Recognition and Classification of Figures in PDF Documents

Conference Paper · August 2005

DOI: 10.1007/11767978_21 · Source: DBLP

Mingyan Shao
Northeastern University
6 PUBLICATIONS **70** CITATIONS
SEE PROFILE

Robert P Futrelle
Northeastern University
70 PUBLICATIONS **977** CITATIONS
SEE PROFILE

Recognition and Classification of Figures in PDF Documents

Mingyan Shao and Robert P. Futrelle

Northeastern University, Boston, MA 02115, USA
myshao, futrelle@ccs.neu.edu

Abstract. Graphics recognition for raster-based input discovers primitives such as lines, arrowheads, and circles. This paper focuses on graphics recognition of figures in vector-based PDF documents. The first stage consists of extracting the graphic and text primitives corresponding to figures. An interpreter was constructed to translate PDF content into a set of self-contained graphics and text objects (in Java), freed from the intricacies of the PDF file. The second stage consists of discovering simple graphics entities which we call *graphemes*, e.g., a pair of primitive graphic objects satisfying certain geometric constraints. The third stage uses machine learning to classify figures using grapheme statistics as attributes. A boosting-based learner (LogitBoost in the Weka toolkit) was able to achieve 100% classification accuracy in hold-out-one training/testing using 16 grapheme types extracted from 36 figures from BioMed Central journal research papers. The approach can readily be adapted to raster graphics recognition.

Keywords: Graphics Recognition, PDF, Graphemes, Vector Graphics, Machine Learning, Boosting.

1 Introduction

Knowledge mining from documents is advancing on many fronts. These efforts are focused primarily on text. But figures (diagrams and images) often contain important information that cannot reasonably be represented by text. This is especially the case in the Biomedical research literature where figures and figure-related text make up a surprising 50% of a typical paper. The importance of figures is attested to in the leading Open Access Biomedical journal, *PLoS Biology* which furnishes a “Figures view” for each paper.

The focus of this paper is on figures which are diagrams, rather than raster images such as photographs. Our group has worked on diagram-related topics for some years, including work on diagram parsing, spatial data structures, ambiguity, text-diagram interrelations, vectorization of raster forms of diagrams, and summarization. This paper deals with graphic recognition in the large, describing a system that begins with the electronic versions of papers and leads to a classifier trained by machine learning methods that can successfully classify diagrams from the papers. This will then allow knowledge bases to be built for organized browsing and diagram retrieval. Retrieval will normally involve related text and should be able to retrieve diagrams from queries that use diagram examples or system-provided exemplars.

To apply machine learning, we first convert the original electronic format of the diagram into machine-resident objects with specified geometric parameters. Then we design algorithms to generate attribute statistics for each diagram that will successfully characterize a diagram. There are thus three sequential stages in the processing/analysis chain: *Extraction* of the figure-related graphics from papers, *attribute computation*, and *machine learning*.

In online papers in PDF format, diagrams may exist in raster or vector format. Most published diagrams are in raster format. However, BioMed Central (BMC), a leading Open Access publisher, has to date published about 14,000 papers, of which approximately 40% contain vector formatted figures. In the preliminary research reported here, we have used a small number (36) of BMC vector figures. Although this paper focuses on diagrams available in vector format, the approach is equally applicable to raster formats. They would require an additional preprocessing step, vectorization, a sometimes imperfect process for deriving a vector representation [1, 2, 3].

It might seem straightforward to extract graphic objects from PDFs, which are already in vector format. This is not the case. PDF is a page-space, geometry-based language with various graphics/text state assignments and shifts that must be untangled. PDF has no logical structure at any high level, such as explicitly delimited paragraphs, captions, or figures. Even white space in text is not explicitly represented, other than by a position shift before the next character is rendered. A small number of studies have attempted to extract vector information from PDFs, typically deriving an XML representation of the original [4]. For our analysis work, we use in-memory Java objects.

The document understanding community has been focused on text, perhaps overly focused. For example, Dengel [5] in a keynote devoted to "all of the dimensions" of document understanding, doesn't even mention figures. Much of the work on graphic recognition for raster images has been devoted to vectorization of technical drawings and maps [1]. It rarely goes on to extract structure, much less to apply machine learning techniques to the results. One piece of research on chart recognition from raster images, for bar and pie charts, used hand-crafted algorithms for recognition [6].

For vector figures in CAD and PDF, hybrid techniques have been used which rasterize the vector figures and then apply well-developed raster-based document analysis algorithms. This settles the issue of where on the page the various items appear, but the object identity of the items is lost [7, 4]. Our approach is different, because we render (install) the object references in a *spatial index*, a coarse raster-like spatial array of objects [8, 9, 10]. This combines the best of both worlds; it allows us to efficiently discover sets of objects that obey specified spatial constraints.

The spatial indexing approach can operate at the level of full document analysis to locate and separate out graphics on pages irrespective of the placement and order of the vector commands in the underlying files, be they PDF, SVG, or the result of graphics recognition (vectorization).

There is some work on vector-based figure recognition. A system was developed for case-based reasoning that did matching of new diagrams to a CAD database [11]. Graph matching was used in which the graphs had geometrical objects at the nodes and geometric relations on the arcs.

For PDFs, a brief but useful description of PDF file structure can be found in [12]. The Xed system converts PDF to XML [4]. This is one of the few papers we could find that shows the results of extracting geometric state and drawing information from PDF. Such a result would have to be converted back to in-memory objects, as we do, before further analyses could be done. We have no requirement for XML in our work, since Java objects can be serialized to files and visualized using Java 2D. Their paper describes four similar tools, only one of which, the commercial system, *SVG Imprint*, appears to generate geometric output; the other three produce raster output for figures or entire pages only.

2 Graphics Recognition System for PDF

We accomplish graphics recognition for vector figures in PDF in the three stages as shown in Fig 1. The first stage consists of extracting the graphic and text primitives corresponding to figures. The mapping from PDF to the rendered page can be complex, so an interpreter was constructed to translate the PDF content into self-contained graphics and text objects, freed from the intricacies of the PDF file. Our focus is on vector-based figures and their internal text. Heuristics were used in this study to locate the figure components on each page. The target form for the extracted entities is Java objects in memory (or serialized to files). This allows us to elaborate them as necessary and to do the processing for the next two stages.

The second stage consists of discovering simple graphics entities which we call *graphemes*. A grapheme is a small set of graphic primitives satisfying specified geometric constraints [13] and Fig. 4. We can also consider graphemes in a larger sense as including point statistics such as the number of polygons in a figure, or statistical distributions such as a histogram of line lengths. A number of different grapheme types can be defined in order to extract enough information from a diagram to classify it. In certain cases, graphemes may contain many primitives. Examples include a set of tick marks on an axis or a set of small triangles used as data point markers in a data graph. Such large sets are described as obeying *generalized equivalence relations* [8, 14]. Discovering geometrical relations between objects is aided markedly by a preprocessing stage in which primitives are rendered (installed) in a *spatial index* [9, 10].

The third stage uses machine learning to classify figures using grapheme statistics as descriptive attributes. In this paper we report on supervised learning studies. Statistics for 16 different grapheme types were collected for 36 diagrams extracted from BioMed Central papers. The diagrams were manually pre-classified and used for training and hold-out-one evaluation. A boosting algorithm, LogitBoost from the Weka toolkit [15], was used for multi-class learning. LogitBoost was able to achieve 100% classification accuracy in hold-out-one training/testing. Other learning algorithms we tried achieved less than 100% accuracy. We can't expect any machine learning algorithm to achieve 100% accuracy in the scaled up work we will do involving tens of thousands of diagrams. Nevertheless, the preliminary results are encouraging. Using a large collection of atomic elements (graphemes) to characterize complex objects (entire diagrams) is analogous to the "bag of words" approach which has been so successful in text document categorization and retrieval. Once trained, the learning system can classify new diagrams presented to it for which the grapheme statistics have been computed.

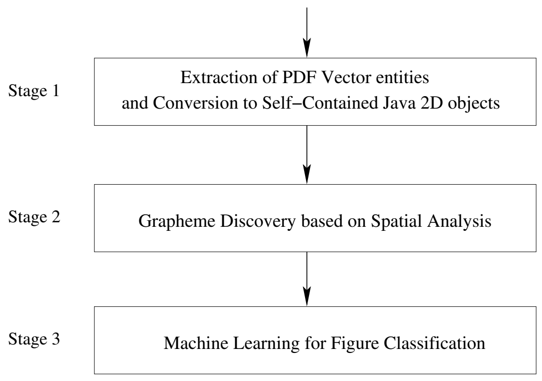
Stage 1
Extraction of PDF Vector entities and Conversion to Self–Contained Java 2D objects
Stage 2
Grapheme Discovery based on Spatial Analysis
Stage 3
Machine Learning for Figure Classification

Fig. 1. Stages of our PDF vector figure recognition system. The first stage consists of extraction of the PDF vector entities in the file and their conversion to *self-contained objects*, Java instances compatible with Java 2D. The second stage involves the discovery of simple items in the figure, *graphemes*, a typical one being two or three primitives obeying geometric constraints such as an arrowhead, or a large set of simply related objects such as a set of identically appearing (congruent) data point markers. The third stage is to use the statistics of various graphemes found in a figure as a collection of attributes for machine learning.

Combining extraction, grapheme discovery, and machine learning for diagram classification is a new approach that bodes well for the future.

3 Extraction of Figure-Related PDF Entities

3.1 Features of PDF Documents and Their Graphics

A PDF document is composed of a number of pages and their supporting resources (Fig. 2). Both pages and resources are numbered objects. Each PDF page contains a resource dictionary and at least one content stream. The resource dictionary keeps a list of pairs of a resource object number and a reference name. A resource object may be a font, graphics state, color space, etc. Once defined, resource objects can be referenced anywhere in the PDF file.

The content streams define the appearance of PDF documents. They are the most essential parts of PDF; they use resources to render text and graphics. A content stream consists of a sequence of instructions for text and graphics. Text instructions include text rendering instructions and text state instructions. Text rendering instructions write text on a page. Text state instructions specify how and where text will be rendered to a page, such as location, transform matrix, word space, text rise, size, color, etc.

Graphics instructions include graphics rendering instructions and graphics state instructions. Graphics rendering instructions draw graphics primitives such as line, rectangle, and curve. Graphics state instructions specify the width, color, join style, painting pattern, clipping, transforms, etc. PDF also provides a graphics state stack so that local graphics states can be pushed or popped to change the graphics state temporarily.

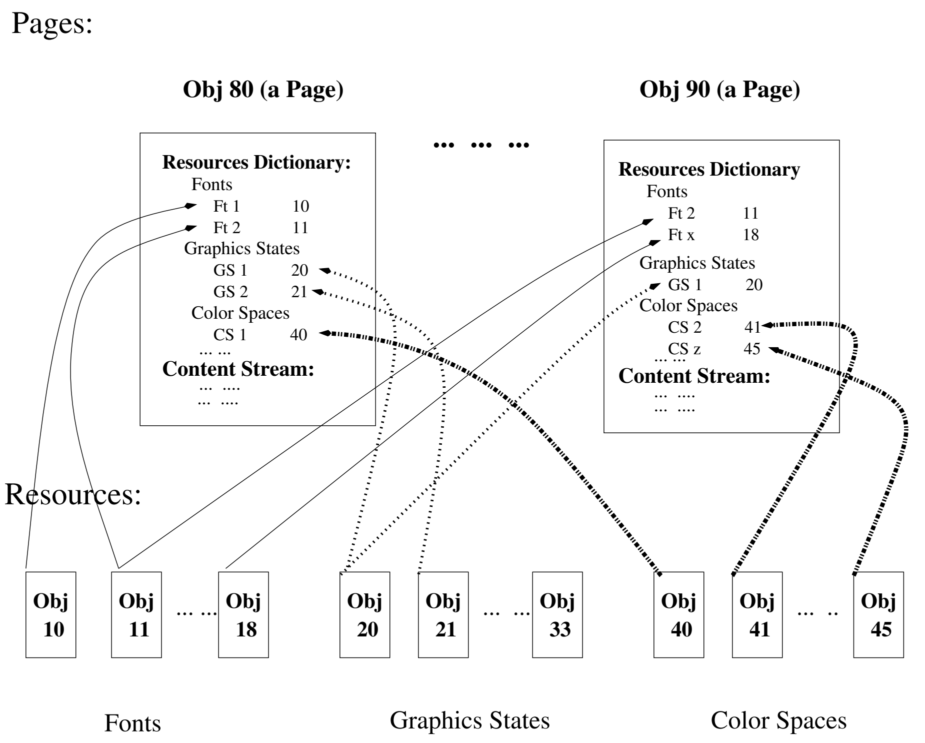

Fig. 2. A simplified PDF structure example. A PDF file is composed of pages and resources such as font, graphics state, and color space. Both pages and resources are defined as objects with a sequence number, a UID. In this example, page 1 is object #80, and font 1 is object #10. These sequence numbers are used as reference numbers when the object is referenced in another object. In this example, object #10 that defines a font is referenced in a page (object #80) object's resource dictionary as "Ft1 10" in which 10 is the font object's sequence number. Once the resource objects are defined, they are globally available, i.e., they can be referenced by any pages in the same PDF file. For instance, object #20 is referenced by two page objects: object #80 and #90. For a useful brief description of PDF structure, see [12].

3.2 Extraction Strategies

To extract graphics, we first translate PDF documents into a format that we can manipulate in software. We apply the open source package, Etymon PJX [16], to translate entire PDF documents into a sequence of Java objects corresponding to PDF objects or instructions. Etymon PJX defines a class for each member of the set of basic PDF commands. It parses the PDF file to create a sequence of object instances, corresponding to the commands in the PDF file, including the argument values given to each command. Thus, for a PDF document, we get Java objects for pages, resources, fonts, graphics states, content streams, etc. Next, we need to determine which Java objects should be extracted. These objects should be the graphics and text inside of figures, as opposed to blocks of text outside the figures proper. The extraction procedure is complicated due to the structural nature of the PDF content stream, and the lack of a simple mapping between positions in the PDF file content stream and positions on the page.

The PDF content stream is a sequential list of instructions. The sequence is important because the sequence of resources (graphics states and text states) defines the local environment in which the graphics and text are rendered. The values of resources can be changed in the sequence, affecting only the instructions that follow. This property makes extraction complicated because to extract either graphics primitives or text inside graphics with all of their related state parameters, we need to look back through the instruction sequence to find the last values of all the parameters needed.

Despite the fact that the content stream is sequential, the instruction sequence in the content stream is not necessarily in accord with their positions on the page. The content stream instruction sequence and positioning on a page are distinct issues in PDF. A PDF document may apply different strategies to write content streams, all leading to the same appearance, though their instructions may be arranged in different orders. Except in specific cases, we cannot apply content stream position information to aid extraction. The drawing order does affect occlusion when it occurs, but occlusion was not a part of this study.

Extraction of Figures. Since some PDF pages only contain pure text or a few simple figures such as tables, which are not dealt with in this study, we can apply the statistics of line primitives to eliminate such a page — if a page has only a few line primitives, then this page does not contain any figure we need to extract. If there are more than a certain number of non-line primitives such as curves or rectangles, we can conclude that this page must contain one or more figures. If there are neither curves nor rectangles in a page, we can still conclude that a PDF page has figures if the count of line primitives is large enough. A similar strategy was used in our preliminary analyses to determine which papers in the BMC collection contained vector-based figures.

Once we conclude that the graphics in a PDF page contains figure material, we extract both graphics rendering instructions and their supporting graphics states. Graphics states can be specified in either the content stream or in separate objects. Graphics state instructions in content streams can be easily extracted as normal instructions, while graphics states in separate objects are extracted using reference and resource dictionaries.

Extraction of Text Within Figures. After extracting all of the graphics elements in a figure, the text inside the figures needs to be extracted. As explained in Section 3.1, the sequence of text and graphics instructions is not necessarily in accord with the sequence in the rendered page. This makes it difficult to decide which part of text instructions in content stream renders the text inside of graphics. The PDF articles published by BioMed Central (BMC) use an Adobe FrameMaker template that results in the PDF content stream structure shown in Fig. 3.

Once the figures and their text have been extracted, we can create PDFs for viewing and validation. This is done by using Etymon PJX tools to generate PDF from the extracted subset of Java objects. This PDF should contain the figures and their text, nothing more nor less.

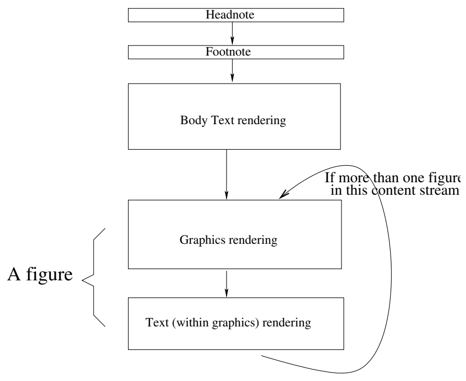
Headnote
Footnote
Body Text rendering
If more than one figure in this content stream
Graphics rendering
A figure
Text (within graphics) rendering

Fig. 3. The content stream structure of BioMed Central (BMC) PDF pages. The content stream of all the BMC PDF pages is organized in the following sequence: head-note, footnote, body-text, graphics instructions (including rendering instructions, graphics state instructions, graphics state references), and text inside the graphics. Graphics, if any, are rendered at the end of each content stream, and text inside the graphics follows the graphics rendering instructions. This structure help us to locate and extract the text inside graphics. The use of this BMC-specific structure is in no way a limitation on our methods. Spatial indexing can be used to locate the objects forming the graphics content in a page irrespective of the content stream sequence.

3.3 An Interpreter to Create Self-contained Objects

The results of the extraction step are Java objects of graphics/text drawing instructions, graphics/text states, etc. This sequence of Java objects exactly mirrors the PDF instructions in the content stream. PDF rendering instructions usually depend on the local environment defined by state instructions. Thus, rendering a graphics object requires the current graphics state and rendering text requires the current font definition. In principle, the entire preceding content stream must be read to get the state parameters needed to render graphics or text.

We have implemented an interpreter to translate these interdependent Java objects into *self-contained objects*. Each self-contained object, either a graphics primitive or text, contains a reference to a state object describing its properties. To enhance modularity, multiple self-contained objects may reference the same state object.

In PDF, the graphic state stack is used to temporarily save the local graphics state so that it will not affect the environment that follows. We deal with this problem by implementing a stack in our interpreter to simulate the PDF state stack so that the local graphics state and the pushed prior state(s) are preserved. Then every self-contained object, no matter how its graphics state is defined, by internal graphics state instructions, external graphics state objects, or via the graphics state stack, references the correct state.

Our interpreter reads all extracted objects and translates and integrates them into self-contained objects that extend Java 2D classes so that they can be manipulated independently from the PDF specification.

4 Spatial Analysis and Graphemes

Up to this point, we have described the extraction of graphics primitives. The ultimate utility of the extracted primitives is for the discovery of the complex shapes and constructions that they comprise, and beyond that to use them in systems that index and retrieve figures and present them to users in interactive applications. A thorough analysis of a figure can involve visual parsing, for example to discover the entire structure of an x,y data graph with its scale lines and annotations as well as data points and data lines, and so forth [17, 9]. Here we describe an alternate approach based on *graphemes*, which is simple compared to full parsing, but still quite useful. A grapheme is typically made up of only two primitives; examples are shown in Fig. 4.

Graphemes allow us to classify figures using a variety of machine learning techniques, as we will see in Section 5. Classification, in turn, can enable indexing and retrieval systems to be built.

A particular grapheme class is described as a tuple of primitives, usually just a pair, that obey constraints on the individual primitives as well as geometrical constraints that must hold among them. For example the *Vertical Tick* tuple in Fig. 4 can be described as a pair of lines, L_1 and L_2 that obey the constraints described in Algorithm 4.1.

Algorithm 4.1. VERTICAL_TICK(L_1, L_2)

```
Comment: Check if a pair of lines ($L_1$, $L_2$) construct a Vertical_Tick
if { $short(L_1)$;
     $vertical(L_1)$;
     $long(L_2)$;
     $horizontal(L_2)$;
     $below(L_1, L_2)$;
     $touch(L_1, L_2)$;
  then $Vertical\_Tick \leftarrow L_1, L_2$
Comment: If $L_1$ is a short vertical line and $L_2$ a long horizontal line, $L_1$ is
 below $L_2$, and they touch at one end of $L_1$, then they form a Vertical_Tick.
```

Graphemes such as *Vertical_Tick* can be discovered by simplified versions of the Diagram Understanding System developed earlier by one of us [9, 18]. One difficult aspect of such analyses is exemplified by the predicates *short()* and *long()* in Algorithm 4.1. This is dealt with by a collection of strategies, e.g., line length histogram analyses, or comparing lengths to the size of the smallest text characters for *short()*.

4.1 Spatial Indexes Aid Grapheme Parsing

The parsing algorithms that define graphemes operate efficiently because a preprocessing step is used to install the primitives in a *spatial index*, allowing constraints such as *below()* and *touch()* to be evaluated rapidly.

A spatial index is a coarse 2D-array of cells (array elements) isomorphic to the 2D metric space of a figure [8, 9, 10, 18] . Each graphics primitive is rendered into the spatial index so that every cell contains references to all graphics primitives that occupy or pass

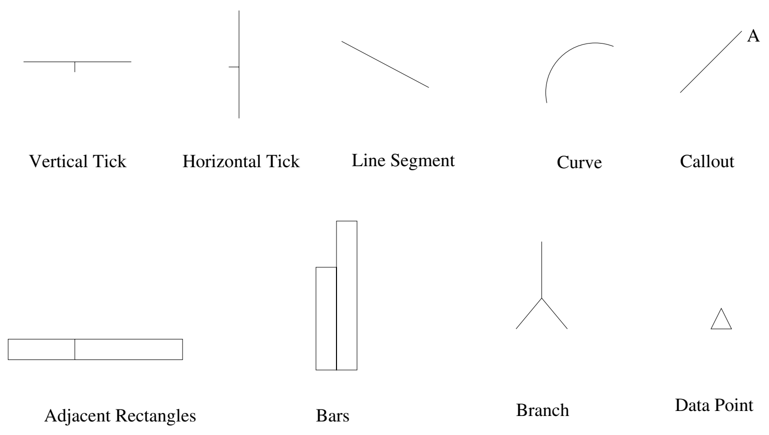

Fig. 4. Some grapheme examples: Vertical Tick, Horizontal Tick, Line, Curve, Callout, Adjacent Rectangles, Bars, Branches, and Data Point

through the cell. Each primitive contains its position in the original PDF drawing sequence in order to faithfully represent occlusions that can occur accidentally or by design.

The spatial index provides a efficient way to deal with spatial relations among graphics primitives, and enables us to deal with various graphics objects such as lines, curves, and text in a single uniform representation. For example, the *touch()* predicate for two primitives simply checks to see if the intersection of the two sets of cells occupied by the primitives is non-empty.

5 Machine Learning for Graphics Classification and Recognition

We analyzed vector graphics figures in PDF articles published by BMC, and defined the following five classes as shown in Fig. 5.

- A *data point figure* is an x, y data graph showing only data points;
- A *line figure* is an x, y data graph with data lines (may also have data points);
- A *bar chart* is an x, y data graph with a number of bars of the same width;
- A *curve figure* is an x, y data graph with only curves;
- A *tree* is a hierarchical structure made of some simple graphics such as rectangles or circles that are connected by arrows or branches.

5.1 Results: Machine Learning of Diagram Classes Using Graphemes

To the extent that distinct classes of figures have different grapheme statistics (counts for each grapheme type), we can use machine learning techniques to distinguish figure classes. We have used supervised learning to divide a collection of figures into the five classes described in Fig. 5.

We extracted figures from PDF versions of articles published by BioMed Central. We examined 7,000 BMC PDFs and found that about 40% of them contain vector graphics.

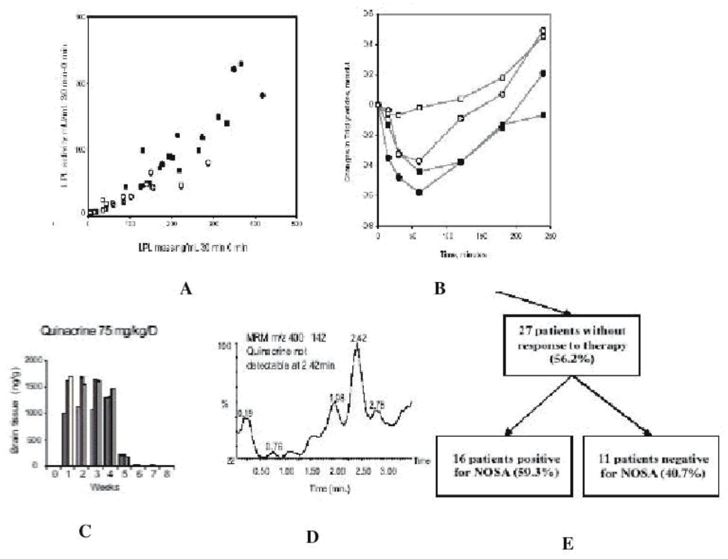

Fig. 5. Five figure classes: **A**: Data graph - points. **B**: Data graph - lines. **C**: Bar chart. **D**: Data graph - curves. **E**: Tree/Hierarchy. The figures used were drawn from BMC papers.

This high percentage, compared to other publishers, appears to be because of specific guidance and encouragement in the BioMed Central instructions for authors.

We extracted vector data from 36 diagrams. A total of 16 different grapheme classes were used as attributes, all geometrical in nature. The counts of grapheme instances in particular diagrams varied from 0 to 120, the latter value being the number of data points in one of the data graph diagrams. Two multi-class learners in the Weka 3, Java-based workbench were used, the Multilayer Perceptron, and LogitBoost. In hold-out-one testing, the perceptron was 94.2% accurate. Its failure on some cases was not unexpected. LogitBoost is a member of the new class of boosting algorithms in machine learning and was able to achieve 100% accuracy on this particular set of 36 diagrams. This excellent result is a testament both to the power of graphemes as indicators of diagram classes and to the power of modern boosting methods. In the future, we will extend these results by analyzing much larger collections of diagrams. As the size and complexity of these collections increases, the accuracy will most certainly be below 100%.

6 Conclusions

This paper has described the design, implementation, and results for a system that can extract and analyze figures from PDF documents, and classify them using machine learning. The system, made up of three analysis stages, was applied to the content of diagrams from research articles published by BioMed Central.

Stage 1. *Extraction* of the subset of PDF objects and commands that comprise vector-based figures in PDF documents. The process required building an interpreter that led to a sequence of self-contained Java 2D graphic objects mirroring the PDF content stream.

Stage 2. *Graphemes* were discovered by analysis of the objects extracted in Stage 1. Graphemes are defined as simple subsets of the graphic objects, typically pairs, with constraints on element properties and geometric relations among them.

Stage 3. *Attribute vectors* for multi-class learners were generated using statistics of grapheme counts for 16 grapheme classes for 36 diagrams, divided into five classes. The best of these learners, LogitBoost from the Weka 3 workbench, was able to achieve 100% accuracy in hold-out one tests.

6.1 Future Work

Besides purely geometrical graphemes, it will be useful to create attributes based on various statistical measures in the figures such as histograms of line lengths, orientations, and widths, as well as statistics on font sizes and styles.

We will include additional classes of vector-based PDF papers that are not created with the standardized FrameMaker-based structure that BMC papers have. The spatial indexing techniques we have described will allow us to locate the figures and figure-related text in such papers irrespective of their position in the PDF content stream sequence.

The approach described here has focused on vector-based diagrams. The great majority of figures published in electronic form are raster based, typically JPEGs. Vectorization of these figures [1, 3, 2, 19], even if imperfect, can generate a vector-based representation of the figure that will allow graphemes to be generated. This in turn will allow systems to be built that can take advantage of figure classification. Such systems could, in principle, deal with all published figures, though most successfully when operating on line-drawn schematic figures, that is, diagrams.

Grapheme-based approaches can form a robust foundation for building full-fledged knowledge-based systems that allow intelligent retrieval of figures based on their information content. In practice, indexing and retrieval of figures will be aided by including figure-related text as a component. We intend to use graphemes as one component of the new diagram parsing system we are developing, which will substantially extend the capabilities of our earlier systems [9, 18]. The fully parsed diagrams that result will allow the construction of more fine-grained knowledge-based systems. These will allow user-level applications to be built that include interactions with diagram internals, linkage between text descriptions and diagram content, and more.

This paper extends our earlier results [10] that also used spatial indexing and machine learning techniques to classify vector-based diagrams. Our papers on a variety of aspects of diagram understanding can be found at http://www.ccs.neu.edu/home/futrelle/papers/diagrams/TwelveDiagramPapersFutrelle1205.html

References

1. Ablameyko, S., Pridmore, T.: Machine interpretation of line drawing images : technical drawings, maps, and diagrams. Springer (2000)
2. Tombre, K., Tabbone, S.: Vectorization in graphics recognition: To thin or not to thin. In: Proceedings of 15th International Conference on Pattern Recognition. Volume 2. (2000) 91–96

3. Lladós, J., Kwon, Y.B., eds.: Graphics Recognition, Recent Advances and Perspectives, 5th InternationalWorkshop, GREC 2003, Barcelona, Spain, July 30-31, 2003, Revised Selected Papers. In Lladós, J., Kwon, Y.B., eds.: GREC. Volume 3088 of Lecture Notes in Computer Science., Springer (2004)
4. Hadjar, K., Rigamonti, M., Lalanne, D., Ingold, R.: Xed: A new tool for extracting hidden structures from electronic documents. In: First International Workshop on Document Image Analysis for Libraries (DIAL'04). (2004) 212–224
5. Dengel, A.: Making documents work: Challenges of document understanding. In: Proceedings ICDAR'03, 7nd Int'l Conference on Document Analysis and Recognition, Edinburgh, Scotland (2003) 1026–1035 Key Note Paper.
6. Huang, W., Tan, C.L., Leow, W.K.: Model-based chart image recognition. In: GREC'03. (2003) 87–99
7. Chao, H., Fan, J.: Layout and content extraction for PDF documents. In: Document Analysis Systems (DAS). (2004) 213–224
8. Futrelle, R.P.: Strategies for diagram understanding: Object/spatial data structures, animate vision, and generalized equivalence. In: 10th ICPR, IEEE Press. (1990) 403–408
9. Futrelle, R.P., Nikolakis, N.: Efficient analysis of complex diagrams using constraint-based parsing. In: ICDAR'95. (1995) 782–790
10. Futrelle, R.P., Shao, M., Cieslik, C., Grimes, A.E.: Extraction, layout analysis and classification of diagrams in PDF documents. In: ICDAR'03. (2003) 1007–1014
11. Luo, Y., Liu, W.: Interactive recognition of graphic objects in engineering drawings. In: GREC'03. (2003) 128–141
12. Hardy, M., Brailsford, D., Thomas, P.: Creating structured PDF files using xml templates. In: In Proceedings of the ACM Symposium on Document Engineering (DocEng'04), Milwaukee, USA, ACM Press (2004) 99–108
13. Futrelle, R.P.: Ambiguity in visual language theory and its role in diagram parsing. In: VL'99. (1999) 172–175
14. Futrelle, R.P., Kakadiaris, I.A., Alexander, J., Carriero, C.M., Nikolakis, N., Futrelle, J.M.: Understanding diagrams in technical documents. IEEE Computer **25** (1992) 75–78
15. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques. 2nd edn. Morgan Kaufmann,San Francisco (2005)
16. Etymon: (Pjx 1.2) http://www.etymon.com/epub.html.
17. Chok, S.S., Marriott, K.: Automatic generation of intelligent diagram editors. ACM Trans. Comput.-Hum. Interact. **10** (2003) 244–276
18. Futrelle, R.P.: (http://www.ccs.neu.edu/home/futrelle/diagrams/demo-10-98/) The Diagram Understanding System Demonstration Site.
19. Shao, M., Futrelle, R.P.: Moment-based object models for vectorizaiton. In: IAPR Conference on Machine Vision Applications (MVA2005). (2005) 471–475