TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ANSSI NURMINEN

ALGORITHMIC EXTRACTION OF DATA IN TABLES IN PDF DOCUMENTS

Master's Thesis

Examiners:
Prof. Tapio Elomaa,
MSc. Teemu Heinimäki
Examiners and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on 9 January 2013.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Degree Programme in Information Technology
NURMINEN, ANSSI: Algorithmic Extraction of Data in Tables in PDF Documents
Master of Science Thesis: 64 pages, 4 appendices (8 pages)
April 2013
Majoring in: Embedded systems (software emphasis)
Examiners: Prof. Tapio Elomaa, MSc. Teemu Heinimäki
Keywords: PDF, portable document format, Qt, data extraction, data mining, Adobe, tables, table row, table column, discovering tables, table discovery, table recognition, HTML, XML, table structure recognition, table structure definition, table detection, table extraction, convert PDF to HTML, convert PDF to XML, layout analysis, document understanding, big data, information extraction, information extraction system, Poppler

Tables are an intuitive and universally used way of presenting large sets of experimental results and research findings, and as such, they are the majority source of significant data in scientific publications. As no universal standardization exists for the format of the reported data and the table layouts, two highly flexible algorithms are created to (*i*) detect tables within documents and to (*ii*) recognize table column and row structures. These algorithms enable completely automated extraction of tabular data from *PDF* documents.

PDF was chosen as the preferred target format for data extraction because of its popularity and the availability of research publications as natively digital *PDF* documents, almost without exceptions. The extracted data is made available in *HTML* and *XML* formats. These two formats were chosen because of their flexibility and ease of use for further processing.

The software application that was created as a part of this thesis work enables future research to take full advantage of existing research and results, by enabling gathering of large volumes of data from various sources for a more profound statistical analysis.

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
NURMINEN, ANSSI: Algorithmic Extraction of Data in Tables in PDF Documents
Diplomityö, 64 sivua, 8 liitesivua
Huhtikuu 2013
Pääaine: Sulautetut järjestelmät (ohjelmistopainotuksella)
Tarkastajat: professori Tapio Elomaa, dipl.ins. Teemu Heinimäki
Avainsanat: PDF, taulukko, taulukot, data, talteenotto, big data, HTML, XML, Qt, Poppler, big data

Lähes poikkeuksetta kaikki nykyisin tehtävä tutkimustyö julkaistaan verkossa, ja yhä enenevässä määrin ”open access”-journaaleissa. Saatavilla olevan tutkimusdatan räjähdysmäinen kasvu on johtanut monilla aloilla tilanteeseen, jossa sen käsittely manuaalisesti on erittäin työlästä, ellei jopa mahdotonta. Jotta tulevaisuuden tutkimustyö voisi hyödyllisellä tavalla rakentua jo olemassa olevan tiedon päälle, tarvitaan siis automaattisia menetelmiä datan keräämiseen ja käsittelyyn.

Taulukot ovat intuitiivinen ja selkeä tapa esitellä pientä suurempia määriä tilastoja, tutkimustuloksia ja muita löydöksia. Suuri osa tieteellisten julkaisujen tärkeistä tuloksista julkaistaankin juuri taulukkomuodossa. Mitään standardisointia taulukoillle eri julkaisijoiden välillä ei kuitenkaan ole, ja taulukot esiintyvät julkaisuissa hyvinkin monimuotoisina, hyvin vaihtelevilla rakenteilla ja ykstyiskohdilla.

Näitä ongelmia varten tämän diplomityön yhteydessä on kehitetty kaksi täysin uutta, joustavaa algoritmia taulukkomuotoisen datan talteenottamiseen ja prosessoimiseen tietokoneiden paremmin ymmärtämään muotoon (HTML, XML). Ensimmäisen algoritmin tehtävä on taulukoiden paikantaminen PDF (Adoben Portable Document Format) dokumenttien sivuilta. Toinen algoritmi jäsentelee taulukoiden tietoalkiot data- ja otsikkoriveihin, ja määrittelee niiden rivi- ja sarakerakenteen. Nämä kehitetyt algoritmit mahdollistavat täysin automaattisen taulukoiden talteenoton ja jatkokäsittelyn PDF-dokumenteista. PDF-dokumentit valittiin kohdemediaksi, niiden yleisyyden ja tieteellisten julkaisujen saatavuuden perusteella, natiivisti digitaalisina PDF-dokumentteina.

Tämä opinnäytetyö ja sen myötä kehitetyt algoritmit ovat etupäässä suunnattu bioinformatiikan ja bioteknologian käyttötarkoituksiin, toimimaan osana ”big data”-tyylistä tutmustyötä, jossa suuresta määrästä olemassa olevaa tutkimusdataa tiivistetään muuten piiloon jääviä korrelaatioita ja muita olennaisia havaintoa. Mikään ei kuitenkaan rajoita algoritmien käyttöä juuri tällaisiin tarkoituksiin.

PREFACE

This thesis work was completed in a time period of 5 months including the development and implementation of the software algorithms. All the algorithms used and described in this thesis work were developed by the author.

I would like to thank Prof. Tapio Elomaa for supervising this thesis work, and Teemu Heinimäki for his careful proofreading and excellent suggestions on my writing. I would also like to thank Tamir Hassan for his helpful correspondence for making the performance evaluation metrics of the algorithms more standardized. Most of all, I would like to thank my wonderful girlfriend Henna for her continuing patience and support throughout the whole process.

Last revised: 2013-04-16.

—Anssi Nurminen, anssi.nurminen@iki.fi

TABLE OF CONTENTS

TERMS AND DEFINITIONS

Words and abbreviations appearing with an *italicized* font within this document are explained in the following table.

API	“Application Programming Interface is a protocol intended to be used as an interface by software components to communicate with each other.”, Wikipedia.
ASCII	“The American Standard Code for Information Interchange is a character-encoding scheme originally based on the English alphabet.”, Wikipedia.
C++	“The C++ programming language is a statically typed, free-form, multi-paradigm, compiled, general-purpose programming language.”, Wikipedia.
CPU	“Central Processing Unit is the hardware within a computer that carries out the instructions of a computer program by performing the basic arithmetical, logical, and input/output operations of the system.”, Wikipedia.
Flash	“Adobe Flash (formerly called "Macromedia Flash") is a multimedia and software platform used for authoring of vector graphics, animation and games which can be viewed, played and executed in Adobe Flash Player.”, Wikipedia.
GPL	“The GNU General Public License is the most widely used free software license, which guarantees end users (individuals, organizations, companies) the freedoms to use, study, share (copy), and modify the software. Software that ensures that these rights are retained is called free software.”, Wikipedia.
GUI	“A Graphical User Interface is a type of user interface that allows users to interact with electronic devices using images rather than text commands.”, Wikipedia.
HTML	“HyperText Markup Language is the main markup language for creating web pages and other information that can be displayed in a web browser.”, Wikipedia.
OCR	“Optical character recognition is the mechanical or electronic conversion of scanned images of handwritten, typewritten or printed text into machine-

	encoded text.”, Wikipedia.
PDF	Portable Document Format (Adobe) is a digital file format with small file size, system independent representation, portability and printability as its key features.
Proto-link	Proto-links describe the adjacency relationships of cells in a table.
Python	“Python is a programming language that lets you work more quickly and integrate your systems more effectively.”, source: http://www.python.org/
Qt	“Qt is a full development framework with tools designed to streamline the creation of applications and user interfaces for desktop, embedded and mobile platforms.”, source: http://qt.digia.com/Product/
Unicode	“Unicode is a computing industry standard for the consistent encoding, representation and handling of text expressed in most of the world's writing systems.”, Wikipedia.
XML	“Extensible Markup Language is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable”, Wikipedia.
XSL	“Extensible Stylesheet Language is used to refer to a family of languages used to transform and render *XML* documents.”, Wikipedia.

1 INTRODUCTION

Most, if not all contemporary scientific publishing is made available, and distributed online. The ubiquitousness of the Internet and the increasing popularity of open access publishing are making an increasing amount of publications easily accessible to a global audience. Our ever expanding collective knowledge and the rapidly increasing amounts of available data in just about any field of study are making manual gathering and processing of such reported data an inefficient and laborious task; if not altogether impossible. Therefore, in order for future research to be able to build adequately on top of existing results and data, as well as being able to interpret the existing data correctly and more profoundly, a system for automatic extraction and processing of data is needed.

Regardless of the scientific discipline, results of studies and experiments are often reported in a tabular format. Tables are an intuitive and efficient way of reporting large sets of data. However, while tabular representation of data is universally used in all types of publications, no standardization of any kind exists in the way data is presented between publications of different publishers or organizations, or in some cases, even within the publications of a single publisher. A software tool for extracting such data will therefore need to be highly adaptable to be able to correctly extract data from an eclectic corpora of different types of tables.

The main focus of this work is to develop a practical software tool for easy and automatic extraction of relevant data from large volumes of *PDF* (Portable Document Format, by Adobe) documents. *PDF* was chosen as the preferred target format for data extraction, because of its popularity and the availability of publications as natively digital *PDF* documents, almost without exceptions. In addition, the release of the patent rights on the *PDF* standard in 2008, has made the *PDF* format even more supported and widely accessible.

The biomedical domain currently offers the most exciting aspects for “big data” research. The current and massive influx of genetic data has created a demand for systems that are able to combine and process the available information from a variety of sources, into a more meaningful ensemble. This thesis work is intended to be a part of a larger system, capable of processing large volumes of published data, but in no way limited to such use.

However, for large enough number of documents, such a method can never achieve perfect results. Therefore, it would be paramount to push publishers to make it a mandatory requirement for their publications' authors to include their relevant experimental data and research findings in a more computer and algorithm friendly way. This could easily be accomplished by including hidden metadata objects in the *PDF* documents

(e.g. in *XML*, “Extensible Markup Language”, format). Such features are already well supported by the *PDF* standard.

The technical background for this thesis is presented in Chapter 2. The problems that need to be solved to create an automated table data extraction system are presented in Chapter 3, while Chapter 4 addresses the methods that have been used to solve them. Chapter 5 focuses on evaluating the performance of the used methods. Chapter 6 takes a look at existing similar systems and compares them to the algorithms developed as a part of this thesis. The final chapter (Chapter 7) discusses the overall achieved results.

2 BACKGROUND

Tabular data extraction falls under a data processing category known as Information Extraction (IE). “Information Extraction is a form of natural language processing in which certain types of information must be recognized and extracted from text” [1]. An Information Extraction System (IES), such as the one proposed in this thesis work, analyzes the input text in order to extract relevant portions. IESs do not attempt to understand the meaning of the text, but they only analyze portions of input text that contain relevant information [2].

There are roughly two main approaches to building IE systems: a rule-based approach, and an active learning approach. Both have significant advantages and disadvantages. This thesis employs a rule-based approach, with some learning-based parameter adjustments. The rule-based approach of the algorithm is rooted in the rules of written language, in all the so-called western languages, such as left-to-right and up-down direction of writing.

In addition to the rules of the written languages, the only usable universal guideline is, that all tables are meant to be read by humans. Considering this rudimentary principle, two general rules for the arrangement of the elements that a table contains can be established:

1. Alignment of rows, and
2. Alignment of columns.

There always exists a visual way for determining which elements within a table are associated with each other. Without any association between the elements of a table, it would simply be a list. Whether the elements are separated from each other by separator lines, drawn rectangles, or just by spacing, there always exists a visual pattern to the placement of the table elements, because otherwise, it would be impossible even for humans to interpret the presented data.

There are generally two different types of *PDF* documents: natively digital documents and scanned paper documents. The natively digital documents differ from the scanned paper documents in a few important ways. Scanned documents have their contents drawn as images, while natively digital documents specify regions and text that is drawn using fonts. To be able to process scanned documents in a useful way, the image would first need to be processed using an optical character recognition (*OCR*) algorithm to discover the written text in the image. Other issues with scanned documents include poor quality images and tilted page orientation, which is the result when a scanned paper is not placed completely straight on the scanning bed. These issues make processing

scanned documents a very different task from processing natively digital documents, and therefore, processing scanned *PDF* documents is left outside the scope of this thesis work.

2.1 Table anatomy

One of the more well-known conceptual models of a table has been proposed by Wang [3], and later extended by Hurst [4]. Wang defines the table being divided into four main regions: (*i*) *the stub* that contains the row- and subheaders; (*ii*) *the boxhead* that contains the column headers (excluding the stub head); (*iii*) *the stub head* that contains the header for the stub, and (*iv*) *the body* that contains the actual data of the table. In this thesis, Wang's definitions have been adapted slightly, so that the stub head is considered being included in the stub.

It is worth mentioning that, of course, not every table has all of the four regions present in it. For example, for a good percentage of tables, the stub and row headers do not exist at all, and the column headers are not “boxed”. In addition to these definitions, this thesis work uses the table definitions: header, column, row, title, caption, superheader, nested header, subheader, block, cell and element. Figure 1 illustrates the definitions.

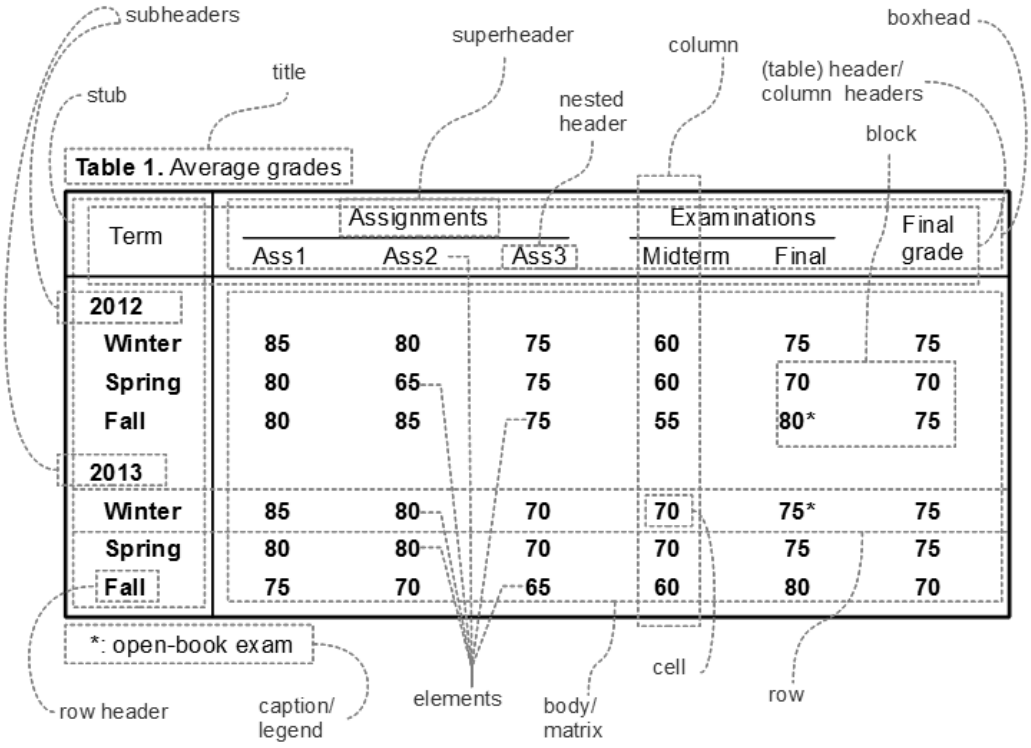


Figure 1: Table anatomy, terms and definitions of table elements.

An element is defined as a single word or a number on a *PDF* page. The difference between an element and a cell in a table, is that a cell can contain multiple elements. This is the case in many tables where multiple words (elements) form a sentence inside the table body or header, and the whole sentence is assigned with the same column and row indices, becoming a single table cell. A block consists of multiple cells.

While the title and the caption may not be considered to be a part of the actual table, they are included in the definitions and the extraction process, because they often contain important information about the contents of the table, and therefore should be extracted and associated with the table, especially when further functional or semantical processing of the data is required.

A superheader is a column header that is associated with multiple columns and has other column headers under it (typically nested headers, each associated with a single column). A subheader is a cell in a table that usually exists on a row that contains no table body elements, and it is associated with all the stub elements below it, or until the next subheader below is found. Only in tables where the stub contains more than one column, the subheaders may exist on body data containing rows.

The left-to-right style of writing used by all western languages, is guarantee enough that the stub can be trusted to be located at the left end of the table, in Column 1. There are of course exceptions, but the percentage of such tables, where the stub columns are not at the left end of the table is negligible. Slightly more commonly, a duplicate of the stub can exist in the middle of, or at the rightmost column of a table.

2.2 Portable Document Format (PDF)

The portable document format (*PDF*) is a file format developed by Adobe Systems in the early 1990s. The main purpose, or idea of the *PDF* file format is the ability to represent printable documents in a manner that is independent of software, hardware, and operating systems [5]. In other words, a *PDF* document should look, read and print exactly the same no matter what system it is used with. The *PDF* specification was made available free of charge in 1993, but it remained a proprietary format, until it was officially released as an open standard in 2008 (ISO 32000-1:2008) [6][7], when Abode published a Public Patent License to ISO 32000-1. This license grants royalty-free rights for all patents owned by Adobe that are necessary to make, use, sell and distribute *PDF* compliant implementations [8]. In addition to these features, *PDF*s offer a good compression ratio, reducing file size and making the format ideal for online distribution. The Adobe *PDF* logo in shown in Figure 2.

Figure 2: The Adobe PDF logo is recognizable to many because of the popularity of the PDF file format.[*]

Because of these qualifications and attributes, the *PDF* format has emerged as one of, if not the most widely used "digital paper" of today, and as such, a preferred method of online distribution of scientific publications for many publishers.

The basic types of content in a *PDF* are: text, vector graphics and raster graphics. The format, however, supports a wide variety of other types of content, such as interactive forms, audio, video, 3D artwork, and even *Flash* applications (PDF-1.7). For the purposes of table data extraction, only the text content and visual clues such as separator lines are relevant.

It is important to mention that the *PDF* document format also supports metadata streams by using the Extensible Metadata Platform (XMP) [9] to add *XML* standards-based extensible metadata to *PDF* documents. Using embedded metadata, it would be possible to include all reported data in a publication in a way that is easily sorted, categorized and understood by computers. If such a practice would be enforced or even encouraged by publishers, extracting and mining relevant data from large sets of publications would become much easier and less error prone.

2.3 Poppler PDF rendering library

The Poppler PDF rendering library [10] is a xpdf-3.0 [11] based *C++* open source project (under GNU General Public License), that is freely available online. The Poppler library provides a convenient way of reading and handling the *PDF* format and files, giving easy access through an *API* to the text in the *PDF* document, as well as rendered (image format) versions of its individual pages.

Poppler is still a young and ongoing project, with the latest release being version 0.22 (released on 2013-02-10). Current relevant limitations of the library *API* include: no proper font information is available (font family, size), no text formatting information is available (**bolded**, *italic*) and some problems with character encodings (some special characters have wrong numerical code values).

---

* Image source: Wikimedia commons.

The software tool that is developed as a part of this thesis project does not handle the *PDF* files, and the *PDF* standard directly, but all reading and interpreting of the contents of a *PDF* is done through the Poppler *PDF* rendering library.

2.4 Project goals

The goal of this thesis project is to create a useful software tool that is freely available online to anyone interested in data extraction from *PDF* documents. While the focus of the project is in scientific publishing, the usability of the created software application is in no way limited to any one type of publications. The developed software tool is intended for use with natively digital *PDF* documents, written in western style, left-to right languages.

The application will be created in a way that allows standalone usage of the program directly with *PDF* documents, as well as using it as a part of other software tools and projects through an *API*. The output of the created software application is designed so that it allows further automatic processing of the extracted data, and conversions between different digital formats.

All the source code of this thesis project will be made available online and it will be released under the GNU General Public License (*GPL),* version 3*.

*http://www.gnu.org/licenses/gpl.html

3 DATA EXTRACTION

The data extraction process begins with defining the information the algorithms are working with. The Poppler *PDF* rendering library (Chapter 2.3) handles the *PDF* file format, and the data extraction algorithms of this thesis process only information interpreted and channeled through the Poppler library.

The available data is not as detailed as one would imagine. For instance, no information about the used font or the style of the font (such as **bold** or *italics*) is available. Neither is any information about the super- or subscript status of a word.

The text in the *PDF* document is received for each individual page in the form of rectangular areas containing a single word of text called an (text) element. Each rectangular element area (text box) is defined by coordinates, giving it a precise, unambiguous location and size (width and height) on the page. The sides of the text boxes are always aligned with the sides of the pages, i.e. there are no tilted or skewed rectangles. Figure 3 illustrates how the textual elements are available through the Poppler *API*.

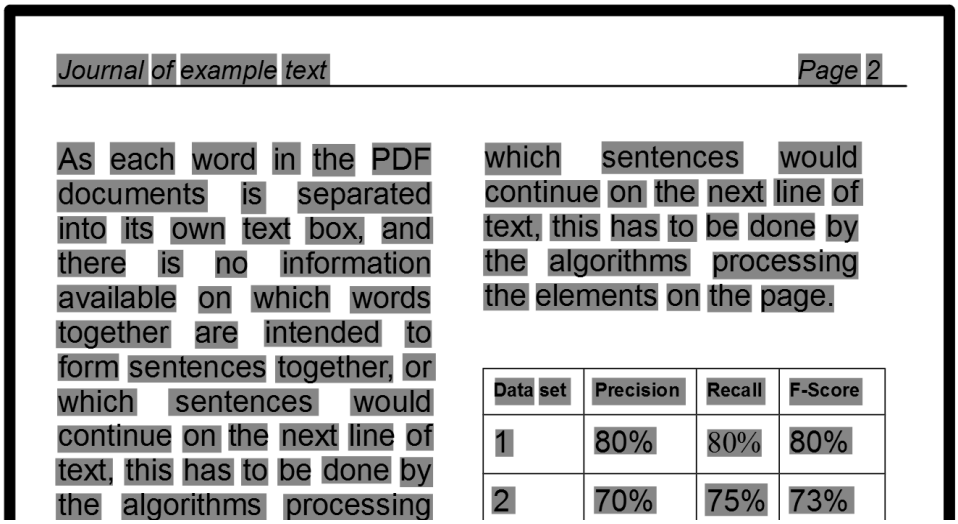


Figure 3: The textual data in PDF documents is available as text boxes (grey areas) with defined text content, location and size through the Poppler PDF rendering library. The text boxes have no defined associations; further processing is needed to establish which text boxes form sentences or tables together.

Each word in a *PDF* document is separated into a text box, and there exists no information on which words together are intended to form sentences or table rows and columns together, or which sentences would continue on the next line of text. Associating words and forming sentences (and table cells) has to be done by the data extraction al-

gorithms, by examining and processing the elements on the page. Also, if a word is hyphenated at the end of a line, and continues on the next, it is separated into two completely unassociated text boxes. Each element also contains the bounding rectangles of each individual letter (or number, or other character) in the word it contains.

The coordinates of the text boxes are available as *points* (abbreviated as **pt**). A *point* is a typographical unit that equals 1/72 of an inch. This is due to the nature of the *PDF* standard, designed as a printable "digital paper". The actual, physical distances when printed out in physical paper format are not really relevant for the algorithmic data extraction process (more relevant are the relative distances between the text boxes). However, the physical distance can help in determining the limits in which text boxes, containing words, would be able to form readable sentences. The following equation shows how *points* can be converted into other units of length.

$$1\ point = \frac{1}{72} inches = \frac{25.4}{72} mm = 0.3527\, mm.$$

Any other kind of information on a page, such as separator lines, images or any other type of more complex embedded data that the *PDF* standard supports, is not directly available through the Poppler API. Other methods must be used for taking such information into consideration.

In addition to providing the textual data, the Poppler library can render the individual pages of *PDF* documents into images. These images can be used for detecting rectangularly outlined sections on a page, and either vertical or horizontal separator lines, which often exist when information is displayed in a table format. These outlines are often essential visual aids in being able to interpret (even for humans) the row and column associations of elements in a table (see Chapter 3.1.3). This is especially true for tables that do not align the contents of their cells vertically, so that the text elements of the table do not appear on the same imaginary horizontal baseline between columns. Any more complex shapes, tilted or handwritten lines are very rarely used in other purposes than visual gimmickry (which is not often present in scientific publications), and good results can be achieved without considering such shapes at all.

3.1 Defining the problems that need to be solved

The extraction of data from tables in *PDF* documents with only limited information is not a trivial task. The fully automatic data extraction process can be subdivided into smaller individual tasks, that must each be completed successfully for correct extraction results:

1. Reading the contents of a *PDF* file.
2. Rotating the table to upright orientation.
3. Discovering separator lines and grids.
4. Discovering table areas in the document.
5. Defining the row and column structure of a table.
6. Defining the header rows of a table.
7. Formatting and outputting table data.

Failing at any of these defined sub-tasks of the extraction process will result in less-than-desirable results. Defining the table stub is not mentioned in this list, because it is not critical by itself for correct data extraction. The two main structural features of the stub: (*i*) defining subheader rows and (*ii*) defining split data rows, are included in sub-tasks 6 and 5 respectively.

The following chapters provide a more detailed look of each sub-task of the full extraction process. In addition to these defined problems of the extraction process, some consideration needs to be given to possible issues with character encoding (Chapter 3.1.8).

3.1.1 Reading the contents of a PDF file

As all the handling of the *PDF* file format is done by the Poppler *PDF* rendering library, this sub-task is a problem that has already been solved, and does not need to be addressed further in this thesis. The Poppler library makes the contents of a *PDF* file available as text boxes and rendered images, as described in the beginning of Chapter 3.

3.1.2 Rotating the table to upright orientation

Because the standard paper formats (such as A4, Letter, …) that are used in publishing are not square in their dimensions, often the best fit, especially for large full-page sized tables, is achieved by rotating the page 90 degrees; from portrait to landscape orientation. In order for algorithmic table detection and table cell associations to work properly, it is paramount that the table can be processed in upright orientation.

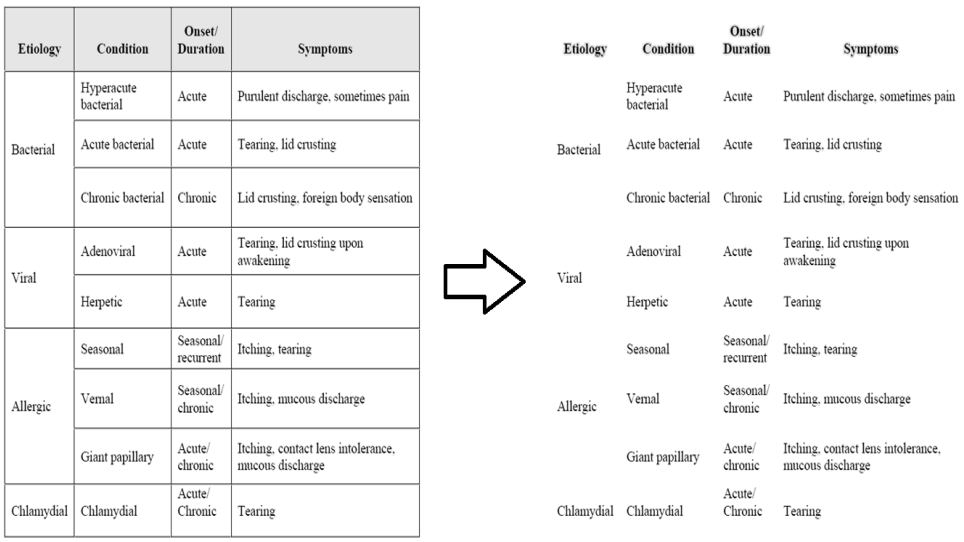
The rules of written western languages and perhaps certain ubiquitous conventions assert a few principles that most tables automatically follow. Such principles that seem intuitive and self-explanatory include:

- The header of the table is most likely to be at the top of the table.
- The stub column is most likely to be at the leftmost column or columns of the table.

While these principles are not in any way mandatory rules of creating tables, a simple observation of tables from a variety of different sources quickly establishes that these principles are inherited by most tables by a very large margin. Furthermore, these principles make it clear that the directions (up-down, left-right) within the table must be known, in order to interpret its header, row, and column structure correctly. The Poppler library (Chapter 2.3) makes no claims about what the intended upright orientation of a page is, it simply serves the page as the author of the document has created it.

3.1.3 Discovering separator lines and grids

Tables that use a definitive grid structure, often do not align their contents into vertically aligned rows, but instead rely on the alignment of the visible grid structure. Without the information about the lines and rectangular areas on a *PDF* page, defining the cells correctly would be in most cases a nearly impossible task (as illustrated in Figure 4). Therefore, a method of taking into account the drawn separator lines and rectangular areas that function as visual aids for the reader is needed.



Etiology	Condition	Onset/ Duration	Symptoms
Bacterial	Hyperacute bacterial	Acute	Purulent discharge, sometimes pain
	Acute bacterial	Acute	Tearing, lid crusting
	Chronic bacterial	Chronic	Lid crusting, foreign body sensation
Viral	Adenoviral	Acute	Tearing, lid crusting upon awakening
	Herpetic	Acute	Tearing
Allergic	Seasonal	Seasonal/ recurrent	Itching, tearing
	Vernal	Seasonal/ chronic	Itching, mucous discharge
	Giant papillary	Acute/ chronic	Itching, contact lens intolerance, mucous discharge
Chlamydial	Chlamydial	Acute/ Chronic	Tearing

Etiology	Condition	Onset/ Duration	Symptoms
	Hyperacute bacterial	Acute	Purulent discharge, sometimes pain
Bacterial	Acute bacterial	Acute	Tearing, lid crusting
	Chronic bacterial	Chronic	Lid crusting, foreign body sensation
	Adenoviral	Acute	Tearing, lid crusting upon awakening
Viral	Herpetic	Acute	Tearing
	Seasonal	Seasonal/ recurrent	Itching, tearing
Allergic	Vernal	Seasonal/ chronic	Itching, mucous discharge
	Giant papillary	Acute/ chronic	Itching, contact lens intolerance, mucous discharge
Chlamydial	Chlamydial	Acute/ Chronic	Tearing

Figure 4: Determining row and cell associations in a table can be difficult without grid structure information.[*]

There are only two types of separator lines in natively digital *PDF* documents that need to be considered: straight vertical lines and straight horizontal lines. Because of their rare, marginal existence, diagonal or curved lines do not need to be taken into account.

---

* *Source: Optometric clinical practice guideline care of the patient with conjunctivitis, Reference guide for clinicians, 2nd edition. American Optometric Association, 2002.*

3.1.4 Discovering table areas in the document

PDF documents contain a lot of different elements other than text in a table format. Therefore, a method of separating the non-table text elements of a page from the elements of a table is crucial. This also invites the question: what qualifies as a table? For the purposes of this thesis work, it is not the actual definition of the term “table” that needs to be concerned with, but rather with what kind of data should be extracted as a table.

As the focus of this thesis work is on data extraction and collection for database storage and further processing, the absolute minimum requirements for document page text elements to qualify as a table can be set to a minimum size of 2 columns and 2 rows. Any table that is below these limitations can be disregarded, for simply not being able to have enough data. These limits cannot be straightforwardly applied to recognized table grid structures, as many types of grids can contain subdivisions of rows and column within the grid cells, as described in more detail in Chapter 3.2. There should be no upper limit to the size of a table, and tables can be split onto multiple pages.

The table areas should also be inclusive of the table title and caption texts, because these table elements often contain important information about the table body elements (actual table data), that is necessary for further functional and semantical processing of the data.

There are four types of errors in table detection that should be recognized and taken into consideration:

1. Table has an incomplete set of elements assigned to it (completeness).
2. Table has non-table elements assigned to it (purity).
3. Elements of a single table are assigned to multiple tables (split errors).
4. Elements from multiple tables are assigned to a single table (merge errors).

3.1.5 Defining row and column structure for the table

After some, or all of the elements on a page in a *PDF* document have been assigned belonging to a table, their row and column associations within the table need to be defined in order to determine the cell structure of the table.

For tables with a fully defined grid structure (see Figure 5 below), this is a relatively straightforward task. The cells of the grid determine the row and column structure of the table autonomously, and no further processing in this regard is needed.

Descrizione	Saldo iniz.	Incrementi	Decrementi	Saldo finale
Ratei	1.669	0	1.269	400
RATEI ATTIVI	1.669	0	1.269	400
Risconti	26.676	0	26.079	597
RISC. ATTIVI	26.676	0	26.079	597
Ratei	49.734	0	14.467	35.267
RATEI PASSIVI	49.734	0	14.467	35.267

Figure 5: Example of a table with a fully gridded structure. Source: PDF-TREX data set [12].

Other types of gridded tables include table types that: only have their outermost outline defined, only have their header separated from the body, have their body elements separated or any mixture of these. All grids that do not define the table row and column structure completely are defined as tables with a supportive grid (Figure 6).

DESCRIZIONE	Consistenza 1/01/2004	Acquisizioni	Spostamenti dalla voce nella voce	Alienazioni	Arrotondamenti	Ammortamento 2004	Consistenza 31/12/2004
Costi di ricerca, sviluppo e pubblicità	38.948	30.240	-	-	-	16.203	52.985
Concessioni, licenze e marchi	5.751	12.595	-	-	-	4.572	13.774
Altre immobilizzazioni immateriali	78.452	4.164	-	-	-	16.245	66.370
Immobilizzazioni in corso	0	46.137				0	46.137
TOTALI	123.151	93.136	-	-	-	37.021	179.266

Figure 6: Example of a table with a supportive grid structure. Source: PDF-TREX data set [12].

At the other end of the table grid structure spectrum lie the tables that have absolutely no defined grid structure at all (Figure 7). All these different types of tables are commonly used, and need to be considered in creating an algorithm that extracts their data.

PROSPETTO VARIAZIONI IMMOBILIZZAZIONI MATERIALI

DESCRIZIONE	S.DO INIZ.	VARIAZIONI	AMM.TI	S.DO FIN.
ATTREZZ. COMMERCIALI	15.219	7.048	2.633	19.634
IMPIANTI	919	0	77	842
ALTRI BENI	4.041	-4.041	0	0
TOTALE IMMOB. MAT.	20.179	3.007	2.710	20.476

Figure 7: Example of a table completely without any grid structure. Source: PDF-TREX data set [12].

For tables without a fully defined grid structure, the algorithm needs to be able to determine which rows can me merged together. For example, when a cell in a table contains so much text it has been split and continued on the next row (line), these rows should be merged together so that the whole text is assigned to a single table cell.

3.1.6 Defining the header rows of the table

For correct data association, an essential step of the data extraction process is finding the header of the table. Without making a distinction between a header cell and a table body data cell, it is impossible to further process the data in a table into more meaningful categories.

The textual elements in a table header can often span multiple columns and rows, be nested under other headers and in general have a lot more varied structure than the body of the table. Therefore, the table header elements need to be identified to process them differently from the table body data elements.

Second, in order of importance, is defining the subheaders rows of the table. A subheader can be defined as a non-data row within the table body that is associated with all the data rows below it (see Chapter 2.1, “Table anatomy”), or until another subheader is encountered (moving down in the table). If the subheaders are misinterpreted as table data, the association mapping between the table cells will be incomplete.

3.1.7 Formatting and outputting table data

The processed tables that become the output of the developed software tool should be formatted in such a way that they can be easily imported into other software applications for further processing. Primary candidates for further processing of the extracted table data could include, for example, databases, spreadsheet applications and web-

pages, among others. The output should be designed to accommodate all of these different further processing methods.

3.1.8 Character encoding

Some special *Unicode* characters embedded in a variety of *PDF* documents have proven problematic with the Poppler *PDF* rendering library. Part of the problem is also due to the misuse of certain look-a-likes of more commonly used characters, such as the hyphen-minus (“-”) character (*ASCII* hexadecimal code 2D). The full *Unicode* character set contains more than 12 characters that look deceptively similar to the common hyphen, as illustrated in Table 8.

Hexadecimal code	Character name	View
002D	HYPHEN-MINUS	-
058A	ARMENIAN HYPHEN	֊
05BE	HEBREW PUNCTUATION MAQAF	־
2010	HYPHEN	‐
2011	NON-BREAKING HYPHEN	‑
2012	FIGURE DASH	‒
2013	EN DASH	–
2014	EM DASH	—
2015	HORIZONTAL BAR	―
2212	MINUS-SIGN	−
FE63	SMALL HYPHEN-MINUS	﹣
FF0D	FULLWIDTH HYPHEN-MINUS	－

Table 8: A non-exhaustive table of Unicode hyphen look-a-likes.

Publication authors, whether they feel that the regular hyphen is too short or not visible enough, sometimes choose to use any of these look-a-likes in the place of regular hyphens. For human readers, this is not a problem at all, but for machines and algorithms, all these “impostor” characters, that look almost or exactly alike on print, are as different as A and B. This can affect the performance of an algorithm, for example when trying to decide whether two rows should be combined in a table. If a line of text ends

in a hyphen, it is likely to continue on the next line and these two lines can be safely combined into a single table cell.

Another example of how the character encoding problem becomes evident, and could have an effect on further processing of the table data, is when considering a data column with Boolean yes/ no, on/ off values. Now, if instead "0" and "1" the author of the document has decided to use "+" and "-" to describe the two values, but instead of "-" (*ASCII* hexadecimal code 2D) she has used a "figure dash" (*Unicode* hexadecimal code *2012*, see Table 8), the interpretation of the data fields becomes much harder for a machine that is only looking at the character numerical codes. This problem is not only common, but involves a lot of different characters (such as "+", "<", ">", "*", "‘") for similar reasons.

3.2 Table examples

There is understandably no single uniform or standard way of presenting data in a table format, and so, tables tend to have a plethora of unique features between them. These features are best elucidated by taking a closer look at a few examples (Figures 9–11).

Table 1 Clinical features of patients

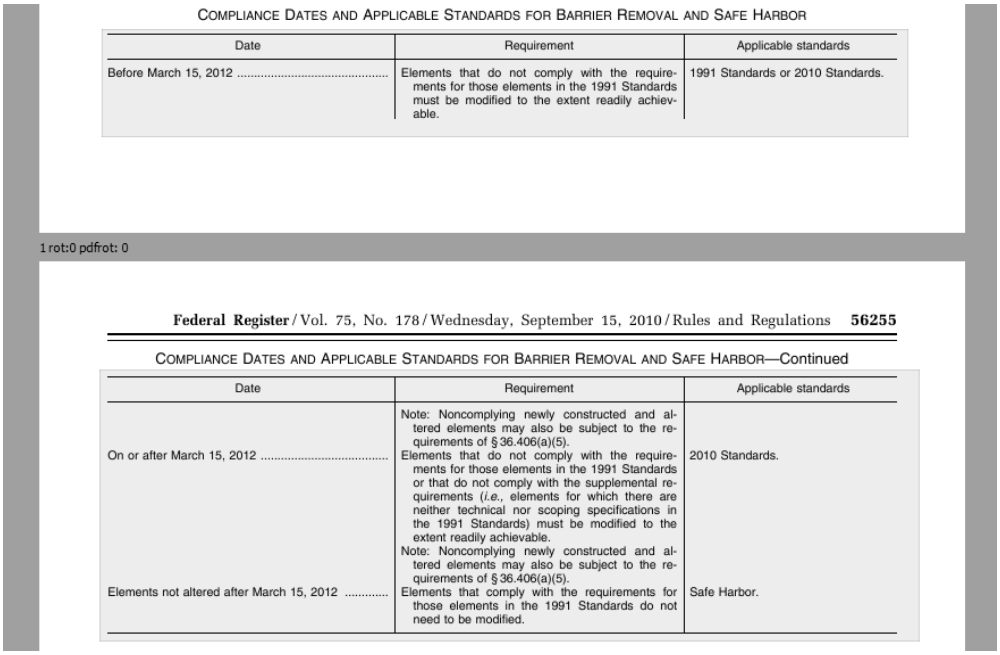
Patient	*Sex*	*Age (years)*	*Onset (years)*	*Family history*	*Ptosis/ PEO*	*Other symptoms*	*MtDNA alterations*
PA	F	63	50	Sporadic	+	None	Multiple deletions in muscle
PB†	**M**	**15**	**8**	**Sporadic**	+	**SANDO**	//
PC	**M**	**63**	**40**	**Sporadic**	+	**Weakness of lower limbs, ataxia, cognitive impairment**	//
PD	M	57	56	Sporadic	+	Exercise intolerance, dysphagia, diabetes, sensory axonal neuropathy	//
PE†	**M**	**4**	**3**	**Recessive**	–	**Alpers syndrome**	Depletion in liver
PF	M	82	78	Sporadic	+	Myopathy, dysphagia, dysphonia, ataxia, tremor, hypoacousia, diabetes	Multiple deletions in muscle
PG	**F**	**45**	**18**	**Dominant**	+	**Myopathy, dysphagia, dysphonia, dyspnea**	//
PH	F	52	Ch	Recessive	–	Myopathy	//
PI	M	62	58	Dominant	–	Myopathy, ataxia, frontal dementia	//
PJ	F	63	54	Dominant	–	Optic atrophy, myopathy, ataxia, sensory axonal neuropathy	//
PK	M	54	18	Sporadic	+	Cardiomyopathy, myopathy, strokes, epilepsy	//
PL	**F**	**27**	**20**	**Sporadic**	+	**SCAE**	//
PM	F	30	5	Sporadic	–	Epilepsy, bilateral deafness, sensory-motor axonal neuropathy	//
PN†	F	Nb	Nb	Sporadic	–	Hepatocerebral failure	Depletion in liver
PO†	**F**	**15 m**	**7 m**	**Sporadic**	–	**Alpers syndrome**	//

PEO: progressive external ophthalmoplegia; SANDO: sensory ataxic neuropathy dysarthria ophthalmoplegia syndrome; SCAE: spinocerebellar ataxia epilepsy syndrome; †: Died; m: months; Ch: childhood; Nb: newborn; (+): present; (–): absent; //: same as upper. Patients for whom the disease causing mutations were identified are in bold.

Figure 9: Typical table from the biomedical research domain features cleanly laid out columns and rows.[*]

While typical tables in research publications in the biomedical domain are well aligned and cleanly laid out, for the algorithms to be applicable universally, a lot of different types of tables need to be considered. Comparison of Figure 9 to Figures 10 and 11 shows a contrast with typical low-complexity tables to more challenging table types.

* *Source: Naïmi et al."Molecular analysis of ANT1, TWINKLE and POLG inpatients with multiple deletions or depletion of mitochondrial DNA by a dHPLC-based assay", European Journal of Human Genetics (2006).*



COMPLIANCE DATES AND APPLICABLE STANDARDS FOR BARRIER REMOVAL AND SAFE HARBOR

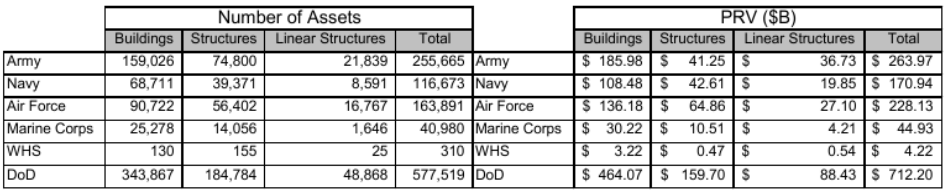
Date	Requirement	Applicable standards
Before March 15, 2012	Elements that do not comply with the requirements for those elements in the 1991 Standards must be modified to the extent readily achievable.	1991 Standards or 2010 Standards.

1 rot:0 pdfrot: 0

COMPLIANCE DATES AND APPLICABLE STANDARDS FOR BARRIER REMOVAL AND SAFE HARBOR—Continued

Date	Requirement	Applicable standards
On or after March 15, 2012	Note: Noncomplying newly constructed and altered elements may also be subject to the requirements of § 36.406(a)(5). Elements that do not comply with the requirements for those elements in the 1991 Standards or that do not comply with the supplemental requirements (*i.e.*, elements for which there are neither technical nor scoping specifications in the 1991 Standards) must be modified to the extent readily achievable. Note: Noncomplying newly constructed and altered elements may also be subject to the requirements of § 36.406(a)(5).	2010 Standards.
Elements not altered after March 15, 2012	Elements that comply with the requirements for those elements in the 1991 Standards do not need to be modified.	Safe Harbor.

Figure 10: Table (light gray area) that is split mid-cell onto two consecutive pages. It also features a semi-gridded structure, where only the middle column cells are encased with four sided rectangles. Source: US-data set [13].



	Number of Assets					PRV ($B)			
	Buildings	Structures	Linear Structures	Total		Buildings	Structures	Linear Structures	Total
Army	159,026	74,800	21,839	255,665	Army	$ 185.98	$ 41.25	$ 36.73	$ 263.97
Navy	68,711	39,371	8,591	116,673	Navy	$ 108.48	$ 42.61	$ 19.85	$ 170.94
Air Force	90,722	56,402	16,767	163,891	Air Force	$ 136.18	$ 64.86	$ 27.10	$ 228.13
Marine Corps	25,278	14,056	1,646	40,980	Marine Corps	$ 30.22	$ 10.51	$ 4.21	$ 44.93
WHS	130	155	25	310	WHS	$ 3.22	$ 0.47	$ 0.54	$ 4.22
DoD	343,867	184,784	48,868	577,519	DoD	$ 464.07	$ 159.70	$ 88.43	$ 712.20

Figure 2. DoD Facilities Portfolio

Figure 11: One or two tables? Source: US-data set [13].

With a large enough sample size, there will always exist a set of tables to break every rule. Taking into account every type of exceptional table is practically impossible, not to mention tables that are misleading and hard to interpret even for human readers. Therefore, for a large enough number of tables from a variety of different sources, an algorithmic approach can never achieve perfect results.

4 ALGORITHMS

This chapter describes the various methods and algorithms that are required and used for extracting tabular data from *PDF* documents. The algorithms described in this chapter provide solutions to the data extraction problems presented in Chapter 3.1.

Some of the algorithms are described using *C++* style pseudo-code, while some are explained using illustrative images and textual descriptions. The algorithms, when combined together, enable fully automatic *PDF* table data extraction.

4.1 Rotation of pages

Each individual page in a *PDF* document can have its main body of text oriented in four possible ways in reference to the upright (text written from left to right, from top to bottom) orientation. The four possible clockwise rotations are: 0°, 90°, 180° and 270°; where pages with 0° rotation are already in an upright orientation. To distinguish between these different rotations, the following pseudo-code algorithm is applied for each individual page (comments in green):

```
//Each element is examined in its original (unrotated) page
//coordinates
Loop for each text element on page:
{
    Skip element that has < 3 characters;

    if( element.height > element.width )
    {
       distanceFromTop = DISTANCE( element.firstChar.top, element.top);
       distanceFromBottom = DISTANCE( element.firstChar.btm,
                                      element.btm);

       //Increase word count for either 90 or 270 degrees rotated words
       if(  distanceFromTop <  distanceFromBottom ) ++rotations90;
       else ++rotations270;
    }
    else
    {
       distanceLeft = DISTANCE( element.firstChar.left, element.left);
       distanceRight = DISTANCE( element.firstChar.right,
                                 element.right);

       //Increase word count for either 0 or 180 degrees rotated words
       if(  distanceLeft <  distanceRight ) ++rotations0;
       else ++rotations180;
    }
}

pageRotation = MAXIMUM( rotations0,  rotations90,  rotations180,
                        rotations270 );
```

The original rotation of a text element is defined here as the rotation that the element is in the *PDF* file with unmanipulated page coordinates. The rectangular text box areas for each element have no orientation themselves. The way to distinguish between upright (rotation 0°) and upside down written text (rotation 180°), because the element areas are exactly alike in shape, is to compare whether the first letter in the element area resides closer to its left or right edge. For upright text, the first character will always be closer to the left edge of the element area rectangle. The same applies for text with 90 or 270 rotations, but instead of comparing the first character of an element to the left or right edges, it can be compared to the top and bottom edges of the element area rectangle.

To distinguish between horizontally written text (0° and 180° rotations), and vertically written text (90° and 270° rotations), element area widths and heights are compared. For text elements that have three or more characters, this comparison will give a good estimation on whether the text is written either horizontally (width > height) or vertically (height > width). For text elements that have only one or two characters, this is not a reliable estimate, because the length of the written word is too small in comparison to the height of the font it is written in. For example, an imagined rectangle drawn around the word “in” would be approximately square in shape, where a three letter word such as “out” would be encapsulated by a rectangle clearly wider in size than tall. This effect is of course emphasized for even longer words.

By calculating the numbers of differently rotated text elements on a page, the algorithm is eliminating the effect of a few words or sentences being written in a different direction, affecting the estimated rotation of the page. This is the case with the publications of many publishers, where for example, the name of the publication or journal appears written in up-down direction in the side margin along the side of the page.

4.2 Edge detection

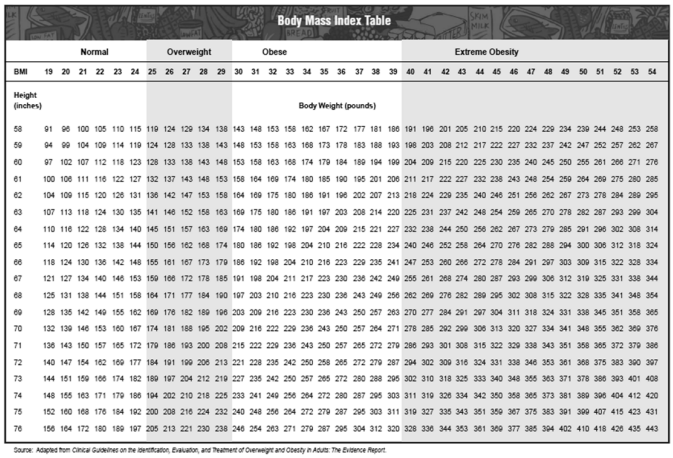
The edge detection algorithm processes rendered image files. The Poppler *PDF* rendering library API provides a convenient function for getting rendered versions of the pages. Example of how the rendered images of pages are acquired using the Poppler library *Qt C++ API* is shown here:

```
// Access page of the PDF file
Poppler::Page* pdfPage = document->page( pageNumber);
// Document starts at page 0
if( pdfPage == 0) {
  // ... error message ...
```

```
    return;
  }
  // Generate a QImage of the rendered page
  QImage image = pdfPage->renderToImage( xres,yres,x,y,width,height);
```

After the image has been rendered, it is converted into gray-scale format, that contains only shades of gray in 255 steps from black to white. Processing the image in a gray-scale format is necessary, because the algorithm is only interested in the pixels intensity values (can also be called brightness for gray-scale images) and their differences between neighboring pixels.

An edge in an image is defined as an above-threshold change in intensity value of neighboring pixels. Choosing a threshold value too high, some of the more subtle visual aids on a page will not be detected, while a threshold value too low can result in a lot of erroneously interpreted edges. Figures 12 and 13 illustrate the goal for the edge detection algorithms.



Body Mass Index Table

| | Normal | | | | | | Overweight | | | | Obese | | | | | | | | Extreme Obesity | | | | | | | |

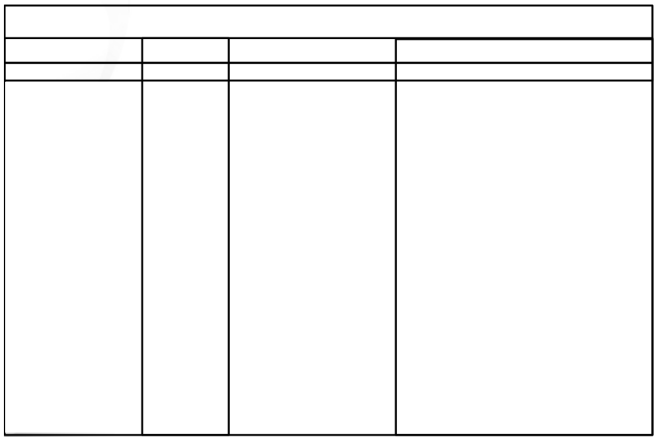


Figure 13: Optimal edge detection result for the table in Figure 12.

The edge detection process is divided into four distinct steps that are described in more detail in the following chapters:

1. Finding horizontal edges.
2. Finding vertical edges.
3. Finding crossing edges and creating “snapping points”.
4. Finding cells (closed rectangular areas).

4.2.1 Finding horizontal edges

The horizontal edge detection algorithm starts by examining the pixels of a gray-scale image from the top left corner. The algorithm compares every top-bottom pair of adjacent pixels, looking for intensity value changes above a set threshold value. Once a pixel-pair with enough difference in their intensities is found, the algorithm proceeds to the right, comparing multiple pixels (in up-down direction) until the edge is no longer present or the right edge of the image is encountered. If the found edge is of sufficient length, it is accepted and registered as a horizontal edge in the image. The flow of the algorithm is visualized in Figure 14.

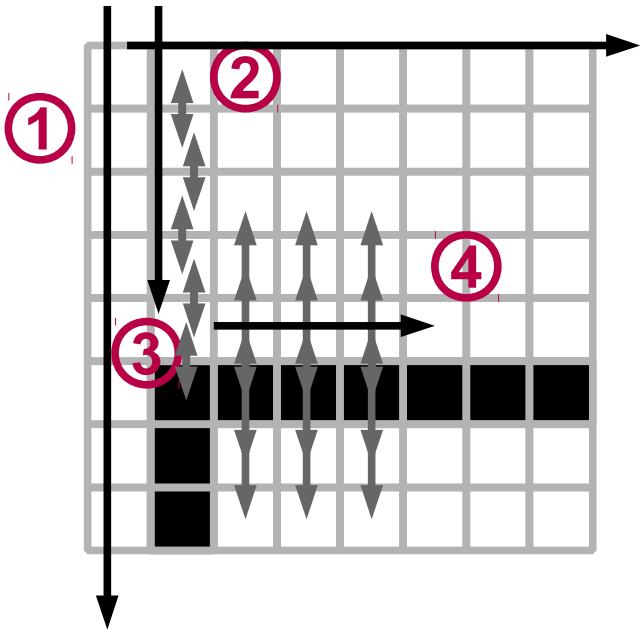


Figure 14: Graphical representation of the horizontal edge detection algorithm. The algorithm moves down the first column of pixels from the top-left corner of an image (1), comparing adjacent top-bottom pairs of pixels. Once it reaches the bottom of the image, it moves on to the next column to the right (2). When a horizontal edge is found (3), the algorithm proceeds along the edge to the right (4) comparing multiple pixels, and stopping when no edge is detected anymore (not shown). The search is then continued from the previously found edge starting-point (3) downward.

Using multiple pixels for detecting horizontal edge preservation, after the initial edge starting-point is found, helps the algorithm deal with edges that are not perfectly aligned, as well as with text elements that are on the edge, obscuring the underlying edge for short lengths at a time. Sufficiently long edges that have been found are saved into memory, so that when the algorithm encounters the same edge again in a subsequently examined column of pixels, it will not be examined again, but instead only skipped over.

If the *PDF* page is rendered as an image too small in pixel size, the bottoms of serifed fonts can blend or blur together, forming horizontal pseudo lines, as illustrated in Figure 15. To avoid this from happening the processed image has to be rendered in large enough size for the individual characters in a word to be separated adequately. Excluding the page text elements areas completely from the edge detection processed image areas, is not possible because in many tables the rows are packed together so tightly, that the text element areas overlap on real horizontal grid edges. The text elements areas often have some extra space under the actual text rendering, to accommodate characters that are partly drawn below the fonts baseline, such as characters “q”, “g” and “p” for example.

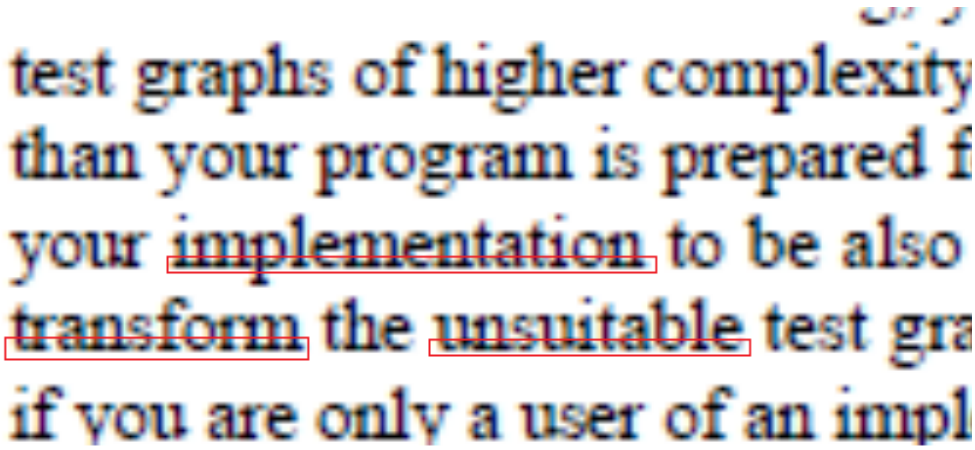


Figure 15: Image quality is important for edge detection. Blurry text (rendered too small) can cause false detections. The three highlighted areas form continuous horizontal edge areas.

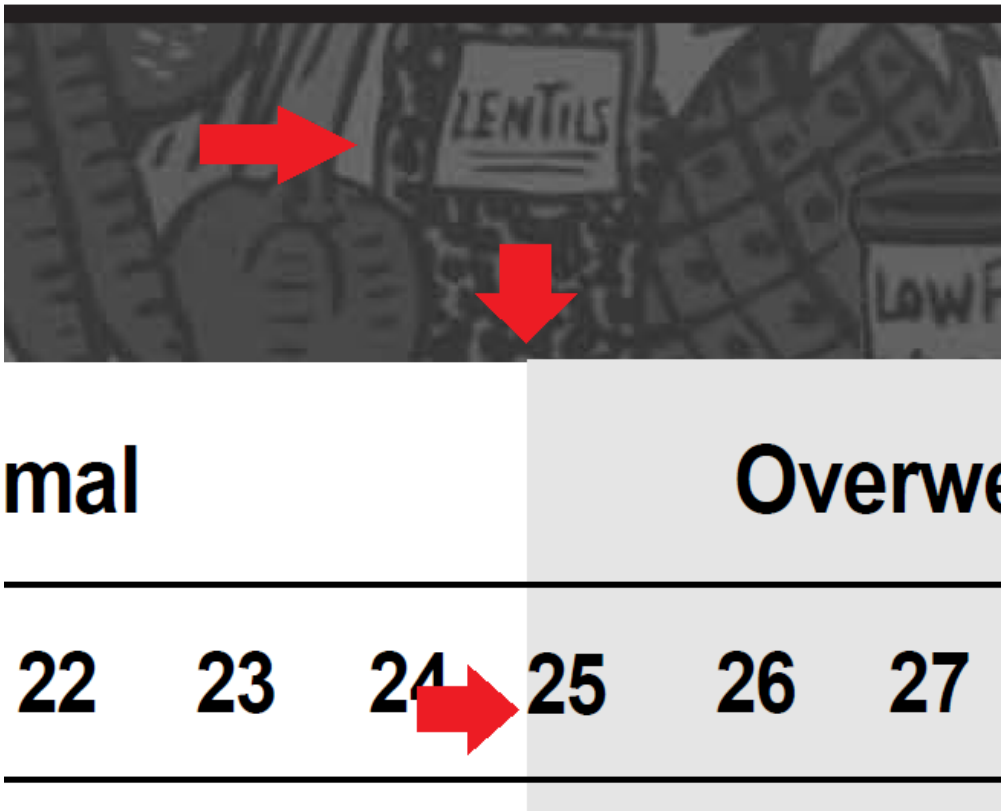


Figure 16: Common problems for edge detection are pointed out with red arrows. The illustrated problems include: graphic backgrounds, misaligned rectangles (or lines), edge-overlapping text, and low-threshold edges.

Using multiple pixels for finding continuing edges, avoids a few common problems as illustrated in Figure 16. Even though most *PDF* documents are created digitally (natively digital), not all edges can be assumed being in perfect alignment.

4.2.2 Finding vertical edges

The vertical edge finding process works much alike the horizontal edges finding process, with one major exception: the text element areas can be excluded from the processing. Unlike with horizontal edges, the real vertical edges very rarely overlap with the text element areas, because there are no issues with font baselines and empty areas within the text element in the left-right direction. Excluding the text element areas avoids the problem of setting the minimum edge length.

Without the exclusion of text elements, many characters within a text element cause false positive vertical edges. For example the leading edges of "I", "P", "L" and other long vertical lines containing characters are prime candidates for causing false positives. If the minimum vertical edge length is set too low (under row character height), all these characters are likely to show up as false positive edges. On the other hand, setting the minimum edge length too high the vertical separators will not show up at all in tightly gridded tables.

4.2.3 Finding and aligning crossing edges

Due to the nature of the edge finding algorithm, described in previous chapters, the crossing points of greater than 1 pixel thick vertical and horizontal edges represent discontinuities in the edge, as shown in Figure 17. The horizontal edge has a gap in it at the position of the original table separator line, and the same applies to the vertical edges. For the edges and table cells to become connected at these points, some further processing is required.

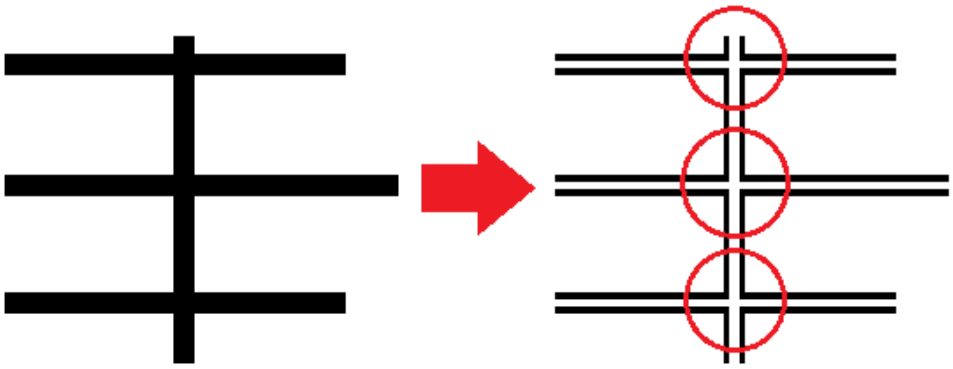


Figure 17: Thick table separator lines produce an edge detection pattern that has unconnected edges. "Snapping points" (red circles) are created to connect all the edge end-points, withing the area of a snapping point, to a single point.

A special snap-to-grid feature in employed. The algorithm searches for both vertical and horizontal positions in the image where many edges end. All edge end-points within a, for example, 10 pixel range from each other are averaged, and this average value will become a "snapping point" with a single pixel center point. All the edge end-points within the snapping point's area are reassigned to this single pixel center-point value. This procedure will connect all the edges in the vicinity of a snapping point.

There are some limitations to this method however. If the snapping point radius is set too big, some of the real grid crossing points become merged together, and if the radius is set too small, some of the edge end-points do not become connected and some rectangular areas are not found at all. Tables with thicker separator lines require longer snapping point radius lengths to become properly connected at their crossings.

4.2.4 Finding rectangular areas

The final step of the edge detection process is to identify closed rectangular spaces within the vertical and horizontal separator lines, and separating unconnected rectangular areas into different tables. This is done is several steps.

After the horizontal and vertical edge detection processing has finished, and the edges are aligned into appropriate snapping-points, points where a horizontal edge contacts a vertical edge are registered as crossing-points. The set of crossing-points is inclusive of points where both a vertical and a horizontal edge end. In other words, one edge does not have to continue through another, it is simply enough that the edges share a common pixel coordinate point.

The following pseudo-code algorithm is applied to find rectangular areas in a set of edges with defined crossing points:

```
array foundRectangles;

//All crossing-points have been sorted from up to down,
//and left to right in ascending order
Loop for each crossing-point:
{
    topLeft = NextCrossingPoint();

    //Fetch all points on the same vertical and horizontal
    //line with current crossing point
    array x_points = CrossingPointsDirectlyBelow( topLeft);
    array y_points = CrossingPointsDirectlyToTheRight( topLeft);

    Loop for each point x_point in x_points:
    {                                  //Skip to next crossing-point
       if( NOT EdgeExistsBetween( topLeft, x_point)) next crossing-
                                                          point;
       Loop for each point y_point in y_points:
       {
          if( NOT EdgeExistsBetween( topLeft, y_point)) next crossing-
                                                             point;
          //Hypothetical bottom right point of rectangle
          btmRight = Point( y_point.x(), x_point.y());

          if( CrossingPointExists( btmRight) AND
              EdgeExistsBetween( x_point, btmRight) AND
              EdgeExistsBetween( y_point, btmRight))
            {
               //Rectangle is confirmed to have 4 sides
               foundRectangles.append( Rectangle( topLeft, btmRight);
               //Each crossing point can be the top left corner
               //of only a single rectangle
               next crossing-point;
            }
       }
    }
}
```

This method of finding rectangular areas ensures that the smallest possible rectangular areas are found with the more common types of table grids. Figure 18 shows some of the possible ways of defining rectangular areas within a grid of horizontal and vertical edges.

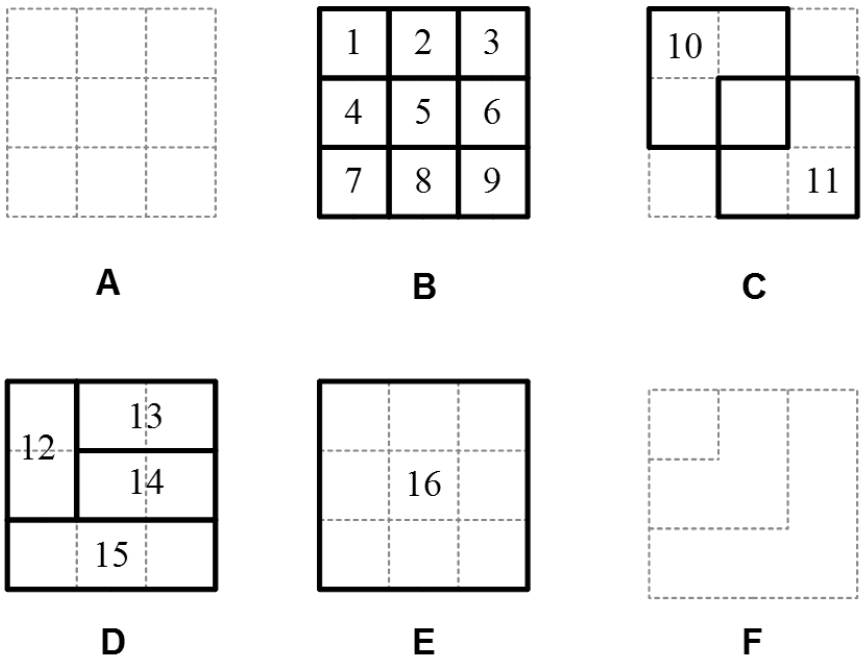


Figure 18: Panels B through E show some of the possible ways of defining rectangular areas within the edges (dashed gray lines) shown in Panel A. The algorithm described in this chapter would define rectangles 1-9. For atypical table edges, such as in Panel F, the algorithm would only define the a single rectangle, the smallest of the possible 3.

After the rectangles (grid cells) of a table have been defined, there is only one more task left to do: assigning the rectangles to tables. In case of pages that contain multiple tables, it is important to separate them from each other. To do this, the corner points of a rectangle are examined. If rectangles share a common corner-point, they are assigned to being in the same table. In case of a 3-by-3 grid table, such as depicted in Figure 18, Panel B, Cell 1 shares two corners with Cell 2 and they are assigned to the same table. Next, Cell 3 is examined, and as it shares corners with Cell 2, it is assigned to the same table as cells 1 and 2. All found cells are examined this way and are assigned together into tables if they themselves, or their connected table partner cells share common corner-points.

4.3 Detecting tables

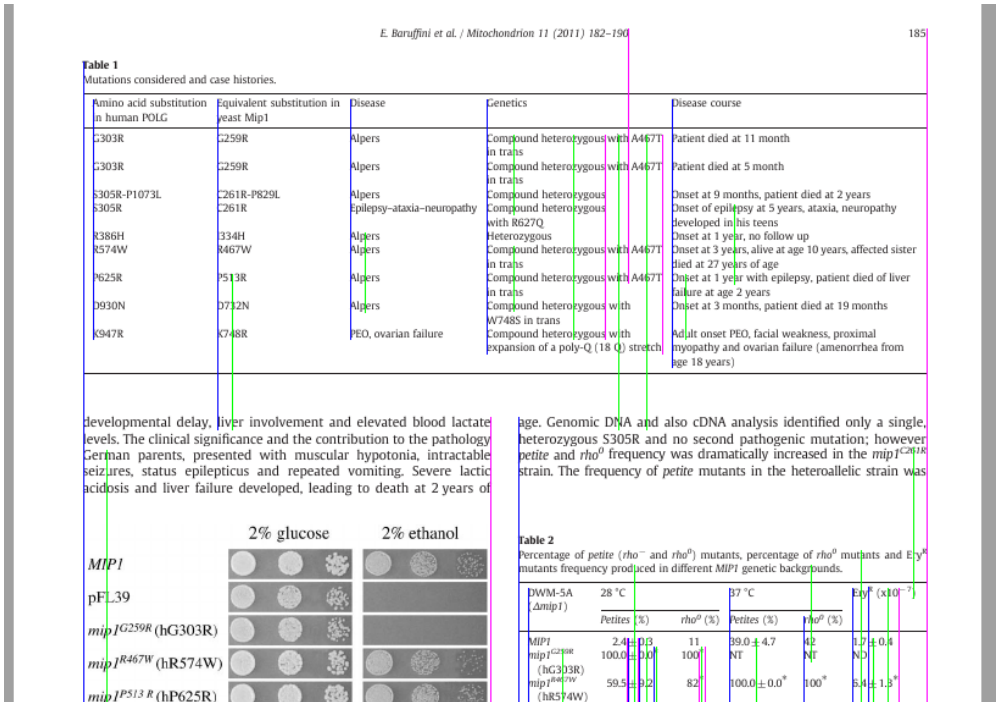
Detecting tables from individual *PDF* document pages is essentially a segregation task. The goal is to separate table elements from non-table elements on the page. After this has been done, the table elements need to be further separated into different tables or merged into a single table.

The main challenge for the table detection algorithm is finding a balance between detecting too much (low purity) and not detecting enough (low completeness). Discovering areas on a page that contain text elements that could have a table structure, is done in several consecutive high level steps:

1. Remove text elements in page margins
    - The page margins often contain superfluous information about the document; such as page numbers, institution logos and names, or publisher information. The first step is to ensure that this information that is irrelevant for the extraction process is weeded out. All text that is displayed in disagreement with the upright orientation of the page is removed completely from further processing.

2. Assign elements into rows
    - A strict initial row assignment is made. Elements are required to be of the same height and to have almost identical vertical coordinates to qualify of being on the same row. After this initial row assignment has been made, some of the rows are merged together based on overlapping areas. This method ensures that super- or subscript text will be merged into the correct row. Merging the super- and subscripts is vital for the next step of processing.

3. Find text edges
    - Text edges are defined to exist in locations where multiple rows have either their element left edges, right edges or center-points at the same vertical line. The minimum number of elements to define a text edge is defined as 4. Elements that break the edge line, also stop the edge from crossing over the element. Figure 19 shows an example of the edges found on a page.



E. Baruffini et al. / Mitochondrion 11 (2011) 182–190 185

Table 1
Mutations considered and case histories.

Amino acid substitution in human POLG	Equivalent substitution in yeast Mip1	Disease	Genetics	Disease course
G303R	G259R	Alpers	Compound heterozygous with A467T in trans	Patient died at 11 month
G303R	G259R	Alpers	Compound heterozygous with A467T in trans	Patient died at 5 month
S305R-P1073L	C261R-P829L	Alpers	Compound heterozygous	Onset at 9 months, patient died at 2 years
S305R	C261R	Epilepsy–ataxia–neuropathy	Compound heterozygous with R627Q	Onset of epilepsy at 5 years, ataxia, neuropathy developed in his teens
R386H	R334H	Alpers	Heterozygous	Onset at 1 year, no follow up
R574W	R467W	Alpers	Compound heterozygous with A467T in trans	Onset at 3 years, alive at age 10 years, affected sister died at 27 years of age
P625R	P513R	Alpers	Compound heterozygous with A467T in trans	Onset at 1 year with epilepsy, patient died of liver failure at age 2 years
D930N	D732N	Alpers	Compound heterozygous with W748S in trans	Onset at 3 months, patient died at 19 months
K947R	K748R	PEO, ovarian failure	Compound heterozygous with expansion of a poly-Q (18 Q) stretch	Adult onset PEO, facial weakness, proximal myopathy and ovarian failure (amenorrhea from age 18 years)

developmental delay, liver involvement and elevated blood lactate levels. The clinical significance and the contribution to the pathology German parents, presented with muscular hypotonia, intractable seizures, status epilepticus and repeated vomiting. Severe lactic acidosis and liver failure developed, leading to death at 2 years of

age. Genomic DNA and also cDNA analysis identified only a single, heterozygous S305R and no second pathogenic mutation; however *petite* and rho^0 frequency was dramatically increased in the $mip1^{C261R}$ strain. The frequency of *petite* mutants in the heteroallelic strain was

Table 2
Percentage of *petite* (*rho*⁻ and *rho*⁰) mutants, percentage of *rho*⁰ mutants and Ery^R mutants frequency produced in different *MIP1* genetic backgrounds.

DWM-5A (*Δmip1*)	28 °C		37 °C		Ery^R (x10^{-7})
	Petites (%)	*rho*⁰ (%)	*Petites* (%)	*rho*⁰ (%)	
MIP1	2.4 ± 0.3	11	39.0 ± 4.7	42	1.7 ± 0.4
$mip1^{G259R}$ (hG303R)	100.0 ± 0.0*	100*	NT	NT	N.D.
$mip1^{R467W}$ (hR574W)	59.5 ± 9.2*	82*	100.0 ± 0.0*	100*	5.4 ± 1.3*

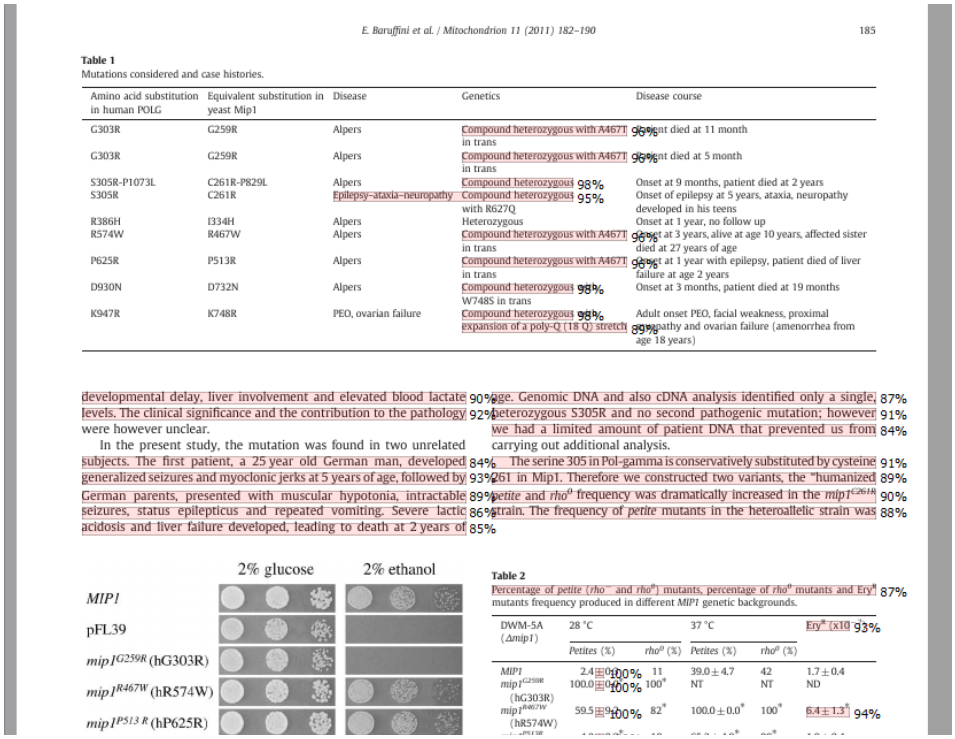
Figure 19: Page excerpt shows the text edges found by the algorithm. Left element edges are shown in blue, right edges by purple, and element center-lines by green color.[*]

- Edges are mostly concentrated to page areas that are tabular in nature. Justified text blocks in multiple page columns need to be identified to not mistake them for tabular areas. Some of the edges also extend beyond the table area limits, connecting with an element that is positioned on the same edge, just by chance.

4. Find justified text blocks

- Justified text blocks need to be identified to prevent mistaking them for tabular regions on the page. A page that has three text columns side by side will contain 6 edges on each horizontal line drawn through the page width, and it is easy for an algorithm to mistake such rows as being a part of a table. This step of the process is illustrated in Figure 20.

* *Source: Baruffini et al., Predicting the contribution of novel POLG mutations to human disease through analysis in yeast model, 2011.*



E. Baruffini et al. / Mitochondrion 11 (2011) 182–190 185

Table 1
Mutations considered and case histories.

Amino acid substitution in human POLG	Equivalent substitution in yeast Mip1	Disease	Genetics	Disease course
G303R	G259R	Alpers	Compound heterozygous with A467T in trans	95% nt died at 11 month
G303R	G259R	Alpers	Compound heterozygous with A467T in trans	96% nt died at 5 month
S305R-P1073L	C261R-P829L	Alpers	Compound heterozygous	98% Onset at 9 months, patient died at 2 years
S305R	C261R	Epilepsy–ataxia–neuropathy	Compound heterozygous with R627Q	95% Onset of epilepsy at 5 years, ataxia, neuropathy developed in his teens
R386H	I334H	Alpers	Heterozygous	Onset at 1 year, no follow up
R574W	R467W	Alpers	Compound heterozygous with A467T in trans	98% t at 3 years, alive at age 10 years, affected sister died at 27 years of age
P625R	P513R	Alpers	Compound heterozygous with A467T in trans	98% t at 1 year with epilepsy, patient died of liver failure at age 2 years
D930N	D732N	Alpers	Compound heterozygous W748S in trans	98% Onset at 3 months, patient died at 19 months
K947R	K748R	PEO, ovarian failure	Compound heterozygous expansion of a poly-Q (18 Q) stretch	98% Adult onset PEO, facial weakness, proximal 89% pathy and ovarian failure (amenorrhea from age 18 years)

developmental delay, liver involvement and elevated blood lactate levels. The clinical significance and the contribution to the pathology were however unclear. 90% 92%

In the present study, the mutation was found in two unrelated subjects. The first patient, a 25 year old German man, developed generalized seizures and myoclonic jerks at 5 years of age, followed by German parents, presented with muscular hypotonia, intractable seizures, status epilepticus and repeated vomiting. Severe lactic acidosis and liver failure developed, leading to death at 2 years of 84% 93% 89% 86% 85%

age. Genomic DNA and also cDNA analysis identified only a single, heterozygous S305R and no second pathogenic mutation; however we had a limited amount of patient DNA that prevented us from carrying out additional analysis. 87% 91% 84%

The serine 305 in Pol-gamma is conservatively substituted by cysteine 261 in Mip1. Therefore we constructed two variants, the "humanized *petite* and *rho*^{0} frequency was dramatically increased in the *mip1*^{C261R} strain. The frequency of *petite* mutants in the heteroallelic strain was 91% 89% 90% 88%

Table 2
Percentage of *petite* (*rho*$^-$ and *rho*^{0}) mutants, percentage of *rho*^{0} mutants and Ery$^+$ mutants frequency produced in different *MIP1* genetic backgrounds. 87%

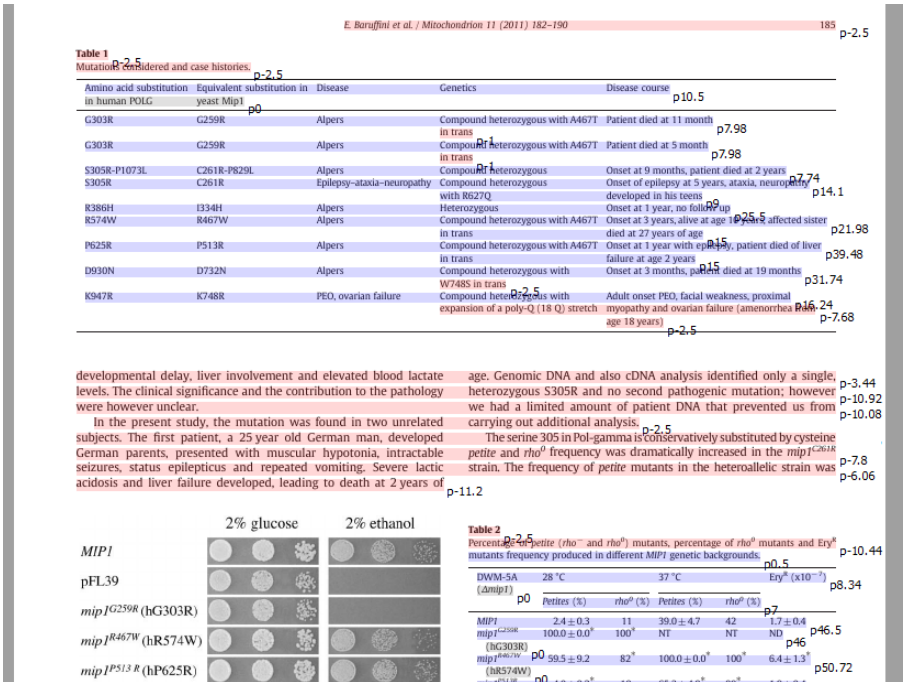
DWM-5A (Δ*mip1*)	28 °C		37 °C		Ery$^+$ (x10^{-7}) 93%
	Petites (%)	*rho*^{0} (%)	*Petites* (%)	*rho*^{0} (%)	
MIP1	2.4 ± 0.1 100%	11	39.0 ± 4.7	42	1.7 ± 0.4
mip1^{G259R} (hG303R)	100.0 ± 0.0 100%	100*	NT	NT	ND
mip1^{R467W} (hR574W)	59.5 ± 9.2 100%	82*	100.0 ± 0.0*	100*	6.4 ± 1.3* 94%

5. Rank each row for its probability of being a part of a table

- Each row on the page is ranked based on the number and the types of edges it contains, as well as the justified text blocks the row contains. This approach does have its limitations. Tables that contain a lot of justified text are easily misclassified as non-table rows. This step of the process is illustrated in Figure 21.

Figure 20: Page excerpt shows regions that are identified as justified text blocks by the algorithm with a red highlight.[*]

* *Source: Baruffini et al., Predicting the contribution of novel POLG mutations to human disease through analysis in yeast model, 2011.*



E. Baruffini et al. / Mitochondrion 11 (2011) 182–190 185

Table 1
Mutations considered and case histories.

Amino acid substitution in human POLG	Equivalent substitution in yeast Mip1	Disease	Genetics	Disease course
G303R	G259R	Alpers	Compound heterozygous with A467T in trans	Patient died at 11 month
G303R	G259R	Alpers	Compound heterozygous with A467T in trans	Patient died at 5 month
S305R-P1073L	C261R-P829L	Alpers	Compound heterozygous	Onset at 9 months, patient died at 2 years
S305R	C261R	Epilepsy–ataxia–neuropathy	Compound heterozygous with R627Q	Onset of epilepsy at 5 years, ataxia, neuropathy developed in his teens
R386H	I334H	Alpers	Heterozygous	Onset at 1 year, no follow up
R574W	R467W	Alpers	Compound heterozygous with A467T in trans	Onset at 3 years, alive at age 18 years, affected sister died at 27 years of age
P625R	P513R	Alpers	Compound heterozygous with A467T in trans	Onset at 1 year with epilepsy, patient died of liver failure at age 2 years
D930N	D732N	Alpers	Compound heterozygous with W748S in trans	Onset at 3 months, patient died at 19 months
K947R	K748R	PEO, ovarian failure	Compound heterozygous with expansion of a poly-Q (18 Q) stretch	Adult onset PEO, facial weakness, proximal myopathy and ovarian failure (amenorrhea from age 18 years)

developmental delay, liver involvement and elevated blood lactate levels. The clinical significance and the contribution to the pathology were however unclear.

In the present study, the mutation was found in two unrelated subjects. The first patient, a 25 year old German man, developed German parents, presented with muscular hypotonia, intractable seizures, status epilepticus and repeated vomiting. Severe lactic acidosis and liver failure developed, leading to death at 2 years of age. Genomic DNA and also cDNA analysis identified only a single, heterozygous S305R and no second pathogenic mutation; however we had a limited amount of patient DNA that prevented us from carrying out additional analysis.

The serine 305 in Pol-gamma is conservatively substituted by cysteine in several patients. To test the pathogenicity of this mutation, we introduced the equivalent mutation in yeast Mip1, the yeast DNA polymerase gamma. The *petite* and *rho*^{0} frequency was dramatically increased in the *mip1*^{C261R} strain. The frequency of *petite* mutants in the heteroallelic strain was

Table 2
Percentage of *petite* (*rho*$^-$ and *rho*^{0}) mutants, percentage of *rho*^{0} mutants and Ery^R mutants frequency produced in different *MIP1* genetic backgrounds.

DWM-5A (Δ*mip1*)	28 °C		37 °C		Ery^R (x10^{-7})
	Petites (%)	*rho*^{0} (%)	*Petites* (%)	*rho*^{0} (%)	
MIP1	2.4 ± 0.3	11	39.0 ± 4.7	42	1.7 ± 0.4
mip1^{G259R} (hG303R)	100.0 ± 0.0*	100*	NT	NT	ND
mip1^{R467W} (hR574W)	59.5 ± 9.2	82*	100.0 ± 0.0*	100*	6.4 ± 1.3*

Figure 21: Page excerpt shows rows with a high probability of being a part of a table highlighted in blue color. *

6. Assign table and extend table areas

   - The last step of the table detection process entails defining the limits or boundaries of tables. If grids and defined rectangular areas exist on the page, all four or more connected rectangular areas found by the edge detection algorithm (Chapter 4.2) are classified as tables. In the absence of grids, the rows defined as containing tabular content are unified to form rectangular areas. These areas are then extended to cover rows above and below them, based on their separation, to include table title and caption areas. This method of extending the boundaries of tables can produce erroneous results in documents with only narrow spacing between the table boundaries and page body text.

* *Source: Baruffini et al., Predicting the contribution of novel POLG mutations to human disease through analysis in yeast model, 2011.*

4.4 Defining table structure

The table structure definition (or recognition) process is the next step after the table has been detected and the limits of its area have been defined by the table detection algorithm. The algorithm in this thesis has been designed to process table areas that include their title and captions, even though most literature available on table structure definition does not recognize these elements as being a part of a table. This decision, to include the title and captions, was made based on their usefulness for further semantical or functional analysis (see [14]). The table detection algorithm can make its own predictions about where the table title and captions are, but further processing is still needed to separate them from the table header and body.

The algorithm for recognizing the table structure processes the elements of the table in several high level steps:

1. Find and merge super- and subscripts
    - Super- and subscripts in a document can be problematic for table row detection. A superscript lying just between two rows can, in the worst cases, cause splitting or merging of two separate rows. All text elements within the area of a table are examined for super- or subscript status. Found super- and subscript elements are merged with the normal text elements in their immediate vicinity to form a single element.

2. Assign to rows
    - The algorithm needs to assign each table element to a row. All elements with an adequate vertical alignment are defined as being on the same row. Even this fairly simple sounding step has its own problems with elements that are not completely aligned. In some cases better results would be obtained by defining the rows more liberally and in others a more conservative method of separating the rows would yield better results.

3. Merge spaces
    - Sentences are created out of words assigned on the same rows. The spaces in the document, between words that belong to the same sentence are decided by averaging the spaces widths on the page. The average spacing length (in *PDF* coordinates) is used to determine which words can be merged into a single element. Elements are never merged over vertical separator lines or from different grid cells. Justified text (aligned to both, left and right edge of

the paragraph) is problematic for this step of the process, since it contains sentences with variable width spacing, and often very wide spaces. Fortunately, tables rarely contain justified text blocks without a grid structure.

4. Find obvious column edges

- Column edges are defined to locations where multiple rows have either element left edges, right edges or center-points at the same vertical line. The minimum number of elements to define an edge is set as 4. This step of the algorithm is identical to Step 3 of the table detection process but includes only the table area.

5. Find rows that do not fit the apparent column structure

- Every row that breaks the column edge structure defined in Step 4, is marked as being a "column breaking row". These rows have a higher probability of belonging to either the title, header (superheader), subheader, or caption row categories and they are excluded from Step 10, *Assign to columns*.

6. Examine the grid

- Find out if the edge detection algorithm has defined rectangular areas (Chapter 4.2.4) to exist within the limits of the table area. The table is defined as having one of the following grid styles: full, supportive, outline, none. A full grid means that the cellular structure of the table is completely defined by the rectangular areas and no further processing of the table body is needed, and the algorithm skips to Step 11, *Finding the header*. All elements outside a full or an outline style grid are set as being a part of the title or the caption. A Supportive grid helps to determine cell row- and column spans, but otherwise the elements are processed just like the grid would not exist at all. For fully gridded tables, the performance of the edge detection algorithm is critical. Any mistakes, or missed borderlines, directly show up as errors in the row and column definitions.

7. Examine underlines

- Underlines are horizontal lines defined by the edge detection algorithm, that are not a part of a rectangular area, and do not cover more than 80% of the table width. If a horizontal line covers more width, it is classified as a horizontal separator. If an underline has only a single element on top of it, at a reasonable distance, this element is extended to the width of the underline. This procedure helps in discovering elements spanning multiple columns.

8. Find super- and subheaders
    - A superheader row is defined as a row that has elements that span over two or more elements on either the row above or the row below. This is quite a promiscuous way of defining superheader rows, and it will classify a lot of rows erroneously. The main idea of this step is to remove rows that might be problematic for the column definition step. A subheader row is simply a row that only has elements in the table stub or the first column, if no stub exists.

9. Find title and caption
    - The title and legend rows are segregated from the table header and body rows. Several types of text withing the table area have a higher probability of being a part of the table title text: text that is in the top rows of the table, text that is defined as “column breaking” and runs through almost the entire width of the table, text that is above the first horizontal separator, text that is centered to the table width or text that is placed only to the left side of the table. Any text that fits these descriptions adequately is classified as a title. For the caption (legend), similar qualities are looked for, but on the bottom rows of the table.

10. Assign to columns
    - Columns of the table are determined by finding empty vertical areas through the table width. This empty area detection, excludes the rows that in the previous steps have been classified as either “column breaking rows”, subheader rows, superheader rows, title rows, or caption rows.

11. Finding the header
    - Separately described techniques are used for detecting the table header rows, see Chapter 4.5.

12. Merge columns
    - Columns without a header (if a boxhead has been found) or columns that do not contain any data, are merged to the column on their left.

13. Format header
    - The header rows often have more variance in their layout on the page, and often span multiple rows and columns. Header cells that are adjacent to empty cells in the header are extended to fill these empty cells.

14. Merge rows

- Based on the header row information (Step 11) and row indentations; some of the rows are merged together. This is an essential step in creating single cells out of blocks of elements that have been split onto multiple rows. When examining the rows, the algorithm looks for rows that contain no data in the first column (stub), and contain data above every other non-empty cell. Because most tables contain a stub that includes only the first column of the table, this approach can achieve good results. For rare tables, with a more complex, multi-column stub, a more sophisticated stub definition and processing method would be required.

15. Set column and row spans for cells

- The final step is to extend elements within grid cells to fill their full available areas and define their row and column spans.

4.5 Finding the header rows

After the table data has been sorted into appropriate rows and columns, it is time to find its header rows (if any). As described in Chapter 2.3, the information provided by the Poppler *PDF* library is somewhat limited. An even larger obstacle for the header finding algorithm to identify the column header rows, in contrast to the abilities of a human reader, is the lack of contextual and semantical understanding of the table data. For an (English speaking) human reader, identifying the header rows is a quite trivial task in most cases, as illustrated in Table 22.

	1	2	3	4	5	6	7
1	**Patient**	**Sex**	**Onset**	**LA**	**Depletion**	**Gene**	**Allele1**
2	*A*	M	28 months	NA	10%	TK2	c.C462W
3	*B*	F	15 months	++	22%	POLG	p.Y154N
4	*C*	M	2 years	normal	12%	TK2	c.Q340S
5	*D*	M	15 months	normal	11%	TK2	c.T460A
6	*E*	F	5 years	normal	32%	POLG	p.A155P
7	*F*	M	30 years	++	8%	TK2	c. G34T
8	*G*	F	34 months	normal	21%	TK2	c. A35T

Table 22: For a human reader, identifying the column header rows *(only Row 1) is a trivial task.*

	1	2	3	4	5	6	7
1	?	?	?????????	??	????	????	????? ?
2	?	?	?? ??????	??	???	???	?? ????
3	?	?	?? ??????	??	???	????	?? ????
4	?	?	? ?????	??????	???	???	?? ????
5	?	?	?? ??????	??????	???	???	?? ????
6	?	?	? ?????	??????	???	????	?? ????
7	?	?	?? ?????	??	??	??	?? ????
8	?	?	?? ??????	??????	???	??	?? ????

Table 23: For an algorithm, without any contextual or semantical understanding, Table 22 looks effectively like this. As some tables do not have separator lines or rectangles, they have been removed from this example, along with font family and style information, because it is not available through the Poppler API (2.3).

Table 23 illustrates the starting-point for the header detection algorithm. Because every table does not have separator lines (or has lines between every row), they alone are not an adequate way of determining the column header rows. Also, the Poppler *API* does not provide information about the font families, or font styles used in the table. Because of the eclectic and non-standardized nature of tables, no single method can work on every table. Therefore, an "expert" voting system is implemented.

A "toolkit" of different algorithms is used to examine the contents of the table cells. Each algorithm casts a vote on the probability of each row being a column header row. Once every algorithm in the toolkit has had its chance to cast a vote, all the votes are collated, and a final conclusion (consensus) is drawn. The following chapters present some of the algorithms in the toolkit.

Each of these individual header prediction components is parametrized with a "weight" for its vote. So that the predictor components that are more often correct in their predictions for certain kinds of tables, are given extra votes for the final evaluation and decision making. Future plans include automating the parametrization of the components using a machine learning-based method. This, however, requires developing a testing data set that has ground truth values for the correct amount of header rows in each table of the data set, against which the predicted values can be compared.

4.5.1 Header predictor: Numbers

Numbers, if they exist in a table, are quite a reliable source of column header row identification. The Numbers algorithm looks for columns that have cells with only text in the top rows, and a long list of cells with numerical content below them. This type of prediction is illustrated in Table 24. This method is very effective with tables that contain numerical data in their body.

	1	2	3	4	5	6	7
1	?	?	**Onset**	??	**Depletion**	????	????? ?
2	?	?	28 months	??	10%	???	?? ????
3	?	?	15 months	??	22%	????	?? ????
4	?	?	2 years	??????	12%	???	?? ????
5	?	?	15 months	??????	11%	???	?? ????
6	?	?	5 years	??????	32%	????	?? ????
7	?	?	30 years	??	8%	??	?? ????
8	?	?	34 months	??????	21%	??	?? ????

Table 24: The Numbers algorithm looks for columns that have text-format cells on top of a column of numerical cells. Recognized column headers are shown in red.

The Numbers predictor cannot make any predictions with tables that do not contain any numerical data. Also, problematic for the predictor are tables that have numerical column headers. These types of columns are quite common, especially with financial data tables, where the column header can simply contains a year-number for the column body data.

Another type of pitfall for the Numbers predictor involves column headers that have only a few, or a single word per row. Imagine a column header such as “Number of families with income less than $50 000”, where “$50 000” is set alone on the last row (line) in the header cell. If the column body below the header then contains only numerical data, the Numbers predictor could easily mistake the last row of the column header as being a part of the table body.

4.5.2 Header predictor: Repetition

Repetition of row values within a column can be used as an indicator on where the header stops. There usually exists no reason to repeat rows within the column header,

and therefore, looking for repeated cells within a column can be used as an effective measure for determining, which rows cannot be a part of the header. The Repetition algorithm needs to be more conservative and reserved in its voting for positive header rows, because if the first data cell in a column happens to be not repeated, it would be easily mistaken as a header row, as illustrated in Table 25, Column 4.

	1	2	3	4	5	6	7
1	?	Sex	?????????	**LA**	????	**Gene**	????? ?
2	?	F	?? ??????	NA	???	TK2	?? ????
3	?	M	?? ??????	++	???	POLG	?? ????
4	?	M	? ?????	normal	???	TK2	?? ????
5	?	F	?? ??????	normal	???	TK2	?? ????
6	?	M	? ?????	normal	???	POLG	?? ????
7	?	F	?? ?????	++	??	TK2	?? ????
8	?	F	?? ??????	normal	???	TK2	?? ????

Table 25: While less reliable in making an accurate prediction of the exact number of header rows than the Numbers algorithm, the Repetition algorithm, is very efficient in identifying rows that are not a part of the column header.

While the Repetition algorithm is less reliable in determining what is the last row of the header, it is very reliable in telling which rows cannot be a part of the column headers. In the case illustrated in Table 25, the Repetition algorithm can say with a good degree of reliability that Row 2, is not a part of the header. This prediction is made by observing that the cell contents “F” and “TK2” are repeated multiple times in Columns 2 and 6 respectively.

4.5.3 Header predictor: Alphabet

Many tables, especially ones that have a stub and row headers, order their rows under the stub header alphabetically, or numerically, in either ascending or descending order. A long line of alphabetically ordered cells below non-ordered cells, is a tell-tale sign of the column header rows, as illustrated in Table 26. In small tables, the ordering in the stub can sometimes be coincidental, therefore, the amount of consecutive ordered cells needs to be limited to a minimum size of four or five, depending on the table size for accurate predictions.

	1	2	3	4	5	6	7
1	**Patient**	?	?????????	??	????	????	????? ?
2	*A*	?	?? ??????	??	???	???	?? ????
3	*B*	?	?? ??????	??	???	????	?? ????
4	*C*	?	? ?????	??????	???	???	?? ????
5	*D*	?	?? ??????	??????	???	???	?? ????
6	*E*	?	? ?????	??????	???	????	?? ????
7	*F*	?	?? ?????	??	??	??	?? ????
8	*G*	?	?? ??????	??????	???	??	?? ????

Table 26: A long list of alphabetically ordered consecutive cells in a column, under non-ordered cells ("Patient") in the table stub, is often clear indication of where the header starts and stops.

A common pitfall for this type of predictor is an accidental stub header ordering. Imagine a stub column that has the following cells from up to down: "Country", "Finland", "Germany", "Italy", "Sweden"; with "Country" being the only cell of the stub header. For the Alphabet predictor it is easy to mistake the whole column not having a header at all, because it has alphabetical ordering starting from Row 1.

4.5.4 Header predictor: other methods

Sometimes, none of the easy ways of identifying the table header rows are effective. In such cases some more subtle methods in the header prediction toolkit are required. Such methods include:

- **Empty stub header**: If the stub head is empty, the first non-empty cell in the stub indicates the first row of the table body.
- **Font size**: Some tables have their header in a larger font size. Comparing element heights withing a column can help identify the header rows.
- **Data types**: If a column has integer numbers in the top rows and decimal numbers in all the rows below, it could be an indication of the header rows.
- **Lists**: The table header is less likely to have comma-separated lists than the cells of the table body.
- **Natural Language**: If the top rows have natural language-like words (3 or more consecutive characters of the alphabet), while the rows below contain only non-

alphabet characters (such as “+” or “*”), or a mixture of numbers and letters, this is a good indication of the header rows.

- **Text alignment**: If the elements are aligned to the center of the column in the top rows, and to the left or the right edge of the column in the rows below, it could be an indication of the header rows.
- **Separators or boxed areas**: Horizontal separator lines often separate the header from the table body.
- **Superheaders and nested headers**: It is uncommon for a column to have **only** a shared header with another column. If a cell in the top rows of the table spans multiple columns, the row below it is more likely to be a header row as well.

See Chapter 2.1, “Table anatomy”, for a description of the used terms for the table parts and elements.

4.6 Outputting extracted data

The extracted data is outputted in two possible formats: *HTML* and *XML*. The *HTML* output can be viewed on any web browser without modification, and imported into modern spreadsheet applications with ease. The *XML* format output is quite similar to the *HTML* style output with the exception that its tags (“<tag>”) can be customized with an *XSL* stylesheet, for easy integration to existing software components.

4.6.1 Application programming interface

The *C++ API* provided by the software application developed as a part of this thesis enables easy access to the table data through different functions. The processed page data is available through the *API* as `Table` and `TableCell` objects. First, a class `TablerInstance` object is created to extract and retrieve the table information. It is also possible to provide specific regions where the tables are located. This feature enables the possibility of using a different algorithm for detecting the tables. The following *C++* code shows the main interface functions of the *`TablerInstance`* class:

```
class TABLER_LIB TablerInstance
{
  public:
    //Extract tables from a PDF document
    //Ownership of returned Table pointers transferred to caller
    QList<Table*> ExtractTables( const QString& aFilePath, const
                                 QString& aPageRange = QString());

  ...
```

```
    //Extracts Tables from specified regions in a Poppler Document

    //Ownership of returned Table pointers transferred to caller
    QList<Table*> SortTables( Poppler::Document* aPopDoc, const
                              QList<TableRect>& aTableRects );
...

};
```

If a *PDF* document has already been opened using the Poppler *PDF* rendering Library extracting table information is done in the following way:

```
Tabler::TablerInstance* tabler = Tabler::Instance();
QList<Tabler::Table*> tables = tabler->GetTables( iPopDoc);
delete tabler;
```

The `Table` class provides easy access to the extracted table's structured contents:

```
class Table
{
  public:

    int PageNum() const;
    int NumberOfHeaderRows() const;
    int Rows() const;
    int Cols() const;

    QList<TableCell*> Cells() const;

    TableCell* Cell( int aRow, int aCol ) const;

    QRectF TableArea() const;
    QRectF CellArea( int aRow, int aCol ) const;

    QString XMLDescription() const;
    QString HTMLDescription() const;

    QString HTMLCellData( int aRow, int aCol ) const;
    QString CellData( int aRow, int aCol ) const;

    Poppler::Page::Rotation Rotation() const;
    QSizeF PageSize() const;

    QStringList TableInfoTypes();
    TableInfoList GetTableInfo( const QString& aInfoType);

    …

};
```

4.6.2 Standalone usage

Extracting table data can also be done using an executable file and the command line interface of the operating system (for more details see Chapter 5.5). Extracting table data from an *PDF* file is done by typing the following command:

```
tabler.exe --html --pages=1-6 tables.pdf > output.html,
```

where “tabler.exe” is the executable program (containing the algorithms), “tables.pdf” is a *PDF* file containing tables, and “output.html” is the desired name for the file where the output of the program is written. “--html” and “--pages=1-6” are optional “flags” for the program, specifying that the desired output format is *HTML,* and that only pages 1 through 6 should be processed. All of the possible options for command line usage of the program can be viewed by using the flag “--help”.

5 EMPIRICAL EVALUATION

The focus and purpose of this thesis work is in practical applications of extracting data from scientific publications. These publications often employ a conservative color palette and a limited amount of visual gimmickry. The publications are, for the most part, carefully written and their layouts are designed by professionals working for different publishers. The typical tables in such publications have multiple header rows that are associated with the entire body of the table.

For these reasons, getting a good evaluation on the "good", lucid, and well formed tables is more important than it is on the nonconforming, very complex, or "badly" formed tables. In other words to get a perfect score on easy tables is more important than to get a good score on difficult tables.

5.1 Evaluation metrics and performance

The evaluation of a table data extraction algorithm is not a completely trivial matter. The evaluation is divided into two separate parts: evaluating table detection and boundary recognition, and evaluating table structure recognition. This method of evaluation has been proposed by *Göbel et al.* [14] and it provides a reasonable, standardized way of comparing the performance of different algorithms.

5.1.1 Evaluating table structure recognition

The performance of the table structure recognition algorithm is based on the cellular structure of the table. The cell structure is defined as a matrix of cells. Ground truth values are set manually for each table in the test data set for comparison as described in Chapter 5.2.

Instead of comparing absolute row and column index values for each cell, only neighboring cell relationships are evaluated. This method of table structure evaluation has been proposed by Hurst [15] and it has a number of advantages against the more simple row and column index number evaluation.

The method developed by Hurst evaluates the performance of a table structure recognition algorithm with an abstract geometric model, where spatial associations between the table cells are known as *proto-links,* that exist between immediate neighboring cells. With this model, a variety of errors that may occur can be considered separately (e.g. cells can be split in one direction, merged in another; entire blank columns can appear). The main idea is that the model allows for errors that are insignificant for the overall structure of a table. One extra column in the middle of the table does not ruin the scor-

ing for the remaining columns. A visualization of *proto-links* in a table is shown in Figure 27.

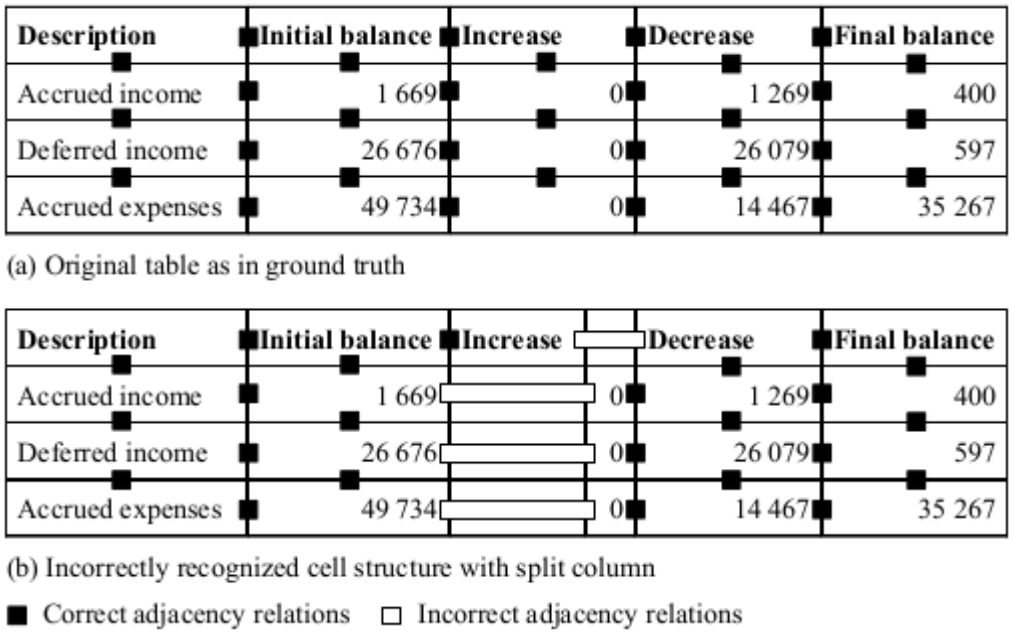


Figure 27: Comparison of an incorrectly detected cell structure with the ground truth. source: Göbel et al. [14]

Table structure recognition evaluation uses an F-score to quantify the performance of the structure definition algorithm. F-score is defined as:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall},$$

where recall and precision are defined as:

$$Recall = \frac{correct\ adjacency\ relations}{total\ adjacency\ relations},$$

$$Precision = \frac{correct\ adjacency\ relations}{detected\ adjacency\ relations}.$$

Panel **a** in Figure 27 shows the correct *proto-links* as dark squares, of which there are 31 (total adjacency relations). Panel **b** in Figure 27 shows an example case of algorithm output, with an incorrectly split 3rd column, resulting in only 24 correct adjacency relations, and 4 incorrect adjacency relations, making the total of detected adjacency relations 28 (24+4). In this example case the F-score would be calculated as follows:

$$Recall = \frac{24}{31} \approx 77.4\%$$

$$Precision = \frac{24}{28} \approx 85.7\%$$

$$F = 2 \cdot \frac{0.857 \cdot 0.774}{0.857 + 0.774} \approx 81{,}3\%$$

The F-score is calculated so that each document in the test set has the same weight in the average precision and recall scores, no matter how many tables it contains. Tables found within the pages of a single document are usually quite uniform in their layout. Calculating the F-score on a document level, rather than table level, eliminates the effects of long documents with a lot of similarly (or identically) laid out tables affecting the score disproportionately.

Establishing the correct adjacency relations requires manual examination of each table in the used data set, and sometimes the correct division of cells is ambiguous. This method of comparison does not allow for either-or relationships between the table cells. Cells that span through multiple columns or rows can have more than four neighboring cells.

Because the table *proto-links* that are compared to the ground truth *proto-links*, do not have any cell coordinates, only the cell contents are matched to asses if the right adjacency relationship exists. In the implementation of the structure performance analysis method, in case of tables where multiple cells contains the exact same content, the same *proto-link* can exists multiple times, without any problems. For example if the ground truth table would contain the relationship “A above B” two times, and the comparison table three times, recall for this particular relationship is 2/2, and precision 2/3.

5.1.2 Evaluating table detection

Table detection, in its essence, is a segregation task. The goal is to separate the elements of a page into table-, and non-table elements. The table detection evaluation measures the ability of the algorithm to find tables within the pages of a *PDF* document in terms of *completeness* and *purity*. The definitions of completeness and purity are taken from Silva [16]. The two terms are defined in the context of table detection evaluation as follows:

- **Completeness**: proportion of tables containing all of their elements with respect to the total number of tables on the page. In order for a table to be complete, it must contain all of its elements.

$$Completeness = \frac{completely\ identified\ tables}{total\ ground\ truth\ tables}.$$

- **Purity**: proportion of tables containing only correctly assigned elements with respect to the total number of tables on the page. In order for a table to be pure, it must contain only correctly assigned elements.

$$Purity = \frac{purely\,identified\,tables}{total\,identified\,tables}.$$

The harmonic mean of completeness and purity (CPF) is used as the measure for the overall performance of the table detection algorithms. It is defined as:

$$CPF = 2 \cdot \frac{Completeness \cdot Purity}{Completeness + Purity}.$$

CPF is calculated so that each document in the test data set, no matter how many tables it contains, has the same weight in the purity and completeness average score.

The resulting purity-score is an indicator for how well the recognized area is within the bounds of the ground truth area. The completeness-score is an indicator of how well the recognized area covers the whole defined ground truth table area. The CPF score, is an indicator of the overall performance of the algorithms.

Why this method of comparison is chosen over the element-based F-score comparison used in table structure recognition, is that it provides a more useful indication in typical table recognition error scenarios. Comparing a single table on a page to another table using the element based F-score would work just fine. The usefulness of the completeness and purity is best described by examining a few examples such as a table detection split error, shown in Figure 28.

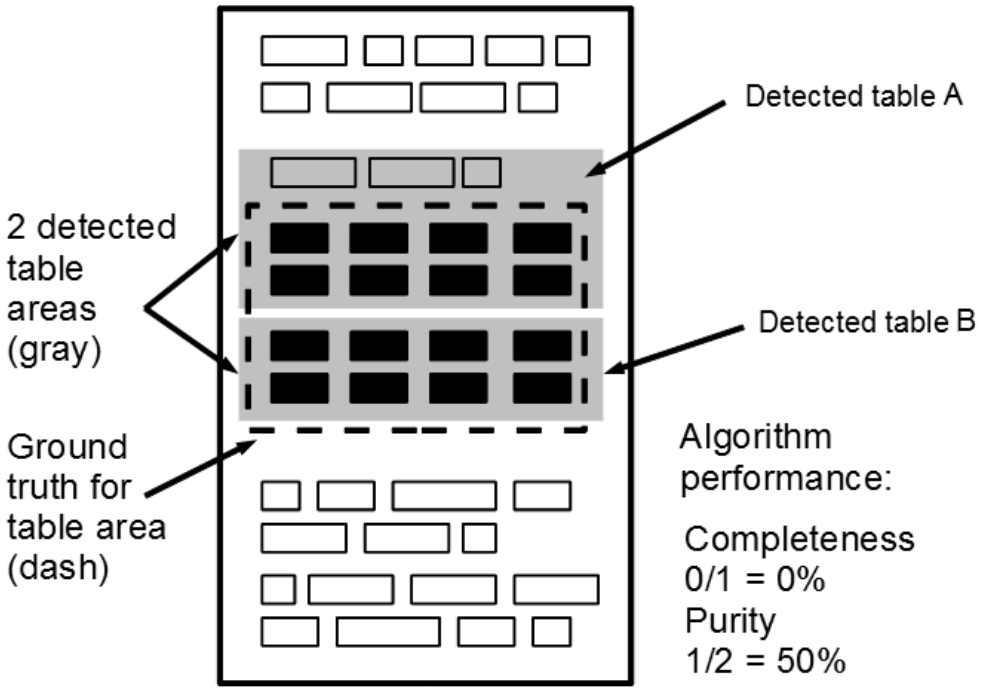


Figure 28: Split errors are common in table detection. Table elements in the ground truth definition for the page are shown as black rectangles, and non-table elements as white rectangles.

One ground truth table is associated with only one comparison table (if any). The association is determined by comparing the elements on the page. For each ground truth table, the association is established with a comparison table that shares the most elements with it. In rare cases, such as depicted in Figure 28, where two tables share the exactly same amount of common elements with a ground truth table (8 in this case), the table with the least amount of overall elements (Table B) is chosen as the associated table. In the case shown in Figure 28, the unassociated, detected Table A would be classified as a false detection.

Another type of common table detection errors is a merge error, where multiple ground truth tables are recognized as a single table (Figure 29). The difference between merge- and split errors, is that a split error affects the completeness score negatively, while a merge error affects the purity score negatively.

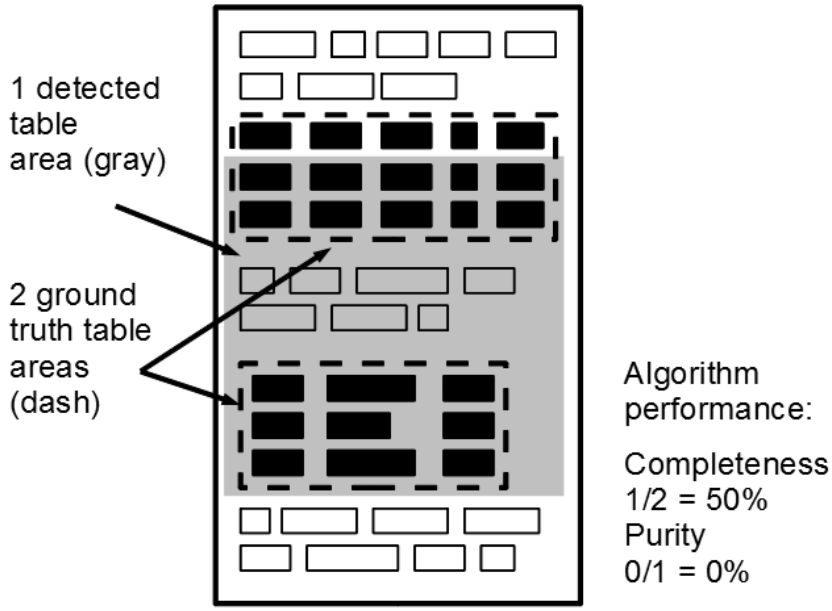


Figure 29: Merge errors are common in table detection. Table elements in the ground truth definition for the page are shown as black rectangles, and non-table elements as white rectangles.

There is no additional false detection penalty scoring; the falsely detected tables only increase the purity score denominator, lowering the purity score. With table merge errors, the purity score is affected directly, because two or more ground truth tables detected as one, are never pure.

The tables are rated as either pure or impure, complete or incomplete; there is no middle ground. If a table area contains even a single non-table element, or an element from another table, it is assigned as being impure. This method of evaluating performance requires quite a large set of test data documents to give an accurate estimation of the algorithm's performance.

5.2 Performance evaluation implementation

For performance evaluation, the algorithm outputs an *XML* file that is compared to a ground truth *XML* file. Comparison is done using a *Python* script. The script reads as its inputs an element file, that defines the text elements of a page; a ground truth file, that defines "the correct answers"; and a comparison file, that contains the "opinion" of the algorithm on the table locations and cellular structure. The element file is needed for determining association between the ground truth and comparison tables.

Due to the way that the *PDF* standard is specified, the word boundaries for textual elements can be somewhat ambiguous. In other words, the *PDF* standard does not define a single correct way of defining words in a document, and two different *PDF* readers or libraries can have some differences in the way words are specified. To avoid problems with unwanted textual elements becoming included inside a table area, only the element center-points are considered when determining whether an element is a part of a table area or not.

5.3 Test data

The performance evaluations for the algorithms are done using two different data sets, provided by *Göbel et al.* [14], called the EU-data set and the US-data set. The EU-data set currently consists of 34 public domain documents, gathered from various European Union government websites. The US-data set consists of 25 public domain United States government website *PDF* documents. Both data sets contain an eclectic set of tables, that for the most part surpass the typical scientific publication table complexities by a large margin (see Chapter 3.2). The test data ground truth table areas exclude both the table title and legend. The table detection algorithm had to be modified to accommodate these changes for the testing phase.

The two used data sets (EU and US) sets are a part of an International Conference on Document Analysis and Recognition (ICDAR) 2013 table competition* and they are freely available on the Internet [13]. The data sets are likely to be expanded in the future, while the authors of *Göbel et al.* [14] are working towards a more unified toolkit for standardized testing methods for table detection and structure recognition. The ground truths for the data sets are provided by *Göbel et al.* [14]. Some examples of the test set tables are shown in Chapter 3.2.

5.4 Performance results

The performances of the table structure recognition algorithm and table detection algorithm were evaluated using two sets of *PDF* documents, the EU- and the US-data sets [13]. Measuring the structure recognition performance was done by using manually determined ground truth areas for tables. In other words: completely independently of the table detection algorithm.

The documents in both test data sets represent a disproportionate amount of complex tables, compared to average tables found in scientific publications. Also, both data sets contain many documents with tables that are either split into multiple table areas on a single page, or tables that are seemingly connected to each other but should be separated

* http://www.icdar2013.org/

into two or more tables. Both of these types of tables are quite a rare occurrence in scientific publications.

5.4.1 Table structure recognition performance results

Long continuous strings of dots and underscores (“.”, “_”) were removed from the table cell contents before comparison. These characters are commonly used as visual aids in tables to align stub row headers with their table body data. While these strings of characters can be considered being a part of the textual content of a table, they serve no semantical purpose, and therefore do not need to be extracted. Chapter 5.1.1 describes the evaluation methods in more detail. The results of the table structure recognition performance analysis are presented in Table 30.

Data set	Documents	Perfect scores	Precision	Recall	F-Score
EU	*34*	18	96.70%	96.43%	**96.57%**
US	*25*	9	86.34%	87.38%	**86.85%**

Table 30: Table Structure recognition performance results.

The overall performance of the algorithm could be evaluated as “very good” or “excellent” based on the results. A major part of the incorrect output of the structure detection algorithm is due to erroneous output of the edge detection algorithm. The measured performance score is somewhat affected by the complexity of the test data set tables, and not directly by the performance of the algorithm, but by ambiguity of some of the adjacency relations (*proto-links*) of the table elements. See chapter 7.1 for a more detailed look at the low scoring tables, and for a more in-depth discussion on the performance of the algorithm.

Future plans for developing the algorithm include creating a new test data set with typical scientific publication tables, for an even more accurate evaluation of the performance of the algorithm for its intended purpose. Output of the table structure precision and recall script is supplied for the EU- and the US-data sets as appendices A and B respectively.

5.4.2 Table detection performance results

The table detection algorithm was adapted to exclude table titles and caption texts, to suit the test data set ground truth definitions. Table 31 presents the achieved results of the table detection algorithm.

Data set	Documents	Perfect scores	Purity	Completeness	CPF
EU	*34*	9	77.82%	49.12%	**60.22%**
US	*25*	11	73.96%	64.00%	**68.62%**

Table 31: Table detection performance results.

Overall performance of the algorithm could be evaluated as “modest” based on the results. The table detection scores are influenced by at least five significant factors:

1. The algorithm has not been designed to detect tables without a defined grid structure with less than 4 rows (minimum size for gridded tables is 2x2).
2. The algorithm has been designed to include table titles and captions.
   - Including a title or a caption sets purity to 0% for a table when compared to the used data sets' ground truth definitions. The algorithm was adapted to exclude table titles and captions for the comparison, but performs better with them included.
3. Strict yes/ no method of assigning completeness and purity for each table.
   - One single extra or missing element sets the purity or completeness score of a table to 0%; there are no *almost* correct answers.
4. Merging multiple separate table areas into a single table has not been implemented.
5. Splitting unified table areas into multiple separate tables has not been implemented.

Output of the table detection purity and completeness script is supplied for the EU- and the US-data sets as appendices C and D respectively.

5.4.3 Performance in terms of time

Most of the processing time used by the algorithm is spent by the edge detection algorithm. The Poppler library renders the document pages into greater than 1000 x 1000 pixel images, depending on the average font size on the page. The large size of the rendered image is required to avoid blurring and cluttering of the table separator lines and text. There are still a lot of options available for optimizing and improving the edge detection process in the future.

Table 32 shows actual measured times for different parts of the table extraction process, on a typical 2013 desktop 64-bit computer with Intel i7-2600k *CPU,* for an idea of the time required to process *PDF* documents.

Data set	Page Rendering (Poppler)	Edge Detection	Table Detection	Table Structure recognition
EU	49.8 ms	647.7 ms	20.7 ms	75.3 ms
US	217.0 ms	679.1 ms	18.3 ms	68.5 ms

Table 32: Average page processing times for the algorithms.

Typically processing a single page that contains tables takes under 1 second. Pages that do not contain any tables are processed approximately 100 ms faster. Both used test data sets contain tables on most pages, but also pages that do not contain any tables. The pages that do not contain any tables are not calculated for the table structure recognition algorithm processing time. The Poppler library page rendering time for the US-data set was offset from the EU-data set result by a few documents with over 2000 ms page rendering times. The Poppler library does not yet provide a thread safe interface (planned) which would allow processing multiple pages with multiple threads at the same time.

5.5 Implementation

The practical implementation of the algorithms is made with *Qt C++*. *Qt* is an open source framework for developing applications for multiple platforms, such as Windows, Mac and Linux systems and various mobile phone platforms. It allows for easy portability between these systems and offers an extensive *GUI* framework. *Qt* is available under the LGPL 2.1 and *GPL* version 3 licenses, and also offers a commercial license for a fee.

The implementation is made into two separate parts: a shared dynamic link library module (DLL) and a *GUI* application for visualizing and debugging the performance of the algorithm. The DLL is also compiled into a self-standing executable file, for direct command line usage. Both parts of the implementation are made available online, including their source codes.

Scripts for performance evaluations are created with *Python* scripting language. The scripts read in *XML* files in a specified format [13], and evaluate the performance of the algorithms as described in Chapters 5.1.1 and 5.1.2.

The whole project consists of more than 20 000 lines of *C++* and *Python* code, and took several months to complete. The project is initially made available only as source code, but future plans include releasing it as an installable application package. The working title for the project *GUI* is “Harvezter” and “Tabler” for the component containing the table extraction algorithms.

6 RELATED WORK

The need for automatic table data extraction has been recognized by others as well. A few such systems for *PDF* document table extraction are represented in this chapter. The lack of standard data sets and performance evaluation methods hinders the comparison of the performance of different approaches to other existing similar approaches. A standard test set and testing methods would be required for better comparisons.

6.1 PDF-TREX

The PDF-TREX table data extraction system was published in 2009 by Oro and Ruffolo [17]. The PDF-TREX system employs a heuristic approach (experience based) to table detection and structure definition. In comparison to the software application created as a part of this thesis work, it has the following limitations:

- Only single column documents are supported.
- Ruling lines or other visual aids on the page are not considered.

The PDF-TREX system itself is not reported to be available for public use, but the data set used for evaluating its performance is currently (March, 2013) available on the Internet [12]. The test data set is provided without any ground truth definitions, and it consists of 100 Italian, single column *PDF* documents (mostly financial data). The authors of the PDF-TREX system report an F-score of 91.97% for table detection and 84.61% for table structure definitions. These reported results are not directly comparable to the scores obtained in the performance evaluation of this thesis work (Chapter 5), due to the different data sets, as well as, and more significantly, the performance evaluation methods.

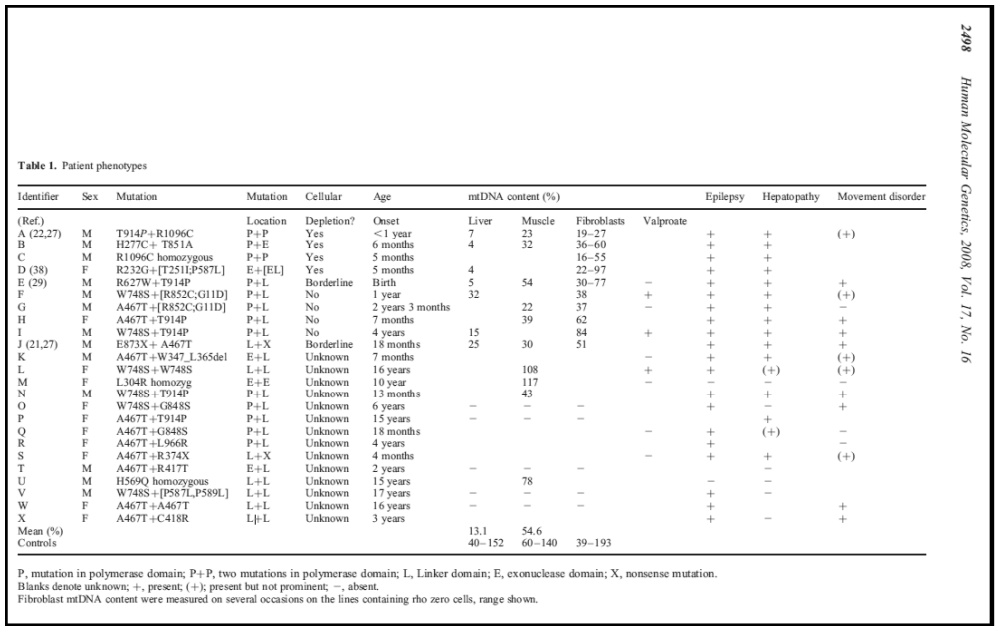
A somewhat more comparable measurement is reported for the PDF-TREX system by Hassan [18], who is one of the authors of the suggested performance evaluation method [14] adapted by this thesis work. Hassan reports a table detection F-score of 55.9% and table structure recognition F-score of 85.1% for the PDF-TREX system.

6.2 Pdf2table

Pdf2Table is a *PDF* table extraction system developed by Yıldız as a Master's thesis work at Vienna University of Technology in 2004 [19]. It uses a software tool called “pdf2html” to converting the contents of a *PDF* document to *HTML* which it processes further to detect table areas and table structures. The pdf2table system is designed as a semi-automatic extraction system with a *GUI* that allows for the user to make modifica-

tions to the table structures proposed by the extraction algorithm. The author writes: “You should expect that a post-process in form of changes in the user interface must be done in almost each case.”.

In a later paper in 2005 by *Yildiz et al.* [20], the pdf2table system is reported of having a table detection F-score of 92.33% and table structure recognition F-score of 85.42%. As the test data set, the authors report to have used 150 *PDF* documents gathered using web search engines. The test data set of 150 document was further divided into 50 lucid or unsophisticated tables and 100 complex tables without further explanations. It is also unclear if the system had been developed further into a more automated version considering the achieved scores. The pdf2table system and its source code are available on the Internet [21]. The used data set is not available, and so, a direct comparison with the approach is impossible.



2498 Human Molecular Genetics, 2008, Vol. 17, No. 16

Table 1. Patient phenotypes

Identifier	Sex	Mutation	Mutation	Cellular	Age	mtDNA content (%)				Epilepsy	Hepatopathy	Movement disorder
(Ref.)			Location	Depletion?	Onset	Liver	Muscle	Fibroblasts	Valproate			
A (22,27)	M	T914P+R1096C	P+P	Yes	<1 year	7	23	19–27		+	+	(+)
B	M	H277C+ T851A	P+E	Yes	6 months	4	32	36–60		+	+	
C	M	R1096C homozygous	P+P	Yes	5 months			16–55		+	+	
D (38)	F	R232G+[T251I;P587L]	E+[EL]	Yes	5 months	4		22–97		+	+	
E (29)	M	R627W+T914P	P+L	Borderline	Birth	5	54	30–77	−	+	+	+
F	M	W748S+[R852C;G11D]	P+L	No	1 year	32		38	+	+	+	(+)
G	M	A467T+[R852C;G11D]	P+L	No	2 years 3 months		22	37	−	+	+	−
H	F	A467T+T914P	P+L	No	7 months		39	62		+	+	+
I	M	W748S+T914P	P+L	No	4 years	15		84	+	+	+	+
J (21,27)	M	E873X+ A467T	L+X	Borderline	18 months	25	30	51		+	+	+
K	M	A467T+W347_L365del	E+L	Unknown	7 months				−	+	+	(+)
L	F	W748S+W748S	L+L	Unknown	16 years		108		+	+	(+)	(+)
M	F	L304R homozyg	E+E	Unknown	10 year		117		−	−	−	−
N	M	W748S+T914P	P+L	Unknown	13 months		43			+	+	+
O	F	W748S+G848S	P+L	Unknown	6 years	−	−	−		+	−	+
P	F	A467T+T914P	P+L	Unknown	15 years	−	−	−			+	
Q	F	A467T+G848S	P+L	Unknown	18 months				−	+	(+)	−
R	F	A467T+L966R	P+L	Unknown	4 years					+		−
S	F	A467T+R374X	L+X	Unknown	4 months				−	+	+	(+)
T	M	A467T+R417T	E+L	Unknown	2 years	−	−	−			−	
U	M	H569Q homozygous	L+L	Unknown	15 years		78			−	−	
V	M	W748S+[P587L,P589L]	L+L	Unknown	17 years	−	−	−		+	−	
W	F	A467T+A467T	L+L	Unknown	16 years	−	−	−		+		+
X	F	A467T+C418R	L|+L	Unknown	3 years					+	−	+
Mean (%)						13.1	54.6					
Controls						40–152	60–140	39–193				

P, mutation in polymerase domain; P+P, two mutations in polymerase domain; L, Linker domain; E, exonuclease domain; X, nonsense mutation.
Blanks denote unknown; +, present; (+); present but not prominent; −, absent.
Fibroblast mtDNA content were measured on several occasions on the lines containing rho zero cells, range shown.

Figure 33: Original table used for pdf2table test run.[*]

* *Source: Ashley et al.,”Depletion of mitochondrial DNA in fibroblast cultures from patients with POLG1 mutations is a consequence of catalytic mutations”. Human Mol. Genetics, 2008.*

TABLE ON PAGE 1

Identifier	Sex	Mutation	Mutation	Cellular	Age	mtDNA content (%)				Epilepsy	Hepatopathy	Movement disorder	
(Ref.)			Location	Depletion?	Onset	Liver	Muscle	Fibroblasts	Valproate				tics,
A (22,27) B	M M	T914PþR1096C H277Cþ T851	PþP PþE	Yes Yes	.1 year 6 months	7 4	23 32	19 27 36 60		þ þ	þ þ	(þ)	2008,
C	M	R1096C hom gous	PþP	Yes	5 months			16 55		þ	þ		
D (38) E (29)	F M	R232Gþ[T251I,P587L] R627WþT914P	Eþ[EL] PþL	Yes Borderline	5 months Birth	4 5	54	22 97 30 77	2	þ þ	þ þ	þ	Vol.
F G	M M	W748Sþ [R852C;G11D A467Tþ[R8 1D)]	PþL PþL	No No	1 year 2 years 3 months	32	22	38 37	þ 2	þ þ	þ þ	(þ) 2	17,
H I	F M	A467TþT914 W748SþT914P	PþL PþL	No No	7 months 4 years	15	39	62 84	þ	þ þ	þ þ	þ þ	No.
J (21	M	E873Xþ A 467T	LþX	Borderline	18 months	25	30	51		þ	þ	þ	16
K	M	A467TþW347_ L365del	EþL	Unknown	7 months				2	þ	þ	(þ)	
L	F	W748SþW748S	LþL	Unknown	16 years		108		þ	þ	(þ)	(þ)	
M	F	L304R homozyg	EþE	Unknown	10 year		117		2	2	2	2	
N	M	W748SþT914P	PþL	Unknown	13 months		43			þ	þ	þ	
O	F	W748SþG848S	PþL	Unknown	6 years	2	2	2		þ	2	þ	
P	F	A467TþT914P	PþL	Unknown	15 years	2	2	2			þ		
Q	F	A467TþG848S	PþL	Unknown	18 months				2	þ	(þ)		
R	F	A467TþL966R	PþL	Unknown	4 years					þ			
S	F	A467TþR374X	LþX	Unknown	4 months				2	þ	þ	(þ)	
T	M	A467TþR417T	EþL	Unknown	2 years	2	2	2			2		
U	M	H569Q homozygous	LþL	Unknown	15 years		78			2	2		
V	M	W748Sþ [P587L,P589L]	LþL	Unknown	17 years	2	2	2		þ	2		
W	F	A467TþA467T	LþL	Unknown	16 years	2	2	2		þ		þ	
X	F	A467TþC418R	LþL	Unknown	3 years					þ	2	þ	
Mean (%)						13.1	54.6						
Controls						40 - 152	60 140	39 193					

Figure 33: The pdf2table system test run results. The pdf2table table extraction system was able to recognize the table structure quite well with the column headers defined correctly. The mistakes in this example table (red arrows) were: 4 incorrectly merged rows and an extra column at the table right edge. These errors are most likely due to text written in up-down direction in the page right margin that interferes with the row structure.

A few example tables were used to test pdf2table performance. Figure 33 shows the original example table and Figure 34 shows the pdf2table *HTML* output. Most significant mistakes were made with tables in multiple column documents (clearly an unsupported document type for pdf2table). Tables with subheader rows were often split into multiple tables erroneously.

6.3 Other products

An Internet search turns up a few other products for the purposes of PDF table extraction. None of them offer the same automated functionality as the software application created as a part of this thesis work.

- VeryPDF *PDF* Table Extractor.
  - According to the VeryPDF website, the tool allows the user to manually (using a GUI) specify the table limits and row and column structures. *Commercial license*.

- Okular *PDF* reader
  - The Okular *PDF* reader allows the user to manually specify table area limits, row and column separators and to extract a table. The interface of the program is shown in Figure 35. *Freeware*.

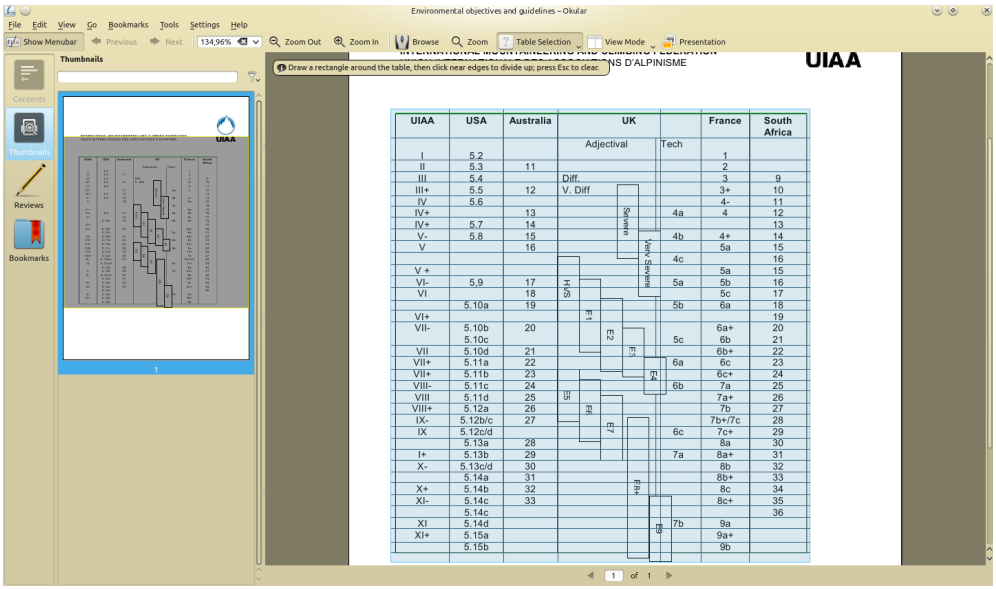


Figure 35: The Okular PDF reader lets the user manually specify table areas and to define the column and row limits. The table can then be exported to other applications.

- Adobe Professional
  - The commercial version of the free Adobe PDF reader has a functionality to export table areas defined manually. *Commercial license*.

7 CONCLUSIONS

As a tool for “big data” extraction, the created software application performs very well in defining table structure in correctly defined table areas. The table detection algorithm performs well with tables that have more than 2 columns, but is still not very accurate in including all the relevant elements at the top and bottom extremities of the tables.

Tables that are split on to multiple pages or unified table areas that should be split into multiple tables are not supported by the table detection algorithm. These are not essential features in most practical applications for the algorithms, because in most cases, the relevant data will still be extracted with correct header-element associations.

7.1 Discussion on accuracy

Several different factors, that are trade-offs with each other, affect the performance of the algorithms. Fine-tuning of several of the parameters of the algorithm could still yield significant improvements. Most room for improvement exists in the table detection and edge detection algorithms.

Few distinct features of the algorithms that have not been evaluated with any quantifiable way are: header row recognition, title row recognition, and caption row recognition. These features of the algorithm serve an important role in further semantical or functional processing of the table data.

The header row recognition algorithm (Chapter 4.5) is a major part of the table structure recognition algorithm (Chapter 4.4). Thus, its performance directly affects the performance of the table structure recognition algorithm (Chapter 5.4.1). As a rough estimation it would seem to be performing equally as well as the full table structure recognition algorithm.

Evaluating the title and caption row recognition parts of the algorithm is a bit more difficult, because they were completely removed when adapting the algorithm for evaluation according to the test data set ground truth definitions. As a rule of thumb, they are able to find the most common types of titles (centered on tables or running the entire width of a table), with some difficulties in recognizing more imaginative ways of presenting table titles and captions. Overall, errors in this part of the algorithm usually only result in the title becoming a column header or vice versa, a column header being marked as a table title. Erroneously detected captions are typically insignificant, since the caption text will typically only become a part of the table body.

7.1.1 Edge detection performance

The main source of errors for the edge detection algorithm, are edges that should not exist at all. One such example is shown in Figure 36. The problems in edge detection in this example are not caused by erroneous function of the algorithm, but rather by the design and construction of the document and the table, that have resulted in unwanted white lines around textual elements in the table header.

Importance of the objective	Creating European centres of excellence through collaboration between laboratories.	Launching European technology initiatives	Stimulating the creativity of basic research through competition between teams at European level
very important	48.4%	41.9%	45.9%
Important	42.4%	44.1%	35.3%
not important	4.5%	6.3%	8.1%
Unnecessary	2.8%	1.5%	6.4%
don't know	1.3%	6.1%	4.2%

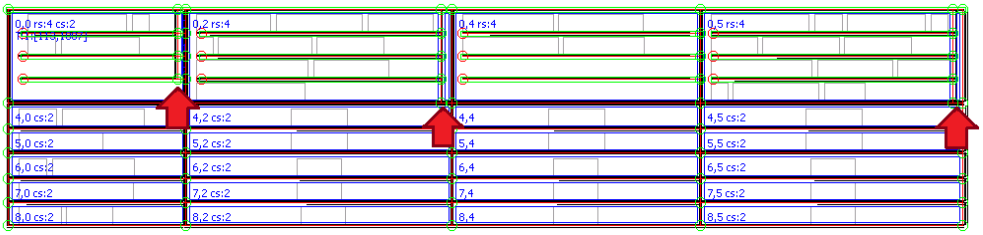
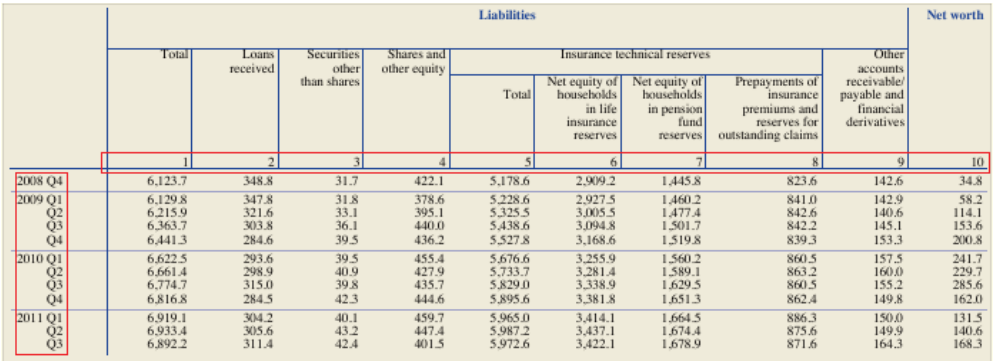


Figure 36: The top panel shows the original PDF document table. Due to the way the document has been constructed, white outlines surround column header text blocks. The lower panel shows the raw output of the edge detection algorithm, with incorrectly detected rectangular areas highlighted with red arrows. *Source: EU-data set [13].*

To address these kind of problems in edge detection, the robustness of the algorithm could be further improved by adding more rules to the formation of grid cells. One viable strategy to avoid such unwanted behavior by the algorithm, would be to enforce minimum sizes (width) for empty rectangular areas.

7.1.2 Problematic tables and data set ambiguities

The used data set ground truth values for the table locations and table structure definitions have been evaluated by three experts in the area of table structure recognition [14], and all tables that they could not agree on were removed from the data set before it was published. However, some definitions of the ground truth can still be argued in a few different directions. Figures 37 through 39 show some of the worst scoring tables in the US- and EU-data sets.

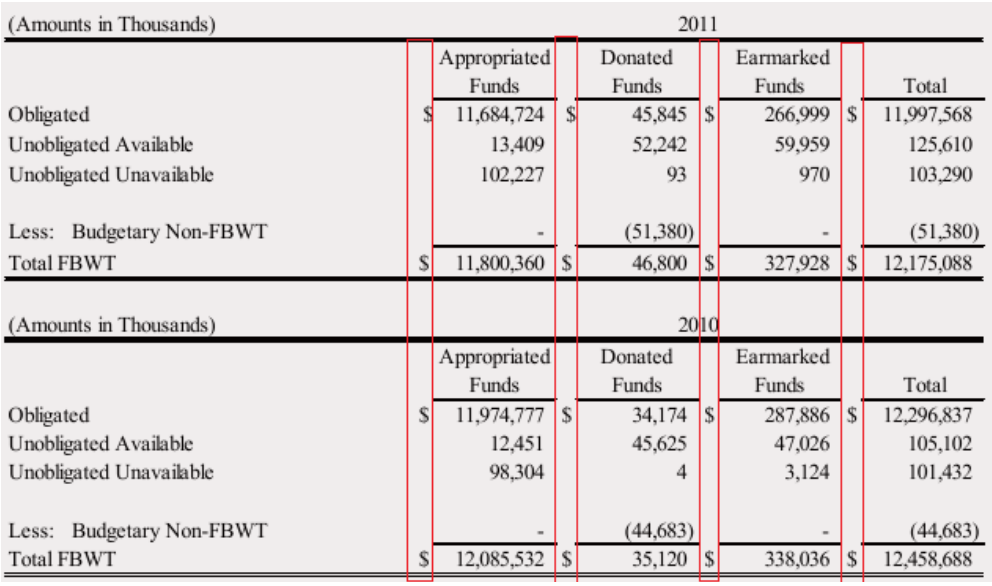


	Liabilities									Net worth
	Total	Loans received	Securities other than shares	Shares and other equity	Insurance technical reserves				Other accounts receivable/ payable and financial derivatives	
					Total	Net equity of households in life insurance reserves	Net equity of households in pension fund reserves	Prepayments of insurance premiums and reserves for outstanding claims		
	1	2	3	4	5	6	7	8	9	10
2008 Q4	6,123.7	348.8	31.7	422.1	5,178.6	2,909.2	1,445.8	823.6	142.6	34.8
2009 Q1	6,129.8	347.8	31.8	378.6	5,228.6	2,927.5	1,460.2	841.0	142.9	58.2
Q2	6,215.9	321.6	33.1	395.1	5,325.5	3,005.5	1,477.4	842.6	140.6	114.1
Q3	6,363.7	303.8	36.1	440.0	5,438.6	3,094.8	1,501.7	842.2	145.1	153.6
Q4	6,441.3	284.6	39.5	436.2	5,527.8	3,168.6	1,519.8	839.3	153.3	200.8
2010 Q1	6,622.5	293.6	39.5	455.4	5,676.6	3,255.9	1,560.2	860.5	157.5	241.7
Q2	6,661.4	298.9	40.9	427.9	5,733.7	3,281.4	1,589.1	863.2	160.0	229.7
Q3	6,774.7	315.0	39.8	435.7	5,829.0	3,338.9	1,629.5	860.5	155.2	285.6
Q4	6,816.8	284.5	42.3	444.6	5,895.6	3,381.8	1,651.3	862.4	149.8	162.0
2011 Q1	6,919.1	304.2	40.1	459.7	5,965.0	3,414.1	1,664.5	886.3	150.0	131.5
Q2	6,933.4	305.6	43.2	447.4	5,987.2	3,437.1	1,674.4	875.6	149.9	140.6
Q3	6,892.2	311.4	42.4	401.5	5,972.6	3,422.1	1,678.9	871.6	164.3	168.3

Figure 37: Structure recognition score P/ R: ***0.82/ 0.81.*** *Highlighted (red outline) areas show problematic stub, and bottom header row. The algorithm merges the table stub incorrectly into a single column. The data set ground truths define highlighted row as merged into the header text cells above it. The algorithm sees it as a separate row. Source: EU-data set [13].*

Figure 37 shows a table where a row of incremental numbers, under the column header texts is included in the same cell with the column header text in the data set ground truths. For a human reader of the document it makes very little difference, but an algorithm will be scored for 20 imprecise and 16 unrecalled *proto-links* in this case (not counting issues with the stub). A more sophisticated performance testing system could allow for either-or types of *proto-linkages* for such special cases.

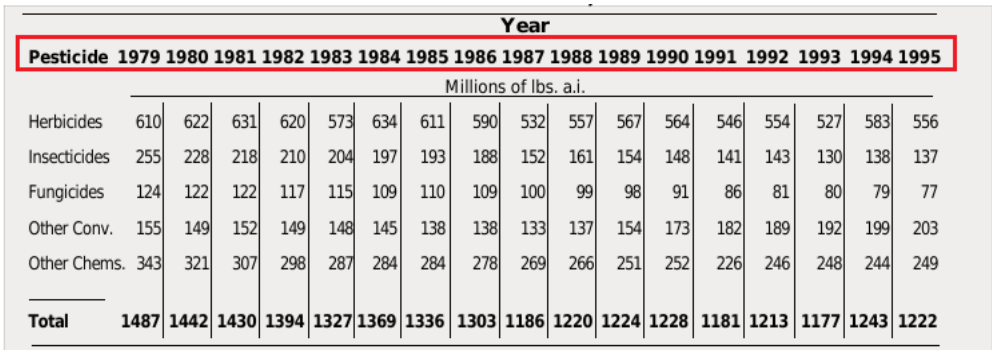
Figure 38 shows a table, or perhaps two tables marked as one in the ground truth definitions of the US-data set. In either case, the $-signs, because they appear in a separate column, that has no obvious column header directly above it, get merged incorrectly into the columns on their left side. This causes all the cells that are either missing or have gained the $-signs into their contents (cells on both sides of the dollar signs), to evaluate having imprecise and unrecalled *proto-links*. This is due to the way that the *proto-links* are compared; by comparing the connected cells contents. A more sophisticated comparison methods could maybe only penalize either the cell that has gained extra content or the one that has lost it, but not both. This particular table also features the possibility to keep the $-signs in their separate columns, because now, only the top and bottom amounts for each year in each column will have the $-signs associated with them.



(Amounts in Thousands)	2011			
	Appropriated Funds	Donated Funds	Earmarked Funds	Total
Obligated	$ 11,684,724	$ 45,845	$ 266,999	$ 11,997,568
Unobligated Available	13,409	52,242	59,959	125,610
Unobligated Unavailable	102,227	93	970	103,290
Less: Budgetary Non-FBWT	-	(51,380)	-	(51,380)
Total FBWT	$ 11,800,360	$ 46,800	$ 327,928	$ 12,175,088

(Amounts in Thousands)	2010			
	Appropriated Funds	Donated Funds	Earmarked Funds	Total
Obligated	$ 11,974,777	$ 34,174	$ 287,886	$ 12,296,837
Unobligated Available	12,451	45,625	47,026	105,102
Unobligated Unavailable	98,304	4	3,124	101,432
Less: Budgetary Non-FBWT	-	(44,683)	-	(44,683)
Total FBWT	$ 12,085,532	$ 35,120	$ 338,036	$ 12,458,688

Figure 38: Structure recognition score P/ R: ***0.55/ 0.58.*** *Highlighted (red outline) areas show problematic "$"-sign columns. The algorithm merges these columns erroneously to the left. The header cell "2011" is not extended to span all of the 4 correct columns.* *Source: US-data set [13].*



	Year																
Pesticide	**1979**	**1980**	**1981**	**1982**	**1983**	**1984**	**1985**	**1986**	**1987**	**1988**	**1989**	**1990**	**1991**	**1992**	**1993**	**1994**	**1995**
	Millions of lbs. a.i.																
Herbicides	610	622	631	620	573	634	611	590	532	557	567	564	546	554	527	583	556
Insecticides	255	228	218	210	204	197	193	188	152	161	154	148	141	143	130	138	137
Fungicides	124	122	122	117	115	109	110	109	100	99	98	91	86	81	80	79	77
Other Conv.	155	149	152	149	148	145	138	138	133	137	154	173	182	189	192	199	203
Other Chems.	343	321	307	298	287	284	284	278	269	266	251	252	226	246	248	244	249
Total	**1487**	**1442**	**1430**	**1394**	**1327**	**1369**	**1336**	**1303**	**1186**	**1220**	**1224**	**1228**	**1181**	**1213**	**1177**	**1243**	**1222**

Figure 39: Structure recognition score P/ R: ***0.64/ 0.40.*** *Highlighted (red outline) header row features words (year numbers) in too close proximity to be recognized as separate columns, which leads to poor performance. Vertical separator lines do not extend into the header.* *Source: US-data set [13].*

Figure 39 shows a table with very narrow spacing between the words in the header. For an algorithm, a row with many narrowly spaced words (the year numbers), that run the entire length of the table, it is very easy to mistake it for a table title text row. Perhaps a strategy of looking at obvious column breaks defined by the table body data with its separator lines and good alignment of the body elements could serve as a clue for keeping the header words in separate cells.

7.2 Future plans

There are numerous ways that the project could still be improved. Here is a list of a few ideas, that would make the software application more useful for interested users:

- Improvements
  - Further improvement of the table structure recognition algorithm.
  - Further improvement of the table detection algorithm.
  - Further improvement of the edge detection algorithm.
  - Speed improvement of the edge detection algorithm.
  - Evaluation and fine-tuning of the performance of the header prediction algorithm.
- Expansions
  - Expansion of the *GUI* part of the project to serve as a ground truth definition tool to create new data sets that include table titles, headers and captions.
  - Installable application package.
  - Support for image files and text recognition.
  - Online server implementation.
  - An easy to set up, comprehensive testing suite for table extraction algorithms.
  - Expansion of the *API* to allow for use of external edge detection algorithm.

The developed algorithms are also submitted to a competition for table detection and structure recognition: International Conference on Document Analysis and Recognition (ICDAR) 2013* Table Competition [13]. The results of the competition were not yet available by the last revision date of this thesis.

* http://www.icdar2013.org/

REFERENCES

[1] E. Riloff. *Information extraction as a stepping stone toward story understanding. In Ram, A., Moorman, K., eds.: Understanding Language Understanding: Computational Models of Reading*. MIT Press, 1999.

[2] A. Bagga. *A Short Course on Information Extraction: A Proposal*. Department of Computer Science, Duke University, Durham, 1998.

[3] X. Wang. *Tabular Abstraction, Editing and Formatting*. PhD thesis, University of Waterloo, 1996.

[4] M. Hurst. *The Interpretation of Tables in Texts*. PhD thesis, University of Edinburgh, 2000.

[5] Adobe Systems Incorporated, *PDF Reference, Sixth edition, version 1.23*. 2006.
Retrieved 2013-02-24: http://www.adobe.com/devnet/acrobat/pdfs/pdf_reference_1-7.pdf

[6] ISO 32000-1:2008, *Document management — Portable document format — Part 1: PDF 1.7*. Iso.org., 2008.
Retrieved 2013-02-24: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=51502

[7] E. Orion. *PDF 1.7 is approved as ISO 32000*. The Inquirer, 2007.
Retrieved 2013-02-24: http://www.theinquirer.net/gb/inquirer/news/2007/12/05/pdf-approved-iso-32000

[8] Adobe Systems Incorporated, Public Patent License, ISO 32000-1: 2008 – PDF 1.7
Retrieved 2013-02-24: http://www.adobe.com/pdf/pdfs/ISO32000-1PublicPatentLicense.pdf

[9] Adobe Extensible Metadata Platform (XMP) becomes an ISO standard
Retrieved 2013-02-24: http://www.iso.org/iso/home/news_index/news_archive/news.htm?refid=Ref1525

[10] Poppler PDF rendering library
Retrieved 2013-02-24: http://poppler.freedesktop.org/

[11] Xpdf, an open source viewer for Portable Document Format (PDF) files.
Retrieved 2013-02-24: http://www.foolabs.com/xpdf/

[12] M. Ruffolo and E. Oro. *PDF-TREX dataset*
Retrieved 2013-02-24: http://staff.icar.cnr.it/ruffolo/files/PDF-TREX-Dataset.zip

[13] Website: http://www.tamirhassan.com/ (last-retrieved: 13-03-2013)

[14] M. Göbel, T. Hassan, E. Oro, G. Orsi. *A Methodology for Evaluating Algorithms for Table Understanding in PDF Documents.* DocEng, 2012.
Retrieved 2013-02-24: http://www.orsigiorgio.net/wp-content/papercite-data/pdf/gho*12.pdf

[15] M. Hurst. *A constraint-based approach to table structure derivation.* In Proc. of International Conference on Document Analysis and Recognition, pages 911–915, 2003.

[16] A. C. e Silva. *Metrics for evaluating performance in document analysis: application to tables*. International Journal on Document Analysis and Recognition, 14(1):101–109, 2011.

[17] M. Ruffolo and E. Oro. *PDF-TREX: An approach for recognizing and extracting tables from PDF documents*. In Proc. Of ICDAR 2009, pages 906–910, 2009.

[18] T. Hassan. *Towards a Common Evaluation Strategy for Table Structure Recognition Algorithms*, DocEng2010, September 21–24, 2010.

[19] B. Yıldız. *Information Extraction – Utilizing Table Patterns*. Master's thesis, Vienna University of Technology, 2004.

[20] B. Yildiz, K. Kaiser, and S. Miksch. *pdf2table: A method to extract table information from PDF files*. In Proc. of Indian Intl. Conf. on AI 2005, pages 1773–1785, 2005.

[21] Pdf2Table website: http://ieg.ifs.tuwien.ac.at/projects/pdf2table/ (last-retrieved: 21-03-2013)

APPENDIX A – STRUCTURE RECOGNITION RESULTS – EU-DATA SET

```
EU-data set results

PDF document: 'eu-001.pdf'
PRECISION:  495/ 495 = 1.0000      100.00%
RECALL:     495/ 495 = 1.0000      100.00%
PDF document: 'eu-002.pdf'
PRECISION:  140/ 143 = 0.9583       97.90%
RECALL:     140/ 146 = 0.9231       95.89%
PDF document: 'eu-003.pdf'
PRECISION:  295/ 295 = 1.0000      100.00%
RECALL:     295/ 295 = 1.0000      100.00%
PDF document: 'eu-004.pdf'
PRECISION:  146/ 146 = 1.0000      100.00%
RECALL:     146/ 146 = 1.0000      100.00%
PDF document: 'eu-005.pdf'
PRECISION:  497/ 499 = 0.9697       99.60%
RECALL:     497/ 499 = 0.9697       99.60%
PDF document: 'eu-006.pdf'
PRECISION:   34/  34 = 1.0000      100.00%
RECALL:      34/  37 = 0.9189       91.89%
PDF document: 'eu-007.pdf'
PRECISION:  292/ 296 = 0.9841       98.65%
RECALL:     292/ 296 = 0.9841       98.65%
PDF document: 'eu-008.pdf'
PRECISION:   67/  67 = 1.0000      100.00%
RECALL:      67/  67 = 1.0000      100.00%
PDF document: 'eu-009.pdf'
PRECISION:   78/  78 = 1.0000      100.00%
RECALL:      78/  84 = 0.9286       92.86%
PDF document: 'eu-010.pdf'
PRECISION:   45/  45 = 1.0000      100.00%
RECALL:      45/  45 = 1.0000      100.00%
PDF document: 'eu-011.pdf'
PRECISION:  380/ 380 = 1.0000      100.00%
RECALL:     380/ 380 = 1.0000      100.00%
PDF document: 'eu-012.pdf'
PRECISION:  204/ 206 = 0.9903       99.03%
RECALL:     204/ 206 = 0.9903       99.03%
PDF document: 'eu-013.pdf'
PRECISION:  273/ 273 = 1.0000      100.00%
RECALL:     273/ 273 = 1.0000      100.00%
PDF document: 'eu-014.pdf'
PRECISION:  127/ 131 = 0.9695       96.95%
RECALL:     127/ 129 = 0.9845       98.45%
PDF document: 'eu-015.pdf'
PRECISION:  160/ 164 = 0.9836       97.56%
RECALL:     160/ 166 = 0.9758       96.39%
PDF document: 'eu-016.pdf'
PRECISION:   70/  70 = 1.0000      100.00%
RECALL:      70/  70 = 1.0000      100.00%
```

```
PDF document: 'eu-017.pdf'
PRECISION:  513/ 513 = 1.0000      100.00%
RECALL:     513/ 513 = 1.0000      100.00%
PDF document: 'eu-018.pdf'
PRECISION:  297/ 299 = 0.9933       99.33%
RECALL:     297/ 299 = 0.9933       99.33%
PDF document: 'eu-019.pdf'
PRECISION:  101/ 103 = 0.9806       98.06%
RECALL:     101/ 103 = 0.9806       98.06%
PDF document: 'eu-020.pdf'
PRECISION: 1148/1148 = 1.0000      100.00%
RECALL:    1148/1148 = 1.0000      100.00%
PDF document: 'eu-021.pdf'
PRECISION:  329/ 421 = 0.7765       78.15%
RECALL:     329/ 346 = 0.9219       95.09%
PDF document: 'eu-022.pdf'
PRECISION:  109/ 134 = 0.8786       81.34%
RECALL:     109/ 139 = 0.8611       78.42%
PDF document: 'eu-023.pdf'
PRECISION:  534/ 539 = 0.9907       99.07%
RECALL:     534/ 539 = 0.9907       99.07%
PDF document: 'eu-024.pdf'
PRECISION:  138/ 138 = 1.0000      100.00%
RECALL:     138/ 138 = 1.0000      100.00%
PDF document: 'eu-025.pdf'
PRECISION:   74/  74 = 1.0000      100.00%
RECALL:      74/  74 = 1.0000      100.00%
PDF document: 'eu-026.pdf'
PRECISION:   25/  25 = 1.0000      100.00%
RECALL:      25/  25 = 1.0000      100.00%
PDF document: 'eu-027.pdf'
PRECISION:  134/ 134 = 1.0000      100.00%
RECALL:     134/ 134 = 1.0000      100.00%
PDF document: 'eu-028.pdf'
PRECISION:  328/ 328 = 1.0000      100.00%
RECALL:     328/ 328 = 1.0000      100.00%
PDF document: 'eu-029.pdf'
PRECISION:  648/ 794 = 0.8156       81.61%
RECALL:     648/ 803 = 0.8068       80.70%
PDF document: 'eu-030.pdf'
PRECISION: 1068/1181 = 0.9041       90.43%
RECALL:    1068/1198 = 0.8907       89.15%
PDF document: 'eu-031.pdf'
PRECISION:  499/ 569 = 0.8770       87.70%
RECALL:     499/ 566 = 0.8816       88.16%
PDF document: 'eu-032.pdf'
PRECISION:  370/ 459 = 0.8075       80.61%
RECALL:     370/ 465 = 0.7839       79.57%
PDF document: 'eu-033.pdf'
PRECISION:  112/ 112 = 1.0000      100.00%
RECALL:     112/ 112 = 1.0000      100.00%
PDF document: 'eu-034.pdf'
PRECISION:   22/  22 = 1.0000      100.00%
RECALL:      22/  22 = 1.0000      100.00%


Processed 34 files

PRECISION AVG: 0.967042     (9752/10315 relations, in 59 tables in 34 files)
RECALL AVG:    0.964284     (9752/10286 relations, in 59 tables in 34 files)

F-SCORE: 0.965661     (96.57%)
```

APPENDIX B – STRUCTURE RECOGNITION RESULTS – US-DATA SET

US-data set results

```
PDF document: 'us-001.pdf'
PRECISION:  142/ 146 = 0.9726       97.26%
RECALL:     142/ 142 = 1.0000      100.00%
PDF document: 'us-002.pdf'
PRECISION:  148/ 148 = 1.0000      100.00%
RECALL:     148/ 148 = 1.0000      100.00%
PDF document: 'us-003.pdf'
PRECISION:   59/  59 = 1.0000      100.00%
RECALL:      59/  59 = 1.0000      100.00%
PDF document: 'us-004.pdf'
PRECISION:   95/  95 = 1.0000      100.00%
RECALL:      95/  95 = 1.0000      100.00%
PDF document: 'us-005.pdf'
PRECISION:  198/ 203 = 0.9811       97.54%
RECALL:     198/ 203 = 0.9811       97.54%
PDF document: 'us-006.pdf'
PRECISION:   67/ 113 = 0.5929       59.29%
RECALL:      67/  94 = 0.7128       71.28%
PDF document: 'us-007.pdf'
PRECISION:   46/  52 = 0.8846       88.46%
RECALL:      46/  50 = 0.9200       92.00%
PDF document: 'us-008.pdf'
PRECISION:   68/  85 = 0.8000       80.00%
RECALL:      68/  77 = 0.8831       88.31%
PDF document: 'us-009.pdf'
PRECISION:   17/  17 = 1.0000      100.00%
RECALL:      17/  17 = 1.0000      100.00%
PDF document: 'us-010.pdf'
PRECISION:    9/  11 = 0.8182       81.82%
RECALL:       9/  22 = 0.4091       40.91%
PDF document: 'us-011.pdf'
PRECISION:   27/  27 = 1.0000      100.00%
RECALL:      27/  27 = 1.0000      100.00%
PDF document: 'us-012.pdf'
PRECISION:  149/ 213 = 0.8090       69.95%
RECALL:     149/ 208 = 0.7412       71.63%
PDF document: 'us-013.pdf'
PRECISION:  149/ 149 = 1.0000      100.00%
RECALL:     149/ 149 = 1.0000      100.00%
PDF document: 'us-014.pdf'
PRECISION:  174/ 193 = 0.9016       90.16%
RECALL:     174/ 189 = 0.9206       92.06%
PDF document: 'us-015.pdf'
PRECISION:  170/ 341 = 0.4985       49.85%
RECALL:     170/ 187 = 0.9091       90.91%
PDF document: 'us-016.pdf'
PRECISION:   85/ 123 = 0.6911       69.11%
RECALL:      85/ 104 = 0.8173       81.73%
PDF document: 'us-017.pdf'
```

```
PRECISION:   47/  47 = 1.0000      100.00%
RECALL:      47/  47 = 1.0000      100.00%
PDF document: 'us-018.pdf'
PRECISION:   40/  40 = 1.0000      100.00%
RECALL:      40/  40 = 1.0000      100.00%
PDF document: 'us-019.pdf'
PRECISION:  112/ 183 = 0.5948       61.20%
RECALL:     112/ 176 = 0.6150       63.64%
PDF document: 'us-020.pdf'
PRECISION:  123/ 223 = 0.5441       55.16%
RECALL:     123/ 213 = 0.5677       57.75%
PDF document: 'us-021.pdf'
PRECISION:  125/ 135 = 0.9259       92.59%
RECALL:     125/ 125 = 1.0000      100.00%
PDF document: 'us-022.pdf'
PRECISION:  173/ 173 = 1.0000      100.00%
RECALL:     173/ 173 = 1.0000      100.00%
PDF document: 'us-023.pdf'
PRECISION:   37/  37 = 1.0000      100.00%
RECALL:      37/  37 = 1.0000      100.00%
PDF document: 'us-024.pdf'
PRECISION:   28/  30 = 0.9333       93.33%
RECALL:      28/  29 = 0.9655       96.55%
PDF document: 'us-025.pdf'
PRECISION:   70/ 110 = 0.6364       63.64%
RECALL:      70/ 174 = 0.4023       40.23%


Processed 25 files

PRECISION AVG: 0.863361     (2358/2953 relations, in 33 tables in 25 files)
RECALL AVG:    0.873792     (2358/2785 relations, in 34 tables in 25 files)

F-SCORE: 0.868545     (86.85%)
```

APPENDIX C – TABLE DETECTION RESULTS – EU-DATA SET

```
PDF document: 'eu-001.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-002.pdf'
PURITY:       2/3 = 0.6667       66.67%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'eu-003.pdf'
PURITY:       5/6 = 0.8333       83.33%
COMPLETENESS: 1/5 = 0.2000       20.00%
PDF document: 'eu-004.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-005.pdf'
PURITY:       2/2 = 1.0000      100.00%
COMPLETENESS: 2/3 = 0.6667       66.67%
PDF document: 'eu-006.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-007.pdf'
PURITY:       3/3 = 1.0000      100.00%
COMPLETENESS: 1/3 = 0.3333       33.33%
PDF document: 'eu-008.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-009.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-010.pdf'
PURITY:       1/3 = 0.3333       33.33%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-011.pdf'
PURITY:       3/3 = 1.0000      100.00%
COMPLETENESS: 3/3 = 1.0000      100.00%
PDF document: 'eu-012.pdf'
PURITY:       2/4 = 0.5000       50.00%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'eu-013.pdf'
PURITY:       1/2 = 0.5000       50.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-014.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-015.pdf'
PURITY:       2/2 = 1.0000      100.00%
COMPLETENESS: 2/2 = 1.0000      100.00%
PDF document: 'eu-016.pdf'
PURITY:       0/0 = 0.0000        null
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-017.pdf'
PURITY:       2/4 = 0.5000       50.00%
```

```
COMPLETENESS: 2/2 = 1.0000      100.00%
PDF document: 'eu-018.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-019.pdf'
PURITY:       1/2 = 0.5000       50.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-020.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-021.pdf'
PURITY:       5/5 = 1.0000      100.00%
COMPLETENESS: 5/5 = 1.0000      100.00%
PDF document: 'eu-022.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/2 = 0.5000       50.00%
PDF document: 'eu-023.pdf'
PURITY:       1/3 = 0.3333       33.33%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-024.pdf'
PURITY:       2/7 = 0.2857       28.57%
COMPLETENESS: 2/2 = 1.0000      100.00%
PDF document: 'eu-025.pdf'
PURITY:       1/5 = 0.2000       20.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-026.pdf'
PURITY:       1/4 = 0.2500       25.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-027.pdf'
PURITY:       1/3 = 0.3333       33.33%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-028.pdf'
PURITY:       2/2 = 1.0000      100.00%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'eu-029.pdf'
PURITY:       2/2 = 1.0000      100.00%
COMPLETENESS: 0/3 = 0.0000        0.00%
PDF document: 'eu-030.pdf'
PURITY:       2/3 = 0.6667       66.67%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'eu-031.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'eu-032.pdf'
PURITY:       2/2 = 1.0000      100.00%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'eu-033.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'eu-034.pdf'
PURITY:       0/0 = 0.0000        null
COMPLETENESS: 0/1 = 0.0000        0.00%


Processed 34 files

COMPLETENESS AVG: 0.491176     (29/59 tables, in 34 files)
PURITY AVG:       0.778199     (54/81 tables, in 32 files)

CPF: 0.602238        (60.22%)
```

APPENDIX D – TABLE DETECTION RESULTS – US-DATA SET

```
PDF document: 'us-001.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-002.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-003.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'us-004.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-005.pdf'
PURITY:       2/3 = 0.6667       66.67%
COMPLETENESS: 1/2 = 0.5000       50.00%
PDF document: 'us-006.pdf'
PURITY:       1/2 = 0.5000       50.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'us-007.pdf'
PURITY:       0/0 = 0.0000        null
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'us-008.pdf'
PURITY:       0/2 = 0.0000        0.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-009.pdf'
PURITY:       1/2 = 0.5000       50.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-010.pdf'
PURITY:       1/4 = 0.2500       25.00%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'us-011.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-012.pdf'
PURITY:       1/3 = 0.3333       33.33%
COMPLETENESS: 2/4 = 0.5000       50.00%
PDF document: 'us-013.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-014.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-015.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-016.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-017.pdf'
PURITY:       1/2 = 0.5000       50.00%
```

```
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-018.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-019.pdf'
PURITY:       2/4 = 0.5000       50.00%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'us-020.pdf'
PURITY:       2/3 = 0.6667       66.67%
COMPLETENESS: 0/2 = 0.0000        0.00%
PDF document: 'us-021.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-022.pdf'
PURITY:       1/2 = 0.5000       50.00%
COMPLETENESS: 2/2 = 1.0000      100.00%
PDF document: 'us-023.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 1/1 = 1.0000      100.00%
PDF document: 'us-024.pdf'
PURITY:       1/3 = 0.3333       33.33%
COMPLETENESS: 0/1 = 0.0000        0.00%
PDF document: 'us-025.pdf'
PURITY:       1/1 = 1.0000      100.00%
COMPLETENESS: 0/1 = 0.0000        0.00%


Processed 25 files

COMPLETENESS AVG: 0.640000     (19/34 tables, in 25 files)
PURITY AVG:       0.739583     (26/43 tables, in 24 files)

CPF: 0.686198        (68.62%)
```