2009 10th International Conference on Document Analysis and Recognition

User-Guided Wrapping of PDF Documents Using Graph Matching Techniques

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

Abstract

There are a number of established products on the market for wrapping—semi-automatic navigation and extraction of data—from web pages. These solutions make use of the inherent structure of HTML to locate instances of data to be wrapped. As PDF documents do not have such a structure, wrapping PDF documents has long been recognized as a challenging problem.

We have developed a novel system for wrapping PDF documents, which is currently at a prototype stage. A PDF document is represented as an attributed relational graph, in which nodes represent physical items on the page and edges represent spatial and logical relationships. A wrapper is defined as a subgraph of the document with additional conditions, and can quickly and intuitively be created by a non-expert using the GUI. An algorithm based on subgraph isomorphism is then used to find the data instances and extract the required data. Experiments show that our approach achieves good results with good execution time.

1. Introduction

In recent years, PDF has become the de-facto standard for the distribution of print-oriented documents on the Web. Its success can be attributed to its roots as a page-description language, and therefore its ability to preserve the visual presentation of a document between screen and printer, and across different computing platforms. However, this is also its main drawback; PDF files contain little or no explicit structuring information, which makes machine processing and automatic data extraction a difficult task.

In the information extraction field, *wrapping* is the process of navigating a data source, semi-automatically extracting data and transforming it into a form suitable for data processing applications. A number of products exist on the market to wrap data from web pages, such as *Lixto*[1], a product of research at our institute. These approaches are successful because the HTML format has an inherent structure. However, as an increasing number of business-critical documents, such as financial reports, price lists and technical specifications, are being distributed in PDF format, there is an increasing need to also wrap data from PDF documents.

In this paper, we present an interactive graph-based approach for wrapping from PDF, which is now at a prototype stage[2]. The idea of using graph matching to locate data instances on PDF files was originally proposed by us in [7].

2. Related work

2.1 Wrapping

The success of the *Lixto Visual Wrapper* provided the motivation to work on extending its capability to PDF files. The Lixto VW is an interactive tool which allows a non-expert user to create wrapper programs in a predominantly visual and interactive fashion by clicking on example instances on a visual rendition of the web page. In the background, the software locates the data using the HTML parse tree. The user can then fine-tune the selected data by adding or removing logical conditions. The system then generates a wrapper program to automatically extract this data from similarly structured sources, or from sources whose content changes over time.

2.2 Document analysis and understanding

HTML wrappers work successfully because the structure of the HTML source code more or less represents the under-

This work has been supported by the Austrian Federal Ministry for Transport, Innovation and Technology under the FIT-IT contract FFG 815128.

[1] Lixto, `http://www.lixto.com`

[2] A test version of the current prototype can be downloaded from: `http://www.tamirhassan.com/graphwrap.html`

978-0-7695-3725-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDAR.2009.238

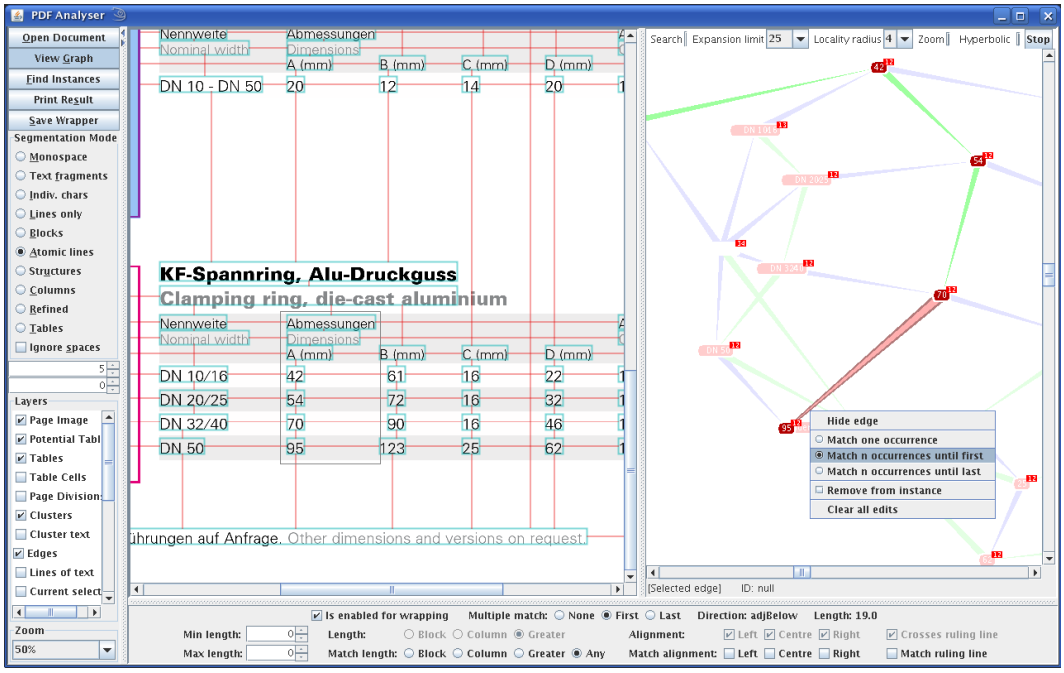


Figure 1. Designing a wrapper using the graphical user interface

lying *logical* structure of the document. In PDF, there is no such explicit structure.[3] There is much literature on *document understanding*, which aims to rediscover this logical information from e.g. scanned documents [1, 2]. There is comparatively little literature which deals specifically with PDF files; examples are [4, 6, 3]. Some of these techniques are used in the generation of our graph representation from the PDF source.

2.3 Wrapping from PDF

Flesca et al. [5] have also addressed the issue of wrapping PDF files, although they use a somewhat more complicated representation of a wrapper; their approach is based on hierarchically organized groups defined by fuzzy-logic constraints. As our graph structure is very similar to the physical layout of the document, we believe our approach to offer increased usability and faster definition and maintenance of wrappers compared to their method, at the possible expense of expressiveness.

We have also worked on employing document understanding methods in the conversion of PDF files into HTML format, so that they can be wrapped using conventional methods. In particular, we have developed algorithms for table recognition [8]. However, these methods have their limitations as they are reliant on an almost perfect understanding of the page. Additionally, wrapping from irregular tabular structures, such as the example used in [7], is not possible at all. Therefore, we have developed a graph-based approach, which offers increased robustness.

3. From PDF to graph

We use the PDFBox[4] library to read in the PDF file and return the visual data as a set of text and graphic objects. PDFBox returns these blocks in the same way as they have been written to the PDF file; text is usually returned in blocks of 2–3 characters in the same order as it was written to the file. The first step is to merge these blocks into complete lines of text, and a set of heuristics achieves this.

The next stage is segmentation; merging of these line objects into blocks that can be said to correspond to one logical entity in the document's structure. These blocks correspond to paragraphs, single table cells and other miscellaneous items of text (such as captions). This provides us with sufficient granularity for wrapping in most cases. For certain applications, however, the user may prefer to generate the graph on the line level, in which case this stage is skipped.

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

[3] PDF 1.4 and above support the addition of logical structure tags in the document, but the vast majority of documents encountered on the Web contain no or insufficient logical structure information for wrapping.

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

neighbours (4-neighbourhood). Our document understanding process then annotates these edges with other geometric properties, 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 such a graph can be seen in the screenshot of the GUI (Fig. 1). More information on the segmentation and graph generation process is given in Section 4 of [8].

4. Wrapper specification

The wrapper specification procedure is carried out using the graphical user interface (Fig. 1), which displays the a bitmap rendition of the document and its corresponding interactive graph side-by-side. Additionally, nodes and edges can be overlaid on the bitmap image using methods from the *XMIllum* framework[5]. The user begins the process by selecting an example data record on the bitmap rendition of the page. The corresponding sub-graph is found and is displayed to the user using methods based on the *TouchGraph*[6] library, which allow interactive navigation and manipulation of the graph.

For example, the user can remove nodes from the example instance and add new nodes corresponding to other items on the page that were not selected before. Most importantly, the user can set conditions for each node and edge, a non-exhaustive list of which is provided below:

Node conditions	Edge conditions
String, substring, regexp	Min & max length
Min & max text length	Alignment left, right, centre
Font size, font name, bold, italic	Crosses ruling line
Ruling line above, below, left, right	Superior/inferior
	Before/after reading order

Additionally, each node can be set as an *output node*, i.e. the data it contains will be extracted and will appear in the resulting XML output. For this purpose, a user-defined XML tag name can also be set.

Essentially, this annotated subgraph defines one level of the wrapper. The system will then look for all possible subgraphs in the document which match the structure of this graph and where the conditions hold. It is worth noting that the wrapper specifies only the items which must appear in the document; any additional items will be ignored by the wrapper, even if they occur within the bounding box of the found instances. Therefore, a common tactic in wrapper design is to define the first and last elements of a data record, allowing any number of items to occur in between.

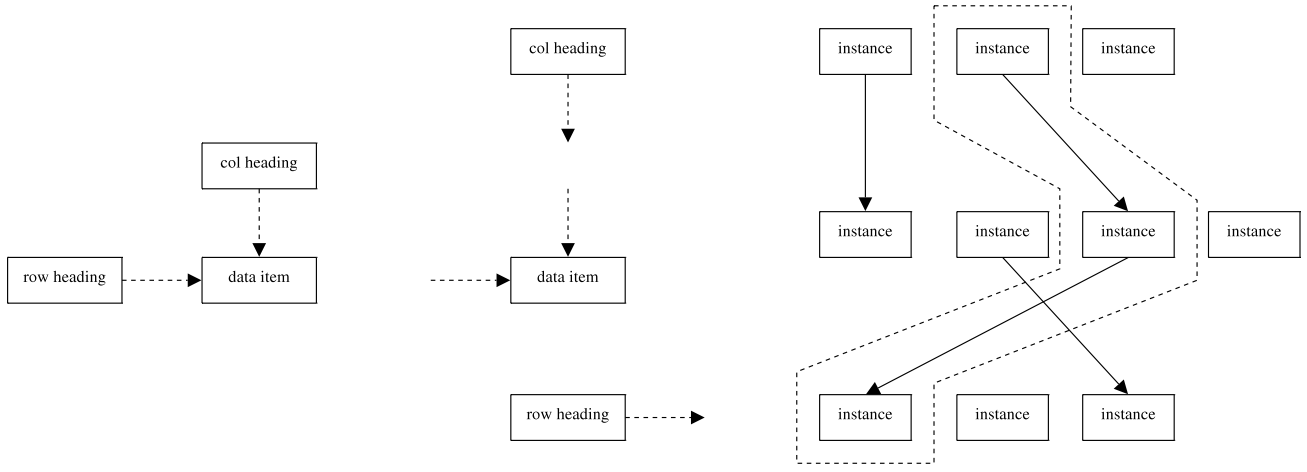
[5]XMIllum, `http://xmillum.sourceforge.net/`
[6]TouchGraph, `http://www.touchgraph.com`

Multiple-match edges. A common occurrence in PDF documents is for data records to be presented with known first and last items but varying numbers of items, e.g. repeating lines or columns, in between. As standard graph matching will only match a fixed structure, we have introduced a new feature, *multiple-match edges*, to make wrapping possible on such documents. Any edge can be selected as a multiple-match edge, and the user has the option to match either the first or the last occurrences of its neighbouring nodes. However, there is a further condition: each multiple match edge in the graph must split the graph into two sub-graphs when it is removed. This means that, for example, a multiple-match edge may not run parallel to another edge. As it does not really make sense to impose a fixed structure at the point where the number of edges is unknown, we do not believe this to be a significant restriction on the expressiveness of the wrapper. The reason for this condition is that, in the background, the wrapper graph is split into sub-graphs, which are matched separately and are then joined together to form the result. This procedure is described in Section 5.

Hierarchical wrapping. In order to wrap nested structures, wrappers can be nested inside each other. There are currently three different modes of nesting: in the `subgraph` mode, which is the most straightforward, only items in the resulting subgraphs are passed as input to the child wrappers, and any unmatched items that lie within the bounding box are not passed on. As most wrappers are defined by just matching a few items in the record, a far more useful mode of nesting is the `area-based` mode. Here, all items whose centres fall within the bounding box of the found instance are also passed on to the child wrappers. The final mode is `whole page`, in which child wrappers are matched on the entire page, but only those results are returned whose output nodes intersect the area of the parent result. In this way, it is possible to wrap a table by first matching the rows and then wrapping the individual columns, as the column headings are still accessible from the child wrapper.

5. The matching process

In order to perform the graph matching, we use a method based on Ullmann's algorithm for subgraph isomorphism [9], which consists of an enumeration of all possible combinations of nodes and a refinement procedure to prune unfruitful search paths early on in the process. As we are working on directed graphs, we use the directed version of the algorithm given in [9]. A major benefit of the algorithm is that it allows us to define, in the start matrix, which pairs of nodes, based on the conditions set in the wrapper, could *a priori* be matched together. As we also need to define conditions for edges as well as nodes, we represent each edge



a) Simple wrapper with multiple-match edges, which are marked with dotted lines

b) The wrapper is split into sub-graphs along the multiple-match edges and each of these sub-graphs is matched independently

c) A sub-instance association graph is formed, in which all pairs of instances are joined with an edge if it is possible to get from one to the other by extending a multiple-match edge. An (undirected) matching step is then used to extract the final solution

Figure 2. Processing a wrapper with multiple-match edges

as a special type of node, and join each of these special edge-nodes with vertices to both neighbouring nodes.

Where multiple-match edges are present, we first split the wrapper graph using these edges into its sub-graphs. A simple example is given in Fig. 2(a), with the resulting sub-graphs in Fig. 2(b), where multiple-match edges are represented with dashed arrows. These sub-graphs are then matched independently, the results of which we term *sub-instances*. After these sub-instances have been found, they then need to be integrated to find the final result. Each sub-instance is compared to each other, and the system tries to find a path from each multiple-match edge in one sub-instance to a multiple-match edge in the other. If a path is found, a check is made to see if the sub-instance reached is the first or last along the path as defined in the wrapper.

A *sub-instance association graph* is then used to represent all these sub-instances, which are joined by a vertex if they have valid paths along multiple-match edges connecting each other (see Fig. 2(c)). Similarly, a *sub-graph association graph* is built from the original wrapper graph, in which each sub-graph is replaced by a single node. A further graph matching step is performed, in which instances of the sub-graph association graph are searched for in the sub-instance association graph. These instances correspond to instances of the final, integrated result.

6. Experimental results

In order to provide an indication of the effectiveness of our wrapping approach, we created three wrappers on three different data sets, which represent possible use-case scenarios of our system.

The first wrapper, `classifieds`, was designed to extract the textual content of all normal (unboxed) classified advertisements from the *Coastal Point* newspaper[7] and was run on six issues. The second wrapper, `catalogue`, was designed to test our hierarchical approach for wrapping tables, and was run on a 72-page catalogue of technical components[8]. Here, the wrapper was designed to extract all items which have a nominal width, order number, price and measurement A; as well as measurements B, C and D if these were given. Finally, we decided to test our system on a much larger data set, a collection of back issues from the *Travel Monthly* magazine[9], which contains 190 PDF files with 445 pages in total. Each destination contains a "Fact file" box with contact information, which we extracted with the wrapper `travel`.

We measured the precision and recall values, as well as execution time. Results are shown in Fig. 5. We found the attained precision and recall to be very dependent on wrapper design, and in some cases we were able to make improvements to the wrapper. For example, some of the fact files from previous years were headed in mixed (title) case instead of upper case. After replacing the respective string condition with a regular expression condition, these documents were also wrapped successfully. The remain-

[7] Coastal Point classifieds, `http://www.thecoastalpoint.com/files/classifieds/classifieds.pdf`

[8] Pink GmbH Vakuumtechnik component catalogue, `http://www.pink.de/pdf/katalog.pdf`

[9] Travel Monthly archives, `http://www.travelmonthly.com.au`

Wrapper	No. docs	No. pages	No. data items	Precision	Recall	Proc. time/page	Match time/page
classifieds	6	24	1095	0.9890	0.9845	7.64 s	3.28 s
catalogue	1	72	4835	1.0000	0.9986	8.48 s	6.13 s
travel	190	445	1385	0.9978	0.9711	4.38 s	0.68 s

Figure 3. Experimental results

ing missed records in the travel dataset were due to irregularly placed logo images (e.g. between the heading and the data), for which there is not yet a suitable representation in our system. Similarly, matching errors in the classifieds dataset were due to inconsistencies in the layout (e.g. alignment) between records.

A further interesting topic is the execution time and complexity of the graph matching algorithm. Unsurprisingly, our experimental results show that more complex documents (i.e. documents with more items on the page) take significantly longer to process. In order to obtain an indication of the effectiveness of the graph matching method itself, we chose to separately measure the amount of time taken to perform just the matching, as well as the total processing time (which includes other tasks, such as reading in the PDF, segmentation and extraction of objects).

Whereas the travel wrapper performed very quickly indeed, the other two wrappers were significantly slower. We attributed this to the fact that these wrappers employed multiple-match edges, which necessitated several graph matching steps. What surprised us, however, was that certain pages of the travel dataset, which contained no data to be extracted, took very long to process (typically over 20 s). On closer examination, these pages contained densely-packed classified information, much like in the classifieds dataset.

In our first experiments, we had found the Ullmann algorithm to be very fast indeed, as long as the wrapper was specified correctly, as the refinement procedure prunes most unfruitful search paths immediately. If the wrapper was specified in such a way to return hundreds of possibly overlapping results, then processing would take significantly longer. As such, we formed the impression that the execution time of the graph matching algorithm was largely determined by the number of results it finds.

We found out that the reason for the slow processing time of the few densely-packed pages in the travel dataset, which did not contain any information to be extracted, was the multi-step approach to graph matching when multiple-match edges were used. Although no results were returned at the end of the matching procedure, the wrapper was defined in such a way that each sub-graph that was matched against the page would return almost every object on the page as an intermediate result. All of these intermediate results were then compared against each other to find the final result, of which there was not a single matching pair.

7. Conclusion

In this paper we have presented a novel approach for extracting data from PDF, which now makes it possible to wrap several classes of documents which could not efficiently be wrapped before. The GUI enables fast creation and maintenance of wrappers even by non-expert users. Experimental results show very good accuracy and good execution performance of the current prototype system.

We believe the next step to be to improve execution time for wrappers with multiple-match edges in large documents. One further advantage of the Ullmann algorithm [9] is that, at each execution step, it is possible to see which nodes in the wrapper have been assigned to which nodes in the document. Thereby, by integrating multiple-match edges into the subgraph isomorphism detection algorithm itself, we can re-work the refinement procedure to also allow for multiple-match edges, pruning unfruitful search paths as early as possible in the matching process. Thus the necessity of several graph matching operations with potentially hundreds of intermediate results would be avoided, leading to significant performance increases.

References

[1] M. Aiello, C. Monz, L. Todoran, and M. Worring. Document understanding for a broad class of documents. *Intl. Journal on Document Analysis and Recognition*, 5(1):1–16, 2002.

[2] O. Altamura, F. Esposito, and D. Malerba. Transforming paper documents into xml format with wisdom++. *Intl. J. of Document Analysis and Recognition*, 4(1):2–17, 8 2001.

[3] A. Anjewierden. Aidas: incremental logical structure discovery in pdf documents. In *ICDAR 2001, Proc.*, 2001.

[4] H. Chao and J. Fan. Layout and content extraction for pdf documents. *Doc. Anal. Sys. VI*, 3163/2004:213–224, 2004.

[5] S. Flesca, S. Garruzzo, E. Masciari, and A. Tagarelli. Wrapping pdf documents exploiting uncertain knowledge. In *CAiSE 2006, Proceedings*, pages 175–189, 2006.

[6] K. Hadjar, M. Rigamonti, D. Lalanne, and R. Ingold. Xed: a new tool for extracting hidden structures from electronic documents. In *DIAL, 2004. Proceedings*, 2004.

[7] T. Hassan and R. Baumgartner. Using graph matching techniques to wrap data from pdf documents. In *15th Intl. Conf. on WWW (Poster track), Proceedings*, pages 901–902, 2006.

[8] T. Hassan and R. Baumgartner. Table recognition and understanding from pdf files. In *ICDAR 2007, Proc.*, 2007.

[9] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, January 1976.