Scalable Algorithms for Scholarly Figure Mining and Semantics

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

Most scholarly papers contain one or multiple figures. Often these figures show experimental results, e.g, line graphs are used to compare various methods. Compared to the text of the paper, figures and their semantics have received relatively less attention. This has significantly limited semantic search capabilities in scholarly search engines. Here, we report scalable algorithms for generating semantic metadata for figures. Our system has four sequential modules: 1. Extraction of figure, caption and mention; 2. Binary classification of figures as compound (contains sub-figures) or not; 3. Three class classification of non compound figures as line graph, bar graph or others; and 4. Automatic processing of line graphs to generate a textual summary. In each step a metadata file is generated, each having richer information than the previous one. The algorithms are scalable yet each individual step has an accuracy greater than 80%.

CCS Concepts

•**Information systems** → **Information systems applications; Information extraction;** *Digital libraries and archives; Information retrieval;* •**Applied computing** → **Graphics recognition and interpretation;**

Keywords

Figure Semantics; Vector Graphics

1. INTRODUCTION

Scholarly figures are abundant on the web. PubMed Central contains more than 1.7 million figures[7]. We extracted more than 15 million figures from the PDFs in CiteSeerX repository. Most of these figures (line graphs/ bar graphs/ pie charts) are generated from data tables and can be treated differently from natural scene images. Content based image retrieval engines such as Google Images maximize visual similarity between the query and result images. In contrast, in scholarly figure search the goal is to query the data itself. One might be interested in the line graph that shows a support vector machine (SVM) algorithm performs better than a Random Forest on Imagenet data. To support these "data based" queries we define a metadata that combines both the context of the figure and the original data table.

The system architecture is described in figure 1. We generate metadata for a figure in increasing order of information richness. For any figure the our initial metadata contains caption and mention (the paragraph(s) that referred to the figure in the text). During PDF generation, figures can be included in a raster graphics (PNG/ JPEG) or a vector graphics (PS/ PDF) format. For the vector graphics we can extract the words inside the image without OCR[6]. The next metadata contains these words.

Compound figures (figures containing multiple sub figures) are not processed further because the subsequent algorithms expect a single figure. These figures are easily detectable (section 4) but are hard to segment[7].

Next we classify each "single" figure as a line graph, bar chart or other. For line graphs and bar charts, the words inside the figure can be classified in at least seven classes: 1. X axis value, 2. X axis label, 3. Y axis value, 4. Y axis label, 5. Legend, 6. Figure label and 7. Other text. Therefore, for these figures a third metadata is generated from the word class labels (section 6). This allows SQL like queries such as "show me the figure where legend contains *SVM* and X axis ranges from *0.1* to *0.8* and X axis label is *precision*". Many line graphs are used to demonstrate comparisons between different experimental methods. Each curve represents a method, usually denoted by a legend. For these figures, we separate out the curves, associate them with their respective legends and classify each curve as having an increasing, decreasing or stable trend to produce the fourth and final metadata. For example, the final metadata for figure 2a will contain the caption, mention, the words and their class labels and sentences such as: "curve *TextRank* has an **increasing trend**".

Our system takes less than 15 seconds to process a single image. All scalability experiments were done on a single thread of a multi core laptop with 16 GB memory.

For a semantic scholarly search engine, the final goal is to enable very specific queries such as "Show me the figure where SVM has better Precision/Recall curve than Random Forest on ImageNet classification". This system is a first step in that direction. More information such as the experimental method and dataset can be extracted in future.

2. RELATED WORK

This work is continuation of our previous work on figure metadata (caption/ mention) extraction[4] and search[5] with the final goal of developing a complete architecture for analysis of scholarly figures[3]. Previous work has explored metadata extraction for figures in PDF documents. Chen et al.[1, 2] reported a system

SBD'16, July 01 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2928294.2928305

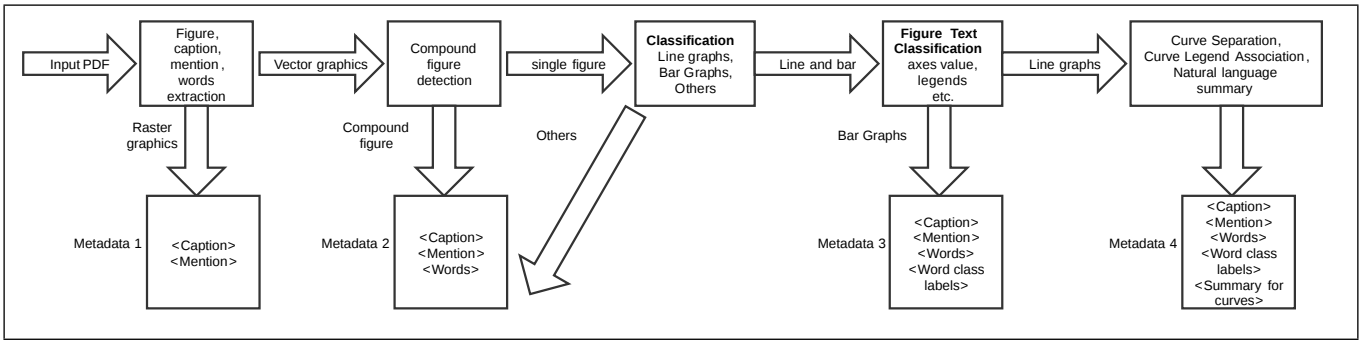


Figure 1: System architecture.

called Diagram Flyer that allows faceted search on the classes of figure text (axes values, legends etc.). However, they didn't discuss the necessary extraction and classification steps in detail. The focus was on the search engine, ranking and query expansion. Linguistics community has long been interested in natural language summary generation for information graphics. While most work has focused on bar graphs, some recent works by Wu et al.[16] and Greenbacker et al[8] have explored line graphs. Most works have considered line graphs collected from the web with a single curve in the plot region. Lu et al.[12] have proposed algorithms for connected curve extraction from scholarly line graphs. But most scholarly graphs do not follow such restrictive assumptions and our algorithm handles more generic cases. Moreover, most previous work has not focused on the scalability aspect of the problem. We report both the time and the accuracy for each algorithm.

3. FIGURE EXTRACTION

Figures are extracted using a system known as pdffigures[6]. Pdffigures produces a JSON file containing the bounding box and caption of the figure and a raster graphics by cropping out the necessary region from the page. If the original figure was in a vector graphics format, words inside the figure and their bounding boxes are also extracted. The code is written in C++ and extremely scalable.

Pdffigures processes one PDF file in less than a second and can be trivially parallelized. We extracted more than 15 million figures from 6.7 million PDF files in CiteSeerX repository by running it on a eight core machine for approximately 20 days. We chose a subset of 10,000 papers published in top 50 computer science conferences from 2004 to 2014 and manually examined 38,000 figures extracted. Around 50% of these figures were compound figures. 22% were stand alone line graphs. These images were subsequently sampled for various experiments in this paper.

Pdffigures has some system dependencies that can not be easily resolved. Therefore we rewrote their algorithm in Scala. Scala is a functional language running on the JVM stack with two advantages: 1. All system dependencies can easily be resolved by Maven and 2. The code can be easily integrated into popular parallel processing architectures such as Apache Spark[1]. Our system extracts both caption and mention for a figure. The code is available on GitHub[2].

4. COMPOUND FIGURE DETECTION

Features	Random Forest	SVM	Logistic Regression
Bag-of-Keypoints	0.7	0.76	0.73
Border Profile	0.78	0.75	0.77
Bag-of-Words	0.71	0.7	0.72
Character Delimiter	0.57	0.65	0.55
Overall	0.85	0.85	0.82

Table 1: Feature performance for compound figure detection.

This module classifies a figure as a compound image or not using both visual features from the raster image and textual features from the figure metadata. Features proposed by Pelka et al.[13] are used. Two visual features are extracted from each image. In an offline step, on average 300 SIFT features are extracted and clustered into a visual words dictionary. During online feature extraction, each image is represented as a frequency distribution over that dictionary. The "Border Profile" feature denotes whether a horizontal vertical separating boarder exists in the figure, i.e., whether there exists a row or column with RGB value [0,0,0] or [255,255,255].

Two textual features are extracted from figure captions: bag-of-words and characteristic delimiter. Stop words from the captions are removed and all remaining words are stemmed. Words co-occuring in both the compound and the non compound figures are removed to keep the distinguishable words. Using the 800 figures in the training set a dictionary with 463 words is created. Each figure is represented as a vector of word frequency over the dictionary. Characteristic delimiters are defined as symbols that separate the sub figures (*"(a)"*). If such patterns exist, the feature is [1,1] else [0,0].

For the experiment, bag of visual keypoints and text words were reduced to 40 features using PCA . A Random forest classifier, a linear kernel SVM and a logistic regression classifier were compared for the binary classification. Five fold cross validation accuracies for different set of features are reported in table 1. All features have significant importance.

There are at most 300 SIFT feature extracted per image. Other features are text based. Therefore, the feature extraction step is extremely scalable: it takes less than one second per image. The code and data is available on Github[3].

5. CLASSIFICATION OF FIGURES

From the last module, we get a single figure in a raster graphics format. This module classifies it one of the following classes: 1.

[1]https://spark.apache.org

[2]https://github.com/sagnik/pdffigures-scala

[3]https://github.com/sagnik/compoundfiguredetection

line graph, 2. bar graph or 3. other. Savva et al.[15]'s method is the state of the art in this problem. As we extract SIFT features in the last module, we tried using them for this task as well. They were not suitable: the F-Score reduced by at least 10% in each class.

5.1 Codebook Generation

First, a codebook of image patches is generated. Each image is converted into a grayscale image; resized into D×D pixels; maintaining the original aspect ratio and padding white pixels whenever necessary. For each image, randomly N points are chosen. For each such point, a P×P pixels patch left cornered at the point is extracted. Patches with variance less than 10% of the image variance are discarded. Selected patches are normalized and reshaped into a $1 \times P^2$ vector.

For the experiment we used D=128, N=100 and P=6[15]. We extracted patches from 498 images (166 from each of the three classes) producing a 498×100×36 matrix. Then ZCA whitening was used to remove non relevant correlations. For ZCA whitening, the original matrix was reshaped into a 498×3600 matrix by concatenating all (100) 36 dimensional patch vectors for each image. After ZCA whitening, the result matrix was reshaped into a 49800×36 matrix and clustered in 200 clusters using K-means clustering. These cluster centers were used as a dictionary of visual words.

5.2 Feature Extraction and Scalability Improvements

The codebook generation is an offline step so it doesn't affect the scalability of the classification process. The next step is to represent each image using the codebook. For that, Savva et al.[15] used a dense sampling approach. For each pixel in a resized (128 × 128) image, a 6 × 6 patch is created and standardized as before. For each patch, a 200 dimensional one hot vector is created with all zeros except the position of the closest cluster center. For each quadrant of the image, these vectors are summed up and four such vectors are concatenated to form a 800 dimensional vector.

The dense sampling step takes on average *92 seconds* for a single image using specialized KD trees[4] for distance computation. For each of the 128^2 pixels the nearest cluster center is calculated, which attributes to its inefficiency.

To improve the scalability, we experimented with two modifications. First, instead of the dense sampling, we randomly sampled 1000 points for the patch creation step. This took approximately *6 seconds* on the same machine but affected the classification accuracy.

For each pixel, our goal is to find out the cluster center that is nearest by Euclidean distance. While there is no direct relation between the Euclidean distance and the Cosine distance, they are the same for unit vectors. Specifically, for two vectors $\mathbf{x}$ and $\mathbf{y}$ with $\|\mathbf{x}\|_2=\|\mathbf{y}\|_2=1$, $\|\mathbf{x}-\mathbf{y}\|_2^2=2-2\cos\angle(\mathbf{x},\mathbf{y})$. The cosine angle ($\cos\angle(\mathbf{x},\mathbf{y})$) between two such vectors is $\mathbf{x}.\mathbf{y}^\mathbf{T}$. Therefore, if we replace the Euclidean distance with the cosine distance, the dense sampling step reduces to following operations: 1. Normalization of row vectors for both the patch and the cluster matrix; 2. Multiplication of the cluster matrix and the patch matrix; and 3. Finding out the indices of the maximum values in the rows of the result matrix. Modern linear algebra libraries are optimized for matrix multiplication, reducing the processing time per image to *6 seconds*. The accuracy does not change much because the direction of patch vectors are more important than the absolute values.

[4]http://docs.scipy.org/doc/scipy-0.16.0/reference/generated/scipy.spatial.KDTree.html

5.3 Classification Experiments and Results

600 images were sampled for the classification experiment. Three classifiers were used: 1. Linear kernel SVM, 2. SVM with RBF kernel and 3. Random Forest with 100 decision trees. The Random Forest classifier performed best. As the focus here is the fast feature extraction process rather than the classifier model, we report the five fold stratified cross validation results using the Random Forest classifier. All other results, code and data for this problem is available on Github[5].

Classification accuracies for three feature extraction methods were compared: 1. Original dense sampling method by Savva et al.[15]; 2. Random sampling of 1000 pixels; 3. Fast dense sampling using Cosine distance and matrix multiplication. Fast dense sampling had comparable accuracy with previous work, with a 15 times improvement in the execution time. On a larger dataset of 2000 images, the fast dense sampling method had F1-scores of 86%, 92% and 82%, for line graphs, bar graphs and other figures, respectively.

6. FIGURE TEXT CLASSIFICATION

Input to this module is a set of text words extracted from line graphs and bar charts. The text is classified in one of the following seven classes: 1. X axis value, 2. Y axis value, 3. X axis label, 4. Y axis label, 5. Legend, 6. Figure Label, and 7. Other text. Most class labels are self explanatory. Sometimes plots contain words to denote regions of particular importance. These words are called "other text".

This step creates a richer metadata that allows faceted search on word labels. Also, for line graphs, the legend words can be combined and associated with respective curves, as done in the next module.

For a word w_i, we have the text, location, and orientation (rotation angle with the horizontal axis) on the image. Multiple features are generated using this information.

1. *Rotation of the word*: Typically Y axis labels are placed vertically and all other texts are placed horizontally.
2. *Distance ratio from the image boundary*: Axes values and labels are typically placed close to the image boundary, whereas legends are placed far inside the image.
3. *Is number/ not*: Axes values usually contain only digits, other words contain both digits and other characters.
4. *Number of words between* w_i *and boundaries of the image*: Usually, Y axis label is placed at the leftmost side of the image, Figure label is placed at the top, and X axis label is placed at the bottom. Therefore, there should not be many words between them and the image boundaries.
5. *Number of words in a rectangle enclosing* w_i: The text density is higher in the legend region compared to any other region. Also, there should be less "pure" numbers in this region.

All features are real valued except the first and the third, which are binary and enum, respectively. The features are designed to ignore the dimension of the input image.

6.1 Experiments

We manually tagged 4363 words from 165 images with class labels. A linear kernel SVM and a Random Forest based classifier with 100 decision trees were compared. Random forest outperformed the other classifier in all evaluation measures. Five fold stratified cross validation results are reported in table 3. Due to space limitation we only report F1-scores. Detailed results, code and data is available on Github[6]. Except for the class label "other

[5]https://github.com/sagnik/figure-classification

[6]https://github.com/sagnik/figure-text-classification

	Feature Extraction Methods								
	Dense Sampling[15]			Random Sampling			Fast Dense Sampling		
Figure Class	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
Line Graph	81	86	83	75	82	78	83	86	**84**
Bar Graph	94	87	**90**	79	90	84	89	89	89
Others	76	76	75	79	63	69	82	79	**80**

Table 2: Figure classification results for multiple feature extraction methods using a Random Forest classifier.

	Classification Models	
Text Class	Linear Kernel SVM	Random Forest
X axis label	82	92
X axis value	91	95
Y axis label	97	97
Y axis value	91	94
Legend	81	92
Figure label	58	78
Other text	5	58

Table 3: Classification results for text words inside a figure.

class", most results are satisfactory. Since only text based features are extracted, this step is extremely scalable, averaging less than one second per image.

7. NATURAL LANGUAGE SUMMARY FOR A LINE GRAPH

This module takes the location and *(word, class label)* pairs for a line graph as input. First an SVG image for the figure is created and the curves are extracted. Then legend words are combined to create legend strings and associated with the respective curves. Finally, curves are analyzed to identify the trends and combined with the previous metadata to produce the final metadata for a line graph (see figure 2).

7.1 SVG Conversion and Curve Separation

A line graph can contain one or multiple connected or disconnected curves drawn with separate markers or colors. Previously, researchers have proposed various methods for this problem. Lu et al.[12] proposed a primitive chain code (PCC) based algorithm for extraction of connected line graphs. Another work[10] proposed connected component analysis.

Most previous approaches used raster graphics for this problem. Figures can be embedded in a PDF as a raster or vector graphics format. In our dataset of CS papers we found almost 75% figures were included in a vector graphics format. A raster graphics is a set of pixel values (binary data) whereas a vector graphics is a set of commands. Figures included as raster graphics can be scalably extracted as the whole image data is embedded in the PDF. Vector graphics are hard to extract because PDF itself is a vector graphics format and the commands for drawing the figure are interleaved with other PDF commands. Existing methods for vector graphics extraction[6, 14] first extract the bounding box of a figure, then the PDF page containing the figure is rasterized (converted into a raster image) and the necessary region is cropped.

Resolution of the resultant image heavily depends on the PDF to image conversion time. For the classification algorithms described before, an image of low resolution suffices. However, the data extraction algorithms expect an image of much higher resolution (at least 70 ppi). Such conversions take 50-60 seconds on average on a standard desktop machine. This severely limits the scalability of the process.

This motivated us to extract the figures as SVG images. SVG is a widely used XML based vector graphics format. Other than the scalability issue, this provides another benefit. PDF and SVG are both vector graphics formats. When any image in a vector graphics format is embedded in a PDF, the commands are just transformed. When the same PDF is converted into an SVG, an inverse conversion happens. All paths or characters in the original image can be restored from the converted SVG, albeit the commands might change because in any vector graphics format the same shape can be drawn by multiple commands. This can be considered as a "loss less" conversion. Whereas if a PDF is rasterized, all commands are lost, making it a "lossy" conversion. This loss less conversion benefits the curve extraction process greatly.

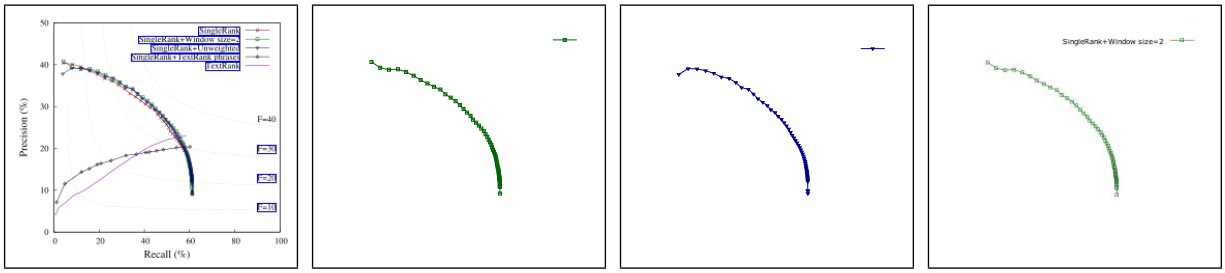
We first split the input PDF into pages. Then the page containing the figure is converted into an SVG image using a popular conversion tool called InkScape[7]. We experimented with multiple such tools; SVGs created by InkScape were found to be most visually similar to the original PDF.

An SVG may contain many commands, of which the most important ones for our purpose are "text", "path" and "image"; the operators for painting text characters, graphics paths and raster images. A path command can contain multiple operators, used for small operations such as drawing a straight line or a Bézier curve. Similarly a text command may contain multiple "tspan" elements, each with multiple characters. Another important command is "transform", which changes the coordinate system of the painting operators using operations such as scale, rotate or multiplication by a transformation matrix. SVG also provides a grouping command "g" that combines multiple paint commands into a group so that a transform command can be applied on the group as a whole. This creates a hierarchical tree structure, as commonly found in most XML files.

Given a figure bounding box, we need a set of SVG paths and characters that are inside. Bounding box for a path command is the rectangle enclosing the bounding boxes of the constituent operations. However, the SVG standard doesn't provide such bounding boxes; they must be calculated. For the text commands, the bounding box for each character needs to be inferred from the font information. If the command has a transform operator, that should be taken into account as well. Also, the commands might belong to one or multiple groups, each with their own transform operation, changing the final bounding box coordinates.

We wrote an SVG parser to solve these problems. The parser takes an SVG produced by InkScape as input and separates out path and text commands. Next, the groups for each such command are identified, effectively reducing the hierarchical tree structure into a flat representation. Specifically, we create a dictionary with the

[7]https://inkscape.org/en/



(a) Legend word identification. (b) Extracted curve. (c) Extracted curve. (d) Curve legend association.

Figure 2: Some steps of our figure summarization pipeline on a line graph extracted from Hassan et al.[9]. The original figure compares precision recall values for unsupervised methods in key phrase extraction. Figure 2a shows the legend words classified and combined. The curves in the original image are overlapping and can not be separated even by the human eye, whereas our algorithms separates them perfectly, as seen in figures 2b and 2c . Other extracted curves are not shown due to space limitations. Figure 2d shows an example of curve legend association. While some texts are wrongly classified as legends in figure 2a, those errors are removed in the association step.

path and text command ids as keys and the group command ids as values.

Next, each path command string is parsed into a sequence of painting operations. SVG paths follow a context free grammar expressed through EBNF (Extended Backus-Naur Form)[8]. We designed a parser combinator for the parsing task. The parsers are regular expression based and capable of parsing a single operation. The combinator accepts a sequence of such parsers and returns a new parser as its output. This approach is extremely modular and scalable.

Next, the bounding box for each operator 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. Bounding box of characters are inferred from the font information available in the "style" attribute of a text command.

The next module is a parser combinator for transform commands. Similar to the previous one, it parses the command string into a sequence of transform operations and each such operation is converted into a transformation matrix. Finally, bounding box for a path or character is calculated by combining the bounding boxes of constituent operations and multiplying that with all transformation matrices for that path or character (from the path itself and the groups it belongs to) in sequence.

More details about the algorithm can be found from our open source repository[9].

Our curve extraction system works on color line graphs. 40% of all line graphs in CS papers are such graphs. These are indeed easier instances of the problem. But extraction from raster graphics can be challenging even for such graphs, especially for overlapping curves. For example, the curves in figure 2a can not be segmented even by the human eye. However, after the SVG creation, such extraction problems reduce to simple text processing.

Each curve in a line graph can consist of one or multiple SVG paths. Each such path is an XML node and the color information is available in the "style" attribute. The text in the plots is usually written in black, background is almost always white or gray and curves themselves are rarely drawn in black. Therefore, we extract paths painted with non black colors using simple regular expressions. Even if a black curve is present, the failure in its extraction wouldn't affect the accuracy for the other curves. Since by the SVG conversion we recover the original vector graphics back, overlapping curves are extracted perfectly, which will be impossible to do with a raster graphics.

While the current system uses only the color information, SVG paths may contain other features such as stroke-width or stroke-opacity. In the future, we plan to use them in a more advanced clustering algorithm for handling black and white line graphs.

7.2 Legend Word Combination and Curve Legend Association

Words classified as "legend" are horizontally combined together to form legend strings. Two words in the same vertical position are combined if there doesn't exist any image pixel between them. To check the presence of an image pixel, the extracted SVG curves are rasterized with a low resolution. A word can have multiple such words as combination candidates. The one with the minimum horizontal distance is chosen. The combined string is put into the queue of words, leading to an iterative algorithm. Our algorithm would fail on multi line legends but they are extremely rare.

Usually for each legend some pixels can be found to the left or right, from the curve associated with it. Such regions can be easily identified, but there are multiple problems with this approach. Our curve extraction algorithm does not extract black curves. Therefore, legends for those curves should not be associated with any other curve. Also, all legend strings can be in the same horizontal line. In the previous steps some words can be wrongly classified and combined as legends (see figure 2a). In these cases a legend string can have multiple curves as candidates.

We cast this as a bipartite matching problem. We define a cost function between a curve C and a legend L as the horizontal distance between L and the pixel from C closest to L. If no pixel from the curve exists within a rectangle of width 20 to the left or right of the legend, the cost is infinity. If the number of extracted curves and legend strings are not equal, the matrix is padded with infinity values as necessary. Then the Munkres assignment algorithm is used to minimize the total cost of the assignment[11].

7.3 Natural Language Summary Generation

Line graphs are generated from data tables and each pixel on a curve is a data point (x,y). For a pair of points $P(x_1, y_1)$ and $P(x_2, y_2)$ on a curve such that $x_2=x_1+1$; if $y_2>y_1$, we say that the curve has an increasing trend at that point. Similarly, if $y_2<y_1$, the curve has a decreasing trend and if $y_2 = y_1$, the curve has a stable trend. Trends for all data points are summed up. If the number

[8]https://www.w3.org/TR/SVG/paths.html#PathDataBNF
[9]https://github.com/sagnik/svgimagesfromallenaipdffigures

of points for a particular trend $\geq 50\%$ of total points, we infer the curve has that trend overall. Else, we report the percentage of each trend in the curve, such as *Curve X has x% increasing, y% decreasing and z% stable trend*. In the future more advanced algorithms can be incorporated. An example summary for a line graph is available online[10].

7.4 Experiments and Scalability

For these experiments we randomly chose a set of 200 color line graphs. For curve extraction, we define precision as number of curves correctly extracted/ total number of curves extracted. The recall is defined as number of curves correctly extracted/ actual number of curves. Precision for curve extraction process is 90.08% and the recall is 88%. Our algorithm does not extract curves drawn with black color. This lowers the recall value. Due to the color based separation, sometimes the algorithm extracts grid lines and other non relevant visual elements as curves. This causes the decrease in precision. For curve legend association the precision as defined as the number of cases where the curve is correctly associated with the legend/ total number of curve legend associations. The precision for our dataset is 81%. The legend combination step is mostly correct. For the curves that were extracted correctly, legends were associated correctly as well. Curves extracted erroneously in the last step contributes to the reduction of precision.

One important aspect of our method is the scalability. Given a figure location and the words inside the figure, it takes on average 4-5 seconds to extract the SVG, and 1-2 seconds to produce the curve summary, including all steps for classification, word combination and legend curve association. This is a huge scalability improvement over standard raster processing methods where it takes 50-60 seconds just to produce the required raster image. Code and data for these experiments are available on Github[11].

8. CONCLUSION AND FUTURE WORK

Scholarly papers contain many figures that are usually ignored in traditional search and data extraction analysis. We report a scalable architecture for analysis of such figures. Figures are extracted, classified and processed to produce a natural language summary. In each step, a searchable metadata is generated. For some tasks we show an improvement in the running time of existing algorithms. Previous work attempted to automatically extract the curves from scholarly line graphs. We show that the process can be made more scalable as well as accurate by adopting a different paradigm. While the current focus is on line graphs, other types of scholarly figures can be semantically indexed later.

9. ACKNOWLEDGEMENTS

We gratefully acknowledge partial support from the National Science Foundation and Qatar Foundation.

10. REFERENCES

[1] S. Z. Chen, M. J. Cafarella, and E. Adar. Searching for statistical diagrams. *Frontiers of Engineering, National Academy of Engineering*, pages 69–78, 2011.

[2] Z. Chen, M. Cafarella, and E. Adar. Diagramflyer: A search engine for data-driven diagrams. In *Proceedings of the 24th International Conference on World Wide Web Companion*, pages 183–186. International World Wide Web Conferences Steering Committee, 2015.

[10] http://personal.psu.edu/szr163/hassan/hassan-Figure-2.html

[11] https://github.com/sagnik/svg-linegraph-processing

[3] S. R. Choudhury and C. L. Giles. An architecture for information extraction from figures in digital libraries. In *Proceedings of the 24th International Conference on World Wide Web Companion, WWW 2015- Companion Volume*, pages 667–672, 2015.

[4] S. R. Choudhury, P. Mitra, A. Kirk, S. Szep, D. Pellegrino, S. Jones, and C. L. Giles. Figure metadata extraction from digital documents. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 135–139. IEEE, 2013.

[5] S. R. Choudhury, S. Tuarob, P. Mitra, L. Rokach, A. Kirk, S. Szep, D. Pellegrino, S. Jones, and C. L. Giles. A figure search engine architecture for a chemistry digital library. In *Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries*, pages 369–370. ACM, 2013.

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

[7] A. García Seco de Herrera, H. Müller, and S. Bromuri. Overview of the ImageCLEF 2015 medical classification task. In *Working Notes of CLEF 2015 (Cross Language Evaluation Forum)*, September 2015.

[8] C. F. Greenbacker, P. Wu, S. Carberry, K. F. McCoy, and S. Elzer. Abstractive summarization of line graphs from popular media. In *Proceedings of the Workshop on Automatic Summarization for Different Genres, Media, and Languages*, pages 41–48. Association for Computational Linguistics, 2011.

[9] K. S. Hasan and V. Ng. Conundrums in unsupervised keyphrase extraction: Making sense of the state-of-the-art. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 365–373, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[10] 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.

[11] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2:83–97, 1955.

[12] 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.

[13] O. Pelka and C. M. Friedrich. Fhdo biomedical computer science group at medical classification task of imageclef 2015. *Working Notes of CLEF*, 2015, 2015.

[14] S. Ray Choudhury, P. Mitra, and C. L. Giles. Automatic extraction of figures from scholarly documents. In *Proceedings of the 2015 ACM Symposium on Document Engineering*, DocEng '15, pages 47–50, New York, NY, USA, 2015. ACM.

[15] M. Savva, N. Kong, A. Chhajta, L. Fei-Fei, M. Agrawala, and J. Heer. Revision: Automated classification, analysis and redesign of chart images. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 393–402. ACM, 2011.

[16] P. Wu, S. Carberry, S. Elzer, and D. Chester. Recognizing the intended message of line graphs. In *Diagrammatic Representation and Inference*, pages 220–234. Springer, 2010.