Analysing layout information: searching PDF documents for pictures

Brigitte Mathiak, Andreas Kupfer, Richard Münch, Claudia Täubner, and **Silke Eckstein**
Institut für Informationssysteme, TU Braunschweig, Germany
Institut für Mikrobiologie, TU Braunschweig, Germany
b.mathiak, a.kupfer, r.muench, c.taeubner, s.eckstein@tu-bs.de

Abstract

Roughly 5 percent of the web's content consists of PDF documents, yet most web mining applications simply ignore them or make a standard to-ASCII conversion. We believe that PDF documents contain valuable information, eg. most scientific publications are stored in PDF, and should therefore be analysed more thoroughly, by not only using the text information, but also the layout including the pictures. In this paper we address some of the issues involved with this approach and present an example application to help scientists search for specific pictures in a scientific PDF corpus.

1 Introduction

Most web mining applications center around HTML only, but in fact the Internet is much more diverse. For example, a short investigation of common terms like "have", "after" and "group" shows a ratio of overall hits to hits in PDF documents between 3.5 and 6.1 percent. The real number of PDFs might be even higher since, not all PDFs convert into legible text (see Chap. 3). Also, PDF documents are quite relevant, especially since research papers are often published in PDF and Postscript (see CiteSeer [Bollacker *et al.*, 1998]).

The basic problem of handling PDFs is that the text information is not freely available. While an HTML file stripped of its tags usually delivers legible text, even the simple task of text extraction from a PDF is rather complicated. Down to the basics, PDF is foremost a visual medium, describing for each glyph (= character or picture) where it should be printed on the page [Ado04, 2004]. Most PDF converters simply emulate this glyph-by-glyph positioning in ASCII, HTML or any other formats [BCL04, 2004], but information about reading order and overall semantic connection of the text is lost.

Still, since the position of all glyphs is known, the original layout can be deduced and the semantical connection can be restored. For HTML, the layout information has successfully been used to improve the classification of web pages [Kovacevic *et al.*, 2004]. We extract the same layout information for PDF documents. To prove the viability of our approach, we used the layout information to implement a special search for PDF embedded pictures in scientific publications. Since the figure captions contain much information about the figure, we implemented a search engine, similar to Google Images, to find images not in regular HTML web pages, but in PDFs. We use layout information to associate the picture with the caption. So, if the user wishes to find pictures containing UML-diagramms, he can just enter "UML" to find what he needs.

For evaluation, we took a problem from bioinformatics: quality assessment of pictures documenting experiments. The special problem posed by experimental data is that keywords searches in abstracts or even full text are ambiguous. The experimental procedures are rarely mentioned in the abstract and in the full text, experimental methods belonging to the overall topic are often referenced eg. in literature overview. By using the figure captions we can accurately find pictures for each experiment.

In bioinformatics, text mining is usually done by inspecting abstracts, since they are most easily available. Yet, more information can be obtained by searching through full text paper [Faulstich *et al.*, 2003]. In [Yeh *et al.*, 2003], it has been observed that analyzing the figure caption is of great value. Classifying the different sections of a paper to analyse them seperately like in [Shah *et al.*, 2003] has been successfully attempted, but curiously, the figure captions have not been examined.

The paper starts with a general section about the internal PDF structure. Section 3 describes how the text is put into PDF and how it can be extracted again, while maintaining the layout information. In section 4 we explain the same process for images and section 5 finally describes image and text coming together as figure and caption. Section 6 describes the overall process of the example application CaptionSearch and in Section 7 we present some results. In the last section we draw some conclusions and discuss further improvements on our search engine.

2 PDF Structure

Before we start discussing the actual text and picture extraction in the sections 4 and 5, we first have to explain some basics about PDF and how its raw data structure works like. PDF documents are all organized the same way: a header, objects and a trailer. The header contains information about the PDF version. The trailer is a bit more complicated containing structural information such as the length of the document, a reference to the root object, and more. Everything else like text information, fonts, images and structuring information is encoded into objects, an object each.

The example below shows two objects. Line numbers are included for readability. Objects are structured into an object designation (line 1 and 9), that gives the number the object can be referenced with and object data. The data is often proceeded by a dictionary (lines 2-4) that gives addi-

tional information about the data.

```
 1 1 0 obj
 2 << /Length 2 0 R
 3    /Filter /FlateDecode
 4 >>
 5 stream
 6 inserted here are 108 bytes of data
 7 endstream endobj
 8
 9 2 0 obj
10  108
11 endobj
```

In line 2 the length of the stream is determined by referencing the second object with the ID `2 0` in line 9 that contains 108 as its data (line 10). In line 3 information is given on how the following stream can be decoded. There are a number of possibilities, `FlateDecode` is the most common and identical to ZIP [Deutsch, 1996]. The semantics of the decoded stream depend on the function of the object given by the context the object is referenced in or given explicitly in the dictionary. This object could for example describe how text is to be printed on a page.

The overall structure of the document is mostly hierarchical. The root object, which is given in the trailer, references a pages container object, which references the pages and so on. Mutual information may be shared by referencing the same object several times. The objects are all readable from all objects and may not contain other objects only reference them. Hencefore, the order of the objects does not matter, as all are treated in the same way. The number of objects in the document varies depending on the way it was produced: an average 5 pages document may contain between 20 and 300 objects.

3 Text in PDF

When looking for captions, we first have to analyse all the text that is on the page. For that, we have to take the page object (like the one in the example before) and decode the text stream (the binary code in line 6), the result is a chain of commands that describes the text to be written on that page. The above example shows a short sample taken from a real PDF:

```
1 BT
2 8 0 0 8 52 757.35 Tm
3 /F2 1 Tf
4 0 -1.706 TD
5 (page 354)Tj
6 T*
7 [ (J) -27 (OURN) 27 (AL) -378 (1) ]TJ
8 ET
```

By convention, the parameters of a command are written before the command and all commands are abbreviated to two letters. All text-related commands are between a BT (line 1) and an ET (line 8) which stands for Begin Text and End Text, respectively.

There are 3 different matrices that keep track of the current writing position. The matrices are given as 6 values, like the values in line 2. The first four represent a rotation matrix. The rotation matrix can be used to write landscape text or up-side-down. Also it gives scaling parameters, which are multiplied with the actual fontsizes. The next 2 values give the position on the paper in pixels. The matrix that is set in line 2 with the Tm (set Text matrix) command is the text matrix. The other two matrices are the transformation matrix, which can be set outside the BT-ET environment to move whole text passages and the CTM (Current Transformation Matrix) that is supposed to keep track of the beginning of the line to enable carriage return.

The font size can be set either by the scaling of the matrix or directly when setting the font. The command Tf (see line 3) has two parameters, first the font object, here referenced by name, and the font size. Although the font size is set to 1pt the matrix scales it up to 8pt. In the next line line spacing is defined in text matrix coordinate system, the next line is supposed to start 1.706 times the current font size below the start of the current line. There are also possibilities to define word spacing and character spacing.

In line 5 at last text is written to the screen. The Tj command has a string parameter. Strings are denoted by the brackets around them. The text is now written, using the current matrices and the chosen font. The T* in line 6 marks a carriage return. The text matrix is set to the CTM that stored the coordinates from the beginning of the line. Then the line padding operation as defined in line 4 is executed and the CTM is set to this new coordinates.

In line 7 the second line of text is written. The TJ command, opposing to the Tj command, allows ligatures inside the string. The numbers between the strings modify the horizontal space between the letters. Contra-intuitively, positive numbers mean less space, while negative numbers mean more space. A very large negative number, like the -378 in the example can even be used to produce a space between the words without using the space literal. Since every kind of movement can be such an implicit space or carriage return, we need some algorithm to decide which one is which.

For indexing purposes it is vital to identify correct word borders, otherwise terms may be glued together or torn apart. In those cases it is not possible to find the terms anymore. Unfortunately, the spaces are sometimes not given directly, but instead the characters are just a little more apart from each other than usual. The problem sharpens as theoretically all characters can be written in any kind of order by jumping around with explicitly set coordinates.

In order to identify the spaces anyway, our first run through the text stream just extracts the characters one by one and calculates their bounding boxes. Then the difference vector $\vec{x}_{diff}$ between two adjacent characters is calculated and rotated in writing direction $\mathbf{R}$.

$$rotationmatrix\ \mathbf{R} = \begin{pmatrix} x_{old,\,right} - x_{old,\,left} & y_{old,\,right} - y_{old,\,left} \\ -y_{old,\,right} + y_{old,\,left} & x_{old,\,right} - x_{old,\,left} \end{pmatrix}$$

$$\vec{x}_{diff} = (\vec{x}_{new,\,left} - \vec{x}_{old,\,right})\frac{\mathbf{R}}{|\mathbf{R}|}$$

The resulting vector is compared to the current modified font size to determine whether this is a space, no space, carriage return or a new block of text. Next, the blocks are sorted and go through a similar procedure. This way the initial information about the order is conserved best.

The blocks bounding boxes are conserved to allow further investigation of their layout, also all changes in fonts or font size and all lines are denoted with their own bounding boxes.

Additional problems which arise are: text overlaps, when e.g. a special font is used to write the accent over à that overlaps the original a and the overall handling of

non-identifiable fonts and fonts that give wrong bounding boxes.

4 Images in PDF

Since we want to present the picture along side with the caption, we need to do two things: firstly, we have to know where the image is and secondly, we have to extract the image into a standard readable image format.

The images themselves are stored in so called XObjects. From the text stream an XObject can be called by using the command Do (execute the named XObject).

```
1  22 0 obj
2  << /Type /XObject
3  /Subtype /Image
4  /Name /Im3
5  /Width 580
6  /Height 651
7  /BitsPerComponent 8
8  /ColorSpace /DeviceGray
9  /Length 31853
10 /Filter /DCTDecode>>
11 stream ... endstream endobj
```

This example object represents an image that can be called by entering `Im3 Do` into the text stream. What happens then is that the object called Im3 is identified and executed. From the object dictionary, we can gain some information like width (line 5) and height (line 6), although this information might not be acurate. The true hight and width are calculated and give, together with the current position the bounding box of the picture.

To actually extract the picture, we need the filter (as given in line 10), in this case DCTDecode, which is the PDF name for Jpeg encoding [International Organization for Standardization, 1994]. The stream simply contains the a jpg-file that can be copied out without further modifications.

Aside from this case, there are a number of possible complications. Natively, all images are given as raster images, like a BitMap. Width and height are given to determine the dimensioning of the raster, BitsPerComponent (see line 7) give the color depth. The Colorspace (given in line 8) maps the colors to the RGB values they are painted in. For filters, there are a number to chose from, some of those are quite old and unfortunately not all have open source libraries to convert them to more publicly known formats. Also it is possible for an image to consist of drawing instructions in the PDF text stream language or they can be inserted in PostScript language.

Since a relatively high number of pictures can not be extracted easily, our next idea is to use a third-party application to extract the pictures seperately. We would then try to match the coordinates given by the program with coordinates of our own. Unfortunately, this is still in prototype phase.

5 Finding the Caption

What we have at this point is the converted text (see section 4), together with its bounding boxes and the pictures with their bounding boxes (see section 5). The next step is to find out which text belongs to which picture.

To achieve this, every text block is weighted according to 2 factors: y_{diff}, the y-proximity in pixels from the bottom line of the picture and x_{diff}, the x-proximity from the left border of the picture. The weighing of these factors can be seen from the following equation.

$$\text{weight } \omega = 10y_{diff} + x_{diff}$$

In order to find captions that are next to the figure or above it, we also introduced a semantic criterion. Figure captions do traditionally begin with ”Figure 1:” or something similar. The block that we found with the method described above, is first checked whether or not it has such a denotation. If not, we look at the other blocks in proximity and check them. If there is no ”Figure”-block to be found, we stick with the block right below. This is the case, for example, when the caption is prefaced with a filler (dots or a special symbol), or when the image is not scientifical, like for example a logo.

Finding the end of the caption is much less deterministic. Fortunately, most captions have a significant gap before the main text begins again. Also, normally captions are single-column even if the text is two-column. Although we do keep track of the font and the captions are usually written in another font, we do not use this information, since there are just too many publications, that do use the same font and font size for both purposes. Instead we keep strictly to the layout information on where an untypical large gap between line is.

6 CaptionSearch: Our example application

As a simple demonstration, how the layout information can be used, we decided to write a web based search engine for images. It is called CaptionSearch, since we are technically not searching through the images, but through the caption text beneath them.

The process consists of 6 steps from the download to the actual query execution (see Fig. 1), all of them are currently implemented in Java.

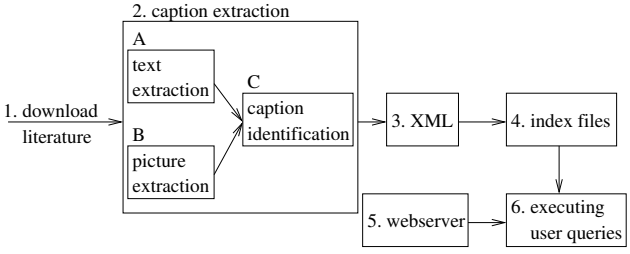

Figure 1: Overview of the CaptionSearch dataflow

Step 1 is straightforward downloading any kind of literature that might be interesting into a file pool. The technical details of step 2 were mainly already given before. In order to allow for future extensions, we split the process into three parts. Part A works like any PDF to text converter and in fact is based on the Java converter PDFBox [PDF04, 2004]. We made a few changes, like refining the coordinate calculations and changing the space detection algorithm. We also keep a lot of the meta-information that is usually lost in the process, like the bounding boxes of all the text blocks, the fonts and font sizes. That information could also be used by a different algorithm.

Part B extracts the pictures into files and adds fake picture text blocks that contain information on the position of the picture and the filename as a link. In part C we perform the algorithm outlined in section 5 to get pairs of picture blocks and text blocks for each pdf.

In step 3 these pairs are written to an xml file (see example below).

```
1 <?xml version="1.0"
              encoding="iso-8859-2"?>
2 <pdf src="10094677.pdf">
3   <img src="pics/10094677.Im4.jpg">
4     <caption>FIG. 4. DNase I
            footprint analysis of ...
5 </caption></img></pdf>
```

Each publication is represented by one file containing a PDF element (line 2-5), which may contain many image elements (like the one in line 3-5), which have the link to the picture as an attribute (line 3) and the caption as a further element (line 4-5).

In Step 4 we use the Digester package [Dig05, 2005] to extract the information from the xml files into the indexer. For each figure a virtual document is put into the indexer containing the figure caption to be indexed and the image link and pdf link as metainformation. For the indexing itself, we use the Lucene package [Hatcher and Gospodnetic, 2004], which offers fast, Java-based indexing, but also some additional functionality, like a built-in query parser and several so-called analyser that allow us to vary how exactly the captions are indexed and what defines a term.

For step 5 we set up a Tomcat webserver [Tom05, 2005], using Java servlets [Coward and Yoshida, 2003] to produce the website and to present the query results. In step 6, all queries are executed by a servlet that uses Lucene to fetch the results from the index files and builds a new web page to display the results according to the pre-selected schema.

7 Results

In the last few years, the number of biological databases has grown exponentially, from 548 in January 2004 to 719 in January 2005 [Galperin, 2005]. Yet, one of the most time-consuming tasks when setting up new databases is the annotation of literature. There already exist a number of very interesting search engines. Via PubMed [PubMed, 2004], for example, almost every abstracts ever written in life science can be searched. Unfortunately, abstracts often do not mention the exact methods that were used, so for databases that contain experimental data, like the PRODORIC database [Münch *et al.*, 2003], literature annotation becomes the proverbial search for the needle in the haystack.

The PRODORIC database contains very special data like DNA binding sites of prokaryotic transcriptional regulators. This data is generated via specific experiments like DNAse I footprints or ElectroMobility gel Shift Assays (EMSA) that are generally not mentioned in the abstracts of scientific literature. The search for general key words classifying this comprehensive field like ”gene regulation”, ”promoter” or ”binding site” results in over 150,000 hits, and even with additional refinement only 10-20% contain appropriate data. Therefore it is necessary to screen all the hits manually to obtain literature references suitable for database annotation. Of these, those are especially valuable that contain pictures of the DNAse I footprint or EMSA assay, because they represent verified information of high quality. This quality assessment can be important on further exploration of the subject.

Out of 188 papers that were known to contain information about DNA binding sites, 170 did contain extractable pictures. All in all there were 1416 pictures in the PDFs of which 586 could not be extracted, but we identified and indexed the caption anyhow. The relatively high number of pictures attributes mostly to the fact that some PDF producing programs use pictures for symbols like list bullets. A random sample of 236 captions showed that 15 % of the found captions were just random text pieces, like page numbers or single sentences, mostly due to the said symbol pictures. 6 % were wrong text that means whole paragraphs of text, just not belonging to the figure. The main reasons here were again symbol pictures, which naturally had no genuine caption and also some cases of figure captions written left or right of the picture. We had 3 cases of duplication, where one figure was internally composed of several pictures, all of which rightly claimed the same caption. We had text conversion problems with only 3 out of these 236 captions. In one no spaces were found, in the second one, some spaces were missing and in the third one some symbols like Δ converted into wrong strings. We had only one case, where the end of the caption was not found correctly. For a summary confer table 1.

No. of paper	No. of papers containing pictures		
188	170 (90%)		
No. of pictures	**No. of extractable pictures**		
1416	830 (59%)		
No. of captions in sample	**short text pieces**	**wrong text**	**wrong conversion**
236	37 (15%)	17 (7%)	3 (1%)

Table 1: Results of Evaluation

To search for DNAse I footprints, we used the keywords ”footprint”, ”footprinting” and ”DNAse”. Overall, 184 hits were scored of which 163 actually showed experimental data. As a byproduct, the thumbnails mostly sufficed to make a fast quality assessment. Another positive effect was that the data was much faster available than with the usual method of opening each PDF independently.

The search for EMSAs was a lttle bit more difficult, since there exist a wide range of naming possiblities. The most significant terms in those names were ”shift”, ”mobility”, ”EMSA” and ”EMS” to catch ”EMS assay”. We had 91 hits of which 81 were genuine.

Recall could not be tested thoroughly, due the sheer numbers of pictures and the limited time of experts, but the random sample did not include pictures that would not have been found by the keywords, which suggests a rather high recall.

We are still in an early test phase with the user acceptance. For legal reasons we cannot just put the information on the Web and see what is coming, but instead, we only have our local biologist work group as users, who supply us with the literature anyway.

8 Conclusion and Future Work

Although we have just started to explore the possibilities of layout-enhanced analysis of PDF files, our first application looks promising. To bypass the legal problems posed by the copyrights, we plan on publicizing a demo version, that does not link to the full text, but to the PubMed entry instead. We also plan to cross-check whether or not the full texts are available on the Internet and then give appropriate addresses, so users can download from the source. This demo version should then be able to give some data about user behaviour.

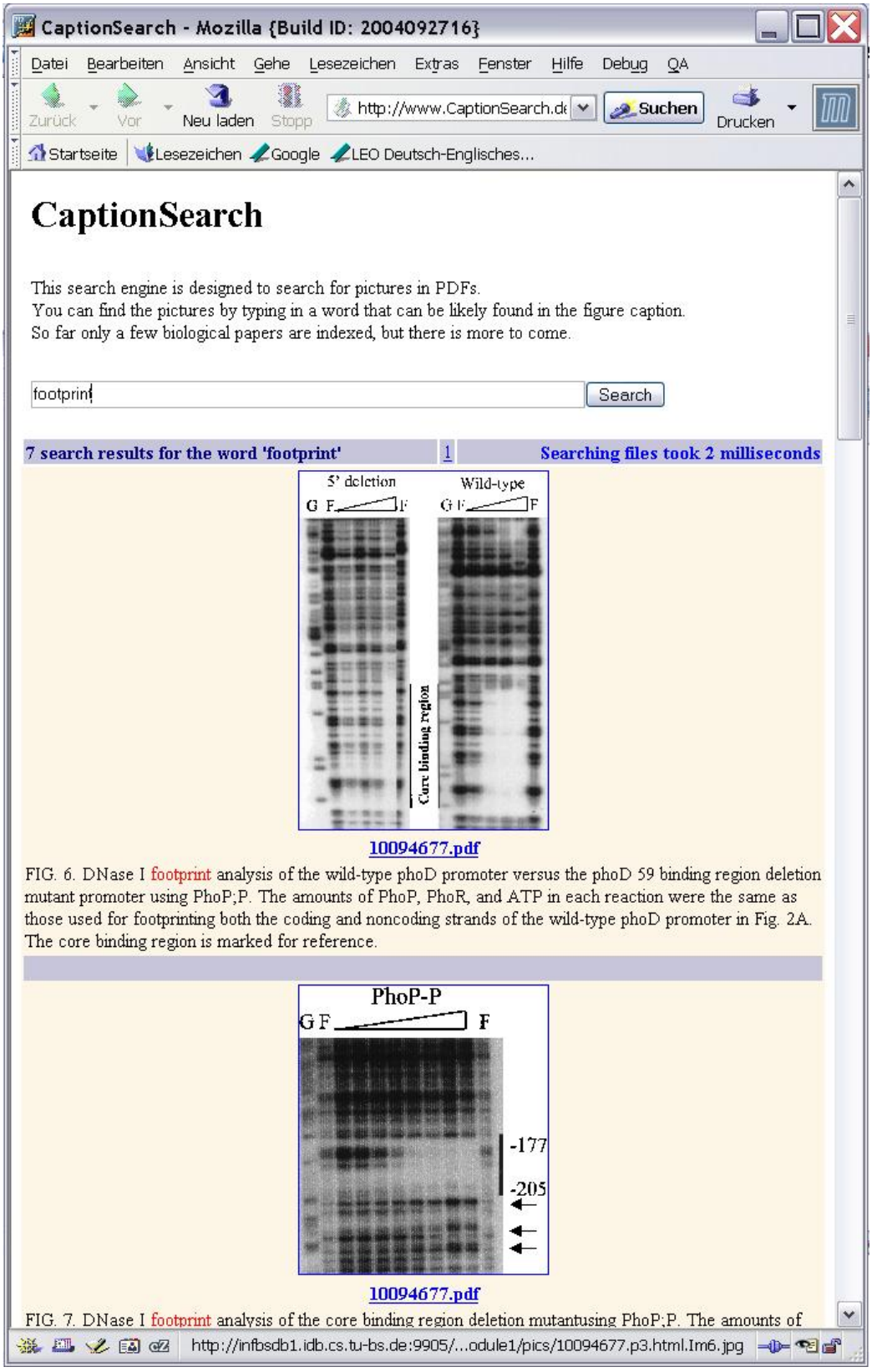

Figure 2: A screenshot from our search engine

We are in process of finding more areas of application for our search engine, as we broaden our spectrum of functionalities, especially those mentioned in the sections above, like finding the context a certain figure is mentioned in (eg. "...as you can see in Fig. 2...") to add more text to be searched through and be presented to the user and the extraction of more pictures.

The groundwork of knowing the layout of the publication can also be used for other purposes. On long term, we are working on reading order recognition to improve shallow parsing, which is still a problem in text mining applications [Schmeier *et al.*, 2003]. Also, we are investigating the feasibility of a "table search engine" similar to what has already been investigated by [Wang *et al.*,] for HTML web pages. The overall goal is to make PDF a multi-functional format that can easily be used with any kind of text mining application, just as easily as HTML or plain text.

References

[Ado04, 2004] PDF Reference Fourth Edition. Adobe Network Solutions, http://partners.adobe.com/asn/acrobat/sdk/publicdocs/PDFReference15_v6.pdf, 2004.

[BCL04, 2004] BCL Jade. http://www.bcltechnologies.com/document/products/jade/jade.htm, 2004.

[Bollacker *et al.*, 1998] Kurt Bollacker, Steve Lawrence, and C. Lee Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–123, New York, 1998. ACM Press.

[Coward and Yoshida, 2003] Danny Coward and Yutaka Yoshida. Java Servlet Specification. http://jcp.org/aboutJava/communityprocess/final/jsr154/index.html, 2003.

[Deutsch, 1996] L. Deutsch. Deflate compressed data format specification. *Request for Comments No 1951, Network Working Group*, 1996.

[Dig05, 2005] Digester. http://jakarta.apache.org/commons/digester/, 2005.

[Faulstich *et al.*, 2003] Lukas C. Faulstich, Peter F. Stadler, Caroline Thurner, and Christina Witwer. litsift: Automated text categorization in bibliographic search. In *Data Mining and Text Mining for Bioinformatics, Workshop at the ECML / PKDD 2003*, 2003.

[Galperin, 2005] Michael Y. Galperin. The Molecular Biology Database Collection: 2005 update. *Nucleic Acids Research*, 33(Database-Issue):5–24, 2005.

[Hatcher and Gospodnetic, 2004] Erik Hatcher and Otis Gospodnetic. *Lucene in Action*. Manning Publications, 2004.

[International Organization for Standardization, 1994] International Organization for Standardization. *ISO/IEC 10918-1:1994: Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines*. International Organization for Standardization, Geneva, Switzerland, 1994.

[Kovacevic *et al.*, 2004] Milos Kovacevic, Michelangelo Diligenti, Marco Gori, and Veljko Milutinovic. Visual Adjacency Multigraphs - a Novel Approach to Web Page Classification. In *Proceedings of SAWM04 workshop, ECML2004*, 2004.

[Münch *et al.*, 2003] Richard Münch, K. Hiller, H. Barg, H. Heldt, S. Linz, E. Wingender, and D. Jahn. Prodoric: prokaryotic database of gene regulation. *Nucleic Acids Research*, 31(1):266–269, 2003.

[PDF04, 2004] PDFBox. http://www.pdfbox.org/ or http://sourceforge.net/, 2004.

[PubMed, 2004] PubMed. http://www.ncbi.nlm.nih.gov/pubmed/, 2004.

[Schmeier *et al.*, 2003] S. Schmeier, J. Hakenberg, A. Kowald, E. Klipp, and U. Leser. Text mining for systems biology using statistical learning methods. In *"3. Workshop des Arbeitskreises Knowledge Discovery"*, 2003.

[Shah *et al.*, 2003] P. Shah, C. Perez-Iratxeta, P. Bork, and M. Andrade. Information extraction from full text scientific articles: Where are the keywords? *BMC Bioinformatics*, 2003.

[Tom05, 2005] Tomcat. http://jakarta.apache.org/tomcat/, 2005.

[Wang *et al.*,] Y. Wang, I.T. Phillips, and R.M. Haralick. Table detection via probability optimization. In *Proceedings of the 5th IAPR Workshop on Analysis Systems (DAS 2002)*, pages 272–283.

[Yeh *et al.*, 2003] A. Yeh, L. Hirschman, and A. Morgan. Evaluation of text data mining for database curation: lessons learned from the kdd challenge cup. *Bioinformatics*, 19(1), 2003.