Using Graph Matching Techniques to Wrap Data from PDF Documents[*]

Tamir Hassan
Database and Artificial Intelligence Group
Vienna University of Technology
Favoritenstraße 9-11, A-1040 Wien, Austria
hassan@dbai.tuwien.ac.at

Robert Baumgartner
Database and Artificial Intelligence Group
Vienna University of Technology
Favoritenstraße 9-11, A-1040 Wien, Austria
baumgart@dbai.tuwien.ac.at

ABSTRACT

Wrapping is the process of navigating a data source, semi-automatically extracting data and transforming it into a form suitable for data processing applications. There are currently a number of established products on the market for wrapping data from web pages. One such approach is *Lixto* [1], a product of research performed at our institute.

Our work is concerned with extending the wrapping functionality of Lixto to PDF documents. As the PDF format is relatively unstructured, this is a challenging task. We have developed a method to segment the page into blocks, which are represented as nodes in a relational graph. This paper describes our current research in the use of relational matching techniques on this graph to locate wrapping instances.

Categories and Subject Descriptors: I.7.5 [Document and Text Processing]: Document Capture—*document analysis*; H.3.3 [Information Systems]: Information Search and Retrieval

General Terms: Algorithms, Experimentation

Keywords: Wrapping, PDF, Document understanding, Logical structure, Graph matching

1. INTRODUCTION

The popularity of the PDF format can be attributed to its roots as a page-description language. Its main advantage is that it preserves the visual presentation of a document between screen and printer, and across different computing platforms, thus making it a useful format for the exchange of documents on the Web. However, this is also its main drawback; PDF files contain little or no explicit structuring information to help us locate wrapping instances.

In HTML, on the other hand, the structure of the code somewhat corresponds to the logical structure of the document. This has led to the development of a number of tools that use this structure to locate data items. One such product is the *Lixto Visual Wrapper*, which allows the user to interactively select data items from a visual rendition of the web page. The system then generates a *wrapping program* to automatically extract this data from similarly structured sources, or from sources whose content changes over time.

[*]This work is partly funded by the Austrian Federal Ministry for Transport, Innovation and Technology under the FIT-IT programme line as part of the NEXTWRAP project, and by the REWERSE network of excellence.

2. OUR APPROACH

Much of our previous work has been concerned with converting PDF files to HTML, which can be directly understood by the Lixto VW. There are many "off-the-shelf" packages that purport to do this, such as *Archisoft PDF2HTML*[1]. However, we found that most of these packages simply use `<div>` elements to recreate the layout of the original PDF, sidestepping the document understanding process. The resulting HTML contains no structure, and is therefore of no use to us for wrapping.

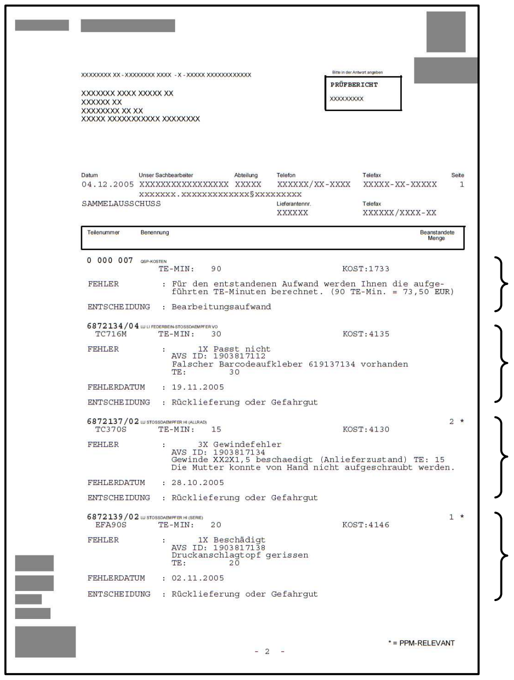
We therefore developed our own HTML conversion process, which attempts to represent the logical structure of the PDF in the resultant HTML code. This now gives us limited wrapping functionality in many documents, although this is heavily dependent on the accuracy of the document understanding process, which is inherently an imprecise task. There are many complex documents, such as the example in Fig. 1, a real use-case example of quality management data from the automotive domain.[2] Such documents can not be fully understood without additional input from the user.

We have identified three main data structures within a PDF document that could be used to locate instances of data to be wrapped:

- geometric structure (explicit in the co-ordinates)
- logical structure (inferred from the layout)
- content and content attributes (the text itself, as well as font, style, size, etc.)

Whilst our HTML conversion allows us to use the content and logical structure to identify wrapping instances, it does not give us direct access to the document's geometric structure. The graph matching method described in this paper allows us to use a combination of all three structures, essentially shifting some of the burden of the document understanding process to the user. We expect this to compensate for the inherent inaccuracies and limitations of document understanding.

[1]`http://www.archisoftint.com/logiciels/recr_us.htm`
[2]In this example, confidential data has been altered or obliterated for publication.



[redacted]

```
XXXXXXXX XX - XXXXXXXX XXXX  - X - XXXXX XXXXXXXXXXXX                Bitte in der Antwort angeben
                                                                     PRÜFBERICHT
XXXXXXX XXXX XXXXX XX                                                XXXXXXXXX
XXXXXX XX
XXXXXXXX XX XX
XXXXX XXXXXXXXXXX XXXXXXXX

Datum        Unser Sachbearbeiter  Abteilung  Telefon          Telefax           Seite
04.12.2005   XXXXXXXXXXXXXXXXX     XXXXX      XXXXXX/XX-XXXX   XXXXX-XX-XXXXX    1
             XXXXXXX.XXXXXXXXXXXXX§XXXXXXXXX
SAMMELAUSSCHUSS                               Lieferantennr.   Telefax
                                              XXXXXX           XXXXXX/XXXX-XX

Teilenummer   Benennung                                                  Beanstandete
                                                                            Menge

0 000 007  QSP-KOSTEN
             TE-MIN:   90                          KOST:1733

FEHLER       : Für den entstandenen Aufwand werden Ihnen die aufge-
               führten TE-Minuten berechnet. (90 TE-Min. = 73,50 EUR)

ENTSCHEIDUNG : Bearbeitungsaufwand

6872134/04 LU LI FEDERBEIN-STOSSDAEMPFER VO
  TC716M     TE-MIN:   30                          KOST:4135

FEHLER       :      1X Passt nicht
               AVS ID: 1903817112
               Falscher Barcodeaufkleber 619137134 vorhanden
               TE:          30

FEHLERDATUM  : 19.11.2005

ENTSCHEIDUNG : Rücklieferung oder Gefahrgut

6872137/02 LU STOSSDAEMPFER HI (ALLRAD)                                  2 *
  TC370S     TE-MIN:   15                          KOST:4130

FEHLER       :      3X Gewindefehler
               AVS ID: 1903817134
               Gewinde XX2X1,5 beschaedigt (Anlieferzustand) TE: 15
               Die Mutter konnte von Hand nicht aufgeschraubt werden.

FEHLERDATUM  : 28.10.2005

ENTSCHEIDUNG : Rücklieferung oder Gefahrgut

6872139/02 LU STOSSDAEMPFER HI (SERIE)                                   1 *
  EFA90S     TE-MIN:   20                          KOST:4146

FEHLER       :      1X Beschädigt
               AVS ID: 1903817138
               Druckanschlagtopf gerissen
               TE:          20

FEHLERDATUM  : 02.11.2005

ENTSCHEIDUNG : Rücklieferung oder Gefahrgut

                                                          * = PPM-RELEVANT
                               - 2 -
```

[redacted]

Figure 1: A sample page of data to be wrapped. Brackets indicate the individual wrapping instances.

3. IMPLEMENTATION

3.1 Obtaining PDF data

We use the PDFBox[3] library to parse the raw PDF file and return the visual PDF data as a set of text and graphic objects. PDFBox returns these text blocks in the same way as they have been written to the PDF file, i.e. as a set of individual blocks, usually with no more than 2–3 characters per block. The first step is to merge these blocks into complete lines of text, and a set of heuristics achieves this.

3.2 Page segmentation

Our next step is to merge the line objects into blocks that can be said to correspond to one logical entity in the document's structure. These blocks correspond to paragraphs, headings, single table cells and other miscellaneous items of text (such as captions). We believe this provides us with sufficient granularity for logical selection of wrapping instances.

3.3 Graph representation

We represent these blocks as nodes in an attributed relational graph. Initially, the graph is built with just the adjacency relation being present, which links all blocks to their neighbours. Our document understanding process then produces other geometric relations, such as alignment; and logical relations, such as reading order and superiority (which, for example, relates a title to its body text). An example of this graph on a single wrapping instance is shown in Fig. 2.

As each of the nodes has a set of co-ordinates, this representation maps easily onto the visual domain, where the user can interactively select an example wrapping instance, and its corresponding sub-graph is found automatically.

[3]PDFBox, `http://www.pdfbox.org`

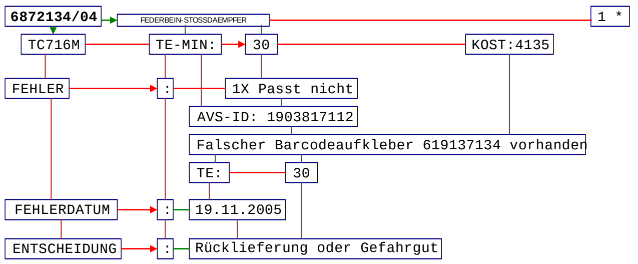


Figure 2: Sub-graph for one record (wrapping instance) from Fig. 1. Note that edges with arrows represent *superior*-to-*inferior* relationships.

3.4 Similarity measures

Once the user has selected the example instance, it must be matched to other *similar* occurrences on the page, and possibly from other pages in the document. There are many algorithms in the literature for graph matching. However, in our case, it is obvious that an algorithm that finds exact matches is of little use to us. Instead, we require a significant and specific *error tolerance* to match objects that are somehow logically or visually similar.

The familiar notion of *edit cost* can be used to define the similarity of two sub-graphs. Allowed operations would include not just additions and deletions of single nodes or edges, but additions and deletions of complete rows of elements. For example, a certain paragraph may be one line longer or a certain table might have an extra row added. Yet, the logical structure with relation to *shape* would remain the same. Thus we are finding wrapping instances using both logical and visual similarity.

Furthermore, this method could be further extended to discriminate between headings and data. The logical relations present in the graph enable us to determine, with some degree of certainty, which blocks contain headings and which blocks contain just "data" (plain body text). Any "edits" that affect heading elements would therefore correspond to a change in logical structure, and this would carry a higher edit cost than the equivalent operation to only body text.

3.5 The matching process

We require an error-tolerant algorithm for relational subgraph matching. The most popular algorithms, such as [3], are tree-based. We are currently developing such an algorithm that uses a *branch-and-bound* strategy. The benefit of this approach is its adaptability to our definition of graph similarity. Although the complexity is exponential in the worst case, the use of application-specific heuristics to prune the search can make the problem tractable. If this turns out not to be the case, there are many other approaches, such as [2], that proclaim to reduce the complexity even further.

4. REFERENCES

[1] R. Baumgartner, S. Flesca, and G. Gottlob. Visual web information extraction with lixto. In *The VLDB Journal*, pages 119–128, 2001.

[2] W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Tran. on Pattern Anal. and Mach. Intel.*, 17(8):749–764, Aug. 1995.

[3] J. Llados, E. Marti, and J. J. Villanueva. Symbol recognition by error-tolerant subgraph matching between region adjacency graphs. *IEEE Tran. on Pattern Anal. and Mach. Intel.*, 23(10):1137–1143, Oct. 2001.