ReVision: Automated Classification, Analysis and Redesign of Chart Images

Manolis Savva[*], *Nicholas Kong*[†], *Arti Chhajta*[*], *Li Fei-Fei*[*], *Maneesh Agrawala*[†], *Jeffrey Heer*[*]

[*]Stanford University
{msavva, achhajta, feifeili, jheer}@cs.stanford.edu

[†]University of California, Berkeley
{nkong, maneesh}@eecs.berkeley.edu

ABSTRACT

Poorly designed charts are prevalent in reports, magazines, books and on the Web. Most of these charts are only available as bitmap images; without access to the underlying data it is prohibitively difficult for viewers to create more effective visual representations. In response we present *ReVision*, a system that automatically redesigns visualizations to improve graphical perception. Given a bitmap image of a chart as input, ReVision applies computer vision and machine learning techniques to identify the chart type (e.g., pie chart, bar chart, scatterplot, *etc.*). It then extracts the graphical marks and infers the underlying data. Using a corpus of images drawn from the web, ReVision achieves an image classification accuracy of 96% across ten chart categories. It also accurately extracts marks from 79% of bar charts and 62% of pie charts, and from these charts it successfully extracts the data from 71% of bar charts and 64% of pie charts. ReVision then applies perceptually-based design principles to populate an interactive gallery of redesigned charts. With this interface, users can view alternative chart designs and retarget content to different visual styles.

ACM Classification: H5.2 [Information Interfaces and Presentation]: User Interfaces – Graphical User Interfaces.

Keywords: Visualization, chart understanding, information extraction, redesign, computer vision.

General terms: Algorithms, Design, Experimentation

INTRODUCTION

Over the last 300 years, charts, graphs, and other visual depictions of data have become a primary vehicle for communicating quantitative informationf [28]. However, designers of visualizations must navigate a number of decisions, including choices of visual encoding and styling. These decisions influence the look-and-feel of a graphic and can have a profound effect on graphical perception [6]: the ability of viewers to decode information from a chart. Despite progress in the development of design principles for effective visualization [6, 11, 16, 20, 21, 24, 28], many charts produced today exhibit poor design choices that hamper understanding of the underlying data and lead to unaesthetic displays.

UIST'11, October 16-19, 2011, Santa Barbara, CA, USA.

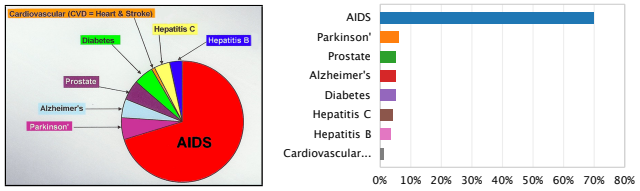


Figure 1: Chart Redesign. Left: A pie chart of NIH expenses per condition-related death. The chart suffers from random sorting, highly saturated colors, and erratic label placement. Right: Plotting the data as a sorted bar chart enables more accurate comparisons of data values [6, 20].

Consider the pie chart in Figure 1 (left), which depicts data concerning the 2005 research budget of the National Institutes of Health (NIH). The design of this chart could be improved in multiple ways: slices are ordered haphazardly, labels are placed erratically, and label text competes with saturated background colors. Moreover, the chart encodes values as angular extents, which are known to result in less accurate value comparisons than position encodings [6, 16, 24]. Figure 1 (right) shows the same data in a redesigned visualization: the bar chart sorts the data, uses a perceptually-motivated color palette [26], and applies a position encoding.

For analysts working with their own data, automated design methods [20, 21] based on visual design principles [6, 11, 16, 28] can lead to more effective visualizations. However, the vast majority of visualizations are only available as bitmap images. Without access to the underlying data it is prohibitively difficult to create alternative visual representations.

We present *ReVision*, a system that takes bitmap images of charts as input and automatically generates redesigned visualizations as output. For example, ReVision produces Figure 1 (right) as a suggested redesign when given Figure 1 (left) as input. Our system identifies the type of chart, extracts the marks (visual elements that encode data) and underlying data, and then uses this information in tandem with a list of guidelines to provide alternate designs. ReVision also supports stylistic redesign; users can change mark types, colors or fonts to adhere to a specific visual aesthetic or brand. We make the following research contributions:

Classification of Chart Images. ReVision determines the type of chart using both low-level image features and extracted text-level features. We propose a novel set of features that achieve an average classification accuracy of 96%. Our image features are more accurate, simpler to compute, and easier to interpret than those used in prior work. While we focus on the task of chart redesign, our classification method is applicable to additional tasks such as indexing and retrieval.

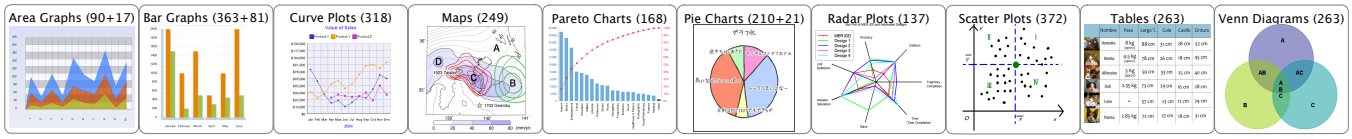


Figure 2: Our 10-category chart image corpus. Numbers in parentheses: (strictly 2D images + images with 3D effects).

Extraction of Chart Data. We present a set of chart-specific techniques for extracting the graphical marks and the data values they represent from a visualization. Our implementation focuses on bar and pie charts. We first locate the bars and pie slices that encode the data. We then apply heuristics to associate the marks with related elements such as axes and labels. Finally, we use this information to extract the table of data values underlying the visualization. We exclude any bar and pie chart images with 3D effects or non-solid shading. We achieve an average accuracy of 71.4% in extracting the marks and an average accuracy of 67.1% in extracting the data after we successfully extract the marks.

Finally, we demonstrate how these contributions can be combined with existing design guidelines to automatically generate redesigned visualizations. We have also compiled a corpus of over 2,500 chart images labeled by chart type. We are making this corpus publicly available in the hope of spurring further research on automated visualization interpretation.

RELATED WORK

Our ReVision system builds on three areas of related work.

Classifying Visualization Images

Classification of natural scenes is a well-studied problem in computer vision [1, 3]. A common approach is to use a "bag of visual words" representation [29] where the "words" are low-level image features such as gradients or local region textures. Each input image is encoded as a feature vector using these words. Machine learning methods then classify the feature vectors and corresponding images into categories.

Researchers have developed specialized techniques for classifying chart images by type: bar chart, pie chart, *etc*. Shao and Futrelle [23] and Huang et al. [17] extract high-level shapes (e.g., line segments) from vector images of charts and then use these shapes as features for classifying six kinds of charts. Their approach relies on an accurate vectorization of charts, which can be difficult to generate from chart images.

Prasad et al. [22] classify bitmap images of charts drawn from five common categories. They apply a series of preprocessing operations to compute global curve saliency and local image segment features. These operations require many input-dependent parameters, complicating generalization to more categories. Additionally, they use Scale Invariant Feature Transform (SIFT) [10] and Histogram of Oriented Gradient (HOG) [9] features to represent higher level properties of image regions. They then compare these features using multi-resolution pyramidal histograms and use the similarity scores to train a multi-class Support Vector Machine (SVM), achieving an average classification accuracy of 84.2%.

The features used by Prasad et al. cannot easily be tied to the presence of specific graphical marks. We are interested in image features at the level of graphical marks because they could be useful in connecting classification to extraction. We desire highly predictive features that (a) are easier to compute, (b) more readily interpreted, and (c) better inform post-classification extraction.

Extracting Marks from Charts

Some researchers have investigated techniques for extracting marks from charts. Zhou and Tan [31] combine boundary tracing with the Hough transform to extract bars from bar charts. Huang et al. [17, 18, 19] generate edge maps, vectorize the edge maps, and use rules to extract marks from bar, pie, line, and low-high charts. Yang et al. [30] built a system incorporating this technique to allow humans to correct automatically generated vector representations of charts. However, their work focuses on extracting the mark geometry rather than the underlying data. Their techniques rely on an accurate edge map, which can be difficult to retrieve from real-world charts, due to large variance in image quality. We designed our techniques to be more robust to such variance.

Automated Visualization Design

Researchers have applied graphical perception principles to automatically produce effective visualizations. Mackinlay's APT [20] generates charts guided by rankings of visual variables for specific data types such as nominal, ordinal, or quantitative data. Stolte et al.'s Polaris [25] is a system for generating small multiples [28] displays based on user queries of a multidimensional data table. Mackinlay et al. [21] extend this work to support a range of automated visualization designs. These systems assist users in producing visualizations directly from data; we seek to first extract data from chart images and then redesign the visualizations. Our current work is complementary to these earlier systems: once we have extracted a data table, we could feed it into any of these systems to generate improved alternative designs.

SYSTEM OVERVIEW

ReVision is comprised of a three stage pipeline: (1) chart classification, (2) mark and data extraction, and (3) redesign. In stage 1, ReVision classifies an input image according to its chart type. This stage uses a corpus of test images to learn distinctive image features and train classifiers. In stage 2, ReVision locates graphical marks, associates them with text labels, and extracts a data table. In stage 3, ReVision uses the extracted data to generate a gallery of alternative designs.

STAGE 1: CLASSIFYING CHART IMAGES

While classification of natural images is a well-studied computer vision problem, chart images are unlike natural images in several important ways. First, marks within data graphics are more discrete and more frequently repeated than textured regions in photographs. Second, there are many areas with constant color, so pixel neighborhoods with low variances are common. Finally, text occurs more frequently and

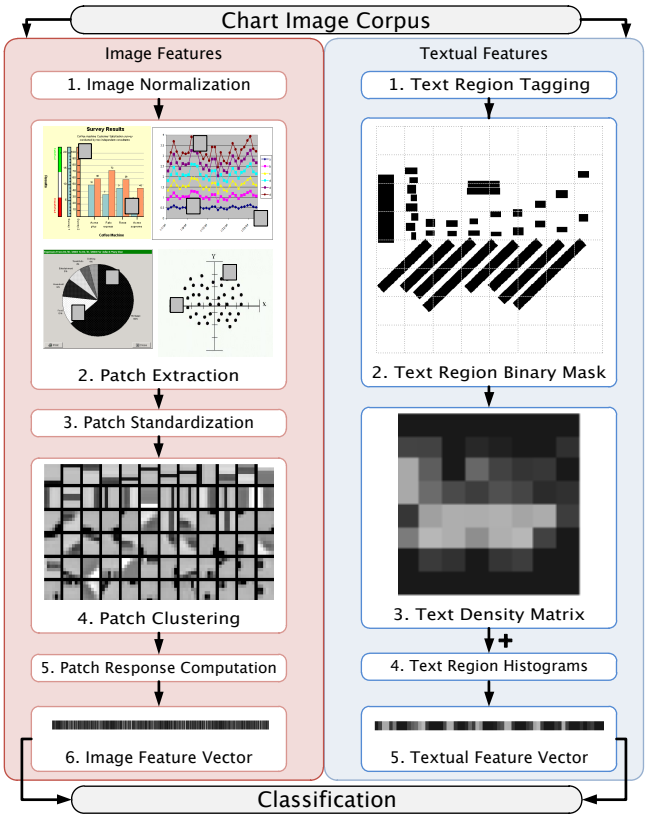


Figure 3: The ReVision classification pipeline, including both image (left) and textual (right) features.

prominently than in natural scenes. The following sections describe how we account for these differences to achieve classification accuracy superior to previous work [22].

Learning Low-Level Image Features

Figure 3 illustrates our classification pipeline. Coates et al. [7] found that random sampling of image patches (small rectangular image regions), followed by K-means clustering of the patches and then classification achieves state of the art performance for classifying natural images. We extend their approach to classifying chart images using a sequence of 7 steps: (1) normalize the input images, (2) extract image patches, (3) standardize the patches, (4) cluster the patches to form a "codebook", (5) compute a "codebook" patch response for each image, (6) formulate a feature vector for each image from the response, and (7) perform classification.

1. Image Normalization. We normalize images to a constant size of $D \times D$ pixels to ensure homogeneous sampling and equivalence of each exemplar in the corpus. We preserve the aspect ratio of the original image by padding with the background color. We convert color images to grayscale, as color is rarely indicative of visualization category. Note that we use the original color images in the mark and data extraction stage of our system. We considered image sizes from $D = 32$ up to $D = 512$ and found that $D = 128$ achieves the best classification accuracy. We believe this is due to the reduction of noise and compression artifacts in the down-sampled images (the average image size is roughly 300×400 pixels).

2. Patch Extraction. We extract square patches from the images by uniform random sampling for 100 locations per image. We tested patch sizes over the range from 2×2 up to 20×20 pixels and found that a patch size of 6×6 pixels gave the best classification performance. The optimal patch size was consistently about 5% of the normalized image dimensions. To filter out frequently occurring constant color regions, we reject sample patches with variance less than 10% of the maximum pixel value.

3. Patch Standardization. For classification, absolute pixel values are not as important as variation within a patch. Thus, we normalize the contrast of each patch by subtracting the mean and dividing by the standard deviation of the pixel values in that patch. This normalization ensures that a single appropriately weighted patch can represent patches with different absolute intensities but similar variations. We perform a "whitening" procedure on the entire patch set which reduces the cross-correlation between patches and improves classification performance [7].

4. Patch Clustering. Given an extracted patch set, we perform K-means clustering to obtain a set of "centroid" patches that correspond to the most frequently occurring patch types. We use a Euclidean L^2 distance metric over the pixel values. In practice we set the number of centroids K to 200 as we found that larger numbers of centroids did not achieve better performance. The centroid patches constitute a feature set, or "codebook", which we can use to describe our image corpus.

These codebook patches capture frequently occurring graphical marks such as lines, points, corners, arcs and gradients. By independently extracting codebooks from each visualization category, we obtain patch sets that reflect the occurrence frequencies of marks characteristic of each category. As an example, in Figure 4 we compare the centroids for 3 categories from the corpus of [22]. We invite the reader to guess which categories are represented.[1]

Using Low-Level Image Features for Classification

To classify an input image, we first normalize it to a 128×128 grayscale image and extract 6×6 sized patches centered on each pixel. We then perform the following steps:

5. Codebook Patch Response. For each extracted patch, we determine the nearest codebook patch by Euclidean distance. We thus obtain a D^2 centroid response map over the entire image, where D (= 128) is the normalized image dimension.

6. Feature Vector Formulation. We reduce the dimensionality of the codebook patch response map by dividing the image into quadrants and summing the activations for each codebook patch in a given quadrant, similar to constructing a histogram of activated patches. We thus obtain a $4K$-length feature vector for each input image. Since we set $K = 200$, we obtain an 800 element image feature vector which is much smaller than the set of all 128×128 patch responses.

7. Classification. We use the feature vectors to perform classification using Support Vector Machines (SVMs) [8] with a quadratic kernel function. We use the SVM implementation provided by the WEKA [14] framework.

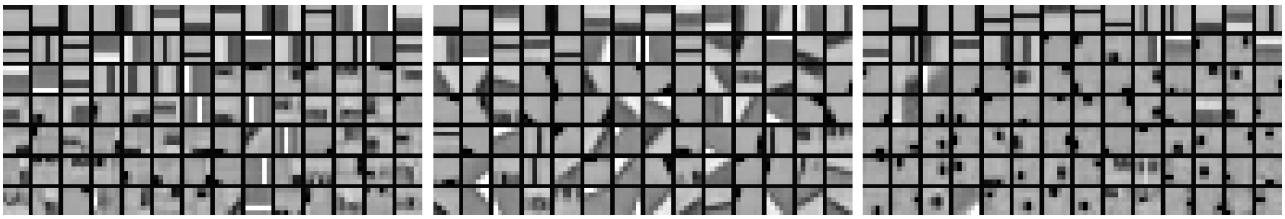


Figure 4: Codebook patches for 3 different visualization categories. We invite the reader to guess which categories were sampled.[1]

Using Text-Level Features to Improve Classification

While image features offer powerful cues to determine the category of a chart image, text-level features are also informative. The position and size of text regions in chart images may correlate with the visualization type. We show that textual information can further improve classification accuracy.

We designed a tagging interface to annotate chart images with the position, size, angular orientation and content of text regions. Our tool extracts the text image region and performs OCR using the Tesseract open source OCR engine; the user can correct the OCR output if desired. While manual text annotation is tedious, it is amenable to crowdsourcing and we designed the tagging UI with this scenario in mind. Furthermore, connected components analysis or more sophisticated approaches [4] can extract text region bounding boxes automatically. These methods could allow fully unsupervised textual feature extraction from a visualization corpus.

To capture the distribution and density of text regions in an input image, we construct a binary mask indicating which pixels belong to text regions (step 2 in Figure 3, right). We subdivide the mask into $N \times N$ blocks, calculate the proportion of text region pixels in each block (step 3), and linearize the values into an N^2 element vector. We found that $N = 8$ maximizes classification accuracy. However, at this resolution the exact positions, sizes and relative orientations of text regions are abstracted. To retain this information we compute normalized 10-bin histograms of the distributions of text region length, width, center position, as well as pairwise orientation and pairwise distance between text region centers (step 4). We concatenate these histograms with the text density vector to create a final textual feature vector (step 5).

Classification Performance

We used three collections of chart images to evaluate our system. To compare our classification results with those of Prasad et al. [22], we used their corpus of 667 images drawn from 5 categories: bar charts (125), curve plots (121), pie charts (134), scatter plots (162) and 3D surface plots (125). We also obtained the corpus of Huang et al. [17], which contains 200 images in 4 categories: 2D bar charts (80), 2D pie charts (48), 3D pie charts (12), and line plots (60). Finally, we compiled a new corpus containing 2,601 chart images labeled with 10 categories using a semi-automated tool for querying Google image search. Example images from each category are given in Figure 2. We use this larger corpus to evaluate how our system scales to classifying images from larger sets of visualization categories.

[1] Bar Charts (left), Pie Charts (middle), Scatter Plots (right). Bar corners, pie arcs and scatter points are prevalent within their respective categories.

	Prasad [22]	Image	Text	All	Binary
Bar	90%	85%	57%	89%	95%
Curve	76%	75%	50%	83%	92%
Pie	83%	93%	84%	95%	97%
Scatter	86%	91%	64%	91%	97%
Surface	84%	90%	71%	94%	97%
Average	**84%**	**88%**	**66%**	**90%**	**96%**

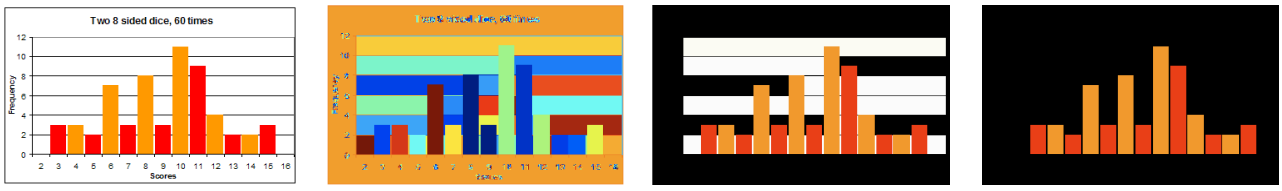
Table 1: Classification accuracy for the corpus from Prasad et al. [22]. We compare Prasad et al. to our method using image features (first two columns), text features (1st and 3rd), and both (1st and 4th) in multi-class SVMs. The last column shows results for both features using binary SVMs.

	Multi-Class	**Binary**
Area Graphs	88%	98%
Bar Graphs	78%	95%
Curve Plots	73%	91%
Maps	84%	97%
Pareto Charts	85%	97%
Pie Charts	79%	97%
Radar Plots	88%	93%
Scatter Plots	79%	93%
Tables	86%	97%
Venn Diagrams	75%	97%
Average	**80%**	**96%**

Table 2: Classification accuracy on our 10-category corpus (2,601 images) using only image features.

We first tested the performance of the learned image features for classification by training a multi-class SVM classifier through 5-fold cross-validation on the 5 category image corpus. In each fold, we randomly picked $4/5$ of the images for training and the remainder for testing. We then averaged results over the 5 folds. The first two columns of Table 1 summarize our results in comparison to Prasad et al. [22]. Our average classification accuracy was 88%, compared to Prasad et al.'s 84%. Incorporating text features further increased our accuracy to 90% (the third column of Table 1 uses only text features, while the fourth column uses both text and image features). For the 200 image corpus used by Huang et al. [17] our average accuracy was 94%, exceeding the 90% reported by those authors. We used the same procedure on our larger 10-category corpus where we obtained an average classification accuracy of 80% (see Table 2).

We have found that charts of different categories often include visually similar elements such as axes, grid lines, and legends. Some charts are hybrids of multiple categories (e.g., a combination bar and line chart). The multi-class SVM classifier we have proposed so far enforces disjoint classification (e.g., the image is purely a bar chart or purely a line chart).



(a) Input chart (b) Connected components (c) Rectangular components (d) Candidate Bars

Figure 5: Bar extraction procedure. We compute connected components (shown false-colored) and discard non-rectangular components. We keep rectangles whose color differs from surrounding colors (see inset figure on this page), that touch the baseline x-axis, and whose height is greater than 2 pixels or whose width is near the average width of the other candidate bars.

The classifier must learn to ignore common elements and cannot handle hybrid charts. A bank of 1–vs–all binary classifiers (1 classifier per class) is more appropriate for a system like ReVision. Each binary classifier computes whether the image contains a visualization of a given class but does not prevent the other classifiers from also labeling the image. A hybrid chart for example might be labeled as both a bar and line chart. We train a binary SVM for each category using in-category images as positive examples and all others as negative examples. Results obtained using this method are given in the rightmost columns of Tables 1 and 2. We achieve an average accuracy of 96% for both our corpus and the Prasad et al. corpus, and 94% for the Huang et al. corpus.

The total time for image feature learning, feature vector extraction and classifier training on the Prasad et al. corpus was 144s on an Intel Xeon E5540 2.53GHz workstation. We classified new images in less than a second.

STAGE 2: MARK AND DATA EXTRACTION

After categorizing charts by type, ReVision proceeds to locate graphical marks and extract data. Our current implementation focuses on mark and data extraction for bar and pie charts, two of the most popular chart types.

We have found that chart images from the web are often heavily compressed and noisy. To reduce such artifacts, we first apply the bilateral filter [27] to each bar or pie chart image given by our classifier. The bilateral filter smooths away small variations in color but retains sharp edges in the image. Next we perform *mark extraction*, which locates graphical marks such as bars and pie slices by fitting models of expected shapes to image regions. Finally, we perform *data extraction*, which infers the mapping between data space and image space, associates labels with marks, and then extracts a data tuple for each mark.

Our algorithms are based on a few simplifying assumptions:

- Charts contain 2D marks and do not contain 3D effects.
- Each mark is solidly shaded using a single color. Marks are not shaded using textures or steep gradients.
- Marks encode a data tuple, where at least one dimension is quantitative and one dimension is nominal. For example, in a bar chart the height of a bar represents a quantitative value, while a nominal label often appears below the bar.
- Bar charts do not contain stacked bars.
- Bar chart axes appear at the left and bottom of the chart.

Based on these assumptions we develop simple, robust data extraction algorithms that apply to a significant number of real-world charts. For example, 42% (52/125) of the bar and 43% (53/125) of the pie charts in the Prasad et al. corpus [22] follow these assumptions.

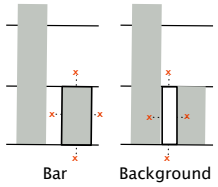
Mark Extraction

In this step, we locate the marks. For bar charts, we extract the bars and axes. For pie charts, we extract the pie location and slice borders. We discuss each in turn.

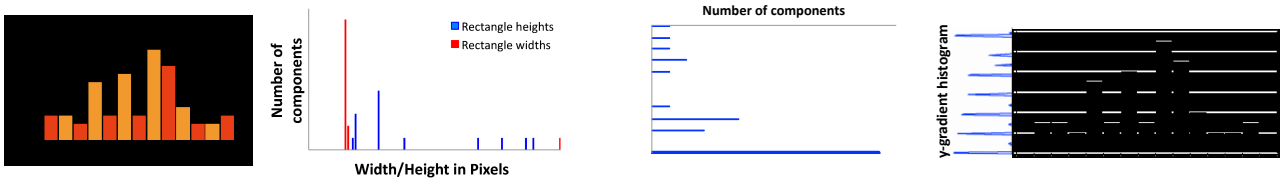
Bar Charts

Figure 5 shows the steps of our bar extraction algorithm. We identify bars by looking for rectangular connected components and then use the bar locations to extract the axes.

Find rectangular shapes. We first extract connected components from the filtered image by grouping adjacent pixels of similar color, i.e., those with an L^2 norm less than 0.04 in normalized RGB space (Figure 5b). We then identify rectangular connected components by computing how much each component fills its bounding box. If component pixels fill more than 90% of the bounding box, we classify the component as a rectangle; otherwise, we discard the component. We also discard very small and thin rectangular components (with width or height less than 2 pixels), as such small components are unlikely to represent bars and are often due to image artifacts (Figure 5c). However, some of these components do represent very small bars, and we add them back to our set of candidate bars later in the extraction procedure.



Remove background rectangles. The remaining rectangles are likely to include all data-encoding bars, but may also include background rectangles formed by gridlines. In Figure 5c, the background rectangles are colored white, while the true bars are orange or red. Background rectangles are usually shaded with the same color as adjacent rectangles, since they adjoin other background rectangles. On the other hand, bars tend to be shaded with different colors than adjacent rectangles, due to bars being shaded differently or gaps between bars. We test for background rectangles by comparing each rectangle's color to the color of four points outside the rectangle, as shown in the inset figure. We choose outside points by first finding the color of pixels bordering the rectangle. We then move away from the rectangle until we find a pixel whose color differs from the border pixel, or is 5 pixels away from the rectangle. If any



(a) Candidate bars **(b)** Width and height histograms **(c)** Histogram of top and bottom y-values of candidate bars **(d)** y-gradient image

Figure 6: Inferring the orientation of the chart and locating the baseline axis. (a) Candidate bars. (b) Histograms of widths and heights of the bars. Note that the mode of the width histogram (red) is stronger than the mode of the height histogram (blue), indicating that the chart is vertical. (c) To estimate the baseline axis, we build a histogram of the top and bottom y-values of candidate bars and treat the mode of this histogram as our initial estimate of the baseline axis. (d) Many charts draw the baseline axis as a solid line. To refine the axis location, we compute the y-gradient image of the initial chart and then sum the rows to create the y-gradient histogram. We treat the peak of this histogram located nearest our initial estimate as the final baseline axis.

one of the outside points has the same color as the interior of the rectangle, we classify the rectangle as part of the background and discard it; otherwise, we classify it as a candidate bar.

Infer chart orientation. Bar charts may be oriented horizontally or vertically. Since bars encode data using length, they vary in one dimension and remain fixed in the other dimension; e.g., vertical bars vary in height but maintain a fixed width. To identify which dimension varies most we build histograms of the widths and heights of the candidate bars and find the mode of each histogram (Figure 6b). The histogram with the strongest mode represents the constant dimension and gives us the orientation of the chart: vertical if the width histogram has the strongest mode, horizontal if the height histogram has the strongest mode. For brevity, we will describe our algorithms assuming a vertical bar chart, but our system works with horizontal bar charts using analogous analysis techniques.

Baseline axis extraction. For most vertical bar charts the baseline axis is the x-axis that touches the bottom of the bars. However, in some bar charts (e.g., those with negative values), the top of some bars may touch the baseline instead of the bottom of the bars. We therefore build a histogram of the top and bottom y-values of the candidate bars. The mode of this histogram represents the horizontal line that most bars touch, and we use the mode as an initial estimate of the baseline axis (Figure 6c). In practice we have found that noise, antialiasing, and other pixel level artifacts in the charts can sometimes lead to small inaccuracies (e.g., a few pixels) in our estimate of the position of the baseline axis. However, many charts draw the baseline x-axis as a solid line. Thus, we further refine our baseline axis estimate by computing the y-gradient of the original chart and summing the rows of this image to form the y-gradient histogram (Figure 6d). We then treat the peak of this histogram located nearest our estimated baseline axis position as the final baseline position.

We do not extract the y-axis for bar charts. As we will show in the data extraction section, we can extract data values for the bars without knowing the location of the y-axis.

Recovering small bars. Recall that when initially finding rectangular shapes we discard very small rectangles. However, we have found that in a few cases these rectangles may be legitimate bars representing small values. Since bars in a vertical bar chart almost always have the same width, we recover small bars by first computing the average width of the remaining candidate bars and then re-introducing any bars we previously discarded with the same width. Finally, we discard any bar that does not touch the baseline axis. This step leaves us with the final set of candidate bars (Figure 5d).

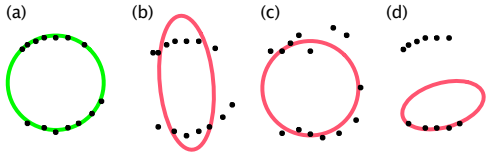


Figure 7: A low scoring ellipse and three high scoring ellipses. (a) A low scoring ellipse maximizes circularity, goodness of fit and coverage. (b) An ellipse with high circularity score, (c) high fit score, and (d) high coverage score.

Pie Charts

Mark extraction in pie charts involves two phases. We first fit an ellipse to the pie. We then unroll the pie and locate strong differences in color to locate the pie sector edges.

Fitting the pie. Although most pie charts are circular, some are elliptical. For greatest generality, we model the pie as an ellipse. Our goal is to fit the ellipse to the set of pixels at the edge of the pie. We start by thresholding the gradient magnitude of the chart to extract gradient pixels (i.e., pixels where the color changes sharply), as these are likely to lie at edges. We set the threshold such that gradient pixels comprise at least $1/30$ of the image pixels, as we empirically found this parameter to be sufficient to fit the ellipse. We use the text region tags to remove gradient pixels due to text.

While the set of gradient pixels is likely to include the pixels at the edge of the pie, it is also likely to include many additional pixels where the color changes sharply. Thus, we adapt the RANSAC [12] algorithm to robustly fit an ellipse to the gradient pixels that are most likely to lie at the edge of the pie while rejecting the other outlier pixels. Our algorithm works as follows: we first randomly select four gradient pixels and compute an ellipse that passes through them using Fitzgibbon et al.'s [13] direct least-squares ellipse fitting method. We then find the set of gradient pixels that are at most 10 pixels away from the ellipse and call them *in-*

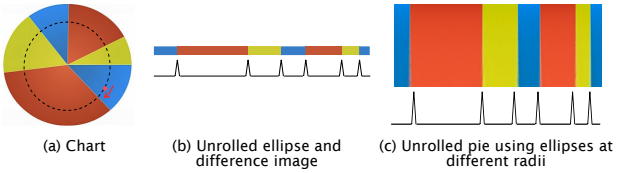


Figure 8: Unrolling the pie. Consider the inner ellipse marked in (a). We unroll the ellipse by sampling points at constant angular intervals (b). Peaks in the horizontal derivative occur at sector edges. To improve edge estimation, we unroll multiple ellipses and sum their horizontal derivatives (c). Peaks in the summed horizontal derivatives occur at sector edges.

liers. Next we check how well the ellipse explains the inliers by computing an ellipse score, which is a weighted sum of three components: the circularity of the ellipse (ratio of the minor to major axis), how tightly it fits the inliers (average distance of an inlier to the ellipse), and coverage (how much of the ellipse is not near inliers). Lower scores indicate better ellipses. Examples of high scoring ellipses are shown in Figure 7. On each iteration, we keep the ellipse if its score is lower than the previous lowest score. We iterate this process 20,000 times, which we experimentally found to work well for the chart images in our corpora.

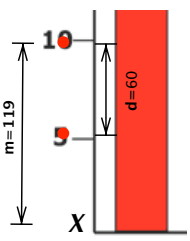
Locating sector edges. To extract the pie sector edges we first "unroll the pie": we sample an ellipse located inside the pie at constant angular intervals to create a one-dimensional ellipse image (Figure 8a). We then take the horizontal derivative of this ellipse image to identify strong changes in color. Color changes indicate transitions between pie sectors, and so the peaks in the derivative give us estimates for the sector edge locations (Figure 8b). To increase robustness of our estimates, we unroll multiple concentric ellipses and sum their horizontal derivatives (Figure 8c). We identify peaks in the summed derivatives by looking for zero crossings of its first derivative, which we find by convolving the summed derivatives with a derivative of a 3-pixel wide Gaussian. This smoothing handles noise and pixel-level artifacts. Some pies include thin borders between pie slices and the smoothing aggregates the peaks at either edge of the border into a single peak. Finally we retain all peaks that are more than one standard deviation above the mean of the summed derivatives.

Data Extraction

In the data extraction step, our goal is to recover the data encoded by each mark. We assume that a mark encodes a tuple of data, where one dimension is quantitative and one is nominal. We recover these tuples by using the geometry of the extracted marks and the text region tags from the classification stage. The output of our data extraction step is a table that contains an ID and a data tuple for each mark.

Bar Charts

To recover the data from a bar chart, we first infer the mapping between image space and data space. We assume a linear mapping, but we believe our data extraction techniques are amenable to other mappings (e.g., logarithmic). The linear mapping is fully defined by (1) a scaling factor between image space and data space, and (2) the minimum value (usually the value at the x-axis).



We first recover the scaling factor by considering the y-axis labels. We identify the y-axis labels that encode data values by finding text regions that are equidistant from the leftmost bar and line up vertically. We then estimate the scaling factor using each pair of value labels, as illustrated in the inset figure. We assume the labels were recovered by our text extraction procedure in the classification stage. The pixel distance between labels "5" and "10" is $d = 60$ pixels, and the estimated scaling factor is $60/(10-5) = 12$ pixels/data unit. We compute the scaling factor for each pair of labels and take the median as our final estimate. For this chart, the median scaling factor across all label pairs is 12.5 pixels/data unit.

We then find the minimum value. We begin with the y-axis label vertically closest to the x-axis. If this label is "0", we set the minimum value to 0. Otherwise, we compute the minimum value using a similar procedure to computing the scaling factors. For each label, we estimate a minimum value by using the y-distance (in pixels) from the label's center to the x-axis, and the chart scaling factor. For example, using the location of the "10" label in the inset figure, we find the pixel distance is $m = 119$. Assuming a chart scaling factor of 12.5, we find $X = 119/12.5 - 10 = -0.48$. The actual minimum is 0; our result is not exact, but close. We set the chart's minimum value to the median of these minimum value estimates. For this chart, the median minimum value was -0.2.

Finally, we assign a nominal value to each bar by associating it with the nearest label below the baseline x-axis.

Pie Charts

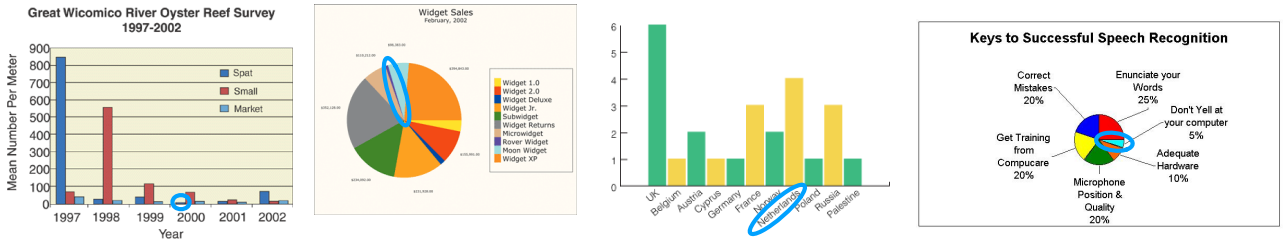
Each sector of a pie encodes two data values: (1) a quantitative value representing the percentage of the whole pie covered by the sector, and (2) a nominal value representing the label of the sector. We compute the percentage as the angular extent between the edges of the sector. We treat the text label nearest the elliptical arc spanning the sector as the nominal label for the sector.

Extraction Results

To test our extraction algorithms, we used the subset of Prasad et al.'s [22] corpus that met our assumptions, which resulted in 52 bar charts and 53 pie charts. For this test we also assumed the chart type was known a priori, noting that our classification stage provides this information.

To generate ground truth, we manually identified the marks and generated data tuples for each mark based on nearby text labels. We used sector labels (for pie charts) and axis labels under the baseline (for bar charts) as the nominal values. If a quantitative label was located near a mark we treated it as the quantitative value of the mark (e.g., Figure 11 left and Figure 12 left). Otherwise, we did not generate a ground truth quantitative value. For pie charts, we converted quantitative values to percentages. For bar charts, we directly used the labeled values.

Using the ground truth data, we found that ReVision successfully extracted all the marks for 41/52 (79%) of bar charts



(a) Small bars are missed (b) Small sectors are missed (c) Mislabeled bars due to rotated labels (d) Mislabeled sectors due to callouts

Figure 9: Mark and data extraction failures. If marks are very small, our algorithms fail to extract them. For example, we fail to extract the circled bar in (a) and the circled sector in (b). Data extraction failures occur when marks are mislabeled, e.g., (c) when labels are rotated, or (d) if the chart places labels far from their associated marks, such as the circled marks.

and 33/53 (62%) of pie charts. Most mark extraction failures occurred because we failed to detect small marks (Figure 9a, b). Our algorithms are designed to treat small, thin regions as decorative elements rather than marks that encode data. With relatively small chart images (on average, 342×452 pixels in our corpus) our algorithms can have trouble separating legitimate marks from these decorative elements.

Accurate data extraction depends on accurate mark extraction. Focusing on the charts for which we were able to extract all marks, we accurately extracted data tuples for 29/41 (71%) of the bar charts and 21/33 (64%) of the pie charts. The most common error was incorrect association of a nominal label to a mark. Our simple closest-label heuristic for associating text labels with marks is especially prone to errors when labels are rotated (Figure 9c), or when marks are small and labeled with callouts (Figure 9d).

We could only generate quantitative ground truth for 12 of the bar charts and 17 of the pie charts for which we extracted the data. The remaining 17 bar charts and 4 pie charts did not have quantitative labels. Examining this subset of charts, we found that our extracted quantitative values were on average within 7.7% of the original data for bar charts and within 4.6% of the original data for pie charts. Figures 11 and 12 show examples of successfully processed chart images.

Running times varied widely with chart size. On a 2.4Ghz MacBook Pro with 2Gb of RAM, extracting marks and data from an average sized pie chart took 1,225s. Extracting marks and data from an average sized bar chart took 100s.

STAGE 3: REDESIGN

The output of the data extraction process is a relational data table. ReVision uses this data to populate a gallery of alternative visualizations (Figure 10). We rank visual encodings by *effectiveness* [20] and display a variety of visualizations in a sorted gallery. ReVision chooses different visualizations depending on the input chart type and extracted data. For the input pie chart in Figure 10a, the gallery presents bar charts to support part-to-part comparisons and divided bar, donut, and treemap charts to support part-to-whole judgments [24]. For the input bar chart in Figure 10b, ReVision generates a bar chart and labeled dot plot to support comparison of individual values, and small dot and box plots to enable assessment of the overall distribution. Note that the y-axis of the input chart does not start at zero; ReVision's bar chart correctly incorporates a zero baseline and displays the data range with more appropriate charts (dot and box plots).

In addition, users can select and compare choices of font and color palette. ReVision includes palettes designed for well-spaced, perceptually discriminable colors [15, 26], as well as palettes from popular charting tools and news magazines. We generate charts using Protovis [2]; viewers can export the Protovis definitions for subsequent modification or reuse. Alternatively, users can export the data to create their own charts using tools such as Microsoft Excel or Tableau [21].

LIMITATIONS AND FUTURE WORK

In its current state ReVision is suitable for generating alternative visualizations for many published bar and pie charts. However, more work is needed to overcome limitations of our classification and extraction methods.

With respect to classification, we currently employ a manual approach to annotating text regions. To achieve fully automatic processing of textual features, we would need to employ text identification and extraction algorithms. We do not yet use the text itself for classification; it is possible that such information might further improve classification accuracy.

Our mark and data extraction algorithms are ripe for additional research. Our extraction techniques do not yet handle textured marks or 3D effects (e.g., due to gradient shading or perspective) both of which are common in our corpora. As 3D effects and textural patterns have been shown to hamper graphical perception [5, 11], it would be beneficial to enable redesigns for such charts. We do not yet parse legends. This causes failures on charts where the legend provides necessary labeling information, as in grouped bar or pie charts with categorical color encodings documented in a legend. We also plan to extend our extraction algorithms to additional chart types, such as bubble, line, and donut charts.

Chart design often involves semantic data or connotations beyond the raw data tuples. For example, the y-axis labels in the leftmost chart of Figure 11 are ordered by increasing educational level. By treating labels as nominal data, our redesign does not respect this ordinal relation. Consequently, our design gallery should support sorting data by both value and the initial observed order. Stylistic choices such as color and typography can also shape the message of a chart; our redesigns do not always retain these messages. However, by extracting the underlying data, our system creates an opportunity for redesign (both automatically and by hand) that would otherwise be prohibitively difficult.

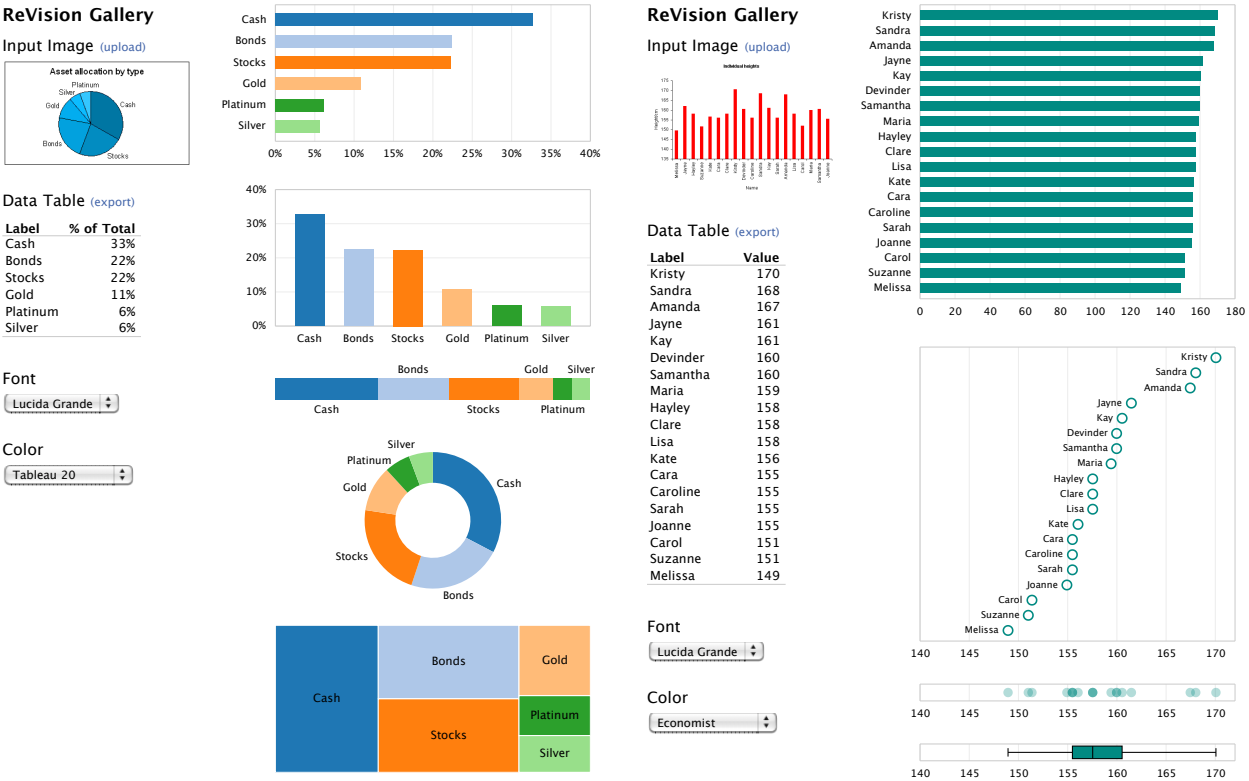


Figure 10: ReVision Design Galleries. Given an extracted data table, the gallery presents a variety of chart types, sorted by proposed perceptual effectiveness rankings [20]. Users can also select and compare color schemes and typefaces.

CONCLUSION

We have presented ReVision, a system that classifies charts, extracts their graphical marks and underlying data table, and then applies perceptually-based design principles to automatically redesign charts. Automated redesign is only one of many possible applications for a system that extracts data from charts. Our techniques could be used to enhance information retrieval systems with chart metadata or to support screen-reading of charts for blind users.

ACKNOWLEDGMENTS

This research has been supported by the Akiko Yamazaki and Jerry Yang Engineering Fellowship Fund, NSF grants IIS-1017745 & IIS-1016920, the Stanford CS Dept., and a gift from Greenplum/EMC.

REFERENCES

1. A. Bosch, A. Zisserman, and X. Munoz. Scene classification via pLSA. *Computer Vision–ECCV*, pages 517–530, 2006.
2. M. Bostock and J. Heer. Protovis: A graphical toolkit for visualization. *IEEE Trans Visualization & Comp Graphics*, 15(6):1121–1128, 2009.
3. M. Boutell, C. Brown, and J. Luo. Review of the state of the art in semantic scene classification. *Rochester, NY, USA, Tech. Rep*, 2002.
4. D. Chen, J. Odobez, and H. Bourlard. Text detection and recognition in images and video frames. *Pattern Recognition*, 37(3):595–608, 2004.
5. W. S. Cleveland. *Visualizing Data*. Hobart Press, 1993.
6. W. S. Cleveland and R. McGill. Graphical perception: Theory, experimentation, and application to the development of graphical methods. *Journal of the American Statistical Association*, 79(387):531–554, 1984.
7. A. Coates, H. Lee, and A. Ng. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. *Advances in Neural Information Processing Systems*, 2010.
8. C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.
9. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE CVPR*, pages 886–893, 2005.
10. G. David. Distinctive image features from scale-invariant keypoints. *Intl Journal Comp Vision*, 60(2):91–110, 2004.
11. S. Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. Analytics Press, Berkeley, CA, 2004.
12. M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, June 1981.
13. A. Fitzgibbon, M. Pilu, and R. Fisher. Direct least square fitting of ellipses. *IEEE Trans Pattern Analysis & Machine Intelligence*, 21(5):476 –480, 1999.
14. S. R. Garner. Weka: The waikato environment for knowledge analysis. In *In Proc. of the New Zealand Computer Science Research Students Conference*, pages 57–64, 1995.
15. M. Harrower and C. Brewer. Colorbrewer.org: an online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1):27–37, 2003.
16. J. Heer and M. Bostock. Crowdsourcing graphical perception: Using Mechanical Turk to assess visualization design. In *ACM CHI*, pages 203–212, 2010.
17. W. Huang and C. L. Tan. A system for understanding imaged infographics and its applications. In *Proceedings of the 2007 ACM symposium on Document engineering*, DocEng '07, pages 9–18, New York, NY, USA, 2007. ACM.

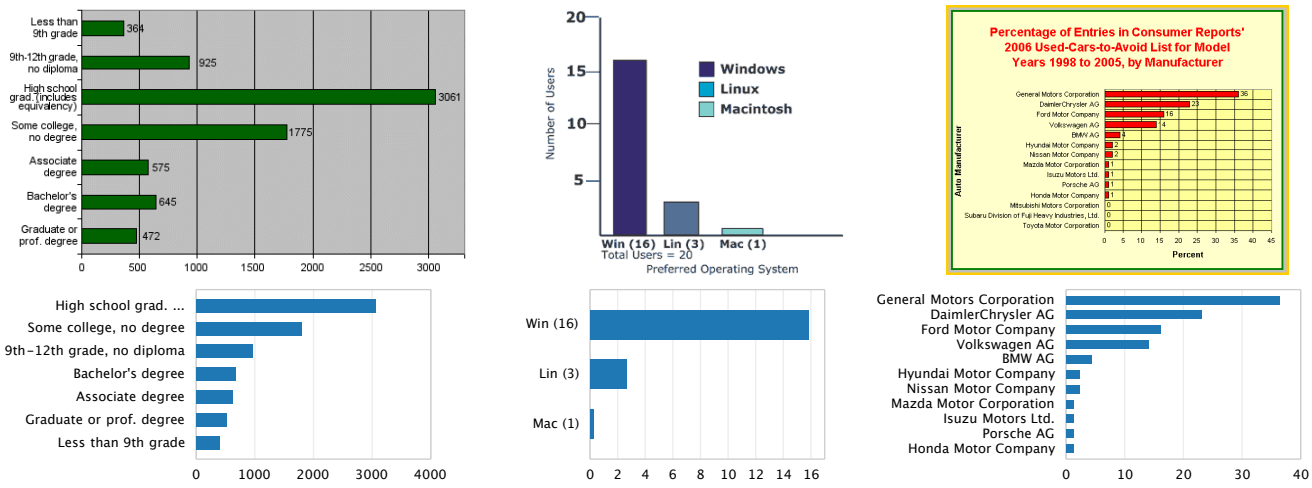


Figure 11: Example ReVision redesigns for input bar charts.

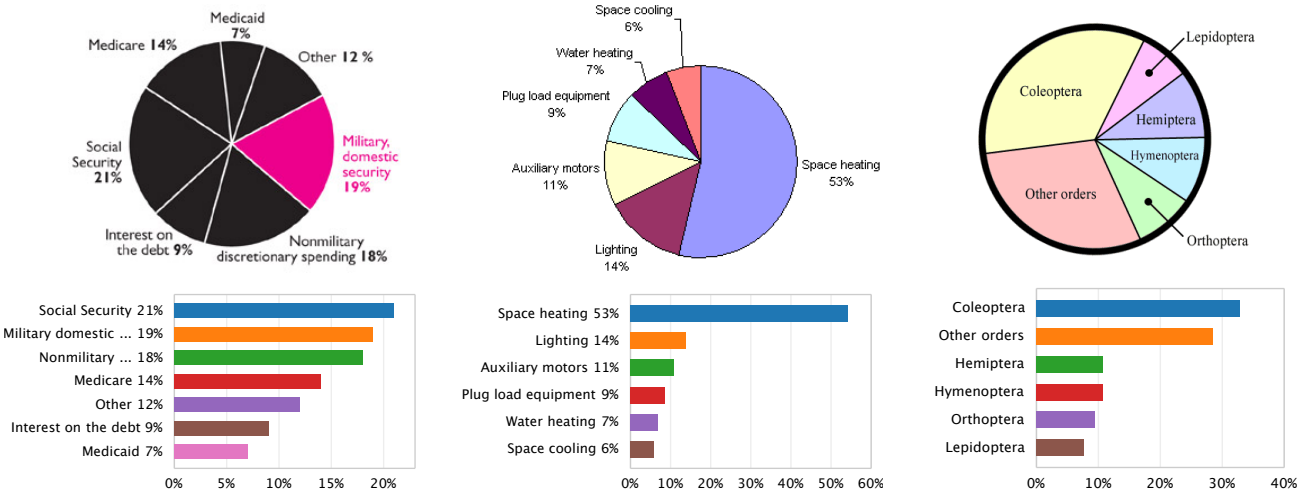


Figure 12: Example ReVision redesigns for input pie charts.

18. W. Huang, C. L. Tan, and W. K. Leow. Model-based chart image recognition. In J. Lladós and Y.-B. Kwon, editors, *Graphics Recognition*, volume 3088 of *Lecture Notes in Computer Science*, pages 87–99. Springer Berlin / Heidelberg, 2004.
19. R. Liu, W. Huang, and C. L. Tan. Extraction of vectorized graphical information from scientific chart images. In *Document Analysis & Recognition (ICDAR)*, pages 521–525, 2007.
20. J. D. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans on Graphics*, 5(2):110–141, 1986.
21. J. D. Mackinlay, P. Hanrahan, and C. Stolte. Show me: Automatic presentation for visual analysis. *IEEE Trans Visualization & Comp Graphics*, 13(6):1137 –1144, 2007.
22. V. Prasad, B. Siddiquie, J. Golbeck, and L. Davis. Classifying Computer Generated Charts. In *Content-Based Multimedia Indexing Workshop*, pages 85–92. IEEE, 2007.
23. M. Shao and R. Futrelle. Recognition and classification of figures in pdf documents. In W. Liu and J. Lladós, editors, *Graphics Recognition. Ten Years Review and Future Perspectives*, volume 3926 of *Lecture Notes in Computer Science*, pages 231–242. Springer Berlin / Heidelberg, 2006.
24. D. Simkin and R. Hastie. An information-processing analysis of graph perception. *Journal of the American Statistical Association*, 82(398):454–465, 1987.
25. C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *IEEE Trans Visualization & Comp Graphics*, 8(1):52 –65, 2002.
26. M. Stone. *A Field Guide to Digital Color*. A. K. Peters, 2003.
27. C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *ICCV*, pages 839 –846, Jan. 1998.
28. E. R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, 1983.
29. J. Yang, Y.-G. Jiang, A. G. Hauptmann, and C.-W. Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Workshop on Multimedia Information Retrieval*, pages 197–206, 2007.
30. L. Yang, W. Huang, and C. Tan. Semi-automatic ground truth generation for chart image recognition. In *Document Analysis Systems VII*, volume 3872 of *Lecture Notes in Computer Science*, pages 324–335. 2006.
31. Y. P. Zhou and C. L. Tan. Hough technique for bar charts detection and recognition in document images. In *Intl Conf on Image Processing*, pages 605–608, sept. 2000.