.1145/3173574.3173813>
- [47] Haotian Li, Yong Wang, Aoyu Wu, Huan Wei, and Huamin Qu. 2022. Structure-Aware Visualization Retrieval. In *CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, 1–14. <https://doi.org/10.1145/3491102.3502048>
- [48] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. 2020. Differentiable Vector Graphics Rasterization for Editing and Learning. *ACM Transactions on Graphics* 39, 6 (Nov. 2020), 193:1–193:15. <https://doi.org/10.1145/3414685.3417871>
- [49] Can Liu, Olivier Chapuis, Michel Beaudouin-Lafon, and Eric Lecolinet. 2016. Shared Interaction on a Wall-Sized Display in a Data Manipulation Task. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 2075–2086. <https://doi.org/10.1145/2858036.2858039>
- [50] Weihong Ma, Hesuo Zhang, Shuang Yan, Guangshun Yao, Yichao Huang, Hui Li, Yaqiang Wu, and Lianwen Jin. 2021. Towards an Efficient Framework for Data Extraction from Chart Images. *arXiv:cs/2105.02039*
- [51] Eva Mackamul. 2022. Improving the Discoverability of Interactions in Interactive Systems. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 57, 5 pages. <https://doi.org/10.1145/3491101.3503813>
- [52] Mark Mitchell, Baurzhan Muftakhidinov, and Tobias Winchen et al. 2015. Engauge Digitizer. <http://markummitchell.github.io/engauge-digitizer/>
- [53] Damien Masson, Sylvain Malacria, Géry Casiez, and Daniel Vogel. 2023. Chagraph: Interactive Generation of Charts for Realtime Annotation of Data-Rich Paragraphs. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 18. <https://doi.org/10.1145/3544548.3581091>
- [54] Damien Masson, Sylvain Malacria, Edward Lank, and Géry Casiez. 2020. Chameleon: Bringing Interactivity to Static Digital Documents. Association for

- Computing Machinery, New York, NY, USA, 1–13. <https://doi.org/10.1145/3313831.3376559>
- [55] Damien Masson, Jo Vermeulen, George Fitzmaurice, and Justin Matejka. 2022. Supercharging Trial-and-Error for Learning Complex Software Applications. In *CHI Conference on Human Factors in Computing Systems (CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 381, 13 pages. <https://doi.org/10.1145/3491102.3501895>
- [56] J. Nathan Matias, Sayamindu Dasgupta, and Benjamin Mako Hill. 2016. Skill Progression in Scratch Revisited. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 1486–1490. <https://doi.org/10.1145/2858036.2858349>
- [57] Gonzalo Gabriel M ndez, Miguel A. Nacenta, and Sebastien Vandenheste. 2016. iVoLVER: Interactive Visual Language for Visualization Extraction and Reconstruction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. Association for Computing Machinery, New York, NY, USA, 4073–4085. <https://doi.org/10.1145/2858036.2858435>
- [58] Tamara Munzner. 2014. *Visualization Analysis and Design* (1st edition ed.). A K Peters/CRC Press, Boca Raton.
- [59] Donald A. Norman. 1994. How Might People Interact with Agents. *Commun. ACM* 37, 7 (July 1994), 68–71. <https://doi.org/10.1145/176789.176796>
- [60] Jason Obeid and Enamul Hoque. 2020. Chart-to-Text: Generating Natural Language Descriptions for Charts by Adapting the Transformer Model. In *Proceedings of the 13th International Conference on Natural Language Generation*. Association for Computational Linguistics, Dublin, Ireland, 138–147.
- [61] Anshul Vikram Pandey, Katharina Rall, Margaret L. Satterthwaite, Oded Nov, and Enrico Bertini. 2015. How Deceptive Are Deceptive Visualizations? An Empirical Analysis of Common Distortion Techniques. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*. Association for Computing Machinery, New York, NY, USA, 1469–1478. <https://doi.org/10.1145/2702123.2702608>
- [62] Panupong Pasupat and Percy Liang. 2015. Compositional Semantic Parsing on Semi-Structured Tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Association for Computational Linguistics, Beijing, China, 1470–1480. <https://doi.org/10.3115/v1/P15-1142>
- [63] Jorge Poco and Jeffrey Heer. 2017. Reverse-Engineering Visualizations: Recovering Visual Encodings from Chart Images. *Comput. Graph. Forum* 36, 3 (June 2017), 353–363. <https://doi.org/10.1111/cgf.13193>
- [64] Jorge Poco, Angela Mayhua, and Jeffrey Heer. 2018. Extracting and Retargeting Color Mappings from Bitmap Images of Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (Jan. 2018), 637–646. <https://doi.org/10.1109/TVCG.2017.2744320>
- [65] T. Poisot. 2011. The Digitize Package: Extracting Numerical Data from Scatterplots. *The R Journal* 3, 1 (2011), 25–26.
- [66] Ghulam Jilani Quadri and Paul Rosen. 2022. A Survey of Perception-Based Visualization Studies by Task. *IEEE Transactions on Visualization and Computer Graphics* 28, 12 (Dec. 2022), 5026–5048. <https://doi.org/10.1109/TVCG.2021.3098240>
- [67] Donghao Ren, Tobias H llerer, and Xiaoru Yuan. 2014. iVisDesigner: Expressive Interactive Design of Information Visualizations. *IEEE Transactions on Visualization and Computer Graphics* 20, 12 (Dec. 2014), 2092–2101. <https://doi.org/10.1109/TVCG.2014.2346291>
- [68] Ankit Rohatgi. 2021. Webplotdigitizer: Version 4.5.
- [69] Quentin Roy, Simon Tangi Perrault, Katherine Fennedy, Thomas Pietrzak, and Anne Roudaut. 2021. Understanding User Strategies When Touching Arbitrary Shaped Objects. In *Proceedings of the 23rd International Conference on Mobile Human-Computer Interaction (MobileHCI '21)*. Association for Computing Machinery, New York, NY, USA, 1–11. <https://doi.org/10.1145/3447526.3472038>
- [70] Quentin Roy, Futian Zhang, and Daniel Vogel. 2019. Automation Accuracy Is Good, but High Controllability May Be Better. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–8. <https://doi.org/10.1145/3290605.3300750>
- [71] Arvind Satyanarayan and Jeffrey Heer. 2014. Lyra: An Interactive Visualization Design Environment. *Computer Graphics Forum* 33, 3 (2014), 351–360. <https://doi.org/10.1111/cgf.12391>
- [72] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2017. Vega-Lite: A Grammar of Interactive Graphics. *IEEE Transactions on Visualization and Computer Graphics* 23, 1 (Jan. 2017), 341–350. <https://doi.org/10.1109/TVCG.2016.2599030>
- [73] Manolis Savva, Nicholas Kong, Arti Chhajta, Li Fei-Fei, Maneesh Agrawala, and Jeffrey Heer. 2011. ReVision: Automated Classification, Analysis and Redesign of Chart Images. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology (UIST '11)*. Association for Computing Machinery, New York, NY, USA, 393–402. <https://doi.org/10.1145/2047196.2047247>
- [74] Mingyan Shao and Robert P. Futrelle. 2006. Recognition and Classification of Figures in PDF Documents. In *Graphics Recognition. Ten Years Review and Future Perspectives (Lecture Notes in Computer Science)*, Wenyan Liu and Josep Llad s (Eds.). Springer, Berlin, Heidelberg, 231–242. https://doi.org/10.1007/11767978_21
- [75] Springer. 2022. Conference Proceedings Guidelines | Springer. <https://www.springer.com/gp/computer-science/lncs/conference-proceedings-guidelines>
- [76] Arjun Srinivasan, Steven M. Drucker, Alex Endert, and John Stasko. 2019. Augmenting Visualizations with Interactive Data Facts to Facilitate Interpretation and Communication. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 672–681. <https://doi.org/10.1109/TVCG.2018.2865145>
- [77] Edward Tufte. 2006. *Beautiful Evidence*. Graphics Press, Cheshire, Conn.
- [78] John Tukey. 1977. *Exploratory Data Analysis* (1st edition ed.). Pearson, Reading, Mass.
- [79] Aditya Vashistha, Pooja Sethi, and Richard Anderson. 2017. Respeak: A Voice-based, Crowd-powered Speech Transcription System. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 1855–1866. <https://doi.org/10.1145/3025453.3025640>
- [80] Chat Wacharamanatham, Lukas Eisenring, Steve Haroz, and Florian Echtler. 2020. Transparency of CHI Research Artifacts: Results of a Self-Reported Survey. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*. Association for Computing Machinery, New York, NY, USA, 1–14. <https://doi.org/10.1145/3313831.3376448>
- [81] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky. 2000. Guidelines for Using Multiple Views in Information Visualization. In *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '00)*. Association for Computing Machinery, New York, NY, USA, 110–119. <https://doi.org/10.1145/345513.345271>
- [82] Keke Wu, Emma Petersen, Tahmina Ahmad, David Burlinson, Shea Tanis, and Danielle Albers Szafir. 2021. Understanding Data Accessibility for People with Intellectual and Developmental Disabilities. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–16. <https://doi.org/10.1145/3411764.3445743>
- [83] Chun Yu, Yizheng Gu, Zhican Yang, Xin Yi, Hengliang Luo, and Yuanchun Shi. 2017. Tap, Dwell or Gesture? Exploring Head-Based Text Entry Techniques for HMDs. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. Association for Computing Machinery, New York, NY, USA, 4479–4488. <https://doi.org/10.1145/3025453.3025964>

A APPENDIX

A.1 Extraction Algorithms

All the extraction algorithms work by taking as input a user selection (i.e., a list of shapes) and outputting an array of coordinates that should be added to the extracted data. A shape is defined by a list of points (corners). To give consistent specifications to extractors, all shapes are pre-processed to be subdivided into smaller units everytime they use the “moveto” feature (C2), see Algorithm 6. For clarity reasons, the pseudo-code focuses on one specific orientation: vertical bars, box plots, and axes and horizontal lines.

Algorithm 1: Extraction of bars

Input: A user selection of shapes *shapes*

Output: An array of 2D coordinates

```

points ← []
foreach s ∈ shapes do
  add (middle of s, top of s) to points
return points

```

Algorithm 2: Extraction of scatters

Input: A user selection of shapes *shapes*

Output: An array of 2D coordinates

```

points ← []
foreach s ∈ shapes do
  add centre of s to points
return points

```

Algorithm 3: Extraction of axis

Input: A user selection of shapes *shapes*
Output: An array of 2D coordinates
points \leftarrow []
foreach *s* \in *shapes* **do**
 if *s* is text **then**
 add (text of *s*, centre of *s*) to *points*
return *points*

Algorithm 4: Extraction of lines

Input: A user selection of shapes *shapes*
Output: An array of 2D coordinates
points \leftarrow []
foreach *s* \in *shapes* **do**
 foreach *pt* \in *s* **do**
 add *pt* to *points*
return *points*

Algorithm 5: Extraction of box plots

Input: A user selection of shapes *shapes*
Output: An array of 2D coordinates
points \leftarrow []
groups \leftarrow group *shapes* with equal horizontal positions
foreach *group* \in *groups* **do**
 min \leftarrow y of *group*[0]
 max \leftarrow top of *group*[0]
 q1 \leftarrow *min*
 q3 \leftarrow *max*
 medians \leftarrow []
 foreach *shape* \in *group* **do**
 if *shape* is horizontal line **then**
 add vertical centre of *shape* to *medians*
 if *shape* is rectangle **then**
 q1 \leftarrow y of *shape*
 q3 \leftarrow top of *shape*
 foreach *median* \in *medians* **do**
 if *median* > *q1* and *median* < *q3* **then**
 add (middle of *group*, *median*) to *points*
 add (middle of *group*, *min*) to *points*
 add (middle of *group*, *max*) to *points*
 add (middle of *group*, *q1*) to *points*
 add (middle of *group*, *q3*) to *points*
return *points*

Algorithm 6: Split a compound shape

Input: A shape *s*
Output: An array of shapes
shapes \leftarrow []
path \leftarrow []
foreach *ope* \in path of *s* **do**
 if *ope* is moveto **then**
 add shape formed from *path* to *shapes*
 path \leftarrow []
 add *ope* to *path*
return *shapes*
