

# Intelligent wrapping from PDF documents with Lixto

Tamir Hassan  
22 April 2005

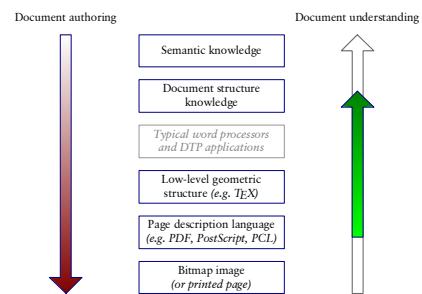
## Introduction

- This talk will present my work on extending Lixto to work with PDF documents
- Structure of the talk:
  - Intro: document authoring process; wrapping
  - Summary of the PDF format
  - Techniques for document understanding
  - Implementation and prototyping
  - Demonstration

## Basics of a document

- A document can be analysed on two levels:
  - geometric, physical or layout structure (text, images, boxes, etc.)
  - logical structure (articles, headings, paragraphs, ...)
- In HTML the logical structure (mostly) corresponds to the structure of the code
- In PDF the logical structure is not (usually) explicitly encoded; this must be rediscovered through document understanding

## Document hierarchy

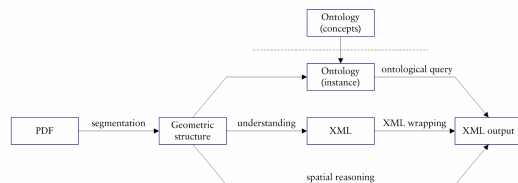


## Wrapping and Lixto

- The Lixto Visual Wrapper is a **wrapper generator** for web pages
- It makes use of the **tree structure** inherent in HTML to locate data
- As it is fully interactive, the underlying structure is hidden from the user
- Ways to extend Lixto to PDF:
  - convert the content of the PDF to a similarly structured format (e.g. XML)

## Wrapping from PDF

- use an **ontological model** to wrap directly from the PDF
- use **spatial reasoning** techniques to wrap directly from the physical structure of the PDF



## The PDF format

## PDF Basics

- Successor to the PostScript printer language
- Why is it so popular?
  - **ease of creation:** any document can be “printed” to a PDF file
  - the PDF is guaranteed to look (almost) identical to the original
- Various libraries to help us access low-level data
  - PDFBox, JPedal, Acrobat API, etc.

## What does PDF give us?

- Pages
- **Text objects:** typically contain no more than a few characters each
  - co-ordinates, font, style, font-size, ...
  - can be merged into lines or paragraphs
  - PDF provides very rich font information
- **Lines and boxes** – drawn by the relevant PostScript commands `moveto`, `lineto`, etc
- **Images** – we require only the co-ordinates

## What about logical structure?

- **Tags** – unfortunately, seldom used
- For PDFs created from software applications:
  - **artefacts of the PDF creation** (“printing”) **process can often give us clues as to the former (logical) structure of the document**
  - reading order
  - spaces between words

## Progress so far

- Found suitable tools for prototyping
- Implemented a simple algorithm for line-finding and segmenting a page
- This will be demonstrated at the end of the talk

## Document understanding

## Overview

- Traditionally seen as the next step after OCR
- Most previous work on segmentation etc. based on images obtained from scans
- Some work uses ASCII text as the source (particularly in table understanding)
- Little work directly on PDF
  - so many more things PDF can tell us...

## Overall process

- The process of document understanding can be split up into the following stages:
  - Merging or line-finding
  - Segmentation of the page into homogeneous blocks
  - Classifying the blocks (text, image, table...)
  - Finding relationships between these blocks
  - Understanding of paragraph structure
  - Understanding of tables

## Merging/line finding

- Most text in a PDF is written in blocks of no more than a few characters at a time
- Text on the same line will (usually) have exactly the same **baseline**
  - a small tolerance can be included for OCR
- Insert a space if the horizontal space sufficiently large between two neighbouring text blocks
- Font metric information can improve result

## Page segmentation

- Several approaches in the literature:
  - convert to bitmap image, series of smoothing algorithms and connected components analysis
  - “bin-sorting” technique using the start, middle and end horiz. co-ordinates of each line
  - “clustering” of PDF text objects if they are sufficiently close to each other
- Extending these approaches to make use of other structuring elements (lines, boxes, etc.)
- A basic method will be demonstrated at the end of this talk

## Classifying the blocks and finding relationships

- Representing physical objects and their relations:
  - **feature vector** (aspect ratio, area ratio, font-size ratio, font style, content size, number of lines) [Aiello et al]
  - **geometric relations** between the blocks – Thick Boundary Rectangle Relations (TBRR) [Aiello et al]
  - Relative **status** of each block (i.e. heading > body)
- Obtaining a high-level understanding of the document:
  - Rules/heuristics [Klink, Kieninger]
  - Abstractions followed by grammars [Anjewierden, Kabel]
  - Statistical pattern recognition (tree classifier) [Aiello et al]
    - only four logical labels: Title, Caption, Body, Page Number
  - Machine learning approaches tend to be doc specific

## Paragraph structure

- Paragraphs need to be merged into one **flow** of text
  - need to differentiate between **inline** and **external** objects (e.g. titles/headlines, images and captions)
  - this should also work across pages
  - NLP might be necessary to fully detect reading order
- Indentation grammars [Rus, Summers]
  - this method attempts to detect structure without any prior knowledge (we will have prior knowledge)
- Attempt to detect a multi-level hierarchical structure (for complex documents)

## Tables

## Tables

- Table detection
  - detecting and isolating tables on a page
- Table understanding
  - partitioning the 2D representation into cells
  - deriving a logical structure from the 2D representation, which may have more than two dimensions
- PDF tables vs ASCII tables

## Detecting tables

- Algorithms for detecting tables can often provide information for cell partitioning
- For PDF the most appropriate methods are:
  - looking for a “grid-like” structure, i.e.
  - examining ruling lines
  - examining whitespace density graphs (horizontal and vertical projection)

## Table understanding

- There are many techniques for doing this
- Most literature deals with ASCII data
  - although these methods can scale up to PDF, they ignore the extra information available to us
- Two parts to this task
  - partitioning the cells
  - obtaining a higher-level logical understanding
- Aside: table models

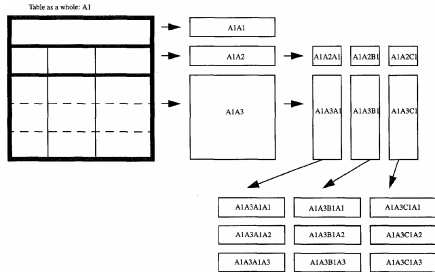
## Table models: 2D

- Grid or box structure
  - closest to physical representation
  - like a spreadsheet
- Hierarchical structure [Green, Krishnamoorthy]
  - easier to locate data in rows than columns (or opposite way round)
  - makes sense when we already know the logical structure (tuples/attributes)
- Directed acyclic graph [Lopresti et al]
  - essentially two hierarchies in one
  - rows can be located as easily as columns

- Grid structure example

	A	B	C	D	E	F
1						
2						
3						
4						
5						
6						

• Hierarchical structure example [Green, Krish.]



• DAG example [Lopresti et al]

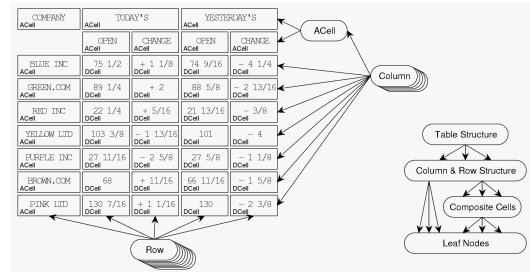


Table models: Wang

- A table is represented on the page in two dimensions
- The underlying logical structure of a table can have more than two dimensions
- Question: As we're only extracting data, is this representation useful for us or an unnecessary complication?
  - simple case: dealing with a table and its transpose

• Wang model example [Wang]

Year	Term	Mark					Grade
		Assignments			Examinations		
		Ass1	Ass2	Ass3	Midterm	Final	
1991	Winter	85	80	75	60	75	75
	Spring	80	65	75	60	70	70
	Fall	80	85	75	55	80	75
1992	Winter	85	80	70	70	75	75
	Spring	80	80	70	70	75	75
	Fall	75	70	65	60	80	70

$\delta(\{Year.1991, Term.Winter, Mark.Assignments.Ass1\}) = 85;$   
 $\delta(\{Year.1991, Term.Winter, Mark.Assignments.Ass2\}) = 80;$   
 $\delta(\{Year.1991, Term.Winter, Mark.Assignments.Ass3\}) = 75;...$

Partitioning a table into cells

- Essentially, two types of approaches:
  - segmentation (top-down)
  - clustering (bottom-up)
- Segmentation
  - Horizontal projection profiles to find candidate column separators
    - e.g. Whitespace Density Graph [Rus, Summers]
  - For PDF we can also use vertical projection profiles to find row separators; there is likely to be an additional space or ruling to differentiate lines from rows
    - also pattern-matching [Rus, Summers], and looking for partial lines [Lopresti et al]

Clustering approaches

- Kieninger's algorithm [Kieninger]
  - Clustering algorithm worked by use of a seed which was the same width as the text block, and three times its height (extending above and below by the same amount)
  - Note that the height of the text block is less than the total line spacing



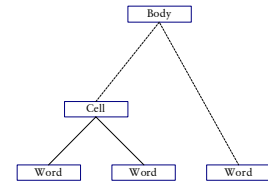
### Kieninger clustering algorithm (contd.)



- This is followed by a series of heuristics to correct common errors by the process (e.g. spanning headings)
- works where cell alignment not perfect
  - more of a problem in ASCII than in PDF

### Hierarchical clustering

- The algorithm of [Lopresti et al]
  - leaf level clusters contain different words
  - these are merged upwards starting with the two nodes with the smallest inter-cluster distance
- Heuristics to determine where to **prune** the tree
- After pruning, leaf nodes correspond to columns



### Problem tables

- “Folded” tables to conserve space
  - i.e. column headings are repeated halfway through
- Ambiguous headings [example from Hurst]
  - usually more of a problem with ASCII than PDF

	YEAR ENDED DECEMBER 31,			
	1991	1992	1993	1994
(UNAUDITED)				
(DOLLARS IN THOUSANDS)				
STATEMENT OF OPERATIONS				
Investment income.....	\$ 8,806	\$ 7,953	\$ 8,333	\$ 8,820
Interest expense.....	4,139	3,509	3,661	4,756

### Problem tables (2)

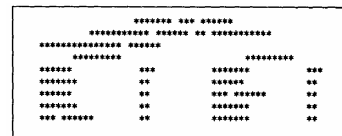
- Equations with matrices and other mathematical objects erroneously detected as tables (or are they tables, perhaps?) [Lopresti et al]

$$\alpha = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \\ \alpha_6 \end{bmatrix} = \begin{bmatrix} 0.26 \\ 2.37 \\ 1.84 \\ 1.31 \\ 0.17 \\ 0.05 \end{bmatrix}$$

### ASCII tables in PDF?

- ASCII tables are sometimes included as monospaced text within a PDF
- Because of the different layout conventions that are used, they should be treated differently
- The first step of the process is to determine whether the table is ASCII or not
  - monospaced text (i.e. x and y positions always multiples of a certain factor)
  - no ruling lines or other non-text elements

### The Acid Test



- What are the captions here? [Pyreddy, Croft]

## Implementation and prototyping

## Implementation

- Prototyping
  - PDFBox and XMillum
- Problems with PDFBox
  - rasterizer is at a very early stage
- Alternatives
  - Adobe API
  - PDFBox (server) + Adobe API (client)
  - PDFBox + Ghostscript (via command line)
  - XPDF is GPL

## Demonstration

## Demonstration

- This demonstration makes use of the following tools:
  - PDFBox for parsing the low-level data of the PDF file
  - XMillum to display the results of the output
- You will see the results of a simple merging algorithm, and a simple segmentation algorithm, as explained earlier in the talk.

## Improvements?

- Neighbourhood function:
  - use modal size instead of mean size
  - look at line spacing
  - make use of ruling lines and graphical objects
- Line-finding
  - make use of font metric information
  - calculate average character spacing
  - word separator if space between blocks  $\gg$  character spacing
- Complexity
  - find lines first
  - segment with naïve algorithm
  - then split up or merge later with second pass that looks at ruling lines etc.

Thank you