ScatterScanner: Data Extraction and Chart Restyling of Scatterplots

Aaron Baucom  
Master of Engineering, EECS  
University of California, Berkeley  
atbaucom@berkeley.edu

Christopher Echanique  
Master of Engineering, EECS  
University of California, Berkeley  
echanique@berkeley.edu

Figure 1: ScatterScanner System Pipeline

ABSTRACT
Commonly, publications do not expose their underlying data, only simple (often poorly designed) visualizations thereof. Many authors want to source data from other papers, yet if they include the pre-designed visualization they are chained to the design choices of the original author. This is a pain point for authors who want holistic control over the design language of their publication. In order to solve this problem, we present ScatterScanner, a web interface that will process an image, extract the data, and allow the user to modify design choices such as chart form, color, spatial relationships etc. In addition, we provide an option to download the extracted data from the visualization in the form of a .csv file so that the user can save and modify this data in other applications. Due to the large space of chart types and thus a wide variety of characteristics defining these charts, we limit the scope of this system strictly to scatterplots.

Author Keywords
Computer vision; information extraction; redesign; visualizations; scatterplot.

ACM Classification Keywords
H.5.m. Information interfaces and presentation: User Interfaces – Graphical User Interfaces.

General Terms
Algorithms; Design; Experimentation.

INTRODUCTION
Publishers of scholarly works or online articles recognize that it can be beneficial to build upon the ideas of their peers. Unfortunately, in order to use existing visualizations and graphics, the author is often forced to give up full design control or sacrifice the cohesive nature of their publication. Similarly, great data can be hidden in old scanned documents or visualizations that are not in the right
form for your work. What an author really wants is a way to redesign these charts to help illustrate and support their message. Ideally, the author could access the raw data and develop a fresh visualization to illustrate their message. Sadly, raw data is rarely available from scholarly works and online reports. We suggest a system that accepts a chart image and extracts the underlying data would bridge this reusability gap and enable better publications in general.

Fully automatic chart extraction is a difficult problem thanks to the large space of possible design choices. As such, we will restrict our problem space to scatter plots, in order to leverage some broad design conventions. Even after this reduction, there is significant variability in the types of plots that can be generated. Furthermore, many published scatter plots would not even be feasible for a human to transcribe data from, usually due to a high degree of clustering and point overlap. Given that no algorithm can accurately process all of the charts, it is exciting to see how different techniques succeed and fail under different cases.

RELATED WORK

The primary reference for our work is a system called Revision [1] developed by Savva et al. in 2011. This system categorizes images that contain charts into various general types such as bar, line, scatter, pie, etc. Their system can also perform some extraction of data from pie charts and bar charts. We plan to complement this approach by adding scatterplots to the range of graphs from which data can be successfully recovered.

Additional research has been done to reclassify data from scatterplots, though this research is very limited. Bezdek and Chiou [2] have presented a feature extraction system to convert three-dimensional scatterplot images into two-dimensional space. However, this does not focus on the actual extraction of data in Cartesian coordinates.

In addition to this, there are a number of prior systems that allow the user to manually extract the data. This is typically done with user input, such as in [3] by Timothee Poisot, which allows users to click on data points and scale labels. These systems are still very time consuming, especially for graphs, which have a very large amount of data.

SYSTEM OVERVIEW

ScatterScanner is comprised of two stages: (1) data extraction and (2) chart revitalization. In stage 1, we use computer vision algorithms to locate chart characteristics of the scatterplot such as axis location, scaling information, and plotted point location in order to extract the data into a table. In stage 1, we present a web interface to allow the user to view extracted data, modify the visualization, and export the data to a .csv file.

STAGE 1: DATA EXTRACTION

The goal of ScatterScanner in this stage is to accurately identify the marks in the scatterplot and extract them into a data table. To achieve this, we must implement an approach to map points from the image space to points in the data space using computer vision. However, due to the large variability of visual elements in different types of scatterplots, this approach is difficult to generalize for all types.

In order to simplify to our algorithm, we make the following assumptions of the scatterplot:

- Chart axes appear on the left and bottom of the scatterplot.
- Charts do not contain heavy gridlines or other plot types (line, bar, etc.) over scatterplot marks.
- Chart points are made up of simple shapes.
- Only one data set is encoded in the points.
- No text or unusual shapes appear in the plotting region of the chart.
- The data plotted is in the first quadrant of the Cartesian plane.

Based on these assumptions, we develop a robust data extraction algorithm that applies a significant number of real-world scatterplots.

Plotted Region Cropping

The first step in the data extraction stage is to identify the plotted region of the chart. We will use Figure 2 as a sample scatterplot to provide a visual aid as we describe the process.

![Figure 2. Sample Scatterplot for reference](image-url)
Generate histogram of edges. Since we assume that the chart has both the left and bottom axis, we can exploit this feature to identify the bounded area. We implement a similar approach as in [1] to do so. Sobel edge detection is used to identify the edges in both the x and the y directions of the chart. Then a histogram is generated for the top-to-bottom sum of the pixel edges in the x direction and left-to-right sum of pixels edges in the y direction. Figure 3 illustrates this concept. However, we recognize the limitations of this approach due to the possibility of borders surrounding the full image. Therefore we crop the image removing any empty space surrounding the full image by detecting if edges exist in these regions. Then we threshold the newly crop region with a five percent margin on each side to avoid mistakenly identify borders instead of axes locations.

Mark peaks of histogram. Once the histograms of each edge direction are found, we use this information to identify the column index of the left axis and row index of the bottom axis. This is done by determining the peaks of the histograms in the region where the axis falls. Due to the fact that some charts (such as the one in Figure 2) may have a full border instead of just a left and bottom axis, we limit our search to peaks in the left or bottom half of the histogram. This prevents from misclassifying the chart axes with misleading cues.

Identify full bounding region. Once the column index of the left axis and row index of the bottom axis is found, we can mark the intersection point as the bottom left corner of the plotted region. To identity the full region we must find the top right corner as well. Since we know the index where the axis line lies, we must find the indices of topmost point of the left axis and rightmost point of the bottom by iterating through these axis vectors. However if we simply iterate through the beginning of the vector and stop at the first edge found, we may run into the case of text or chart elements above the axis and wrongly detect the region. Therefore we find the longest connected line within the vector and mark its beginning and end indices. This is done for both axes to obtain the top point and the rightmost point of the plotted region for cropping.

Location of Marks

The next step in the data extraction stage is to identify the index locations of the marks in the plotted region obtained from the previous step.

Determine mark size. To make our approach invariant to changes in scale of the points, we determine pixel size of the plotted marks. This is achieved by finding the average size of the contours of the image. In order to provide a more accurate estimate, we filter out any contours found that are larger than ten percent of the chart width, assuming that the points generally fall into this range. Also we threshold contours found smaller than 3x3 square pixels as these likely identify noise in the image.

Apply Gaussian filter. Once we have the estimated pixel size of the plotted marks in the image, we convolve the image with a Gaussian filter. The goal of this is to have the mark’s brightness peak at its center. Therefore we select the size of the Gaussian filter to the same as the mark size determined in the previous step. However, it makes the algorithm simpler if we define the size as an odd number to ensure there is a center pixel. If the mark size is even, we extend the size of the filter by one row and column to achieve this. In addition, we set sigma of the Gaussian filter
to be one-third the size of filter to allow for an optimal distribution for convolving [4].

**Find peaks.** The effect of the Gaussian filter over the image is that the there will be local maxima at the regions of the marks. We can exploit this by iterating through to each pixel and comparing the values associated with the pixel to each of the eight adjacent pixel neighbors as shown in Figure 4. If all of the neighbors’ pixel values are less than the value of the pixel of interest, we mark that index as a point and store the coordinates in a vector.

![Figure 4. Checking for peaks at each pixel](image)

**Peak filtering.** In certain circumstances, points on the scatterplot are marked multiples in different pixels within the point region. These clusters of redundant data must be filtered out such that only one mark is used to identify the point. In order to do this, we search through the vector of marked indices to verify if there exists other nearby marks that represent the same point. Given the mark size \(m \times m\), we use a filter of size \(2m+1 \times 2m+1\) to remove any marks identified in the region around a given mark. We realize the pitfalls of this approach as it may potentially remove correct marks that identify other points within the given range of a certain point. However, this approach is chosen in order to limit the number of false positives due to redundant data as opposed to differentiate between points in a clustered region.

**Image Space to Data Space Conversion**

The final step in the data extraction stage is to convert the data from pixel indices in the image space to actual Cartesian coordinates in the data space. First

**Text Recognition.** The scale of the data is determined by examining the tick labels on the axis. Since scatter plots are largely used to illustrate accurate relationships between two variables, it is safe to say that we should expect some axes and some form of tick labeling along them for scale. The only exception to this would be plots that are labeled with callouts near the data points themselves, which we did not encounter in our corpus. Given that the axes can be identified, we can look below and to the left to find labels for the x and y axes respectively. Next we use edge detection and box filter techniques to find hot spots where we are likely to find text. We extract a bounding box using a blob detection algorithm and then feed the image patch into the Tesseract [5] text recognition engine.

**Coordinate Scaling.** Now that we have extracted the axes labels, we can convert data from pixel coordinates to data coordinates by scaling the chart area to the bounds of the axes values. This is done by using dividing the pixel coordinates by the size of the chart image to normalize the indices from zero to one. The text extracted from the Tesseract engine is used to define the scale. We look at the top and bottom as well as leftmost and rightmost values of extracted text from the axis (as in Figure 5) and store them as \(y_{\text{max}}\), \(y_{\text{min}}\), \(x_{\text{max}}\), and \(x_{\text{min}}\) respectively. These values are then used as follows to obtain the coordinates:

\[
X = j \cdot (x_{\text{max}} - x_{\text{min}}) + x_{\text{min}} \\
Y = i \cdot (y_{\text{max}} - y_{\text{min}}) + y_{\text{min}}
\]

where \(X\) and \(Y\) are the Cartesian coordinates, \(i\) is the row index of the image, and \(j\) is the column index of the image.

![Figure 5. Finding axis labels for scaling](image)
generate accurate data. This approach is susceptible to noise perturbations of the data and label locations that can introduce slight offsets or scaling errors. These magnitudes of these errors relative to the data values should scale inversely with input resolution, so we believe this is an acceptable approach.

STAGE 2: RE-VISUALIZATION
The second stage of our system involves revisualizing the data that was extracted from the previous stage. Once the server-side computer vision analysis has finished, the data is ready to be handed off to the user. Our system lets the user download the raw data as well as create simple vector visualization previews using D3. The data is exported in a simple two-column comma separated (CSV) file. This allows the user to easily import the data into any common visualization editor and generate their own graphics. This is the main avenue for our system to empower authors to create better papers. We suggest software like Tableau or Spotfire to generate powerful and meaningful visualizations based on this information.

The D3-powered web previewer was initially intended to be a small visualization editor in its own right. Over the course of this project, we realized that this space is already well explored and we decided to redouble our efforts on the data extraction software instead. That said, the web application does produce nice vector graphic images and gives the user an easy way to assess how accurate the data extraction process was. We implemented features such as different chart modes, between scatter plots and histograms, as well as simple toggle switches to display data labels, axis bars, and axes labels. The user can screenshot this data or copy the svg code in the HTML for use on their website. In the future, we think this front end could be better aligned with our extraction methods to illustrate what has happened and help the user correct any errors. In that sense, the front-end would keep the user in the loop as opposed to being a simple output device; this concept will be discussed further as an area for future work.

RESULTS
In order to test our extraction approach, we used a subset of images Prasad et al.’s [22] corpus of scatterplots that met our assumptions. This subset consists of 48 images.

To validate our results, we compared our predicted locations of the marks in the scatterplot with their actual plotted locations. The metric we used to determine successful data extraction was to verify that over 95% of discernable points were correctly identified with no false positives. Discernable points are defined by the ability to easily distinguish one point from another. Our approach comes with the understanding that clusters of data points can limit the decoding of such points into their real
coordinates even for humans. Therefore we do not penalize our extraction algorithms for errors due to clusters.

Using this metric, we found that ScatterScanner successfully extracted marks for 30/48 (62.5%) of the scatterplots. Most of our mark extraction failures were due to detection of false positives such as tick marks of the axes or the marking of entire lines bordering the plotted chart region.

Using this metric, we found that ScatterScanner successfully extracted marks for 30/48 (62.5%) of the scatterplots. Most of our mark extraction failures were due to detection of false positives such as tick marks of the axes or the marking of entire lines bordering the plotted chart region.

Figure 7. Good data extraction results

Figure 7 above presents some of the successful results obtained through our data extraction method. The “Extracted Marks” figures show red x’s on the pixels that were detected during data extraction. For chart (a) our method was able to accurately detect all the points. One important observation to note here is the diamond shape of the points, indicating a level of robustness to mark shape. In chart (b), most of the discernable points were accurately detected as well. As it can be seen in the figure, the regions with large clusters of marks pose difficulty for identification.

Figure 9 identifies some of the issues with our extraction approach. Taking a closer look at chart (a), it can be seen that multiple red x’s appear on each of the plotted points. This phenomenon also occurs in chart (b). One important observation we noticed in testing our data extraction method was that this redundant detection occurred mainly in charts with hollow mark shapes. This suggests that our approach is not robust to these certain mark types. In addition, chart (b) demonstrates false positives on tick marks along the axes.

Figure 8. Poor data extraction results

Another important thing to note is the performance of the text extraction. There was a large variance in this performance based on a few factors. Image resolution was mainly the limiting factor for the accurate extraction of axis labels. The Tesseract text extraction framework had a hard time accurately extracting axis labels from images with small resolution. In addition, our selection of these text regions affected its performance. At times the axis labels were too close to the tick marks and therefore were read incorrectly. This posed issues with the scaling of our pixels coordinates to Cartesian space.

There were some peculiar behaviors of our system during testing. We hope to study these on case by case basis and determine why, occasionally, we got some good results on bad data, and bad results on good inputs.

Figure 8. Left: an interesting failure case on a simple graph. Right: a surprising success in a busy graph with lots of extraneous chart marks.

Performance

Our C++ code runs on a web server running Ubuntu 12.04 OS. It is equipped with 8 GB of RAM and an Intel "Ivybridge " dual-core CPU clocked at 2.8 GHz. Figure 9 shows the performance of our system. The worst case runtime we experienced was ~2.5 seconds when processing a 1271x801 image file. It can be seen that our system is more or less trending to be O(n) bounded where n is the...
pixel count of the input image. More tests will be needed to characterize full performance the system, but it appears that there is some fixed overhead which is incurred after which the application behaves linearly. Our lowest time-to-pixel ratios show up in the largest images. There is additional room for improvement on this system. Our current system is not leveraging any parallel compute power in the GPU and has not been fully tuned using any profiling software. We believe that it is possible to extend the current system to any reasonable image whilst maintaining an appropriate runtime for the user.

![Run Time vs. Image Size](image)

**Figure 9. Runtime of data extraction by image size**

**DISCUSSION**

In this paper, we have presented a system that extracts data from visualizations and have described the underlying algorithms. Our system addresses a problem of communication in which visualizations are telling a story but are doing so at the expense of portability and reusability. This system helps users aggregate and redesign visualizations to fit their needs. Now that this system is in place, we may be able to learn what kinds of visualizations people are most interested in editing. Now that this system is in the wild, it may be possible to discern what general properties make a graph “non-portable” once enough people use the system. In a sense, this could address the question: “What properties can designers be cognizant of when making graphs to facilitate (or prohibit) reuse and peer citations?” Another interesting avenue that this work opens up is the correlation between machine perception and human perception. It is possible that this sort of system could be used to give a rough estimate of how well a given visualization exposes the underlying data. If our system could function as a first-order estimation of the readability of a graph, it could provide useful feedback to users who want to understand how clearly they are presenting the raw data to the viewer.

**FUTURE WORK**

In the future, we want to improve our own algorithm's accuracy, front-end functionality, and generalizability. In addition to this, there is space to combine our approach with existing work and create a more fully featured system. In that sense, our system is just a start point for developing what would be a truly useful extraction tool.

Primarily, we hope to improve accuracy by developing our algorithms further by characterizing what the failure cases look like to drive future iteration. In addition, we could make the system more useable by returning some level of confidence for each graph processed. We can also keep the user in the loop and leverage their ability to correct errors and improve accuracy, which we discussed in more detail next.

We think there is a lot of room to improve our front-end functionality and would want to take it in a different direction in the future. Initially we had intended to design a system that enables redesign in the browser. At the same time, existing visualization tools deliver better customization and are already far more feature-complete for actual chart design, so reimplementing that doesn't add value for our users. In retrospect, we think it would be a more cohesive system if we instead focused on using the front-end to enhance the data extraction process. If we could convey more information about the underlying extraction algorithms and give the user the tools to correct for errors and failure cases, the system would be much more reliable.

Third, we want to generalize our algorithms further by allowing for scatter plots that don't conform to our rigid structure. For example, many scatter plots encode different data through size, shape, orientation, or additional visual variables. An ideal system could extract more data from these characteristics; a more reasonable intermediate system would be invariant to some of these variables and could still reliably extract position data despite these changes.

Lastly and in a different vein, there is additional work in the space of data extraction on other forms of charts. A paper by one of our colleagues, Eric Yao, addresses line graphs and the previously mentioned Revision system can be used to extract data from bar charts and pie charts. We are beginning to get more coverage into the set of chart categories that can be processed, but there are more chart types to explore such as area charts, treemaps, or even 3D...
visualizations. Furthermore, the Revision system can categorize an image based on what kind of chart is contained within it. A natural progression from these building blocks would be a system that uses their categorization as a preprocessor that forwards the chart to the appropriate data extraction framework. In this way, an adaptable and more general data extraction tool could be created that might become a great tool for researchers.

CONCLUSION
We have presented ScatterScanner, a web interface that will process an image of scatterplot, extract the data, and allow the user to modify design choices such as chart form, color, spatial relationships, etc. Our system can be used to extend the robustness of current data extraction systems like ReVision [1] to apply to a larger range of chart types.

ACKNOWLEDGMENTS
We thank Dr. Maneesh Agrawala and graduate student colleagues who provided helpful comments and feedback on our system approach and design.

REFERENCES