Curve Separation for Line Graphs in Scholarly Documents

Sagnik Ray Choudhury
Information Sciences and Technology
Pennsylvania State University
sagnik@psu.edu

Shuting Wang
EECS
Pennsylvania State University
sxw327@psu.edu

C. Lee. Giles
Information Sciences and Technology
Pennsylvania State University
giles@ist.psu.edu

ABSTRACT

Line graphs are abundant in scholarly papers. They are usually generated from a data table and that data can not be accessed. One important step in an automated data extraction pipeline is the curve separation problem: segmenting the pixels into separate curves. Previous work in this domain has focused on raster graphics extracted from scholarly PDFs, whereas most scholarly plots are embedded as vector graphics. We report a system to extract these plots as SVG images and show how that can improve both the accuracy (90%) and the scalability (5-8 seconds) of the curve separation problem.

Keywords

Data Extraction; Line Graph; Vector Graphics

1. INTRODUCTION

Line graphs are 2D plots with single or multiple curves in the plotting region. They are heavily used in scholarly papers to compare multiple methods. A line graph is generated from a data table. It would be extremely beneficial to automatically regenerate that table[3]. An important step for that is curve separation: clustering each foreground pixel in separate curves.

Previous work in the curve separation problem has used figures extracted as raster graphics. Figures can be embedded in PDF documents in raster (PNG, JPEG) or vector formats (SVG, PS, EPS, PDF). It is hard to extract vector graphics from PDFs and recently some methods were proposed[2]. These methods extract the bounding box of a figure heuristically, then the PDF page containing the figure is rasterized (converted into a raster image) and the necessary region is cropped.

We examined a dataset of 40,000 figures extracted from 10,000 papers published in top 50 computer science conferences between 2004 and 2014. More than 70% figures were originally embedded in vector graphics formats. This motivated us to extract the figures as SVG (an XML based vector graphics format) images. When any image in a vector graphics format is embedded in a PDF, the commands are just transformed. When the same PDF is converted to an SVG, an inverse conversion happens. All paths or characters in the original image can be restored from the converted SVG. Whereas, if we extract the image by rasterizing a PDF page, all such information is lost. This simple and effective observation has gone largely unnoticed in previous work.

Our system converts any SVG into a flat (no grouping element) and "atomic" SVG where each path is separated into a set of painting commands each with exactly one argument and a bounding box. The paths and characters inside a figure bounding box are combined to produce an SVG image for a line graph. These SVG images are processed for curve separation.

JCDL '16 June 19-23, 2016, Newark, NJ, USA

ACM ISBN 978-1-4503-4229-2/16/06.

DOI: http://dx.doi.org/10.1145/2910896.2925469

2. RELATED WORK

Browuer et al. [1] proposed methods to separate overlapping data points in scholarly scatter plots. A more comprehensive version of their architecture [3] reported curve extraction techniques for line graphs, but only for continuous curves. These works used raster graphics; our system uses a different paradigm and handle more generic cases.

3. EXTRACTION OF VECTOR IMAGES

An input PDF is split into pages and each page is converted into an SVG image using InkScape [1]. A "Path" command in an InkScape SVG can have multiple operations for drawing shapes such as lines, smooth and unsmooth Bézier curves and elliptical arcs. Each "operation" is of the form ("operator", "operand") and can be used in the absolute and the relative mode. Also, each operator can have a sequence of operands instead of a single one. Each path command is an XML node; the color and the stroke patterns are provided as the attributes. Multiple path commands can be grouped together with a grouping command ("g"). Each such group can be grouped further, leading to a hierarchical structure. Each path or group can have a "transform" attribute that changes the co-ordinate system by rotation, scaling or matrix multiplication.

An example path from an InkScape SVG is *"<path id="path170" d="m 2209.6, 3082.45 2114.25, 0 0, 1340.53 -2114.25, 0 0, -1340.53 z" />"*. This path has a *moveto* command and a *relative lineto* command each with a sequence of arguments to draw two horizontal and the two vertical lines. This path can come from a marker or axis in a line graph. But unless we parse the path into smaller path commands (four separate line segments), they can not be combined to create such meaningful shapes.

SVG paths follow a context free grammar expressed through EBNF (Extended Backus-Naur Form). We wrote a parser combinator to parse an SVG path into a sequence of underlying operations. The parsers are regular expression based and capable of

[1] http://personal.psu.edu/szr163/svgconversionresults/converted.html

parsing a single operation. The combinator accepts a sequence of such parsers and returns a new parser as its output.

After retrieving such a sequence of operations (each with one or multiple operands) from a path command, we need to convert each operation into a sequence of *absolute* path commands each with a *single* argument. Any SVG path must start with an absolute move command, but all subsequent operations can be relative. Let us assume we have the absolute starting point for a relative lineto operation with three arguments (*m 2209.6, 3082.45 2114.25, 0 0, 1340.53 -2114.25, 0 0, -1340.53"*). Starting point for the first absolute lineto command will be the starting point for the whole path. End point for this command will be the argument itself. Starting point for the second absolute lineto command will be the end point for the lineto command created with the first argument. It is easy to see that a single operation can be transformed into a sequence of path commands using such a recursive formulation. A sequence of operations can again be treated in the same recursive manner to produce the final commands.

Next, the bounding box for each absolute path is calculated. While that is trivial for some operators (lines/ vertical lines), operations such as ellipse or Bézier curves need to converted into their parametric representations. More details are available in the code repository[2].

A path can have a transform attribute and it can belong to multiple groups. Transform operations from all these elements can change the bounding box coordinates. We first recursively traverse the tree structured SVG to create a "dictionary" data structure with the command ids as keys and a sequence of group ids as values. Each transform operation for a path is parsed (using a parser combinator) into a transform matrix and multiple such matrices (from the path and all the groups the path belongs to) are multiplied sequentially to generate a final transform matrix. This transform matrix is multiplied with the bounding box from the last step to produce the final bounding box for a path. Finally, all paths and characters within a figure bounding box are combined.

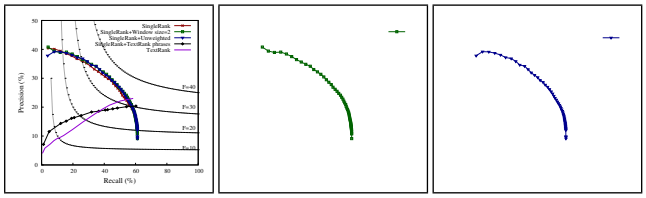
4. CURVE EXTRACTION

A curve in an SVG line graph is a collection of paths. But paths can draw axes or grid lines and tick marks as well. Horizontal and vertical paths painted in black or grey and spanning at least 70% of the image are probable axes or grid lines. Four such lines closest to the image boundary are classified as axes lines. Horizontal and vertical lines intersecting with the axes lines are classified as tic marks.

For color line graphs, all other paths are clustered based on the color and each cluster is output as a curve. Because we have "paths" instead of pixels, overlapping curves are extracted perfectly (figure 1, 2 out of 5 curves are shown).

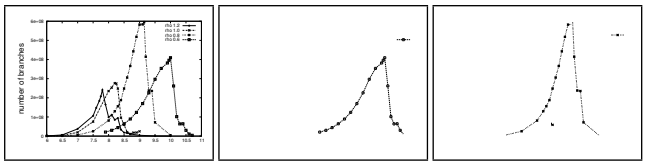
A marker is a collection of paths that looks like a shape: a rectangle, star etc. For black and white (BW) graphs, we create groups of K paths such that any path in a group intersects with at least one other path in the same group. A rectangle is such a group with two horizontal and two vertical lines such that each horizontal line intersects both the vertical lines and vice versa. Similarly, we define rules for all other shapes. More details are available in the code repository[3]. Paths belonging neither to the marker groups nor to the axes and ticks are clustered based on their "style" attribute. If we can't find any marker, each cluster is output as a curve. Else, for each such cluster, we find out the marker group that intersects with the maximum number of elements in that cluster and all paths

[2]https://github.com/sagnik/svgimagesfromallenaipdffigures
[3]https://github.com/sagnik/linegraph-curve-separation



(a) Original line graph (b) Extracted curve. (c) Extracted curve.

Figure 1: Curve extraction from a color line graph.



(a) Original line graph. (b) Extracted curve. (c) Extracted curve.

Figure 2: Curve extraction from a black and white line graph.

in that cluster are added to that marker group (figure 2, best viewed when zoomed in).

We experimented with 200 color and 200 BW line graphs extracted from 400 papers collected randomly from the CiteSeerX repository. Due to the lack of the gold standard data, we use visual evaluation: a curve is considered "correctly extracted" if a person can see at least 80% of the curve in the original plot and at most 10% of any other curve. Precision is defined as the number of correctly extracted curves/ total number of curves extracted and recall as defined as the number of correctly extracted curves/ total number of curves in the plot. The average precision and recall values for the color line graphs are 91% and 90%, respectively. For BW line graphs, these values are both close to 60%. While our approach works well for color graphs, BW graphs need improvement. In comparison, previous work[3] has considered only connected curves (no dashed line) with no markers on mostly synthetic datasets (the graphs were plotted, not extracted). Running time for the algorithm is 5-8 seconds per image on a quad core desktop with 16 GB memory, including the step for the SVG production.

5. CONCLUSION AND FUTURE WORK

We report a method to extract scholarly figures as vector graphics and show how that helps in an important problem: curve separation from line graphs. Compared to previous work, our methods are more robust. They are also scalable and accurate. Future work involves improving the accuracy of the problem for black and white line graphs.

6. REFERENCES

[1] W. Browuer, S. Kataria, S. Das, P. Mitra, and C. L. Giles. Segregating and extracting overlapping data points in two-dimensional plots. In *Proceedings of the 8th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '08, pages 276–279, New York, NY, USA, 2008. ACM.

[2] C. Clark and S. Divvala. Looking beyond text: Extracting figures, tables, and captions from computer science paper. 2015.

[3] X. Lu, S. Kataria, W. J. Brouwer, J. Z. Wang, P. Mitra, and C. L. Giles. Automated analysis of images in documents for intelligent document search. *IJDAR*, 12(2):65–81, 2009.