Graphics Recognition in PDF documents

Mingyan Shao and Robert P. Futrelle

College of Computer and Information Science
Northeastern Univeristy
Boston, MA 02115
E-mail:{myshao, futrelle}@ccs.neu.edu

Abstract

Much of the work on graphics recognition for raster-based input focuses on accurate and rapid discovery of primitives such as lines, arrowheads, and circles. Some of the work goes further to discover higher-level objects such as circuit elements, pie-chart components, and architectural components. Some systems work to discover even higher levels of organization. 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. The mapping from PDF to the rendered page can be complex, so an interpreter was constructed to translate the PDF content into a set of self-contained graphics and text objects, freed from the intricacies of the original PDF file. The second stage consists of discovering simple graphics entities which we call *graphemes*, the simplest being a pair of primitive graphic objects satisfying certain geometric constraints. The third stage uses machine learning to classify figures using grapheme statistics as descriptive attributes. Figure recognition can then be accomplished by applying a classifier so trained to the attribute set of a figure to be recognized (classified). The system is implemented in Java. In the study reported here, 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 diagrammatic figures from BioMed Central research papers. The approach can readily be adapted to raster graphics recognition; once primitives are found, graphemes can be constructed from them and used in the learning-based recognition and classification system.

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

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. We arrived at this figure by tabulating statistics from papers in a variety of journals. The importance of figures is attested 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, e.g., photographs. This new 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 the 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-chosen exemplars.

But to apply machine learning, we must first specify a set of attributes that, taken together, can successfully characterize a diagram. Algorithms must be designed and applied to generate attribute statistics for each diagram. This step in turn depends on converting the original electronic format of the diagram into machine-resident objects with specified geometric parameters. The attribute statistics can then be generated from these objects.

Taken in order then, there are three sequential stages in the processing/analysis chain: *Extraction* of the figure-related graphics from papers, *attribute computation*, and *machine learning*. The three stages are summarized in Section 2 and developed in more detail in the sections that follow it.

In online papers in PDF format, diagrams may exist in raster or vector format; the great majority in the Biomedical literature are in raster format. This paper focuses on diagrams available in vector format. But our 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].

Though most published diagrams are in raster format, BioMed Central (BMC), a leading Open Access publisher, has, to date, published about 12,000 papers, of which approximately 40% contain vector formatted figures. In the preliminary research reported here, we have used a small subset of these BMC vector figures.

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 [2]. But to proceed to the second and third stages of our analysis, we need in-memory, manipulable program objects. We've chosen to use Java for this object representation.

The document understanding community has been focused on text. We would suggest, overly focused. For example, Dengel [3] 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, with the assumption that manual cleanup is often needed, e.g., in CAD applications [1, 4]. One piece of research on chart recognition from raster images, for bar and pie charts, used hand-crafted algorithms for recognition [5]. But much of the research stops at the vectorization stage and does not go on to extract structure, much less to apply machine learning techniques to the results.

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 [6, 2]. Our approach is quite different, because we render (install) the object references in a spatial index, which is a raster-like spatial array of objects, typically a low resolution array [7, 8, 9]. This combines the best of both worlds; it allows us to efficiently discover sets of objects that obey specified spatial constraints.

There is some work on vector-based figure extraction. A system was developed for case-based reasoning, basically a system that did matching of new diagrams to a CAD database of ones found earlier (ref: Yan). Graph matching was used in which the graphs had geometrical objects at the nodes and geometric relations on the arcs. The approach was complicated by attempts to assign direction senses to line segements, which seems unsupportable.

For PDFs, a brief but useful description of PDF file structure can be found in [10]. The Xed system converts PDF to XML and is described in [2]. This is one of the few papers we could find that shows the results of extracting geometric state and drawing information from PDF. As we have said, such a result would have to be converted back to in-memory objects before further analyses could be done. We have no need 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

In our approach, 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 a set of self-contained graphics and text objects, freed from the intricacies of the original PDF file. Since our focus is on vector-based figures and their internal text, heuristics are used to locate this material 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*, the simplest being a pair of primitives satisfying certain geometric constraints [11]. A large number of different grapheme types are defined in an effort to extract as much information from the diagram as can be done using these "atomic" elements. Single 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 a *generalized equivalence relations* [7, 12]. Discovering geometrical relations between objects is aided markedly by a preprocessing stage in which primitives are rendered (installed) in a *spatial index* [8, 9].

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 diagram figures extracted from BioMed Central papers. The diagrams werepre-classified and used for training. A boosting algorithm, LogitBoost from the Weka toolkit [13], 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 be doing that will involve tens of thousands of diagrams, using a mix of supervised and unsupervised methods. Nevertheless the preliminary result is heartening. 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.

Combining extraction, grapheme discovery, and machine learning for diagrams is a new approach. The techniques developed and the results achieved in this preliminary study bode 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. A 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 its reference name. A resource object may be font, graphics states, color space, etc. Once the resource objects are defined, they can be referenced within any page in one PDF file with a reference name.

The content streams define the appearance of PDF documents. They are the most essential parts of PDF since they are the parts that utilize resources to render text and graphics, etc. A content stream consists of a sequence of instructions for text and graphics. Text instructions include text state instructions and text rendering instructions. 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 states include width, color, join style, painting pattern, clipping, transforms, etc. Graphics states can be specified either in internal graphics state instructions or in some referenced external graphics state objects. PDF also provides a graphics state stack so that local graphics states can be pushed or popped to change the graphics state temporarily and then return to a previous state.

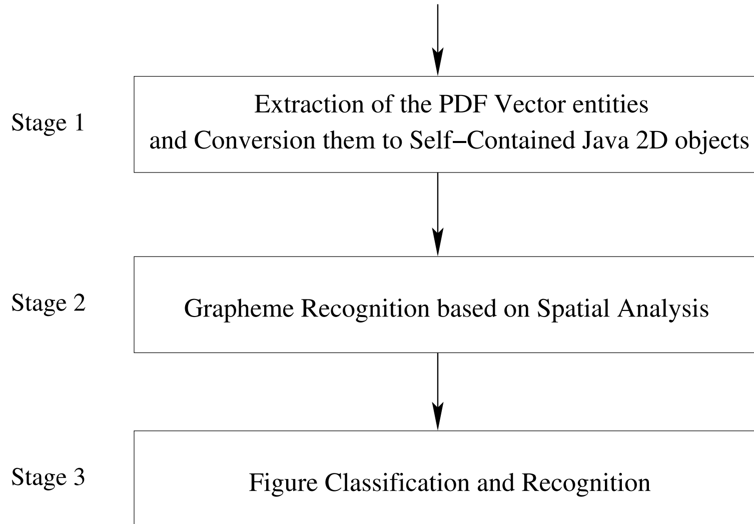

Figure 1: Stages of our PDF vector figure recognition system. The first stage consists of extraction of the PDF vector entities in the file and their conversion to *self-contained objects*, Java instances compatible with Java 2D. The second stage involves the discovery of simple items in the figure, *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. In the study reported here, we have applied supervised learning to classify diagrams.

3.2 Extraction strategies

To extract graphics, first we need to translate PDF documents into some format we can manipulate in software. We apply the open source package, Etymon PJX [14], to translate entire PDF documents into Java objects corresponding to PDF objects or instructions. Thus, for a PDF document, we get Java objects for pages, resources, fonts, graphics states, content streams, etc. Given these Java objects, we need to determine which objects should be extracted and which not. The objects we need to find and extract are 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.

First, 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, and the change affects only the instructions that follow the change and before the next change. This property makes extraction complicated because if we want 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.

Second, 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. In fact, the content stream instruction sequence and positioning on a page are totally different issues in PDF. Moreover, a PDF document may apply different strategies to write content streams to produce the same appearance, though their instructions may be arranged in different sequences. This property also makes extraction difficult. We can't apply content stream position information to help extraction.

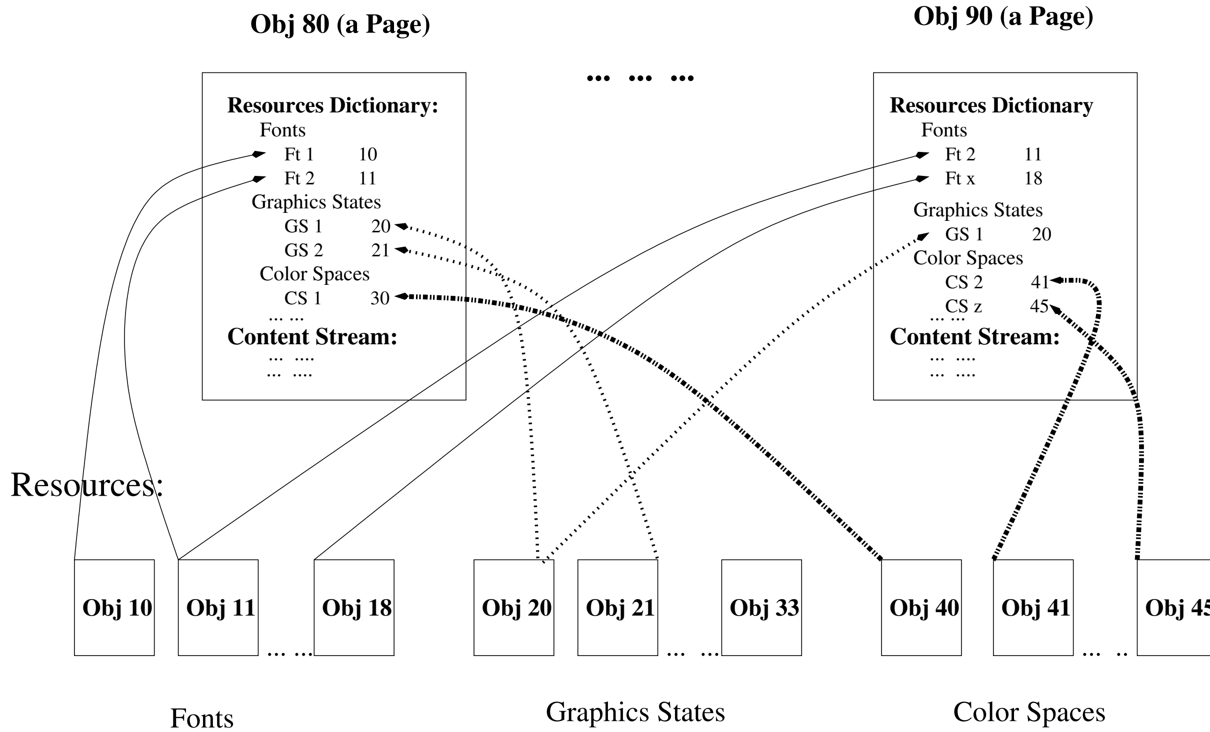

Figure 2: A simplified PDF structure example. A PDF file is composed of pages and resources such as font, graphics state and color space. Both page and resources are defined as objects with sequence number. In this example, page 1 is object #80, and font 1 is object #10. These sequence numbers are used as reference numbers when the object is referenced in another object. In this example, object Font1 is referenced in page1's resource dictionary as 'Font1 10' in which 10 is Font1's object 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.

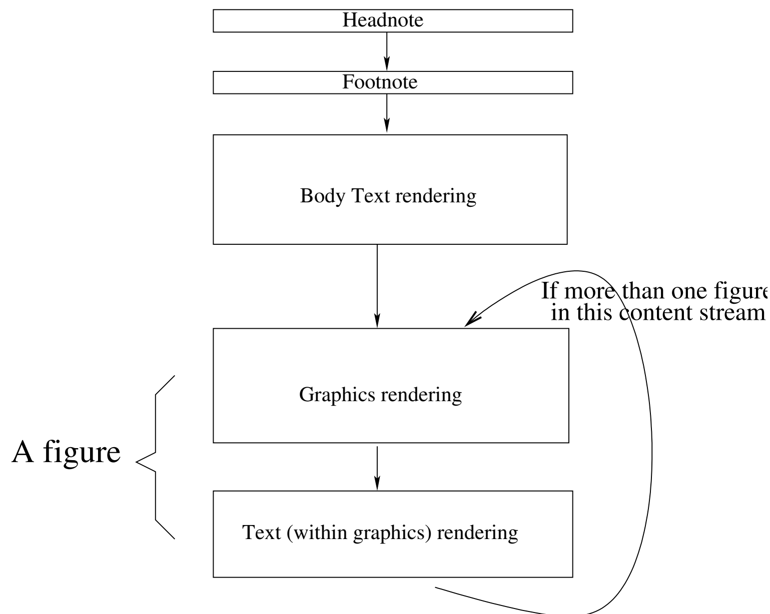

Figure 3: The content stream structure of BioMed Central (BMC) PDF papers. The content stream of all the BMC PDF pages is organized in the following sequence: head-note, footnote, body-text, graphics instructions (including rendering instructions, graphics state instructions, graphics state references), and text inside the graphics. In BMC PDF papers, 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. As of Oct. 2005, there are approximately 12,000 BMC papers published, all with the same PDF structure.

3.2.1 Extraction of Figures

To extract figures, we devised a selection strategy to decide whether a page has figures or not. The selection strategy is based on graphics primitive statistics for each page. Since some PDF pages only contain pure text or a few simple figures such as tables, which are not of interest, 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 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.

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 with the help of reference and resource dictionary. We first read the reference instructions of these graphics state objects and get their reference names, go to resource dictionary to find the object sequence numbers of these reference names, and then access and extract the actual graphics state objects using the sequence numbers.

3.2.2 Extraction of Text within Figures

After extracting the all the graphics elements in a figure, the text inside the figures then 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. Fortunately, since we use PDF articles published by BioMed Central (BMC) they use a standard Adobe FrameMaker template to write content stream for their PDF documents. The content

stream in their PDF documents is arranged so that all the figure content, graphics and the text inside the figures are at the end of the content stream. The text follows the graphics command in this final segment, so that it can be reliably identified and extracted. The BMC content stream has the structure shown in Fig. 3.

Text instructions include text rendering and font references. A font reference is required for all rendered text. Usually the text instructions start with a font reference instruction that the text rendering instructions will utilize until another font reference is specified. If the text following the graphics instructions doesn't start with a font reference, its font must have been declared somewhere before the graphics. This fact requires us to keep the last font reference when we go through the content stream so that once we get the text we need, we can immediately use the last font reference to get the correct font. Given the font reference, we look it up in the resource dictionary to get the object sequence number of this font, then access and extract the font definition object.

As an intermediate result, we can create PDFs for viewing and validation. This is done by using Etymon tools to generate PDF from the extracted subset of Java objects. These should contain only the figures and their associated text.

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, and fonts etc. Since they are simply a translation into Java format of PDF instructions or objects, it is difficult to manipulate them since they directly mirror the sequential PDF ”code”.

The extracted Java objects are stored as a sequence, mirroring the PDF content stream. PDF rendering instructions usually depend on the local environment defined by state instructions. For instance, a graphics rendering object depends on its graphics state object and a text rendering object depends on its font definition object. In principle, the entire preceding content stream must be read to get the state parameters needed for a graphics primitive.

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 it. 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: internal graphics state instructions, external graphics state object, or graphics state stack, references the correct state.

Our interpreter reads every object created by Etymon PJX 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 [15, 8]. But we have found another level of analysis, *graphemes*, which is simple compared to full parsing, but still very useful. A grapheme is a small object 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. Different graphemes can be used to characterize different classes of figures, as we will see in Section 5. Classification, in turn,

enables 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: Decide if a pair of lines L_1 and L_2 construct aVertical_ Tick

$$\textbf{if} \begin{cases} short(L_1); \\ vertical(L_1); \\ long(L_2); \\ horizontal(L_2); \\ below(L_1, L_2); \\ touch(L_1, L_2); \end{cases}$$

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 are touching at one end, then they construct 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 [8, 16]. One difficult aspect of such analyses is exmplfied 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, as well as 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. Each graphics primitive is rendered into the spatial index so that every cell contains references to all graphics primitives that occupy or pass through the cell. Each cell also records the position of each primitive in the 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 primitive 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 analyze vector graphics in PDF articles published by BMC, and define the following five figure classes that are also shown in Fig. 5.

- A *data point figure* is a x, y data graph showing only data points;
- A *line figure* is a x, y data graph with data lines (may also have data points);
- A *bar chart* is a x, y data graph with a number of bars with the same width;
- A *curve figure* is a 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.

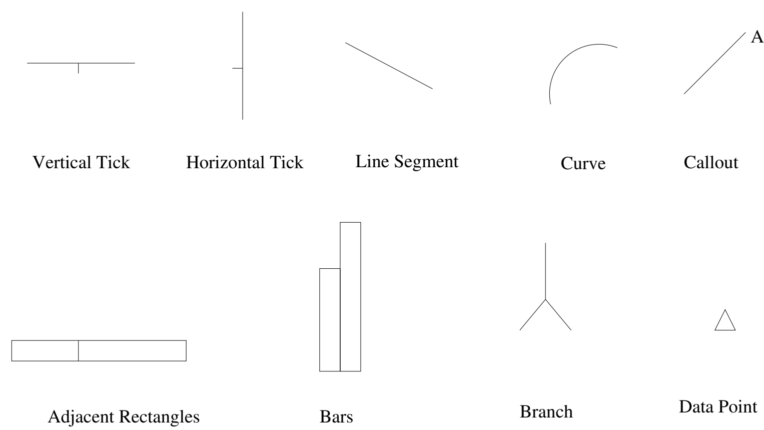

Figure 4: Some grapheme examples: Vertical Tick, Horizontal Tick, Line, Curve, Callout, Adjacent Rectangles, Bars, Branches, and Data Point

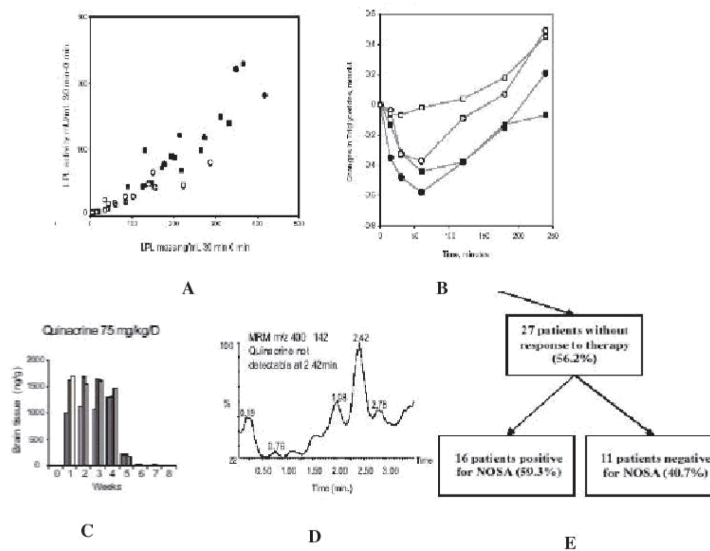

Figure 5: Five graphics classes: **A**: Data point figure. **B**: Line figure. **C**: Bar chart. **D**: Curve figure. **E**: Tree/Hierarchy. The original graphics are from BMC articles.

5.1 Machine Learning - Classifying Diagrams using Graphemes

To the extent that distinct classes of figures have different grapheme statistics (grapheme types and counts), we can use machine learning techniques to distinguish figure classes based on the statistics.

In this study, 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. They publish electronic articles online under an Open Access license, and they actively support data mining. Their vector graphics make it possible for us and others to do data mining of figures. We examined 7,000 BMC PDFs and found that about 40% of them contain vector graphics. (This is an unusually high percentage and is encouraged by BioMed Central's progressive policies)

For the preliminary study reported here, we extracted vector data from 36 diagrams. A total of 16 different graphemes 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. LogitBoost is a member of the new class of boosting algorithms in machine learning and was able to achieve 100% accuracy on the set of 36 diagrams. This excellent result is a testament both to the power of graphemes as indicators of diagram class and to the power of modern boosting methods.

6 Conclusion

This paper has described the design, implementation, and results of a system made up of three analysis stages. The system was applied to the content of diagrams from research articles published by BioMed Central.

Stage 1. The *extraction* of the subset of PDF objects and commands that comprise vector-based figures in PDF documents. The process required building an interpreter that lead 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 between them.

Stage 3. Attributes 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.

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. A typical attribute of this type would be the number of upper and lower case Greek characters. Some of these attributes are redundant, but this presents no problem for contemporary machine learning algorithms.

The approach described here has focused on vector-based diagrams. We fully realize that the great majority of figures published in electronic form are raster based, typically JPEGs. Vectorization of these figures [17, 18], 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.

The grapheme approach can serve as a foundation for building full-fledged knowledge-based systems that allow intelligent retrieval of figures. In practice, indexing and retrieval of figures will be aided by including figure-related text as a component. We intend to use graphemes as an additional component in the new diagram parsing system we are developing. The fully parsed diagrams that result will allow the construction of much more fine-grained knowledge-based systems. These will allow user-level applications to be built that include interactions with diagram internals, linkage to text descriptions, and so forth.

References

[1] S. Ablameyko and T. Pridmore, *Machine interpretation of line drawing images : technical drawings, maps, and diagrams*, Springer, 2000.

[2] K. Hadjar, M. Rigamonti, D. Lalanne, and R. Ingold, "Xed: A new tool for extracting hidden structures from electronic documents," in *First International Workshop on Document Image Analysis for Libraries (DIAL'04)*, pp. 212–224, 2004.

[3] A. Dengel, "Making documents work: Challenges of document understanding," in *Proceedings ICDAR'03, 7nd Int'l Conference on Document Analysis and Recognition*, pp. 1026–1035, (Edinburgh, Scotland), Aug 2003. Key Note Paper.

[4] K. Tombre and S. Tabbone, "Vectorization in graphics recognition: To thin or not to thin.," in *Proceedings of 15th International Conference on Pattern Recognition*, **2**, pp. 91–96, Sep. 2000.

[5] W. Huang, C. L. Tan, and W. K. Leow, "Model-based chart image recognition.," in *GREC'03*, pp. 87–99, 2003.

[6] H. Chao and J. Fan, "Layout and content extraction for PDF documents.," in *Document Analysis Systems (DAS)*, pp. 213–224, 2004.

[7] R. P. Futrelle, "Strategies for diagram understanding: Object/spatial data structures, animate vision, and generalized equivalence," in *10th ICPR*,

[8] R. P. Futrelle and N. Nikolakis, "Efficient analysis of complex diagrams using constraint-based parsing.," in *ICDAR'95*, pp. 782–790, 1995.

[9] R. P. Futrelle, M. Shao, C. Cieslik, and A. E. Grimes, "Extraction, layout analysis and classification of diagrams in PDF documents.," in *ICDAR'03*, pp. 1007–1014, 2003.

[10] M. Hardy, D. Brailsford, and P. Thomas, "Creating structured pdf files using xml templates," in *In Proceedings of the ACM Symposium on Document Engineering (DocEng'04)*, pp. 99–108, ACM Press, (Milwaukee, USA), October 2004.

[11] R. P. Futrelle, "Ambiguity in visual language theory and its role in diagram parsing.," in *VL'99*, pp. 172–175, 1999.

[12] R. P. Futrelle, I. A. Kakadiaris, J. Alexander, C. M. Carriero, N. Nikolakis, and J. M. Futrelle, "Understanding diagrams in technical documents," *IEEE Computer* **25**, pp. 75–78, 1992.

[13] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, Morgan Kaufmann,San Francisco, 2nd ed., 2005.

[14] "Etymon pjx 1.2." http://www.etymon.com/epub.html.

[15] S. S. Chok and K. Marriott, "Automatic generation of intelligent diagram editors," *ACM Trans. Comput.-Hum. Interact.* **10**(3), pp. 244–276, 2003.

[16] R. P. Futrelle, "The diagram understanding system demonstration site." http://www.ccs.neu.edu/home/futrelle/diagrams/demo-10-98/.

[17] J. Lladós and Y.-B. Kwon, eds., *Graphics Recognition, Recent Advances and Perspectives, 5th InternationalWorkshop, GREC 2003, Barcelona, Spain, July 30-31, 2003, Revised Selected Papers*, Lecture Notes in Computer Science **3088**, Springer, 2004.

[18] M. Shao and R. P. Futrelle, "Moment-based object models for vectorizaiton," in *IAPR Conference on Machine Vision Applications (MVA2005)*, pp. 471–475, 2005.