



De-obfuscation of published scientific data: information
extraction using optical character recognition and heuristic
techniques

Submitted in partial fulfilment of the requirements for the degree of

BACHELOR OF SCIENCE (HONOURS)

of Rhodes University

Luke Nicholas Darlow

Grahamstown, South Africa

November 2013

Abstract

Bioinformatics is a fast moving field which is yielding rich data and interesting findings. These interesting findings need to be expressed in a reproducible fashion. Reusability is a point of major importance in scientific research. New areas of research have evolved which make specific and intensive use of computational techniques. To stay true to the characteristic of reusability, scientific publications need to provide sufficient supplementary data and code. Existing formal data definitions are not universal standards. Although it is commonplace to make supplementary data available, the document formats used are often not geared toward reusability. A particularly difficult format to reuse is the Portable Document Format (PDF) as it was never designed for this purpose.

A proof of concept system for the automatic extraction of bioinformatics focused tabular data from spreadsheets and PDF documents is designed and implemented. Image analysis and heuristic techniques are used to determine the table dimensions of tables found in PDFs. Optical character recognition (OCR) is considered as a novel approach as it sidesteps the inner workings of the PDF. OCR is thus used to extract the information from PDF documents and is shown to be a less suitable yet viable method for extracting tabular supplementary data. Improvements of the techniques involved and possible changes to algorithms are suggested.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System (1998 version, valid through 2013) :

I.4.0 [General]: Image processing software

I.7.5 [Document Capture]: Optical character recognition (OCR)

General-Terms: Supplementary data extraction

Acknowledgements

I would like to acknowledge my father for his contribution. Without his help and desire to see me through my education, I would not be what I am today.

I would like to acknowledge my mother for her constant prayers, thoughts, and endless empathy.

Owing to their considerable unbiased input and adept editing, I would like to acknowledge Mrs Jeanne Cyrus and Ms Piette Cyrus. I would also like to thank Piette for her continuous support and patience.

I would like to acknowledge my supervisor, Prof. Philip Machanick, for his openness, direction, and grace in dealing with a worried student.

I acknowledge Mr Greg Pennefather for his argumentative nature and, thus, his constructive criticism.

I would like to acknowledge Telkom, Tellabs, Stortech, Genband, Easttel, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

Contents

1	Introduction	1
1.1	Problem statement	1
1.2	Research goals	2
1.3	Thesis overview	3
2	Literature Survey	4
2.1	What is Biology to a Computer Scientist?	5
2.2	Bioinformatics - what is it?	7
2.3	Input formats	9
2.3.1	BED	10
2.3.2	FASTA format	11
2.4	The MEME Suite	13
2.4.1	What is the MEME Suite?	13
2.4.2	MEME	14
2.4.3	MEME input	15

2.4.4	Other tools in the MEME Suite	16
2.4.4.1	MAST	16
2.4.4.2	FIMO	16
2.4.4.3	DREME	16
2.4.5	Installation and use procedure	17
2.5	Additional tools	18
2.5.1	Galaxy	18
2.5.2	RSAT	20
2.6	Reproducible research	21
2.7	PDF	24
2.7.1	pyPDF	26
2.7.2	PDFMiner	26
2.8	Optical character recognition	27
2.8.1	Tesseract	29
2.8.2	OCROPUS	31
2.9	Information extraction	33
2.9.1	TableSeer	33
2.9.2	VeryPDF	35
2.10	Summary	35

3	System Design	36
3.1	Scraping	37
3.2	PDF information extraction	39
3.3	Spreadsheet information extraction	43
3.4	User interface and user involvement	44
3.5	Repeatable approach	47
4	Implementation	48
4.1	Scraper component	48
4.2	PDF information extraction algorithms	51
4.2.1	OpenCV and data structures	51
4.2.2	Determining cell dimensions	51
4.2.3	Fixing cells and helping Tesseract along	55
4.2.3.1	Fixing rows	55
4.2.3.2	Tesseract and single characters	56
4.2.4	Fuzzy string matching	58
4.3	Keeping track	59
4.4	User interface	60
4.5	How testing will occur	61
5	Findings and Discussion	63
5.1	Scrapy	63

5.2	OCR accuracy	64
5.3	Fuzzy matching improvement	69
5.4	Cell division effectiveness	71
5.5	User interface	75
5.6	Repeatability	75
6	Conclusions	76
6.1	Summary	76
6.2	Future work	78

List of Figures

2.1	Comparison of a section of the Mitochondrial chromosome from the UCSC browser (top track) and the ENSEMBLE browser (bottom track). Differences are emphasised in bold italic.	9
2.2	An example of a simple BED file.	10
2.3	Examples of the FASTA format (MEME, n.d.).	11
2.4	Genomic coordinates in a excel spreadsheet (above) and a PDF (below)(Ramagopalan <i>et al.</i> , 2010).	12
2.5	A visually descriptive representation of the MEME Suite outlining the use and workflow of the tools involved (Bailey <i>et al.</i> , 2009).	13
2.6	A visual representation of a motif called a sequence logo. The height of the letter indicates its probability of occurrence and is scaled for information content. (Bailey <i>et al.</i> , 2009).	14
2.7	An example MEME submission form (Bailey <i>et al.</i> , 2009).	15
2.8	A Galaxy page example. Top right shows input taxonomy data. Middle right shows a history panel. Bottom right shows a workflow (Goecks <i>et al.</i> , 2010).	20
2.9	An example of a “cut and paste” direct conversion of a PDF to HTML using PDFMiner. This highlighted sections are html div elements and the arrows represent the arrangement of these div elements (Shinyama, 2011).	27

2.10	Tesseract quadratic spline line fitting. The top line is straight. The other lines are the fitted lines and are parallel to each other. On careful inspection it can be seen these lines are curves (Smith, 2007a).	29
2.11	Tesseract character classification on a broken ‘O’. The character is segmented into many unit sized features and matched against a prototype with fewer, non unit sized features (Smith, 2007b).	31
3.1	An overview of the proof of concept scraping module. URLs are extracted by the custom built Spider and written to a JSON file. This JSON file is parsed separately. The scraping script ties these components together.	38
3.2	An overview of the PDF table extraction module.	40
3.3	A pixel count for each horizontal line in an arbitrary PDF page. The lighter lines indicate the pixel count. The dark and straight lines are choices for column delimiting.	41
3.4	A simple input form example for the first section of the user input – simple but sufficient. This should accept Microsoft Excel and PDF files. The DPI field refers to dots per inch which is a measure of the quality of the image recovered for OCR. The page field determines which page of the PDF document should be used.	45
3.5	A proposed example for the working output for the user interface. The top figure demonstrates the full row, full column, and single cell selection capabilities. The bottom figure demonstrates the use of the ‘Choose Sequence Name Column’ (in green), ‘Choose Sequence Start Column’ (in yellow), and ‘Choose Sequence End Column’ (in red) buttons. These will be implemented using HTML and the JavaScript jQuery library.	46

4.1	A JSON output example. The links are either relative to the domain being scraped (in this case biomedcentral.com) or static (such as the “Adobe Acrobat Reader” link).	50
4.2	An example of counting horizontal and vertical pixels. The overlaid graph is the pixel count. The lines are inserted based on this pixel count information. Figure a is the horizontal pixel count: counting from left to right for each image y-coordinate. Figure b is the vertical pixel count: counting from top to bottom for each image x-coordinate. Figure c is an added example of what the pixel count looks like when considering non-tabular data.	53
4.3	An example of a heading post analysis – table headings are not uncommon and, although they have little to do with the contained tabular data, do provide useful information. Note the lines cutting letters apart.	55
4.4	Outlining the two concatenation options available to us. The left side shows concatenation per row. The right side shows a rotated cell with concatenation per cell.	56
4.5	Tesseract OCR’s command line options. Option seven is convenient for our use.	57
5.1	An example of constructed data	64
5.2	An example extract of real world style bioinformatics supplementary table data.	64
5.3	Figure a highlights the difference in image quality as the DPI is raised. The difference in text construction is most notable at lower quality and is most noticeable when considering the ‘8’. Figure b shows the difference in text construction at 110 DPI and 250 DPI respectively: at 110 DPI the two eights can easily be confused with ‘BB’.	65

5.4	Accuracy versus Quality - constructed data.	66
5.5	Accuracy versus Quality - Real world data	66
5.6	Overall table OCR analysis time versus quality for constructed data. This corresponds with the data used in Figure 5.4.	68
5.7	Overall table OCR analysis time versus quality for real world data. This corresponds with the data used in Figure 5.5.	68
5.8	Accuracy versus Quality - with and without fuzzy matching (real world data).	69
5.9	Accuracy versus (high) quality - with and without fuzzy matching (real world data).	70
5.10	Horizontal and Vertical pixel count timings – constructed data.	71
5.11	Horizontal and Vertical pixel count timings – real world data.	72
5.12	Pixel threshold ratio versus division counts for tables without noise. The solid lines represent the column division count found by our algorithm. The fuzzy line centers over the required output, with diminishing accuracy and effectiveness as it fades.	73
5.13	Pixel threshold ratio versus division counts for tables with a noise.	74
5.14	A vertical pixel count (like that as shown in Figure 4.2) scaled and overlaid on the pixel ratio. The horizontal lines exemplify how different pixel ratios can cause different algorithmic nuances to come into play.	74

Chapter 1

Introduction

1.1 Problem statement

Fundamental characteristics of research are reusability and repeatability. These principles should ensure scope for published research to be improved and distilled, expanded upon, or debunked. Conformation to the ideals of reusability and repetition has become difficult in research areas involving high levels of computation (such as bioinformatics). The current paradigm for scientific publications has evolved much slower than the relevance of computation in research. There often exists a gap between what is published and what is necessary to repeat the experiments and computation involved. Effective means for publishing relevant supplementary data and code do exist but are rarely used. Scientific journals do not have strict requirements for publishing supplementary data take the formats of this data. Instead, most published supplementary data takes the form of documents not intended for this purpose. Bioinformatics is a data intensive field which results in many scientific publications being accompanied by supplementary data which is less than suitable for reuse.

Manually finding and extracting information from supplementary data is a process found to be needing automation. Manual extraction of tabular data (which is how most useful supplementary data is assumed to be published in bioinformatics publications) from Portable

Document Format (PDF) Documents is haphazard and unreliable. Other formats, such as spreadsheets or databases, provide easier reuse of supplementary data. Developing and exploring a fresh approach to reusing published scientific data is needed up until a paradigm shift in publishing supersedes this need.

1.2 Research goals

The problem is approached from an explorative, proof of concept perspective. There exists current research in extracting content from various file formats but none of these is focused on tabular supplementary data. When considering PDFs, existing extraction techniques rely on the back-end code and representation of the file to extract information. Owing to its pedagogical significance, OCR will be used to extract information.

The use of OCR presents challenges in accuracy. It is our goal to determine how to address these challenges and whether using OCR is a viable alternative to conventional information extraction techniques. Images require considerably more memory to store and work with – the nuances of working with images will be explored.

A web scraping component will be developed to determine the viability of automatically obtaining supplementary data. Our goal is to explore a web scraping framework and determine the viability of future development.

A tracking component will be developed. Keeping track of the state of the system and the parameters involved, is important in order to avoid unnecessary repetition of time consuming stages in the system pipeline when parameters need changing. System parameters and state information will be tracked. It is our goal to show this tracking is relevant and can be used to shortcut the pipeline process.

A heuristic table dimension finding algorithm will be developed. Along with exploring the effectiveness of dividing tabular data using this algorithm, finding possible extensions, improvements, and alterations to this algorithm are our goals.

Peripheral additions and utility functionality will be developed to determine the viability of automating the process of information extraction. Full automation is an unrealistic goal; user intervention must be considered and a simple user interface will be developed to provide scope for it. We will use bioinformatics supplementary data to show the relevance of this system, of which a final output goal will be a BED file which can be used by bioinformatics tool chains.

1.3 Thesis overview

This thesis begins with a literature survey in Chapter 2. An overview of what bioinformatics is and how it is relevant to a computer scientist is presented. The MEME suite is outlined with specific attention given to the flow of data and input formats. The history and structure of the Portable Document Format follows. Optical character recognition (OCR) is explained and current open source OCR engines are discussed. Relevant research in information extraction and reproducible research (with particular focus on PDF documents) are reviewed.

Chapter 3 is a presentation of the design of a proof of concept system. The system is segmented into its constituent parts and described part by part. There is a focus on the PDF information component as it presents the highest level of complexity. Chapter 4 outlines the implementation of the system. Algorithms for cell division and fuzzy matching output tables are described. The implementation of the user interface and a reusability component are described. Future implementation options are also given.

The result summary and discussion of results constitute Chapter 5. Relevant components of the system are tested. Various accuracy and efficiency tests are carried out and discussed (particularly concerning the contributing components to PDF information extraction).

Conclusions and future work follows in Chapter 6.

Chapter 2

Literature Survey

In this chapter context will be given as to what biology is and what it means to a computer scientist. The emergence and significance of bioinformatics will then be discussed. Many researchers involved in bioinformatics have a strong biology background and thus lack the skill to do what computer scientists should be able to do (with varying effort) in order to extract data from supplements. With regard to sequences and genomic coordinates, an outline of common input formats for bioinformatics tools exemplifies the existing gap between what is published (since there are no real standards) and what is required by said tools. A brief summary of some of these tools and what they require as input is given to show the necessity of bridging this gap (a gap that limits the fundamental characteristic of scientific publication: reusability). The importance of reusability and repetition in scientific publications is discussed along with proposed solutions (such as the Galaxy web service's approach to solving this issue).

Current means of publishing and current supplementary data publishing norms are not geared toward reusability. Instead, common documents such as spreadsheets and PDFs are used to publish supplementary data. The PDF is explained and shown to be a document format designed for universal readability, not reusability. Some current information extraction techniques are reviewed. Considering our novel approach to PDF information ex-

traction consisting of image processing and OCR, current OCR techniques and open source OCR engines will be explored.

2.1 What is Biology to a Computer Scientist?

DNA (deoxyribonucleic acid) is a molecule consisting of a combination of paired bases bonded to a sugar phosphate.

There are four possible bases, namely:

- Adenine (A)
- Thymine (T)
- Cytosine (C)
- Guanine (G)

Most DNA is found in the nucleus of the cell. However, there is a small portion residing in the mitochondrial DNA. This mitochondrial DNA is found within the mitochondria of cells. Mitochondria are involved in energy production of cells (Mandal, 2013).

Approximately 99% of DNA is identical between two human beings. The remaining 1% of dissimilar DNA is what is used in paternity tests and the like (Benoît, 2005).

It is DNA that is the determining factor of heredity and consequently how an organism self-replicates in the monumental fashion resulting in the flow and balance of life. Most of DNA processing has been concerned with DNA in terms of protein construction. This is the main coding function of DNA, as understood presently. It cannot decisively be said this is the only function.

DNA consists of two anti-parallel strands comprising the four bases. These strands are bonded to alternating sugars and run in opposite directions. It is possible to determine one

strand's composition by performing a reverse complement on the opposite strand. Performing a reverse complement involves reversing a given DNA sequence and then swapping bases T and A and bases C and G.

Another concept worth understanding is that of the **transcription factor**. A transcription factor is a protein acting on DNA to influence the flow of genetic information. This influences the production of RNA and therefore the proteins that make up an organism.

A transcription factor binding site (TFBS) is a section of DNA where a transcription factor binds. *Cis*-regulatory modules are sequences of DNA known to include a number of TFBS's related to gene expression and are particularly relevant owing to their functional component.

Bayat (2002, p. 1018) defines bioinformatics as :

“The application of tools of computation and analysis to the capture and interpretation of biological data.”

Proteins are comprised of a twenty letter alphabet. Biologists perform a process known as sequencing to analyse DNA and/or protein to determine the order of bases or amino acids. This sequence is linear and (since the representation is simply using an alphabet of letters) results in a large string. This is the point at which computer science becomes useful. The problem at hand is essentially one of string searching, known more specifically as the Approximate Common String (ACS) problem. The biological sequences are the strings for the ACS problem (Bailey, 1995).

Computer scientists are able to develop the tools needed to do the large scale processing and management of raw data (produced by molecular biologists) into a manageable, useful and user-friendly form. This hastens the process of understanding in appropriate fields. Not only are the techniques and algorithms important aspects of this, but so is the interpretation of data and the implementation of tools that make this information more available and user-friendly.

2.2 Bioinformatics - what is it?

Computer science is a rapidly growing discipline naturally lending itself to avenues of thought applicable to multiple other disciplines. The fundamental principles of abstraction and automation associated with computational thinking are inherently powerful devices for the solving of otherwise overwhelmingly complex problems. Problems involving massive data sets, pattern matching, large statistical analysis and multi-database lookups can be managed effectively and efficiently using tools developed within the realm of computer science. A solid core understanding of manipulating data and the processes and algorithms involved (in both the abstracted, higher level and the lower level) are crucial in elegantly using a computer to solve this sort of problem.

The style of problem able to be solved thanks to computer science suits the style of problem present in many other sciences. Molecular biology is an example of a science marrying very well with computer science. Molecular biology concerns itself with the structure, characteristics and chemical processes involved in living cells. These chemical processes result in the formation and self-creation of an organism on a higher level (considering the “one-way street” flow widely believed to be followed in molecular construction; genes are transcribed into RNA and RNA is translated into DNA). The discovery of DNA by James Watson and Francis Crick published in a scientific paper in 1953 resulted in a new area of research with absolutely profound implications (Crick & Watson, 1954).

The key to the link between Biology and Computer Science is the digital nature of the DNA molecule. These building blocks of life are the alphabet living organisms use to self-create. The presence of this, along with the precise mapping between the amino acid building blocks of protein, means computer science, in theory (considering the noise present through the biochemical processes involved), possesses the tools and consequently the potential for tools to store, analyze and process DNA.

The emergence of informational biology and bioinformatics follows naturally as a result of the tools, techniques and processing ability needed from computer science in this quickly

developing area of research. The Human Genome Project was completed in 2003 and became a major contributor to the swift advance of bioinformatics (in terms of popularity and tools). The goals of the Human Genome Project included (amongst others) the development and enhancement of sequencing technology. These can be attributed to sparking much research into novel, state-of-the-art techniques involved in bioinformatics.

Bioinformatics is a rapidly advancing area of research and requires the combined effort of members of different schools of study in order not only to gather and process the vast wealth of information constantly collected, but also to do so, in an efficient manner.

When the question of functionality and use is considered, access to and easy use of bioinformatics tools are paramount. The MEME (Multiple EM for Motif Elicitation) Suite (Bailey *et al.*, 2009) tool set for motif discovery and searching is a stand-alone or web server based tool set designed for use by anybody wishing to do so. The Galaxy project acknowledges computation as an indispensable tool in life science research and provides an open platform for accessible and reproducible web-based tools (Goecks *et al.*, 2010). RSAT (Regulatory Sequence Analysis Tools) is yet another example of a software suite tending the needs arising in bioinformatics (Thomas-Chollier *et al.*, 2008).

These tools, constantly under scrutiny and consequently being improved, all have their flaws and space for growth. A common trend in the use of said tools, is the gap of understanding between computer scientists (the designers and implementers of these tools) and biologists (the users). Whether considering the quirks of software use (command line parameters, for example) or the need for a specific format input (essentially a computer science ‘speed bump’ to be overcome), there is still much space for improvement. The following sections will outline some of these tools and the input formats associated with them.

2.3 Input formats

There are commonly two manners of representing genomic data. The first of these to consider is that of genomic coordinates. These are coordinates referring to specific positions in corresponding sequences (a chromosome in the human genome, for instance). Genomic coordinates are resolved into sequences and are commonly used because they require much less information to be stored.

It is important to have knowledge of the correct version of genome (and is presumed when using coordinates) used in order for these coordinates to resolve into the correct sequences. There are two prominent genome browsers currently being used, namely the UCSC genome browser (Kent *et al.*, 2002) and the ENSEMBL genome browser (Flicek *et al.*, 2013). Variations of genomes between these browsers is attributed to different choice with regard to keeping up to date with sequenced DNA; the ENSEMBL browser keeps consistently (with the warranted trade off of having occasional mistakes released) up to date versions of genomes while the UCSC browser waits for full, confirmed versions of genomes to release. To illustrate this, we compared the Mitochondrial chromosome published by each of these genome browsers (Figure 2.1). These two sequences were 99.7% similar. This may seem adequately similar but this dissimilarity can yield very different results in some experiments.

CATAAAAACCCAA <i>T</i> CCACATCAAA - ACCCCCTCCCATGCTTACAAG
CATAAAAACCCAA <i>C</i> CCACATCAAA <i>C</i> CCCCCCCCCATGCTTACAAG
TACAGCAATCAACC <i>C</i> TCAACTATCACACAAG <i>C</i> CATTTACCGTACATA
TACAGCAATCAACC <i>T</i> TCAACTATCACACAAG <i>T</i> CATTTACCGTACATA
ATTACAGTCAAATCCCTTCTCGTCCCATGGATGACCCCCTCAGAT
ATTACAGTCAAATCCCTTCTCGTCCCATGGATGACCCCCTCAGAT

Figure 2.1: Comparison of a section of the Mitochondrial chromosome from the UCSC browser (top track) and the ENSEMBLE browser (bottom track). Differences are emphasised in bold italic.

Another complication inherent in using coordinates is that of coordinate offsets. Genome coordinates can either be zero- or one- based. The difference is best explained by example: if the start coordinate is zero and the end coordinate is 100, the span of bases is 0-99 for zero-based systems and 1-99 for one-based systems. There is no standard to how this is done. Both the ENSEMBLE and UCSC genome browsers consider the input coordinates to be one-based but researchers may choose which style to publish their coordinate data.

2.3.1 BED

The BED file format is a format for publishing genomic coordinates and is zero-based. There are three compulsory fields in BED files: the chromosome name, the sequence start position and the sequence end position (Figure 2.2).

```
chr7 127471196 127472363
chr7 127472363 127473530
chr7 127473530 127474697
chr7 127474697 127475864
chr7 127475864 127477031
chr7 127477031 127478198
chr7 127478198 127479365
chr7 127479365 127480532
chr7 127480532 127481699
```

Figure 2.2: An example of a simple BED file.

BEDTools

BEDTools is a set of tools for fast and flexible genomic analysis. It consists of many component modules which are runnable as command line tools in a UNIX environment. Of particular interest to us is the tool called *fastaFromBed*. *fastaFromBed* is able to extract FASTA sequences with inputs of BED file coordinates and a FASTA file (typically a large genome file).

2.3.2 FASTA format

As apposed to coordinates, the other manner of representing genomic data is that of sequence data. Although many formats for representing this sequence data are available for use with bioinformatics tools, the FASTA format has become the popular *de facto* standard. As mentioned, other formats exist and if a specific tool does not accept a format, tools exist for format conversion¹.

The FASTA format was first developed for a software package called FASTP (Lipman & Pearson, 1985) and its name has no real significance. The format starts with a header line. The header line starts with a ‘>’ character followed by a unique name (truncated to twenty four characters, if necessary) without spaces. An optional comment/description can be given after a space. This line is then followed by one or more sequence lines (each recommended to be under eighty characters in length). Spaces, blanks, and case are ignored in the sequence lines (Figure 2.3). The accepted alphabets are dependent on which type of analysis is being performed (DNA or protein, for example).

```
>crab_bovin ALPHA
ACTGGGGTCGGCTAGGCTCGAGATATATATTTTCGCGATCTCT
CTATAGGGGCTCTAGAGCTCTCGAGAGAGAGAGCTCTCGAG
>crab_anapl BETA
ATTTGCTGATATAGCTCGCTCGATCGCTATATAGGCTCTAGA
```

Figure 2.3: Examples of the FASTA format (MEME, n.d.).

There is no formal specification of how research data should be published, and specifically how it should be published in the field of bioinformatics (where research input data is plentiful). If authors choose to publish supplementary data, for example, they may choose to store the sequence data or, more commonly, the genomic coordinates in Microsoft Excel or Word documents, PDFs, plain text or TSV files. Each of these has various levels of difficulty associated with the extraction of useful, reliable data.

¹<http://genome.nci.nih.gov/tools/reformat.html>

G87 fx						
	A	B	C	D	E	F
1	Supplementary Table 1. VDR binding intervals defined following calcitriol stimulation					
2						
3	Chromosome	Start	End	Peak value	Average value	
4	chr1	1700188	1700701	71	49	
5	chr1	1701512	1702022	92	55	
6	chr1	2480734	2481127	168	101	
7	chr1	3583145	3583780	115	67	
8	chr1	3806764	3807410	145	97	
9	chr1	8379671	8380465	99	67	
10	chr1	9408885	9409400	148	81	
11	chr1	9925706	9926330	77	48	
12	chr1	10914805	10915186	96	69	
13	chr1	11890423	11891603	252	129	
14	chr1	11892204	11892807	139	96	

Trait	Chr	Start	End
Adiposity related	chr1	845668	1145668
Adiposity related	chr1	72387703	72735027
Adiposity related	chr1	96566581	96959670
Adiposity related	chr1	176030141	176330141
Adiposity related	chr1	217560846	217860846
Adiposity related	chr1	233516751	233816751
Adiposity related	chr10	16189956	16489956
Adiposity related	chr10	37872102	38172102
Adiposity related	chr10	71852445	72303015

Figure 2.4: Genomic coordinates in a excel spreadsheet (above) and a PDF (below)(Ramagopalan *et al.*, 2010).

A non-standard means of storing data becomes, in itself, an unintentional obfuscation of this data. See Figure 2.4 and consider the complications of reusing the data published PDF and Microsoft Excel formats compared to that of data published in a standardised format (such as the BED file format). The purpose of extracting this data is for preparation and use within a toolset (such as the MEME Suite).

2.4 The MEME Suite

2.4.1 What is the MEME Suite?

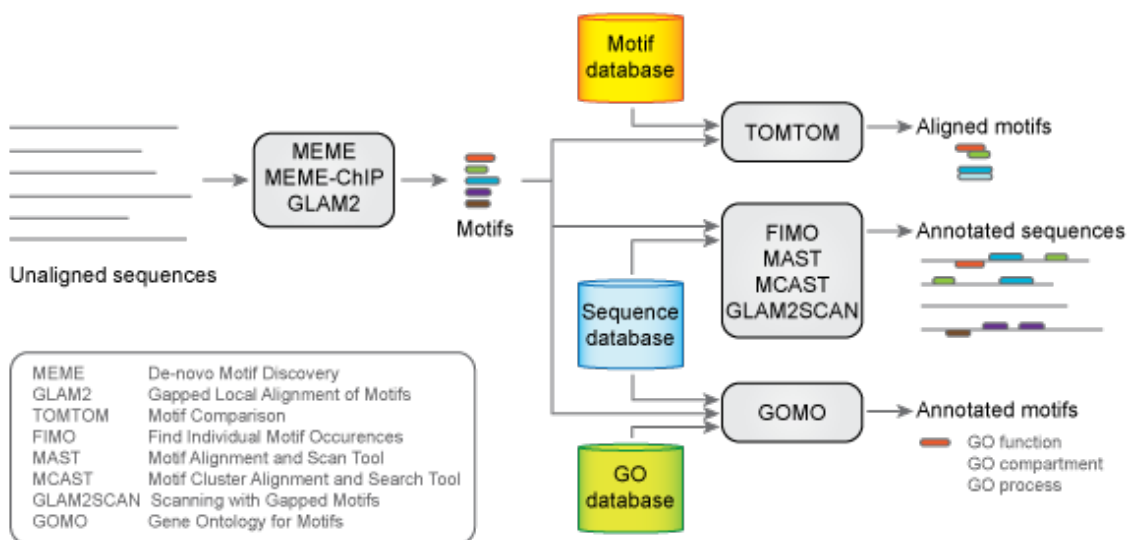


Figure 2.5: A visually descriptive representation of the MEME Suite outlining the use and workflow of the tools involved (Bailey *et al.*, 2009).

The work by Bailey *et al.* (2009) on the MEME Suite is a collaborated collection of tools for discovering, analyzing, comparing and characterizing sequence motifs in DNA or protein sequences. A motif is a statistically significant recurring pattern within the sequence data and is functionally significant in molecular evolution. This suite of tools can be installed locally or on a web server. The flagship tool in the meme suite is MEME. MEME discovers gapless sequence motifs by searching for statistically relevant motifs (see description below) within the supplied unaligned sequences. Other tools in the suite for motif discovery are DREME and Glam2. A motif (see Figure 2.6) is a letter probability matrix for representing statistical probabilities of bases arranged in a certain manner relative to each other.

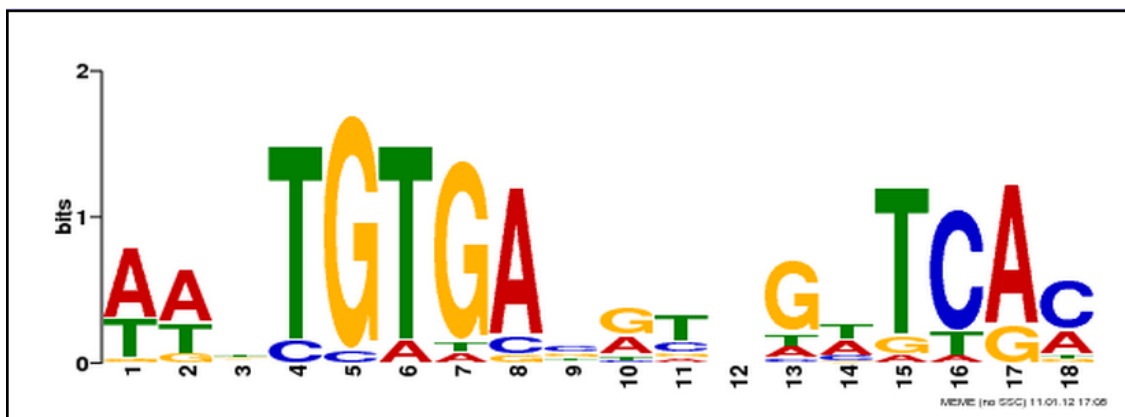


Figure 2.6: A visual representation of a motif called a sequence logo. The height of the letter indicates its probability of occurrence and is scaled for information content. (Bailey *et al.*, 2009).

Tools in the MEME Suite for use in motif search are FIMO, Glam2Scan, MAST, and MCAST. These (along with other, additional tools) will be discussed. Figure 2.5 shows a workflow diagram from the MEME web site depicting how these interrelated tools work together to provide useful information to the user.

2.4.2 MEME

MEME searches for statistically significant motifs in unaligned and related input sequences. It is one of the most widely used bioinformatics tools. The discovery of novel signals within these sequences has many uses in the academic and medical worlds - finding numerous similar motifs in multiple sequences is a good indication these sequences share some biological function (Bailey & Elkan, 1994). The discovery of transcription factor binding sites is but one of many uses for this tool (Bailey *et al.*, 2006). The MEME Suite (running, for example, at meme.nbcr.net/meme/) tries to make things simpler for a user by employing a ‘point and click’ user interface – this still does not bridge the gap of finding and reusing data from publications.

2.4.3 MEME input

The data submission form for MEME is shown in Figure 2.7. Along with an email address and the file containing the input sequences, command line parameters for the MEME algorithm can be set here. This specific tool does not require the input to be in the FASTA format (with other formats dying out), although later tools do. Although multiple formats are accepted as input, none of these formats are coordinate formats. Most research data is published as coordinates for the sake of space.

Data Submission Form

Required

Your e-mail address:

Re-enter e-mail address:

Please enter the **sequences** which you believe share one or more motifs. The sequences may contain no more than **60000 characters** total total in any of a large number of **formats**.

Enter the **name of a file** containing the sequences here:
 No file chosen

or
the **actual sequences** here (Sample Protein Input Sequences):

How do you think the occurrences of a single motif are **distributed** among the sequences?

☐ One per sequence
☒ Zero or one per sequence
☐ Any number of repetitions

MEME will find the optimum **width** of each motif within the limits you specify here:

Minimum width (≥ 2)
 Maximum width (≤ 300)
 Maximum number of motifs to find

Options

Description of your sequences:

MEME will find the optimum **number of sites** for each motif within the limits you specify here:

Minimum sites (≥ 2)
 Maximum sites (≤ 600)

☐ **Shuffle** sequence letters

Perform **discriminative** motif discovery – Enter the name of a file containing '**negative sequences**':
 No file chosen

Enter the name of a file containing a **background Markov model**:
 No file chosen

DNA-ONLY OPTIONS
(Ignored for protein searches)

☐ Search given **strand** only
☐ Look for **palindromes** only

Figure 2.7: An example MEME submission form (Bailey *et al.*, 2009).

2.4.4 Other tools in the MEME Suite

2.4.4.1 MAST

MAST (Motif Alignment and Search Tool) takes as input motif(s) represented as position-dependent scoring matrices along with a specified database or databases for comparison. These input motifs can originate from databases or directly from the user but more usefully can be direct output from MEME. Input must be ungapped (suitable for output from MEME). The MAST algorithm takes each input motif and finds the best matching position in the sequence and calculates p-value statistics based on the length of the sequence (Bailey & Gribskov, 1998).

2.4.4.2 FIMO

FIMO (Find Individual Motif Occurrences) is another tool for scanning sequences for the occurrence of one or more motifs. FIMO can scan DNA or protein sequence databases. Although it is not the first of its kind, it outdoes similar tools in many respects. For example, RSAT (Thomas-Chollier *et al.*, 2008) fails with regard to scanning proteins.

Input into FIMO consists of one or more motifs (thus seamlessly integrating with MEME) and either user-supplied sequences or a database to search. FIMO proceeds to search for statistically significant occurrences of each motif. It does this by scoring each position in the searched sequence with a log-likelihood ratio. It also makes use of dynamic false discovery rates. Output from FIMO consists of statistically ranked motif occurrences (Grant *et al.*, 2011).

2.4.4.3 DREME

A problem facing most motif discovery tools is the difficulty of searching large datasets. DREME is a novel motif search tool tailored for ChIP-seq data (chromatin immunoprecip-

itation followed by high throughput sequencing) experiments as these yield an extremely large number of predictions for TFBS's. What makes the DREME algorithm unique is its linear scalability with regard to large datasets (large sets of short sequences), allowing DREME to discover primary and cofactor ('helper') motifs otherwise overlooked by other tools (which selectively only use some of the data yielded by ChIP-seq experiments).

DREME is limited to finding motifs of length up to eight base pairs wide. This means it may miss information rich wider motifs. Although this is a downfall, the motivation for this choice is rooted in DREME not being a replacement tool for motif discovery but rather a complimentary tool. It is able to discover quickly overlooked motifs.

DREME takes as its input two sets of sequences (a positive and negative set - if no negative set is given the positive set is shuffled to provide this contrasting set) and a significance threshold which is used in the algorithm. Output consists of the discovered motifs, their logos, their reverse compliment logos, statistical significance data (allowing for biologists to distinguish between statistical artefacts and actual functional motifs), an option to download, and an option to submit the output for further analysis (Bailey, 2011; Machanick & Bailey, 2011) .

2.4.5 Installation and use procedure

The MEME Suite is a collection of command line tools with the option of a web service. Any user is able to install this web service. Anybody with some knowledge of programming can use the MEME tools in a pipeline fashion. Galaxy (discussed in Section 2.5.1) uses a visual workflow environment to allow for web service users to do this in a high level manner and would be a suitable direction for the MEME Suite to head in in order to achieve better usability. A number of complications relating to the tying together of the MEME tools arise at this point. A number of command line parameters must be provided for the tools to function optimally (see below for an example thereof).

```
meme crp0.s -dna -mod zoops -nmotifs 3 -revcomp
```

The web service has an overlaying interface for this (see Figure 2.7) but for complete flexibility of use a version must be installed locally. The MEME Suite has extensive documentation regarding this procedure (found at <http://meme.nbcr.net/meme/doc/meme-install.html>) with options for parallel installs, customized installations, and the installation of the web server.

2.5 Additional tools

There exist many additional tools for sequence analysis in the MEME Suite. These range from uses for visualisation to analysis of gapped motifs (and their visualisation) to motif enrichment analysis (SpaMo and CentriMo).

The improvement of tools for use in bioinformatics is a pedagogical venture. The open source nature of most tools show the intent of those who make use of them. This is an environment for research and advance in technology for good. A in depth look at other tools and web services is warranted in this scope – their novel approaches, advantages and downfalls shall be considered.

2.5.1 Galaxy

Galaxy takes a different approach to bridging the gap between computer scientists and biologists. Galaxy places focus on accessibility, reproducibility and (the often underestimated element of) transparency in the context of software development. Many genomic experiments face the issue of reproducibility. Experimental reproducibility is essential for valid scientific research. The complex nature of experiments (with many involved steps, detailed and specific parameters and chaining of computational tools) makes them difficult to reproduce without a framework for doing this. Galaxy proposes a Reproducible Research System in which input, output and representation need to be defined but provide promise for easier and more widespread use of the accessible bioinformatics tools.

Galaxy is a open web based platform for use by those without any, or with little, programming skill (most biologists, for example). This platform provides a mechanism for command line parameter input, tool chaining and visualisation. Galaxy also allows for the installation of other tools. These are required to run on a command line yet have wrappers for graphical interfaces. Developers specify documentation on input and output parameters with galaxy creating a workbench environment for their use. This trades flexibility for user friendliness.

User friendliness is not the crux of the Galaxy’s implementation. Analysis and methods involved are arguably of more importance than simple output figures. Galaxy provides means of output designed for detailed, multi-layered analysis of experiments performed. This output takes the form of Galaxy pages with dynamic workflows and history analysis. See Figure 2.8 for an example of these.

Galaxy makes use of metadata capture on datasets, tools and parameters in an automatic fashion. Reproducibility is also an issue for graphical interfaces, which is why galaxy records the steps a user takes. User descriptions of steps in the history of the experimentation provide a means to annotate and give reasoning for method and parameter choice (amongst other things).

The workflow editor within Galaxy provides a dynamic means of chaining tools and keeping track of this for later use and reproduction.

Galaxy tools can be split into three categories: query operations on datasets, sequence analysis tools, and output display tools. The core of Galaxy does the job of binding all of these in the attempt to fulfil the goals of accessibility, reproducibility and transparency. With approximately five thousand jobs processed daily, Galaxy is a powerful and useful service (Goecks *et al.*, 2010).

Galaxy partially addresses the issue of reusability by allowing storage of genomic coordinates and workflows, but does not provide means of extracting data from supplements.

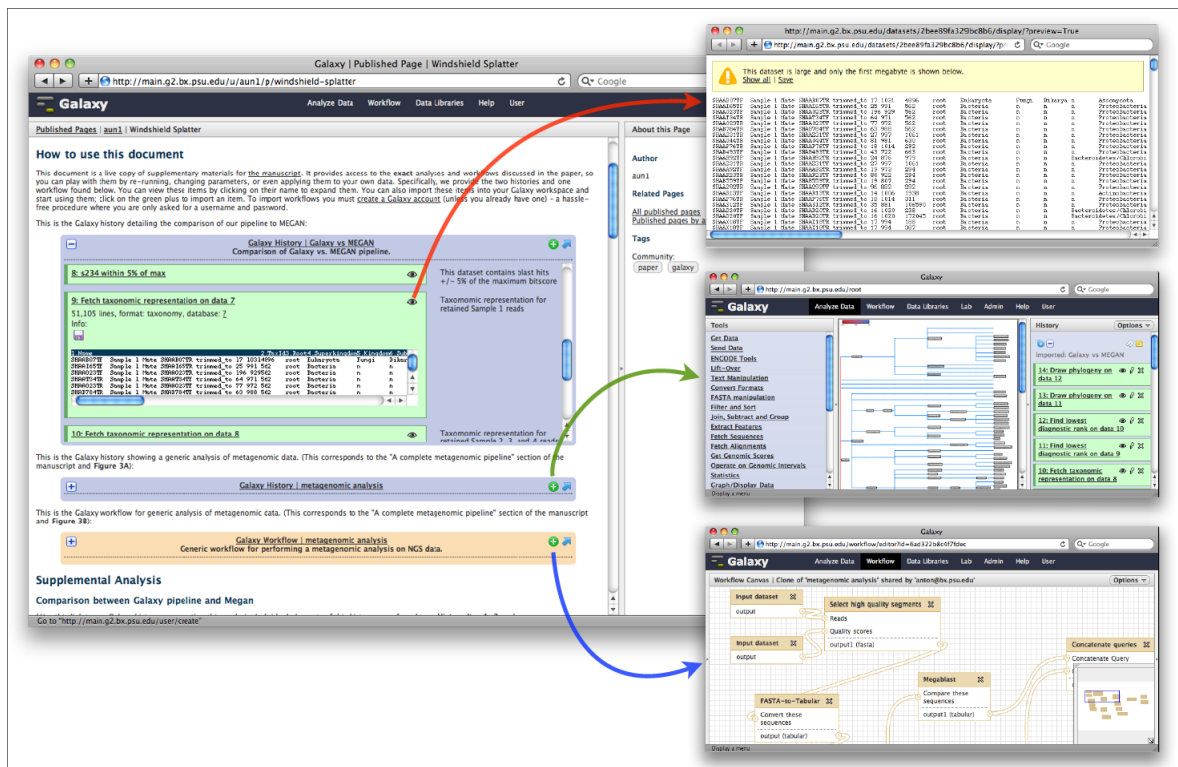


Figure 2.8: A Galaxy page example. Top right shows input taxonomy data. Middle right shows a history panel. Bottom right shows a workflow (Goecks *et al.*, 2010).

2.5.2 RSAT

Regulatory Sequence Analysis Tools (RSAT) is a cluster of sequence analysis discovery and search tools aimed at *cis*-regulatory modules of DNA. *Cis*-regulatory modules are sequences of DNA known to include a number of TFBS's related to gene expression and are particularly relevant owing to their functional component.

RSAT has a series of tools for both pattern discovery and matching. An earlier edition only had the ability to do string matching but the current edition now uses position-specific scoring matrices (similar to MEME) along with a background model for genomic noise. This background model is very important for the correct functioning of the algorithm. RSAT supports over 600 genomes and has pre-computed background models for these.

Without a good background model, much of the RSAT may fail. Making use of these background models, however, allows for random control tests to assess the presence of false positives. RSAT also has drawing facilities and a web service implementation.

The web service allows users to submit input and receive output using a tool installed on the web server but has no interface for chaining tools together. For this, the user needs to have some basic programming skill. A future effort to provide a web interface for tool chaining is needed. As with most bioinformatics tools (implementing a web service worth considering) documentation and demos are available for use of the tools in the suite.

Future efforts to improve RSAT would include increased flexibility with other tools and databases and a means to rope tools together efficiently (Thomas-Chollier *et al.*, 2008).

Bioinformatics is a focal point up until here because it is clearly a research area heavily oriented around data and makes it a rich and useful case study for research data. The tools outlined thus far, form part of an ever-expanding arsenal to be used for effective research. We often speak of research as an almost finite entity, not open to change and evolution after the critical point of ‘publishing’ has been reached. This is truly not the case. In order for research to be fully valuable it should be made as reproducible as possible.

2.6 Reproducible research

One of the main principles of the scientific method is that of reproducibility. Owing to the fact that research only gains true credibility only in the agreeing hands and minds of many (who offer independent verification of observations). No scientific process is complete without peer review. An adequately equipped reader must be able to repeat a published experiment and analysis *caeteris paribus*. Questionable results should be debugged, debunked or confirmed. Interesting findings should be available for testing and to be built upon. Since replication is the purifier and judgement standard for research, a mind-set of reproducibility and repetition is paramount to the scientific process.

Scientific publications should be geared toward repetition. The realm of computationally intense research is that of software packages and data. Thoughtful use of modular software and code yields satisfactory results. This induces a situation whereby the complexity of repeating complicated pipelines for manipulation and analysis of data makes repetition difficult.

In recent years, the world has seen the rise of technology on a massive scale. The increase in availability of technology and, more specifically, computational tools are characteristics of a fresh approach to scientific research. Researchers in the field of bioinformatics, for example, make use of the available tools for analysing large datasets in an automated fashion. While the technology involved in scientific research has taken considerable leaps forward, the convention for publishing this new, computationally focused research, has not changed to keep up with this trend. This results in published research detached from its true environment, its software, and its data. Referring to Claerbout’s philosophy, Buckheit & Donoho (1995, p. 5) state:

“An article about computational science in a scientific publication is not the scholarship itself, it is merely advertising of the scholarship. The actual scholarship is the complete software development environment and that complete set of instructions that generated the figures.”

Mesirov (2010) proposes a Reproducible Research System (RRS). This system is comprised of two parts: the Reproducible Research Environment (RRE) and the Reproducible Research Publisher (RRP). The RRE is where computational work happens and should house the ability to keep track of software and data used and also provide a framework for redistribution. The RRP is a document preparation system and should be able to reproduce readily analysis that extends into the output document itself. The RRP concept is similar to that of a Microsoft Excel worksheet placed within a Microsoft Word document – change the worksheet and the document changes too. Robust code versioning is important in order to allow for computations to stay repeatable.

A user-friendly RRS was created using GenePattern² as the RRE and Microsoft Word as the RRP. The use of a dynamic system such as this, gives researchers the opportunity to become actively involved in the consumption of research.

Goble et al. (2010) took a different approach to solving this problem. They created an online bioinformatics workflow sharing research environment called myExperiment. myExperiment is inspired by websites such as Facebook in that it makes use of and depends upon a social network. Users must register in order to upload workflow information but can browse workflows anonymously. Although – the strength of the social network approach here is realised post registration. The creators of myExperiment recognise the need for a RRS and acknowledge the need for a document production component.

Both myExperiment and a RRS give solutions to the problem of research not being reproducible. What is really needed is the active cooperation of journals. Although it is common to find supplementary data and/or code, there is no fixed format for this and no requirement of its availability. This makes researchers limited in the way they publish findings and thus limits peers’ ability to evaluate these findings.

It should be noted, however, that not all research can be repeated. Time and expense are decisive factors in repeating a study. The Sloan Digital Sky Survey (York *et al.*, 2000) has been continuing for years and would not be a viable candidate for replication. Research techniques may be highly specialised or involve highly specific technologies – in these cases reproducibility becomes limited (Peng, 2011).

These outlying cases do not discount the need for reproducibility and, theoretically, every computation is accompanied by a log of actions taken by the computer. Either code is no longer available or natural language does not provide sufficient detail of the computation. The responsibility of keeping track of computation does not fall into any one party’s hands. Big user interface driven software systems are not geared toward reproducibility and, since old habits die hard, researchers are reluctant to learn how to use new software systems.

²GenePattern is an open source bioinformatics analysis package.

This is not to say there is no research motivated software geared toward reproducibility. The Galaxy web service (Goecks *et al.*, 2010) is designed to make results reproducible through workflow visualisation and tracking. Owing to the rise in popularity of free code publishing websites such as Github (github.com) and Sourceforge (sourceforge.net), making code available has never been easier.

If not for any other reason but for the sake of staying true to the foundational principle of reproducibility of scientific findings, there is a clear need for a paradigm shift in the way researchers (and, more importantly, journals) publish papers in which research is reliant on computation. This will be a slow shift as it is a shift in thinking away from the comfort of how things have always been done. Announcing the result to readers is easily achievable using present conventions, but convincing readers of its correctness requires more than just a paper advertising findings. In the meantime, there is space and scope for effective mechanisms for data extraction for papers published before and outside this paradigm shift (Gentleman, 2004). Supplementary data has no fixed format, although formats such as Microsoft Excel and PDF are fairly commonplace.

2.7 PDF

The PDF dates back to early 1990 – a time when PostScript was fast becoming the standard for describing the printed page. PostScript is a programming language for describing a page to be sent to a printer. A printer, in this case, would house the software and hardware necessary to convert this code into an image. This is called a Raster Image Processor. In the days of PostScript, drawing and page layout were only accomplished by manually typing in code.

The next step in the path toward the PDF is Encapsulated PostScript (EPS) files. An EPS is a single file including the PostScript code and can include a low resolution preview image of the document. This allowed some programs to offer a preview of the document prior to its printing. Another option was to print the PostScript file to disk and view the

result. The PDF might seem (and is often considered to be) a replacement for PostScript and EPS but it is not. Although largely built upon PostScript, PDF is a smart file format capable of storing information about the page behaviour and not only its layout. This includes information such as fonts, images, printing instructions, and so on (Adobe Systems Incorporated, 2013).

PDF has become the standard for exchanging digital documents on the internet. Some printers are even designed to convert all files to PDF before printing. Although a simplification, PDF can be thought of as a PostScript file already interpreted into objects. There are eight types of objects:

1. Null: has a type and value that will always result in inequality with other objects.
2. Boolean: true or false (can be used in arrays and dictionaries).
3. Numeric Objects: integer and Real numbers. Integer objects are within a certain range and centred on zero. Real objects are usually represented in fixed-point and have limited range and precision.
4. String Objects: represented as a series of unsigned integer values.
5. Name Objects: an atomic symbol object; essentially a place holder.
6. Array Objects: a one dimensional sequential arrangement of heterogeneous objects.
7. Dictionary Objects: an associative table of objects. The key must be a Name Object and the value can be any object.
8. Stream Objects: similar to a String Object but unlimited in size and able to be read sequentially (where a String Object must be read in its entirety). Stream Objects can represent large data objects such as images.

Objects in PDFs can be labelled as indirect. An indirect object is assigned a unique object identifier so as to allow other objects to refer to them (in a dictionary, for example) (Adobe Systems Incorporated, 2006).

These objects are displayable on screen as image and text layers and not in code, thus making reverse engineering a displayable PDF file considerably complex. PDFs do not store metadata for entries within a table. Instead, the position of characters on the page are stored. Open source tools exist for extracting information from PDFs – we will outline two of these below.

2.7.1 pyPDF

pyPDF³ is a library built purely in Python and is capable of extracting document information, splitting and merging pages in a document, cropping pages, merging multiple pages into a single page, and encrypting and decrypting PDFs. Since PDF input is as a file stream, pyPDFs ability to work with pure StringIO objects allows it to manipulate PDFs in memory. pyPDF will be used for splitting a PDF into separate pages.

2.7.2 PDFMiner

PDFMiner⁴ is a data extraction tool, written in Python, designed primarily for extracting text from PDF documents. It obtains the exact location of text on a document and is not limited to text alone. It can retrieve font and other document behaviour information. It also includes a PDF converter and an extendable parser which is ideal for non-text related parsing. Owing to PDFMiner’s ability to extract non-text related objects and reconstruct original layouts by automatically grouping text chunks, the need for identifying column and row separating lines (or white space) in a table becomes a possibility (Shinyama, 2011).

The manner in which String Objects are stored within PDFs means that a direct “copy and paste” methodology can easily result in an incorrectly ordered output (see Figure 2.9 for an example).

³<http://pybrary.net/pyPdf/>

⁴<http://www.unixuser.org/~euske/python/pdfminer/>



Figure 2.9: An example of a “cut and paste” direct conversion of a PDF to HTML using PDFMiner. This highlighted sections are html div elements and the arrows represent the arrangement of these div elements (Shinyama, 2011).

Owing to the sporadic nature of directly extracting data stored in PDFs, optical character recognition is a possible alternative.

2.8 Optical character recognition

Optical character recognition (OCR) refers to the process of extracting machine readable characters from input images (usually in the form of scanned documents). OCR can be used to convert books to digital representations (and making them searchable), convert old documents to a digital format for storage and for use by the visually impaired. OCR is still an offline process unlike real time character recognition (which takes the stroke direction and order into account for accuracy) and usually involves multiple passes over a document.

The entire OCR process involves scanning (or other document retrieval), pre-processing, character recognition, layout analysis, and error correction. Pre-processing can include

removing noise in an image (for which libraries such as OpenCV⁵ are suited), converting image to a binary colour layout (i.e. black and white), word and line detection, character spacing and segmentation detection, and scaling of characters.

The actual character detection can and is done in many ways. For simplicity, character recognition is generalised into the techniques of pattern matching and feature matching. Earlier OCR systems relied heavily on pattern matching. Pattern matching attempts to match an unknown possible character to some prototype character. The downfall of this becomes clear when considering various fonts.

When OCR first became popular this was not a problem as fonts were normalised specifically for OCR. Since this has not been the ideal reality for some time, smarter character recognition algorithms for feature detection have become the norm. Not completely dissimilar to pattern recognition, feature recognition recognises the constituent features of a letter and compares these to known letter features. This is, admittedly, a gross underestimation of the technique, but more detail will be given on this when detailing current popular open source OCR systems. Feature matching has the added advantage of weighting invariable characteristics of letters above variable characteristics (Woodford, 2010).

The complexity in recognising characters using OCR will naturally never be completely overcome (with poor quality images and natural handwriting taken into consideration). OCR systems will often employ error correction techniques at multiple stages of recognition. Using a lexicon for possibly occurring words, is not uncommon. With application specific lexicons and language specification, undetermined or uncertain words and/or characters can often be resolved with a fair degree of certainty.

Owing to its rise in popularity and use, commercial OCR systems are used extensively. Open source OCR systems such as Tesseract OCR and OCRopus have also gained momentum and will now be reviewed.

⁵OpenCV is a popular open source image processing library

2.8.1 Tesseract

Tesseract was developed as an open source OCR engine by Hewlett Packard between 1984 and 1994. After its release, it was sent to the 1995 University of Nevada Las Vegas OCR accuracy test in which it showed its mettle when pitted against commercial OCR engines. Tesseract OCR development then went back under development until 2005, at which time it was released for open source⁶.

Owing to the fact that Hewlett Packard owns many products using page layout analysis, there was no need to develop Tesseract's own page layout analysis component. The process involved in Tesseract follows a traditional pipeline approach. Firstly, connected components are analysed to break text into blobs. Blobs are then assigned to words and lines. Text is then categorised as either fixed pitch or proportional (more on that to follow).

The line finding algorithm used by Tesseract assumes page layout analysis is complete with roughly uniform text regions as input. This text is then filtered to remove punctuation, diacritical marks, and noise (these are taken into account at a later stage). The algorithm does not require deskewing of curved text.

Most OCR engines struggle with curved text (such as at the base spine of a scanned book). Tesseract deals with this by making use of quadratic spline⁷ which is reasonably stable but can result in strange discontinuities if multiple spline segments are needed for the function definition (Figure 2.10).

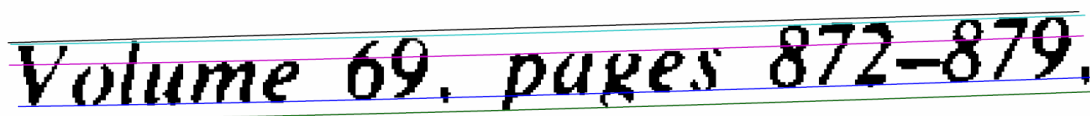


Figure 2.10: Tesseract quadratic spline line fitting. The top line is straight. The other lines are the fitted lines and are parallel to each other. On careful inspection it can be seen these lines are curves (Smith, 2007a).

⁶available at <http://code.google.com/p/tesseract-ocr>

⁷A piecewise defined smooth function

Once line fitting is complete, Tesseract proceeds to classify text as either fixed pitch or proportional. Fixed pitch text is text with a uniform bounding box size for characters and thus makes character recognition a simpler task. Recognising characters in proportional text is a complex task involving chopping and association (when fixed pitch text is found, these steps are skipped). Tesseract circumvents most of these problems by measuring character spacing in a limited region. Spaces close to a certain threshold are only decided upon after word recognition.

Word recognition proceeds on fixed pitch text first. Proportional text characters may very well be joined and must be chopped into characters. This is done by approximating a polygon of the text and identifying concave points as chop point candidates. All possible chops are stored and executed in priority order. Chops improving confidence in the OCR process are kept; chops not improving confidence are undone and given a lower priority order (but possibly used again at a later stage). At the point in which chops are exhausted and confidence in the word is not high enough, broken characters are associated by combinations of the chopped up word. As part of this process, Tesseract is constantly using static character classification and features to classify candidates as characters.

Tesseract assumes features in the unknown need not be exact matches of features in the known. A many-to-few feature match is accomplished here: the unknown is broken down into many uniform, unit size three dimensional features; the known prototype characters are broken down into few three dimensional (i.e. including size) features. The computational cost of calculating distances from the unknown's features to the prototype's features is high but does mean broken characters can be accurately matched (Figure 2.11). This ability to recognise broken characters means that Tesseract required minimal training data as it never needed to be trained for broken characters.

Tesseract does not make extensive use of linguistic analysis. The linguistic module chooses the word amongst scoring categories such as top numeric word. The confidence is calculated as the negative of the normalised distance from prototype to unknown word. The classification of words is done using an adaptive multipass classifier. The adaptive classifier's

first pass over the text strengthens the adaptive classifier. The second pass is for correcting wrongly classified words from the first pass. The final pass resolves spacing in proportional text and finalises the classification (Smith, 2007a).

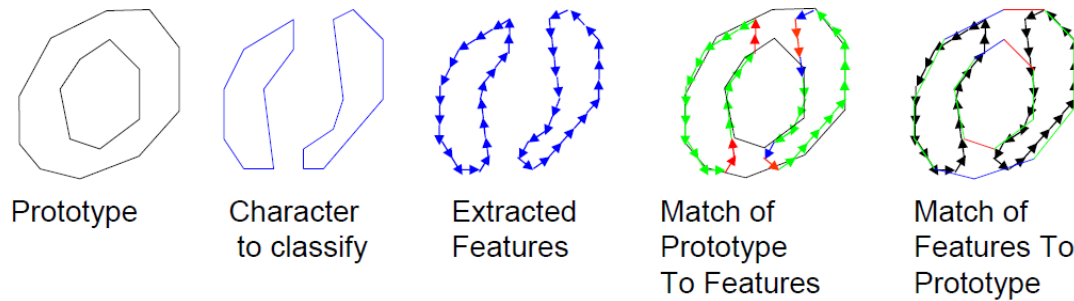


Figure 2.11: Tesseract character classification on a broken ‘O’. The character is segmented into many unit sized features and matched against a prototype with fewer, non unit sized features (Smith, 2007b).

Tesseract is currently behind commercial OCR engines but has been taken over by Google with further work aimed at improving character recognition by using techniques now known to be useful for OCR. A notable feature of Tesseract is its ability to allow defined words – if a certain dictionary of words is expected and so defined, accuracy can be dramatically improved. Owing to the nature of this project, the allowable words can be clearly defined. This makes Tesseract an ideal solution. OCRopus employs additional techniques and is somewhat of a wrapper around Tesseract.

2.8.2 OCRopus

OCRopus is an open source OCR system designed to overcome limitations in OCR software. OCRopus focuses on extensibility with a built-in scripting language and shell and is aimed at the research and commercial communities using an Apache 2 license with few commercial limitations.

In contrast with Tesseract OCR, OCRopus has its own page layout analysis for dividing

an input page into text regions and non-text regions. Text line segmentation and deskewing algorithms are used to prepare the image for OCR.

Once text lines are recognised, they are passed into an OCR engine. Up until recent versions of OCRopus, Tesseract was the default character recognition plugin. The third, and arguably most notable component of OCRopus, is its statistical language modelling. This is done using an open source library called OpenFST (Allauzen *et al.*, 2007; Sproat, 2013). OpenFST is used for creating, manipulating, and searching weighted finite-state transducers. A weighted finite-state transducer is a two tape finite state automata where each state has a input and output and an associated weight. This type of automata is able to accept an input string on its input tape and produce an output string on its output tape. An advantage of using this is the fact that complex language models can be converted into weighted finite-state transducers without the need for language modelling code. It should also be noted these weighted finite-state transducers have multiple functions within an OCR system.

The final result of the OCRopus system is HTML like output (called hOCR). This is aimed at decoupling the output from the system itself. In other words, this output is not dependent on any specific OCR language or script.

OCRopus is currently in its alpha phase and is therefore fairly complex to install and use. It places emphasis on modularity and reusability in many environments but its complexity is an unnecessary addition for the task at hand.

Although the most common use for OCR is digitising written or printed text, having a digital representation of text prior to OCR means noise and inaccuracy issues become redundant. Performing OCR in this scenario seems counter-intuitive, but when text layout and ordering are a problems (as with reverse engineering tabular data in PDFs) then using a different approach as a means of abstraction becomes viable.

It is necessary to build up a toolset for extracting PDF pages as images and for working with those images themselves. pyPDF (Section 2.7.1) is a useful tool for splitting PDFs

into pages while ImageMagick⁸ is used for turning PDF pages into images and OpenCV for processing these images.

The goal of using these tools and techniques is to extract information from documents and reusing this information.

2.9 Information extraction

2.9.1 TableSeer

Liu *et al.* propose the TableSeer system as a solution for searching, crawling for, detecting, extracting, and ranking tables in scientific documents. They argue the point of the lack of metadata for published tables and how this makes tables particularly difficult entities for which to search (even for the likes of public web search engines such as Google). A focus on extracting tabular data from PDFs (due to its inherent complexity in possible layouts and constituent elements) is made with reference to easy extendability into formats such as HTML, Microsoft Word, and the like.

TableSeer consists of a table search engine, a newly defined set of metadata to describe tables, a table detector and extractor (with special focus on a novel ‘box cutting’ algorithm), and a table ranking algorithm (called TableRank).

The search engine is driven by a web crawler designed to crawl open-access digital scientific libraries in search of documents of certain media types.

The next component of TableSeer is its metadata extractor. This accomplishes table extraction by splitting pages into definite boxes and further analysing these boxes. Boxes are put into three categories: boxes with font smaller than the document defined font, boxes with font size equal to the document defined font, and boxes with font larger than the document defined font. Keywords such as ‘table’ and ‘figure’ are kept as a predefined list

⁸ImageMagick is a useful image conversion tool. A python wrapper called PythonMagick (pypi.python.org/pypi/PythonMagick) is convenient for use.

of keywords for box comparison. All boxes with font smaller than the document defined font are analysed for keywords. If a box is found to contain one of these keywords then it becomes a table candidate and proceeds to be analysed for table white space structure. Tables are extracted in no more than three passes (with each pass checking a different set of boxes of the same font size category) of the table detection algorithm. This algorithm is flawed in that it assumes all tables in a document will be of the same font size and thus ignores all boxes of a different font size. Once tables are found, metadata must be associated with them.

The problem of associating meaningful metadata with tabular data is certainly not trivial. TableSeer emphasises the importance of suitable metadata nomenclature for easy searching and recombination. The chosen metadata categories are:

- Table Environment metadata: encapsulating document data (page number, document title, etc.)
- Frame metadata: does the table/box have a surrounding frame? If so, which sides?
- Affiliation metadata: information on the table caption, possible footnote information, and text involved in referencing the table.
- Layout metadata: number of rows, spacing, header information, etc.
- Cell content metadata
- Cell type metadata

Once tables are identified and metadata is associated TableRank has the task of associating relevance of these tables to a query. Owing to the fact that scientific document related search engines (like *Google Scholar*) are simply not geared toward solving this problem, TableSeer considerably out-performs them at table searching. TableSeer is aimed at finding documents by table and not intended as a versatile and robust means of extracting supplementary data from these tables (Liu *et al.*, 2007). There exist commercial pieces of software for extracting table information from various documents.

2.9.2 VeryPDF

VeryPDF (www.verypdf.com) is commercially available software which provides a user with the ability to define table division lines and extract the information to a Microsoft Excel or CSV file. This software proves to be useful and accurate in its table extraction but is not automated and not freely available (with heavy feature and use limitations in the trial version). A user is able to create and destroy boundary boxes and vertical lines. Any further manipulation is completely dependent on the internal algorithms within the system.

2.10 Summary

Bioinformatics is opening many doors into understanding the world around us. Every open door is accompanied by wells of information. Each attempt to process this information is accompanied by complications of keeping track of this distilled content in a sustainable way. A shift of mindset into sustainable and reproducible research methods and styles of publishing, is ideal for the future of scientific publications. In the mean time, however, the need for working with what we have (primarily a non-uniformly populated set of publications and data) is a real and pressing issue. To scrape reliably for, extract, reuse, and keep track of data can aid in steering toward reproducibility.

Chapter 3

System Design

Facing the problem of reusing supplementary data in the field of bioinformatics is not as trivial a task as it may appear to be. The reasoning behind this becomes clear when considering the nature of PDF documents. Attention is given to PDF documents owing to their prevalence and extensive use in the scientific community. There have been general approaches to information extraction from these documents. Broad approaches are unable to pay due attention to the specific details of the task we are attempting to accomplish. For our purposes, a novel broad spectrum approach to extracting tabular data is designed and implemented.

The system is composed of a web scraping component to find possible valid documents, a Microsoft Excel spreadsheet data extraction component, a heuristic, OCR-based table extraction component for PDF documents, a simple user interface, and a parameter and state tracking component. This chapter will outline the system design on a high level. It should be noted our choice of any external components and languages is motivated by the open source nature of said components and languages.

A linear walkthrough of the system will provide a more accurate overall understanding of individual components, as well as the manner in which these components relate to each other.

3.1 Scraping

The first challenge in extracting supplementary data is acquiring the data. The information extraction components of the system assume possession of the document(s). This assumption validates the need to provide aid in finding supplementary data. The challenge of finding specific useful supplementary data is subject to bifurcation with each considered complication and is thus not trivial. With sufficient time, a well-crafted web scraping framework which searches for supplementary data on popular scientific publishing web sites could be designed and implemented. This is essentially not the focus of our work; instead, we chose to make use of an open source web scraping and crawling tool called Scrapy (found at www.scrapy.org).

Scrapy is a very well documented web crawling framework. Scrapy use is broken into three distinguishable parts: an overarching environment (such as a shell) to run the web session in, pre-defined scraping items, and the scraping ‘spider’. Scrapy is used as a proof of concept for extracting PDF and Microsoft Excel document links from a given URL.

The user is prompted (see Sections 3.4 and 4.4) to give a URL upon which the system constructs a file called ‘urls.txt’. The construction of this file is mostly for future expansion, allowing for multiple URLs. Scrapy works conveniently in its own shell environment but is run in a script to launch the Scrapy spider instance. This custom-built Scrapy spider extracts all links with their titles into a JSON¹ file. At this point we have no further need to use Scrapy; instead, we parse the JSON file to extract all possibly interesting links (those with ‘.pdf’ in them, for example). Figure 3.1 shows a visual representation of this process.

Although there is room for improvement in this scraping module (with greater automation and multi-level scraping, for example), it is efficient for simpler cases and serves as a proof of concept. An issue encountered with this design revolves around the dynamic nature of JavaScript and its use in resolving link addresses post-click. This means the links do not

¹JSON (JavaScript Object Notation) files are useful since they provide a notation that is both deserializable and human readable (Figure 4.1)

appear statically within html link tags. To deal with this, the JavaScript associated with said link must be executed in some JavaScript shell environment – this is a complexity to be overcome in future work.

Once the scraping is complete, the user may navigate to the found documents or simply download them. At this point the user will (theoretically) have the document containing supplementary data. This document can be passed onto the system for analysis and extraction. The system will accept PDF and Microsoft Excel documents (with easy extension to other formats).

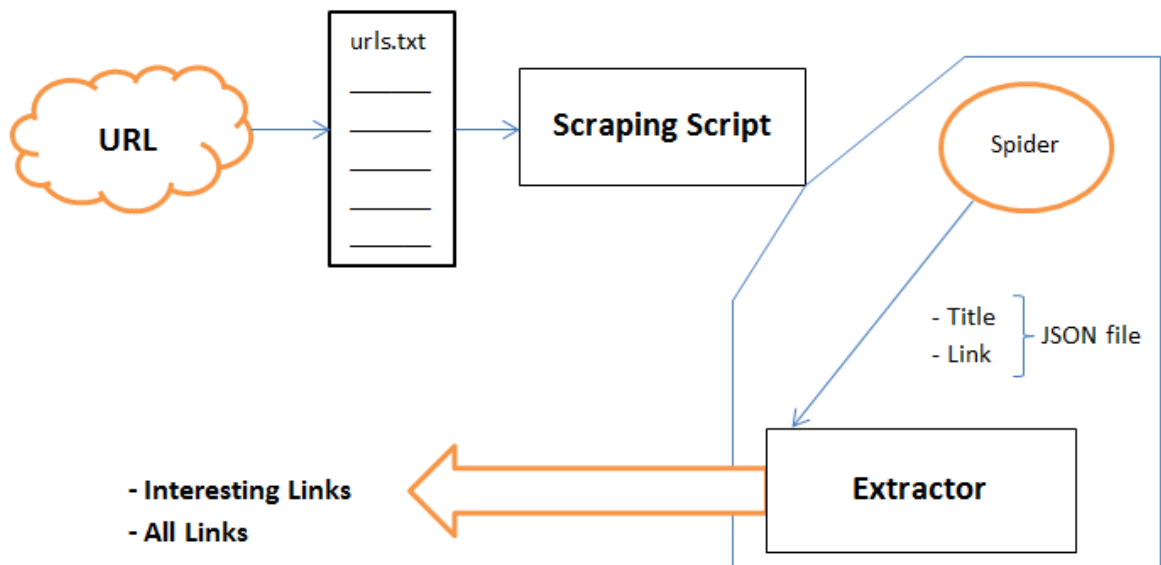


Figure 3.1: An overview of the proof of concept scraping module. URLs are extracted by the custom built Spider and written to a JSON file. This JSON file is parsed separately. The scraping script ties these components together.

3.2 PDF information extraction

PDF information extraction is a major point of complexity in the endeavour of reusing supplementary data. Simple table structures are considered in this development stage. We propose a heuristic model to identifying table cell boundaries in a PDF document and an OCR approach for extracting the text from the table cells. This approach may prove to be computationally heavy, but for the sake of developing a possible universal method for extracting tabular information from published PDFs, this technique must be explored.

Figure 3.2 outlines the table extraction module being designed and implemented. The PDF splitter component is responsible for splitting a multipage PDF document into individual pages and also converting each of those pages into an image (as these images will be used from here on). PyPDF (Section 2.7.1) is used to split the PDF document into its individual pages and PythonMagick is used to convert each individual PDF page into a corresponding image. Another particularly useful mechanism of PyPDF is used at this point: all the text on the page being considered is stored in a simple text file and is also given to Tesseract OCR as its set of user-defined words. The use of this text file will become apparent when considering the fuzzy matching algorithm (Section 4.2.4). It is necessary to trim the excess off the images extracted as images can tend to be computationally heavy to process. A simple edge trimming algorithm is implemented, resulting in a cropped version of the page.

An image has a potential wealth of knowledge associated with it. The trick is to determine which of this knowledge is useful and which is not. Tabular data usually has fairly obvious delimitation (i.e. either lines or white space). An analysis of pixel counts (both horizontally and vertically) is chosen for determining row and column dimensions and coordinates. Figure 3.3 shows a detailed example of column choice. The implementation of this will follow in Section 4.2.2. This information is passed to a text extractor. The interesting components will be detailed in later sections.

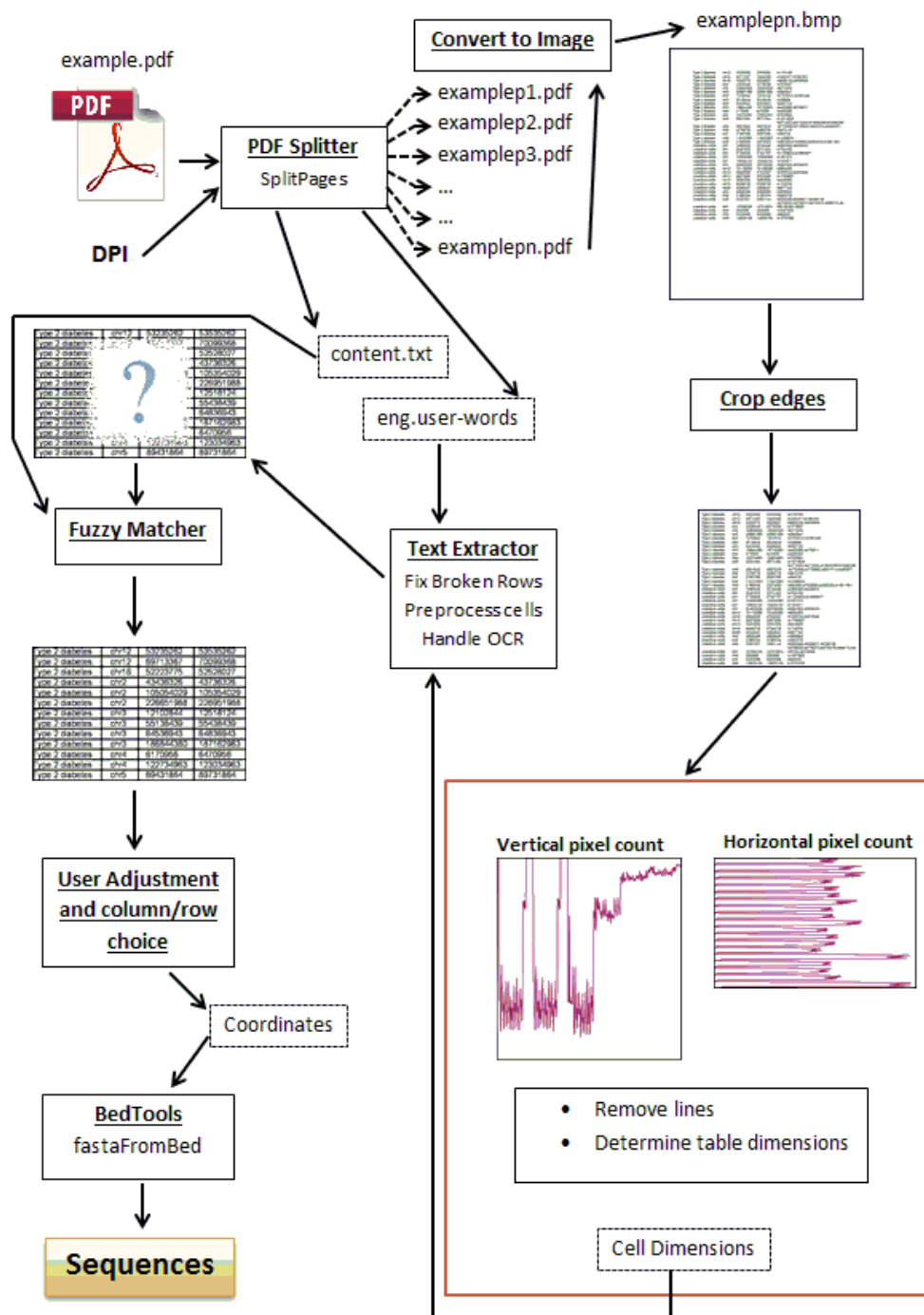


Figure 3.2: An overview of the PDF table extraction module.

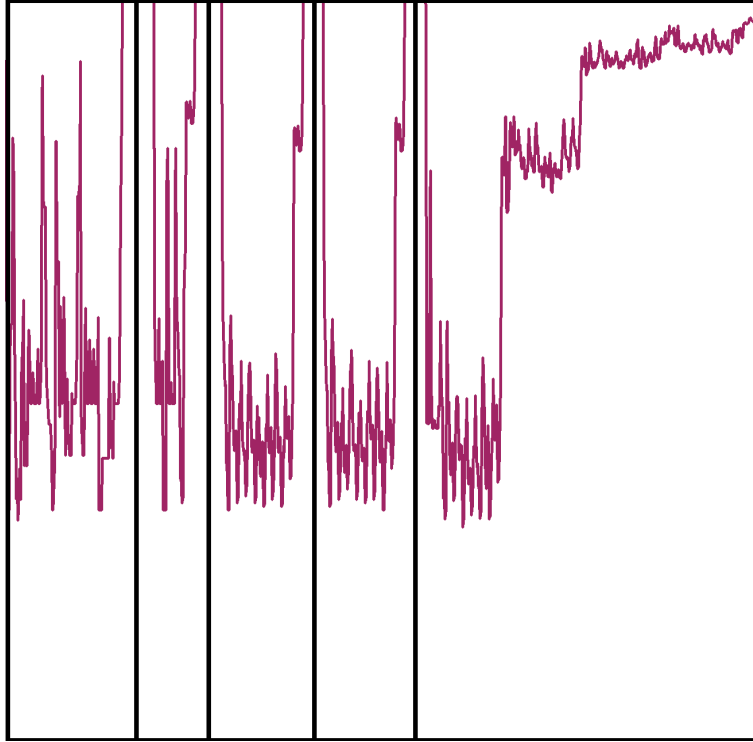


Figure 3.3: A pixel count for each horizontal line in an arbitrary PDF page. The lighter lines indicate the pixel count. The dark and straight lines are choices for column delimiting.

The text extractor is responsible for extracting cells for OCR and performing OCR. The design choice of using OCR when a digital copy of the document already exists, may seem counter-intuitive. OCR accuracy is independant of image quality, therefore having a digital copy of the image at hand allows quality issues to be void. Since we are aiming for a maximum degree of automation, a number of tweaking algorithms are performed, namely:

1. Fixing split rows. To preserve information such as table headings as well as cells spanning multiple columns (considering a flexible layout for tables), each cell is determined to be completed or continuing and dealt with accordingly.
2. Pre-processing individual cells. This involves running a border removal algorithm, as well as, a number of possible OpenCV routines.
3. Dealing with single character/digit cells. Tesseract OCR does not handle these well.

It does have an option for single character input, but fails for single digit input. A check is done to determine the presence of a single character/digit in a cell. If this is proven to be the case, a character is appended to the cell and Tesseract OCR is rerun.

The extracted user words should make Tesseract OCR more accurate. The system is designed to output a text-based table but has the potential to be inaccurate owing to the multipass nature of Tesseract. When only looking at one or a small number of words, the power behind using an adaptive classifier is lost. To regain and actually improve accuracy, an approximate match is sought for each word found through OCR (making use of all the words extracted off the page prior to converting it to an image).

Although there is scope and space for improving and increasing automation, constraints of time and development complexity limit this. The table generated from the PDF page being considered is output. The user is responsible for checking and adjusting cells and choosing columns and rows. A BED file can be generated from three chosen columns (specifically, the name of the chromosome, the start coordinate, and the end coordinate) and passed onto BEDTools in which `fastaFromBed` is used to extract FASTA sequences.

Owing to the nature of this ‘proof of concept’ style work, a vital assumption is made regarding the input page. The page being processed contains a table only (although some text such as a title does not interfere with the algorithm to a large degree). Work by Liu *et al.* on TableSeer (Liu *et al.*, 2007) is capable of determining which block sections of a PDF document are considered to be tables and can be integrated in future work.

Another common format in which supplementary data is published is that of Microsoft Excel spreadsheets and extracting information from them is a less challenging task; we will discuss the design of that module of our system below.

3.3 Spreadsheet information extraction

The strain and manual effort required to access and reuse information contained within a PDF document is considerable and does require developing automated means of doing so. This fact does not prove to be as true for documents in spreadsheet formats. The reason for this is the very nature of these documents is easy access and easy alteration, with a focus on simplicity. PDFs provide a solid means and standard for viewing documents, not for storing scientific data. Extracting data from spreadsheet style documents proves to be an easier task. Many existing libraries exist to this end.

The choice of library is based on programming language and spreadsheet type specifics. Owing to its popularity, working with Microsoft Excel spreadsheets is chosen. When it comes to which library to use, remaining in the realm of a scripted style approach and an open source language, has impact on this decision. For this reason, the xlrd library is chosen for spreadsheet information extraction.

The xlrd library (with a GitHub repository found at github.com/python-excel/xlrd) is a simple Microsoft Excel data reading library² that does not require Microsoft or Excel on the host machine. It is written in Python and represents all strings as Python Unicode Objects (thus compatible with Microsoft Excel 97 and onwards). Its capabilities extend beyond what is needed and become useful for retrieving metadata on the document (number of rows and columns and sheet names, for example) (Simplistix, 2012).

A wrapper is implemented around xlrd for extracting data from spreadsheets. An ideal output from this component should be identical to that from the PDF extraction component. As mentioned previously, this task is done for completeness and for seeking a seamless approach to data extraction in the hopes of reusability.

The goal of full automation is an ideal scenario and a good point to aim for. In reality, however, full automation will not be easily achievable. User involvement via a user interface is necessary and will now be discussed.

²xlwt is an accompanying writing and formatting library

3.4 User interface and user involvement

There exist many means of constructing user interfaces. These range from custom built user interfaces and command line style interfaces or shells, to web service oriented interfaces. Constructing a web service to be deployed in a server environment (of the likes of an Apache server) has the advantage of being able to run on UNIX and Windows platforms (with consideration that all other packages used must have this same property). The ability to make this resource public, makes it more accessible for researchers.

The disadvantages of limited customizability and security concerns are considerable. A web service will adopt the environment of HTML and JavaScript to accomplish a suitable look and feel and usefulness depends upon the capabilities of the designer and implementer. We mention a disadvantage of choosing this option for a user interface is limited customizability. It must be made clear this disadvantage hinges more upon ability to develop this environment from a developers perspective, rather than a limitation in the environment itself (considering the extensive capabilities of HTML and JavaScript). More research must be done in this area but will be reserved for future work.

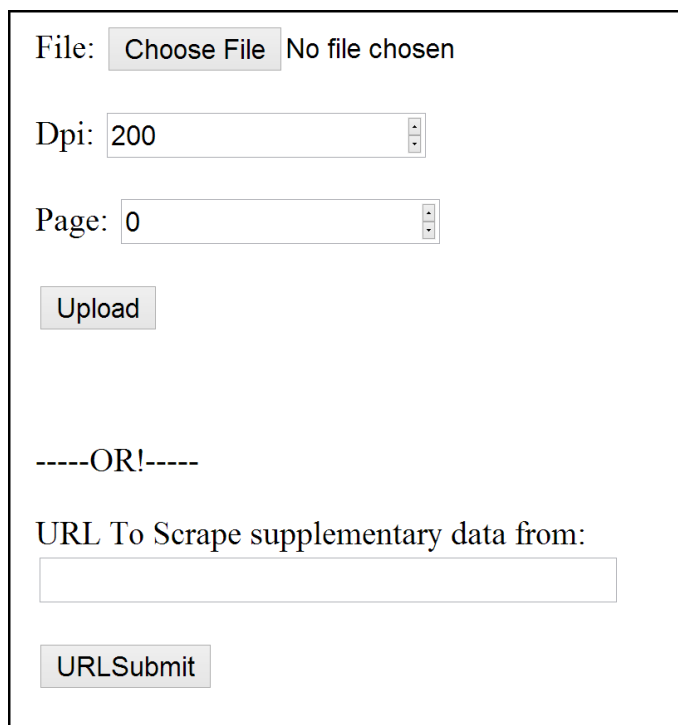
There exists a JavaScript library to aid in creating a good user interface: jQuery (www.jquery.com) has become the *de facto* standard for designing web based user interfaces and will be used for developing our user interface. Another disadvantage of developing a system for use as a web service is security. This is owing to its public facing nature. Dealing with the issue of security will not be a large undertaking because of the simple construction of this user interface and the nature of input data.

The user interface is structured in two parts, namely:

1. an initial ‘upload’ and ‘scrape’ page for uploading documents or a URL to scrape for documents;
2. an output page consisting of the tabular data. The output page is found from analysing the input document with the option to modify this found data, choosing relevant

columns, and removing columns and rows.

An example of the first input form is shown in Figure 3.4 and an example of what output analysis will look like is shown in Figure 3.5.



File: No file chosen

Dpi:

Page:

-----OR!-----

URL To Scrape supplementary data from:

Figure 3.4: A simple input form example for the first section of the user input – simple but sufficient. This should accept Microsoft Excel and PDF files. The DPI field refers to dots per inch which is a measure of the quality of the image recovered for OCR. The page field determines which page of the PDF document should be used.

Since the central theme of this work is to improve reusability of scientific publications, we will now propose a solution to keeping track of the parameters and state of the system. In this way, repeating the entire process of information extraction after parameter tweaking should be made unnecessary.

Choose Sequence Name Column		Choose Sequence Start Column	Choose Sequence End Column		
	Type 2 diabetes	chr12	53235262	53535262	rs1153188
	Type 2 diabetes	chr12	69713367	70099368	rs1495377,rs7961581
	Type 2 diabetes	chr16	52223775	52528027	rs8050136,rs9939609
	Type 2 diabetes	chr2	43436326	43736326	rs7578597
	Type 2 diabetes	chr2	105054029	105354029	rs6712932
	Type 2 diabetes	chr2	226651988	226951988	rs2943641
	Type 2 diabetes	chr3	12102844	12518124	rs17036101,rs1801282
	Type 2 diabetes	chr3	55138439	55438439	rs358806
	Type 2 diabetes	chr3	64536943	64836943	rs4607103
	Type 2 diabetes	chr3	186844380	187162983	rs4402960,rs6769511
	Type 2 diabetes	chr4	6170956	6470956	rs4689388
	Type 2 diabetes	chr4	122734963	123034963	rs7659604
	Type 2 diabetes	chr5	89431864	89731864	rs12518099

Choose Sequence Name Column		Choose Sequence Start Column	Choose Sequence End Column		
	Type 2 diabetes	chr12	53235262	53535262	rs1153188
	Type 2 diabetes	chr12	69713367	70099368	rs1495377,rs7961581
	Type 2 diabetes	chr16	52223775	52528027	rs8050136,rs9939609
	Type 2 diabetes	chr2	43436326	43736326	rs7578597
	Type 2 diabetes	chr2	105054029	105354029	rs6712932
	Type 2 diabetes	chr2	226651988	226951988	rs2943641
	Type 2 diabetes	chr3	12102844	12518124	rs17036101,rs1801282
	Type 2 diabetes	chr3	55138439	55438439	rs358806
	Type 2 diabetes	chr3	64536943	64836943	rs4607103
	Type 2 diabetes	chr3	186844380	187162983	rs4402960,rs6769511
	Type 2 diabetes	chr4	6170956	6470956	rs4689388
	Type 2 diabetes	chr4	122734963	123034963	rs7659604
	Type 2 diabetes	chr5	89431864	89731864	rs12518099

Figure 3.5: A proposed example for the working output for the user interface. The top figure demonstrates the full row, full column, and single cell selection capabilities. The bottom figure demonstrates the use of the ‘Choose Sequence Name Column’ (in green), ‘Choose Sequence Start Column’ (in yellow), and ‘Choose Sequence End Column’ (in red) buttons. These will be implemented using HTML and the JavaScript jQuery library.

3.5 Repeatable approach

The process of information retrieval involves multiple steps. If fault was found in the output, repeating the process should not involve repeating all steps. Keeping system-state and parameter information will improve the reusability of the system.

As an example of this (and as mentioned in Section 2.3), sequence coordinates can be either zero- or one- based. Depending upon what sort of work is being done with the sequences in question, the offset difference of a single base can be irrelevant. However, this irrelevancy is not all-encompassing and cannot be assumed. Which circumstances would yield zero-based or one-based coordinates is not clearly defined: prompting a user for this information will suffice for now.

Consider the following situation: suppose the coordinates extracted from a document are zero-based; suppose also the genome being browsed and the tools used for browsing assume input coordinates are one-based. This would need a simple tweak to the input coordinates in order for the correct sequences to be found. This sort of slight tweak would most certainly not require the entire job to be run again with different input parameters. Instead, a modular system (with pieces running as independently of each other as possible) provides scope for implementing a shell like environment (perhaps in future work) and a scripted approach to tying pieces together. A partial solution is implemented, providing means to store state and parameter information. The reuse of this information is reserved for future work.

Chapter 4

Implementation

The process of de-obfuscation and improving reusability of scientific publications consists of three parts: scraping, information extraction, and user intervention. The information extraction will be the most computationally heavy and intrinsically complex part of this, consisting of an excel extraction module and a PDF extraction module. The user direction is paramount in resolving the data into useful information and works side by side with a tracking module (which will keep track of parameters and various pieces of information relating to system state). The following is an outline of various novel or interesting approaches used in implementing this system.

4.1 Scraper component

Scrapy provides an above standard documentation with walkthroughs on how to accomplish scraping tasks. This framework is very powerful and more than necessary for scraping links to certain types of documents. There are commonly three components to a Scrapy web scraper: the environment in which Scrapy is hosted, the items to be scraped, and the scraping spider itself.

The framework can be utilized in numerous ways, the most common of these being in its own python-like shell environment. This would not be useful for our purposes; instead, running an encapsulating script making use of python's built in event driven network framework (called twisted) is chosen. In order to make this work, a spider must be created and items must be defined.

Scrapy items are used for defining the data to be scraped. This is done by defining an 'Item' class with any choice of associated fields as class level variables. These items are then used by the spider.

The spider is the real backbone of the web scraper. Along with necessities such as the spider name and start URLs, this houses information regarding allowed domains and rules associated with scraping. Spiders can be constructed to run recursively up to a certain depth of link, but for our purposes we will only scrape the given URL. With this information in hand, Scrapy does the job of retrieving information and storing it in an XPath like fashion (XPath is an XML Path Language for addressing an XML document). Detail on this storing mechanism is unimportant – suffice to know Scrapy makes use of selectors for information querying. For example,

```
hxs = HtmlXPathSelector(response)
```

```
links = hxs.select('//a')
```

will return all <a href...> tags from the scraped URL. From this point, all of these tags are broken into their constituent parts and stored as an Item. A question arises at this point: what happens to these items once the scraping is complete? Scrapy has no built-in method for exporting these items. It does, however, usefully employ a JSON exporter (amongst others). The fields stored are the link, the description associated with the link, and the link's title.

```

{"link": ["/bmcbioinformatics/login"], "title": ["log on"]},
{"link": ["/content/supplementary/1471-2105-14-103-s1.pdf"], "title": ["Download file"]},
{"link": ["http://www.adobe.com/products/acrobat/readstep.html"], "title": ["Adobe Acrobat Reader"]},
{"link": ["/about/access/#opendata"], "title": []},
{"link": ["/content/supplementary/1471-2105-14-103-s2.pdf"], "title": ["Download file"]},
{"link": ["http://www.adobe.com/products/acrobat/readstep.html"], "title": ["Adobe Acrobat Reader"]},
{"link": ["/about/access/#opendata"], "title": []},
{"link": ["/content/supplementary/1471-2105-14-103-s3.tiff"], "title": ["Download file"]},
{"link": ["/about/access/#opendata"], "title": []},
{"link": ["/content/supplementary/1471-2105-14-103-s4.tiff"], "title": ["Download file"]},
{"link": ["/about/access/#opendata"], "title": []},
{"link": ["/content/supplementary/1471-2105-14-103-s5.tiff"], "title": ["Download file"]},
{"link": ["/about/access/#opendata"], "title": []},

```

Figure 4.1: A JSON output example. The links are either relative to the domain being scraped (in this case biomedcentral.com) or static (such as the “Adobe Acrobat Reader” link).

The next step is to recover certain useful links only. In our case this would be any links to PDF documents, Microsoft Excel, or FASTA files. Although Scrapy spiders can be tailored and tweaked (using a variety of methods – most notably that of regular expressions) so that selectors return only certain links, doing this apart from Scrapy proves to be much simpler. We have chosen to parse the JSON file and construct two dictionaries for output: a dictionary containing all links and a dictionary containing useful links. These dictionaries are constructed with many-to-one relationships. This ensures a simple link title such as “Download file” can be associated with numerous links (Figure 4.1). The limitations of this scraping component are mentioned in Section 3.1.

Building a truly dynamic web scraper capable of resolving links in a JavaScript shell and recursively following links is an ideal case and is not outside of the scope for future development. Since this is an exercise in providing a means for improving reusability, we will now detail the intricacies of information extraction once these scraped documents (or, for that matter, any suitable document) are supplied to the system.

4.2 PDF information extraction algorithms

As outlined in Section 3.2, the job of reliably and accurately extracting information from PDF documents is no trivial task and can easily be considered enough to form an independent piece of work. A considerably large and decidedly major part of de-obfuscating scientific publications is found in the task of PDF tabular information extraction. Even this, with its intrinsic complexities, does not cover all cases and is but a fragment of what is possible. In the following sections, algorithms associated with the approach of using image analysis and OCR are detailed. The pipeline of actions performed on a PDF document is shown in Figure 3.2. The process of determining cell dimensions is interesting and will be detailed. The data structures related to OpenCV must be considered prior to detailing the algorithmic choices.

4.2.1 OpenCV and data structures

The primary data structures being worked with will inherently be determined by the OpenCV library. This means extensive use of multidimensional arrays and NumPy. NumPy is a Python extension for working with large, multidimensional arrays. NumPy has many useful built-in methods allowing it to perform tasks such as array restructuring. Each image array stores image information in multidimensional arrays – the outer array represents rows; the next depth represents each element of an individual row (i.e. columns); and the inner array is a three element array representing RGB values (ranging from 0 - 255) or a single value (of the same range) for grey images.

4.2.2 Determining cell dimensions

The first step in determining cell dimensions is making decisions regarding what data contained in an image can serve to supply suitable information for splitting into rows (horizontal segmentation) and columns (vertical segmentation). The traditional idea of a table would be one with defined lines. A solution becomes clear when there are clearly defined dividing

lines, i.e. using pre-defined line finding algorithms (like those present in the OpenCV libraries). This has been explored by and is found to be effective when lines are present. The persistence of this ideal case will not be all encompassing and basing algorithmic choices upon it will not work. A reliably present visual characteristic of tables must be determined. One such characteristic is that of space dividing column and row boundaries. Although spacing may be inconsistent and although there do not exist any defined layout conditions, tables involve some form of visual spacing by definition. We make use of the visual layout of tables to gather information.

Useful information for determining the spacing between possible cells is that of a pixel count. This works by counting dark pixels (using a grayscale image) along horizontal lines from top to bottom and vertical lines from left to right. The contrast between a tabular layout and a normal layout shows the usefulness of this information (Figure 4.2). Owing to the separate nature of each line's calculation, counting pixels has the potential to be easily parallelisable and will run in a concurrent fashion using Python's built in multiprocessing module.

At this point we have a cropped image of a PDF page with vertical and horizontal pixel count information. Using this information in a smart manner now becomes the challenge. With practically infinite possible variations of page and table layout, normalising this information is a crucial step in circumventing many classes of strange layouts. For example, a dark image or logo block or highlighting will result in some offset in the pixel information. The offset may be over the entire pixel count (which is not difficult to deal with) or may be over only a section (which is more challenging to deal with). A common minimum pixel count value is determined and all the pixel count values are normalised to this determined minimum.

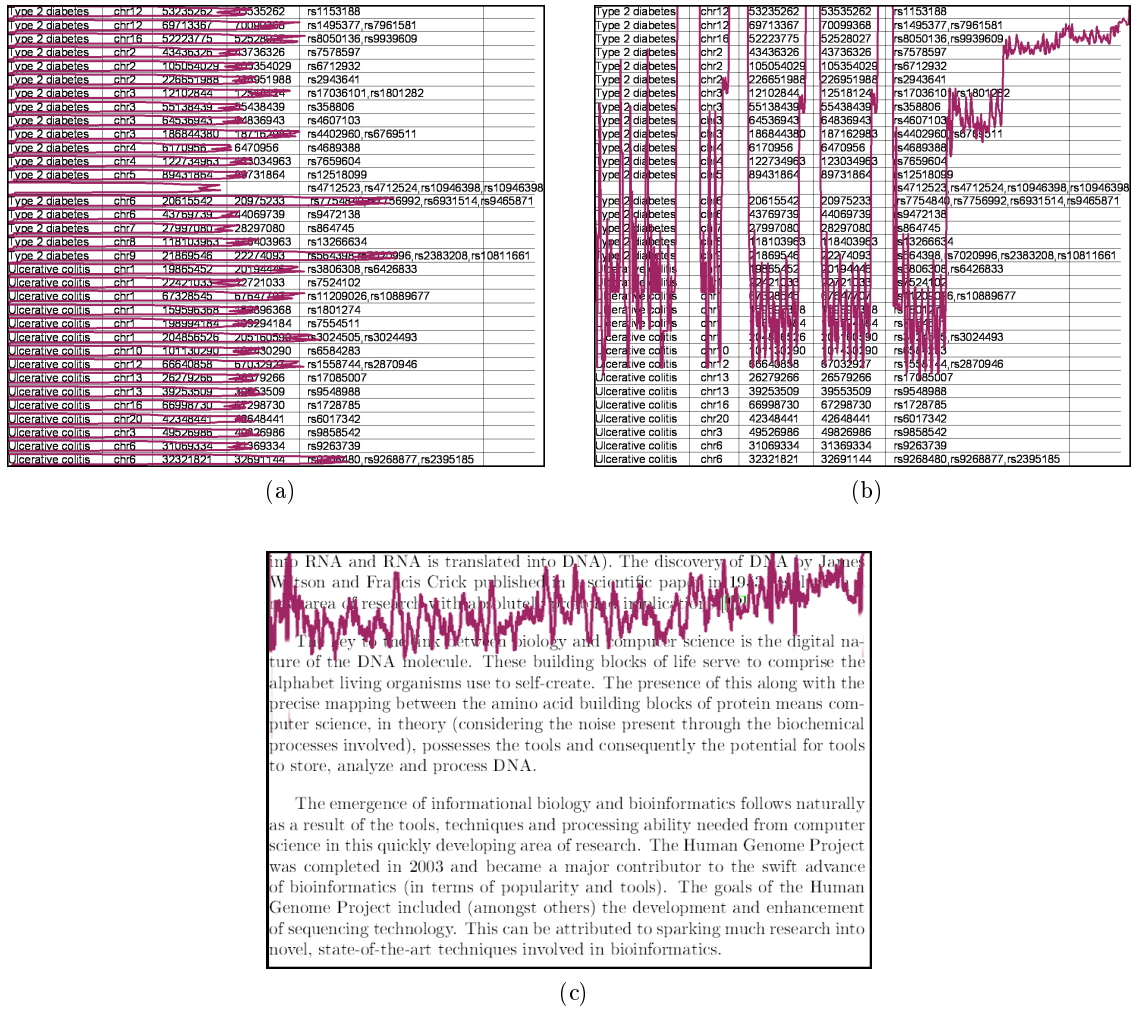


Figure 4.2: An example of counting horizontal and vertical pixels. The overlaid graph is the pixel count. The lines are inserted based on this pixel count information. Figure a is the horizontal pixel count: counting from left to right for each image y-coordinate. Figure b is the vertical pixel count: counting from top to bottom for each image x-coordinate. Figure c is an added example of what the pixel count looks like when considering non-tabular data.

The choice of using space to determine cell divisions means that existing clear table division lines now turns from a possible aid to a problem. To circumvent this problem, lines above a threshold length are removed. This removes all simple cases of dividing lines. An image with only reasonably clear white space divisions and horizontal and vertical pixel counts is what we have at this point. Determining cell boundaries can now occur.

Algorithm 4.1 Cell division point finding algorithm

```

determine pixel count length and calculate threshold
  for each line in pixel count
    if (line pixel count < threshold)
      consider as a gap
    else if (gap exists)
      add gap to gap list
  calculate total gap and average gap
  determine minimum gap length
  for gap in gaps
    if gap exceeds minimum gap length
      add gap to division points
    else
      if first or last gap
        add special cases to division points
        determine division points in percentages
  draw division points on image
  return division points in percentages

```

Algorithm 4.1 briefly summarizes the cell division algorithm. The threshold is calculated based on the image coordinates and a pixel ratio. The average gap multiplied by some multiplier to calculate the minimum gap length – this is also a parameter which can be changed. Tweaking these fields will be the key to determining correct cell dimensions. The division points are returned as relative percentages for each axis of the image (justified by the fact that the pre image analysis can take place on a much lower resolution version of the image than OCR).

Once the cell dimensions are determined, the image is cropped cell by cell in order to perform OCR. A number of complications arise:

- cell dimensions may be incorrect for some cells and/or for headings;
- Tesseract OCR does not work well with single characters and digits. Although there is a command line flag to allow for single characters, this does not fare well for single digits.

4.2.3 Fixing cells and helping Tesseract along

The following sections will outline techniques used to combat possible problems after the cell division step.

4.2.3.1 Fixing rows

Cell dimension analysis can lead to certain circumstances in which post analysis cell fixing must occur (Figure 4.3). The analysis is done row by row. Each row is passed to a fixing algorithm. The deconstructed row is analysed in order to make sure text does not encroach from one cell to another. To do this, each cell's rightmost edge is analysed. If this edge contains dark pixels it is determined that this cell has cut text and should be concatenated with the next cell. Analysis is continued up until the second last cell in the row. The resulting combination of original and concatenated images is returned. This clearly will not pick up on cells that have been cut in positions where there exists spacing (i.e. word or character spaces); the purpose of fixing rows is to allow for Tesseract OCR to have complete letters to perform OCR on.

Supplementary Table 5. Traits, marker SNPs and intervals analysed for VDR binding.					

Figure 4.3: An example of a heading post analysis – table headings are not uncommon and, although they have little to do with the contained tabular data, do provide useful information. Note the lines cutting letters apart.

The construction of the image arrays (see Section 4.2.1) and the need to concatenate images, offers choice as to how images can be concatenated. To run through each row of each image and concatenate in such a manner may prove computationally intense (see the left side of Figure 4.4). Instead, using NumPy's powerful ability quickly to restructure arrays and rotating each image by 90 degrees, means concatenation can then be a one step process, merely involving extending the overall image array (see the right side of Figure 4.4). The entire fixing process can effectively be repeated for each column. This process deals with

text that extends beyond the determined cell boundaries. Text that is particularly short (i.e. an individual character or letter) provides its own complications.

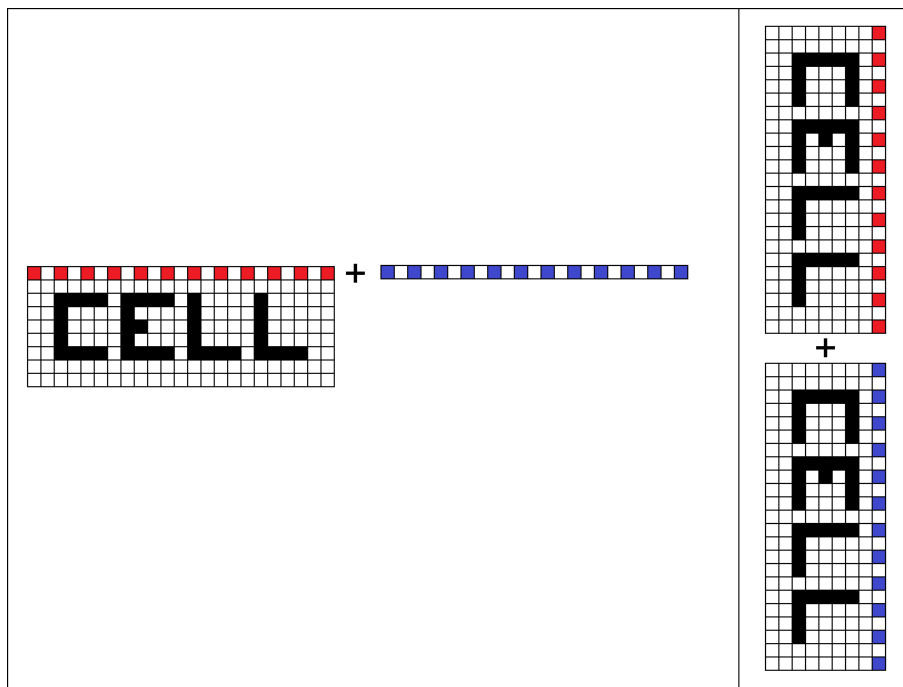


Figure 4.4: Outlining the two concatenation options available to us. The left side shows concatenation per row. The right side shows a rotated cell with concatenation per cell.

4.2.3.2 Tesseract and single characters

Tesseract OCR has numerous command line options for text alignment and input specification (Figure 4.5). Some of these have to do with the fact that Tesseract OCR does not have built in page segmentation. Other options allow a user to specify what the input will look like. Since it is not uncommon to find tabular data consisting of a single character or digit, option 10 (“Treat the image as a single character”) seems like a good choice for attempting to pick up single characters. During initial testing, it was found Tesseract OCR (with its default training) was unable to recognise single characters unless this command line option was set. What is more concerning is its complete inability to recognise single digits even in this mode.

```

Usage:tesseract imagename outputbase [-l lang] [-psm pagesegmode] [configfile...]

pagesegmode values are:
0 = Orientation and script detection (OSD) only.
1 = Automatic page segmentation with OSD.
2 = Automatic page segmentation, but no OSD, or OCR
3 = Fully automatic page segmentation, but no OSD. (Default)
4 = Assume a single column of text of variable sizes.
5 = Assume a single uniform block of vertically aligned text.
6 = Assume a single uniform block of text.
7 = Treat the image as a single text line.
8 = Treat the image as a single word.
9 = Treat the image as a single word in a circle.
10 = Treat the image as a single character.
-l lang and/or -psm pagesegmode must occur before anyconfigfile.

Single options:
-v --version: version info
--list-langs: list available languages for tesseract engine

```

Figure 4.5: Tesseract OCR’s command line options. Option seven is convenient for our use.

A simple mechanism is needed to deal with these cases. The solution to this comes in the form of placing an easily recognisable character alongside these single characters or digits and discarding it once OCR is completed. For no other reason besides for its easily identifiable features, the letter ‘T’ is chosen. OCR is first attempted and if no result is found, a single character or digit is assumed. The cell is analysed to make sure there is space to add an additional character. An image of a ‘T’ is loaded, scaled and added to the cell image. The ‘T’ image is smaller than the cell image, allowing NumPy’s broadcasting technique to be used. Broadcasting refers to NumPy’s ability to match smaller arrays to larger arrays by broadcasting the smaller array across the larger array so that they have the same dimensions and adding them. This mechanism is a means of vectorizing, taking place in C instead of Python and is usually computationally efficient. The row-fixing occurs before OCR takes place and the addition of an extra character to deal with single character faults occurs after OCR (causing OCR to repeat). The overall table output from the OCR step is kept using the same dimensions as is found from the cell dimensioning algorithm. The next step is to clean and improve the text found after performing OCR.

4.2.4 Fuzzy string matching

Traditional use of OCR is on scanned images with limitation on the image quality. Since text in PDFs is vector based, the quality of the image extracted from the PDF has the potential to be much greater than any scanned image can offer. The power behind using OCR is not fully realised nor needed. A suitable quality image must be chosen in order to strike a balance between Tesseract OCR's ability and the computational cost of analysing high resolution images. Accuracy testing for different chosen image qualities will be done. A minimum image quality of approximately 150 DPI is chosen. Regardless of the quality of the image, it is useful to compare OCR output to some known present words.

As mentioned in Section 3.2, pyPdf can be used to extract all text from a page. Each cell's text (determined by OCR) is compared to these words using pythons built in difflib library. The difflib module¹ provides classes for comparisons of sequences of various formats. The routines used from this library are *get_closest_matches* and *SequenceMatcher*. *get_closest_matches* takes in a string to match along with a list of possible matches, and returns an ordered list of best matches above some threshold match accuracy. *SequenceMatcher* is used to determine comparison ratios between two strings by finding the best contiguous match in a recursive fashion. Complications must be taken into account before matching can occur:

- A cell can contain multiple words.

Solution: each individual word is matched and a matched sentence is built up from these best matches.

- OCR can easily be confused between the letter 'O' and the number zero, the letter 'S' and the numbers five and eight.

Solution: All occurrences of possibly confused letters are replaced with their counterpart(s) and a new match is found for each of these.

¹Documentation found at docs.python.org/2/library/difflib.html

These cases result in many possible matches. The matches are the original input (if no matches are suitable), a simple one word match, multiple letter replaced matches, and multiple sentence matches (the original sentence also undergoes letter and number swapping to circumvent confusion). These matches are found using *get_closest_matches*. *Sequence-Matcher* is now used to determine the accuracy ratio of each individual match and the best match is chosen. If the input consists purely of numerical values, the required threshold accuracy is higher – numerical values are usually required to be of higher accuracy. The new fuzzy-matched table is output as the final result.

Along the way there are many possible subtle tweaks to algorithm parameters and command line inputs. Owing to the fact that we are considering cases focused on bioinformatics genome coordinates, specifications of what tables should look like do exist. For instance, there should be a minimum of three columns: the sequence name column, the sequence start coordinate, and the sequence end coordinate. These columns, the coordinate offset (i.e. 0- or 1-based notation), and the genome will be used to construct a query for sequences.

4.3 Keeping track

Individual elements of this system have been outlined and can generally work separately from each other. It would be a waste of time and computational power to recompute much of what has already been computed if a parameter is needed to be changed. The changing of a parameter is just one example of what could change. We will keep a record of all parameters along with information relating to intermediate states of the system. The following is an outline of the parameters.

- The input filename.
- The quality (in dots per inch) used to convert the PDF page needing analysis along with the page number within the PDF document.
- The multiplier for determining the threshold of the minimum pixel count for column

and row ‘gap’ calculations. This is called the *pixel ratio*.

- The multiplier for determining the presence of a gap. This multiplier aids in defining the length of a gap and is called the *minimum gap length multiplier*.
- The multiplier for determining the normalised minimum point for the pixel count. This is called the *minimum pixel count threshold multiplier*.
- The indices of the three necessary columns (as mentioned in Section 4.2.4).

Along with the parameters, certain files relating to the system state will be kept.

- The original PDF document and the original image file extracted from it.
- The pixel count information – both before and after normalisation. This means that normalisation can be repeated with a different *minimum pixel count threshold multiplier*.
- For both user use and debugging (see Figure 4.2 for examples):
 - The image with the pixel count information plotted on this.
 - The image with determined cell boundaries indicated.
- The extracted table before and after fuzzy matching.

4.4 User interface

The supplementary gathering system is designed to improve reusability of scientific publications. As it stands, the scraping, Microsoft Excel extraction, and PDF extraction modules can stand alone (as can their constituent parts) to form an explorative backbone for a proof of concept information extraction system. To move away from this code-oriented environment into a workable piece of software, a user interface is required. This user interface will be responsible for gathering user defined parameters and should be the means through

which tweaks in this entire data extraction process can occur. There exist many possible aspects to this. As an example we will implement a mechanism for editing table structure and data (as exemplified in Figure 3.5). The main mechanism behind this will be the jQuery JavaScript library. The jQuery library makes use of the document object model which is an API defining the logical structure of valid HTML and XML documents. jQuery is designed as an abstraction on top of lower level details and is written in JavaScript. It simplifies the syntax for traversing the document object model in order to select elements, change the content and structure dynamically, handle events, develop socket based applications, and much more. An advantage of using JavaScript is that it runs client side and does not require HTML requests to restructure the working web page.

The HTML page will be dynamically built by a Python script making use of Python's Common Gateway Interface (CGI) scripts. CGI scripts are a means of running python code invoked by an HTML request and run on the server side. They work in much the same way as PHP scripts. The choice of using CGI over PHP is the fact that Python is used throughout the project and will link easily with the CGI scripts. The Apache server setup allows for these (provided the settings are in order). HTML tags allow both 'class' and 'id' fields (with no decisive limitation on the number of classes or ids associated with a tag). Tailoring these as markers for column, row, and cell selection proves useful – jQuery has access to these within the document object model and can select and alter tags.

jQuery-style functions are written to highlight selected cells and keep track of the associated ids and classes. Once the table has been tailored and columns have been selected, this enriched table information can be passed on and turned into a BED file for submission.

4.5 How testing will occur

In order to do testing, a sufficient dataset is necessary. This will require some searching for bioinformatics related supplementary data (since it is this area at which the system is aimed). Any PDF with suitable table structures will also be considered for testing. Various

accuracy tests will be undergone. Accuracy can be broken into two parts: OCR accuracy and cell division accuracy. Using OCR to extract cell information can be replaced in the future (by other means of direct coordinate to text element PDF extraction). Tables will be analysed with this system and also manually. The comparison of manual and automatic analysis will yield suitable accuracy results. The fuzzy matching algorithm will also be tested with a variety of inputs.

Chapter 5

Findings and Discussion

Testing was carried out on parts of the system that would yield results. For instance, the fuzzy matching algorithm is tested to gauge the contributed improvement of using it. To follow is a series of results and discussion associated with several parts of the system.

5.1 Scrappy

The implementation of the scraping component of this system is not at a level where it can resolve dynamic links using JavaScript. It can, however, extract all static links (i.e. links found directly in the HTML source code of a web page) and determine which of these correspond to useful documents. The scraping component is not the bulk of the system being implemented but does serve as a proof of concept and opens doors to future implementation and improvement. The current platform would be considerably improved with the addition of the ability to resolve links dynamically.

5.2 OCR accuracy

Owing to the fact that OCR has the potential to be inaccurate, we measure accuracy to determine the viability of using OCR as a means for extracting information. As a proof of concept we perform tests on two sets of data – a constructed table with an array of different styles of input values (determined by analysing a range of supplements) and a real world bioinformatics example. Figure 5.1 shows an example of constructed data while Figure 5.2 shows an example of real world data.

y				
0	00	O	o	00000
1	1.1	-1.234567890	+0.2 e-3.7	1011001101
9	88	5.3E-23	6655	44321
85%	<1.0	0.0801	(mg/L.) -8*9	34
105,45	789, 758	2006-hello	ABCDEFGHJKLM	mnopqrstuv
chr11	16adf69	16510086	rs297325,rs4756	a bc def ghi
Ooyui	aAaAx!	@ U.S.A. bord	<i>Bollettino</i>	(-1.2349)

Figure 5.1: An example of constructed data

Crohn's disease	chr1	199052488	199352488	rs11584383
Crohn's disease	chr10	35177655	35477655	rs17582416
Crohn's disease	chr10	63958491	64258491	rs10995271
Crohn's disease	chr10	101131582	101431582	rs11190140
Crohn's disease	chr11	75828963	76128963	rs7927894
Crohn's disease	chr12	38738206	39038206	rs11175593
Crohn's disease	chr13	43205924	43505924	rs3764147
Crohn's disease	chr16	49171279	49471279	rs2066847
Crohn's disease	chr17	35144288	35444288	rs2872507
Crohn's disease	chr17	37617726	37917726	rs744166
Crohn's disease	chr18	12619946	12919946	rs2542151
Crohn's disease	chr2	233695148	233995148	rs3828309
Crohn's disease	chr21	15577090	15877090	rs1736135
Crohn's disease	chr21	44289988	44589988	rs762421

Figure 5.2: An example extract of real world style bioinformatics supplementary table data.

An expected output is constructed by manually copying and correcting the table data. The PDF is run through the table extraction pipeline and resulting tables are output (both with and without fuzzy matching). These tables are then compared cell by cell to the expected output table. This comparison is two fold: each cell is checked for an exact match against the expected output (dashed line) and also for similarity against the expected output (solid line). The varying input parameter is the image extraction quality (so as to test the most effective range of OCR). Figures 5.4 and 5.5 show the OCR accuracy versus input image quality for constructed data and real world data respectively. The error rate in higher quality regions (from 160 DPI onwards) of the constructed data and real world data is 6.05% and 0.2% respectively. Tesseract OCR word error rate is found to be between 2.63% and 4.95% with an average of 4.12%. Our achieved error rate is not very different from Tesseract OCR's known error rates. Error rates can be minimized by using maximum quality images – the real world data OCR maximum accuracy is 100% while the constructed data OCR maximum accuracy is 96.77%.

The inconsistency in the trend of accuracy with lower quality images is expected as OCR is dependent on the structure of the text represented in the image. At different qualities certain nuances of text construction change (Figure 5.3).

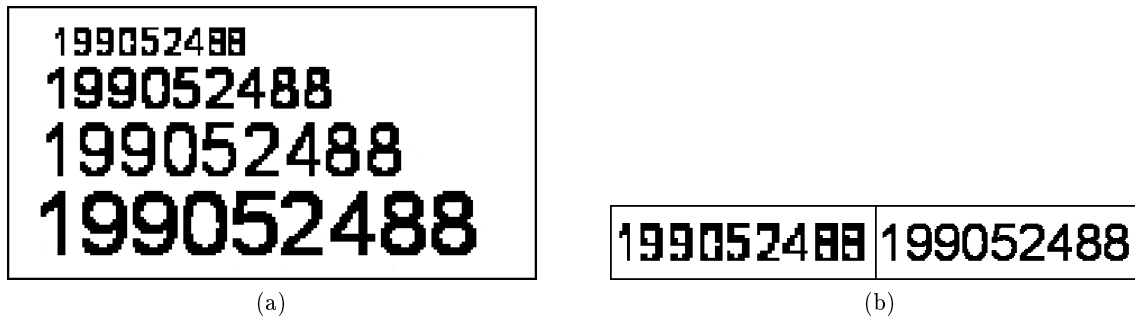


Figure 5.3: Figure a highlights the difference in image quality as the DPI is raised. The difference in text construction is most notable at lower quality and is most noticeable when considering the '8'. Figure b shows the difference in text construction at 110 DPI and 250 DPI respectively: at 110 DPI the two eights can easily be confused with 'BB'.

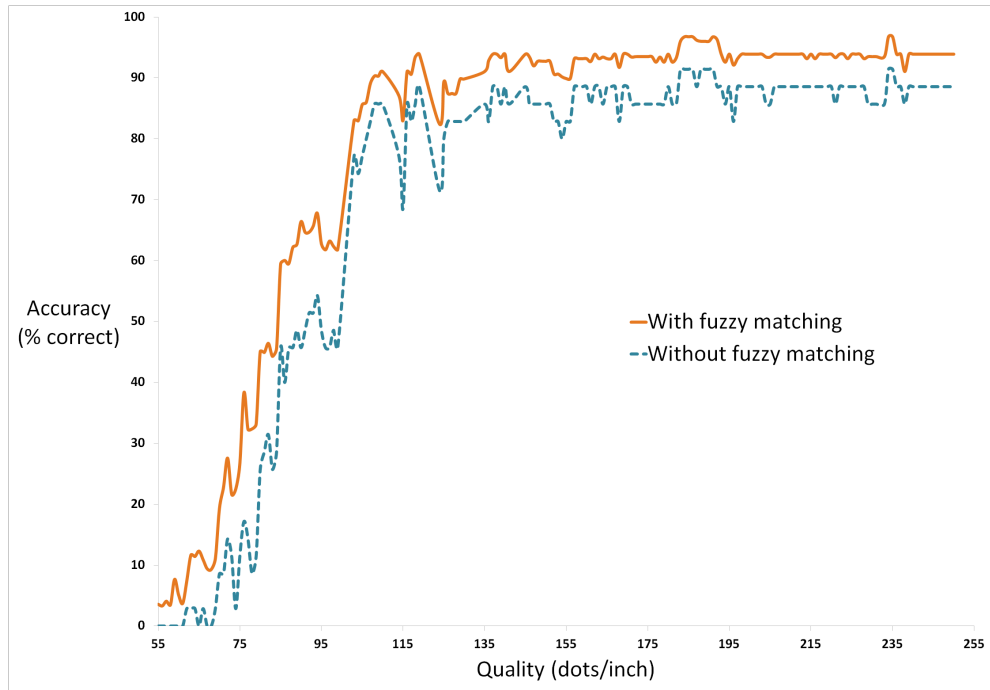


Figure 5.4: Accuracy versus Quality - constructed data.

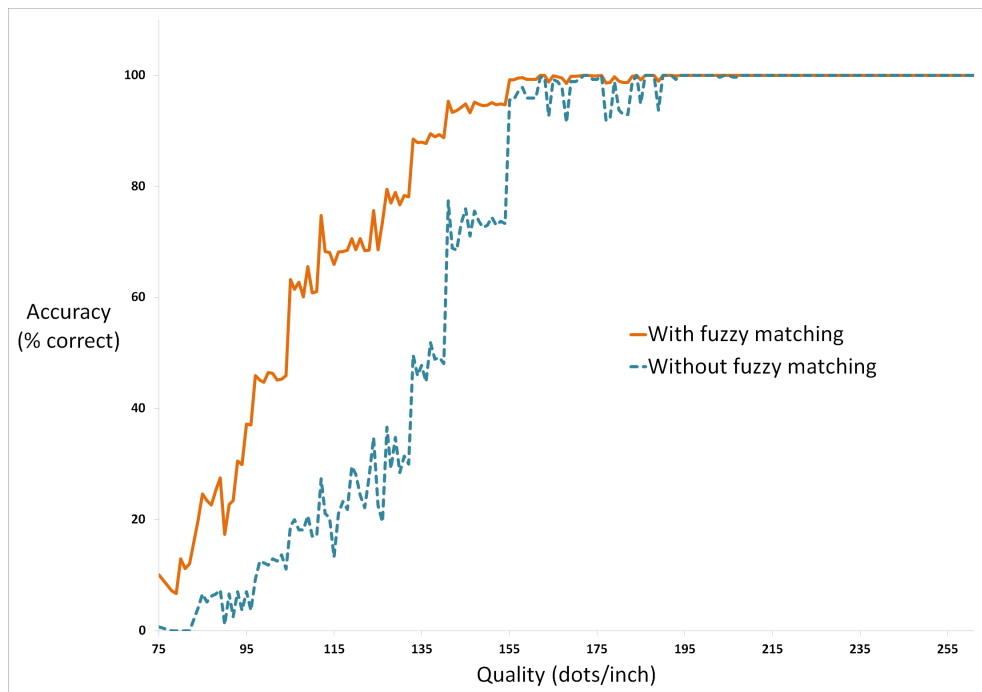


Figure 5.5: Accuracy versus Quality - Real world data .

There is a consistent trend toward more accurate results with higher quality images, although the shape of the graphs in Figures 5.4 and 5.5 is not smooth. The erratic changes in OCR accuracy diminish with higher quality images as does the time taken to perform OCR. Figure 5.7 shows the OCR timing results for constructed data. OCR analysis on images below 97 DPI takes considerably longer than analysis of images over 97 DPI. Figure 5.7 shows the OCR timing results for real world data. In contrast with the previous results, this shows an initial increase in time with higher quality images. The contrast between these results shows the dependence of the OCR stage on many factors (such as the number of cells, the constituent text of these cells, and the image quality). There can be no consistent time related to the OCR stage owing to the many possible table construction variations. The real world data consists of many more cells than the constructed data; the OCR stage takes on average 44.67 seconds to complete.

The time taken to perform OCR becomes linearly consistent (considering a rolling average over multiple qualities) and highly erratic. This erratic nature has been observed throughout implementation and testing. A possible cause of this may reside within Tesseract OCR. This behaviour is because of the inherent difficulty in processing different quality images (Figure 5.3) but would cease as the image quality is raised to a point where image quality and text shape and construction have little or no relationship. Another cause of this may be involved in the large array data structures used in image processing. Certain quality images may result in data structure sizes which are efficiently stored and passed around in memory, while other quality images may result in data structure sizes which require more instructions to handle. These are merely assumptions and future testing would involve determining the instruction counts involved in this step at different image qualities.

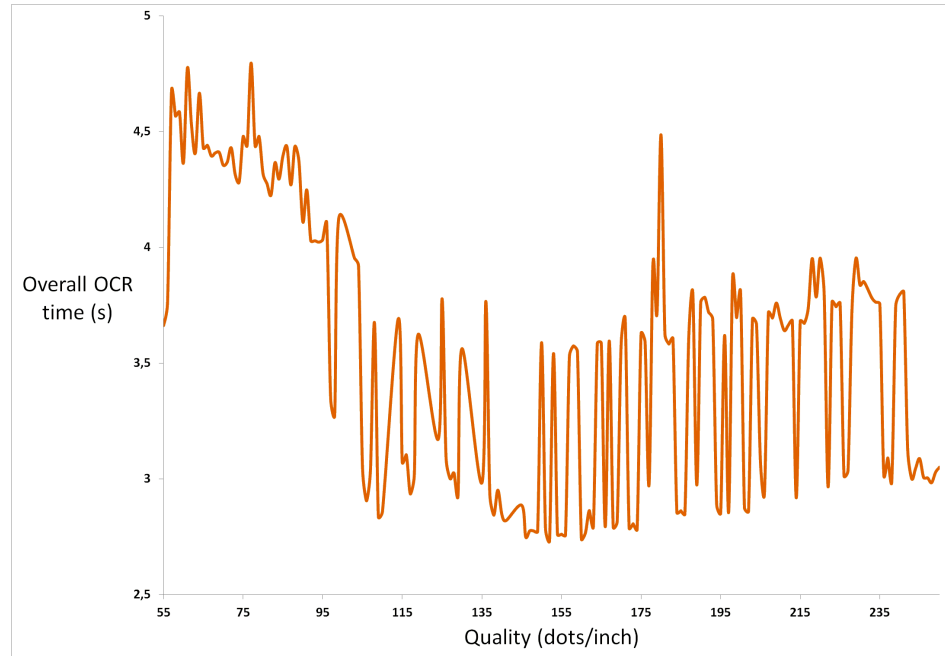


Figure 5.6: Overall table OCR analysis time versus quality for constructed data. This corresponds with the data used in Figure 5.4.

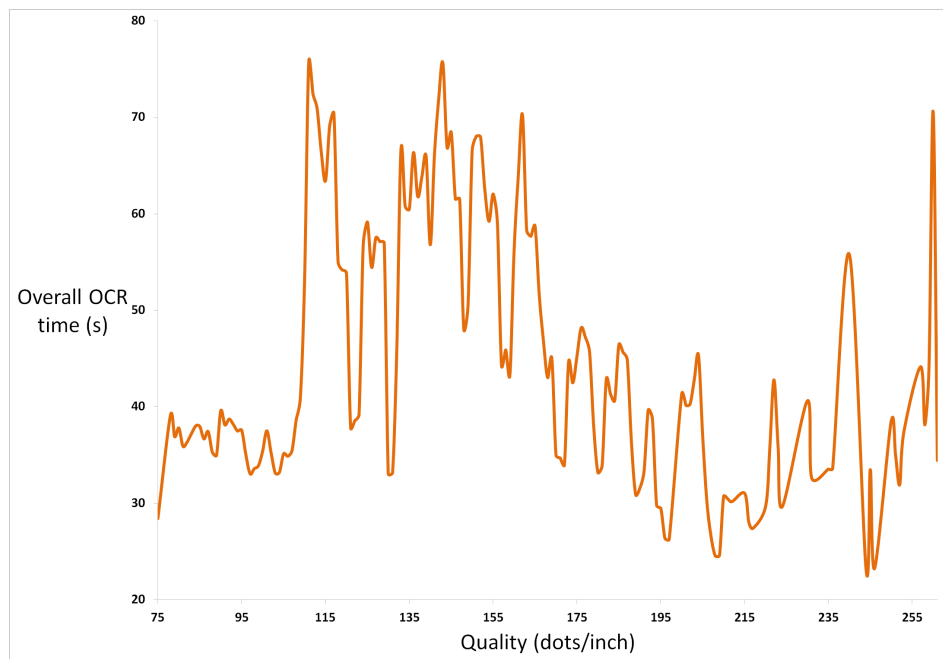


Figure 5.7: Overall table OCR analysis time versus quality for real world data. This corresponds with the data used in Figure 5.5.

5.3 Fuzzy matching improvement

The contribution of the fuzzy matching algorithm is exemplified in Figures 5.4 and 5.5. The difference in accuracy with and without fuzzy matching is tested. At first glance these results look similar to those of the OCR accuracy testing. This is, however, not the case. There may be possible confusion surrounding the difference when fuzzy matching takes place in these two sets of tests. When testing OCR accuracy, only the *SequenceMatcher* is used to determine resultant accuracy levels. In contrast to the OCR accuracy testing, this fuzzy match improvement testing is an actual test of the fuzzy matching algorithm (outlined in Section 4.2.4). This is done by constructing two output tables: both with and without fuzzy matching. The accuracy of these tables is determined (Figure 5.8 and Figure 5.9).

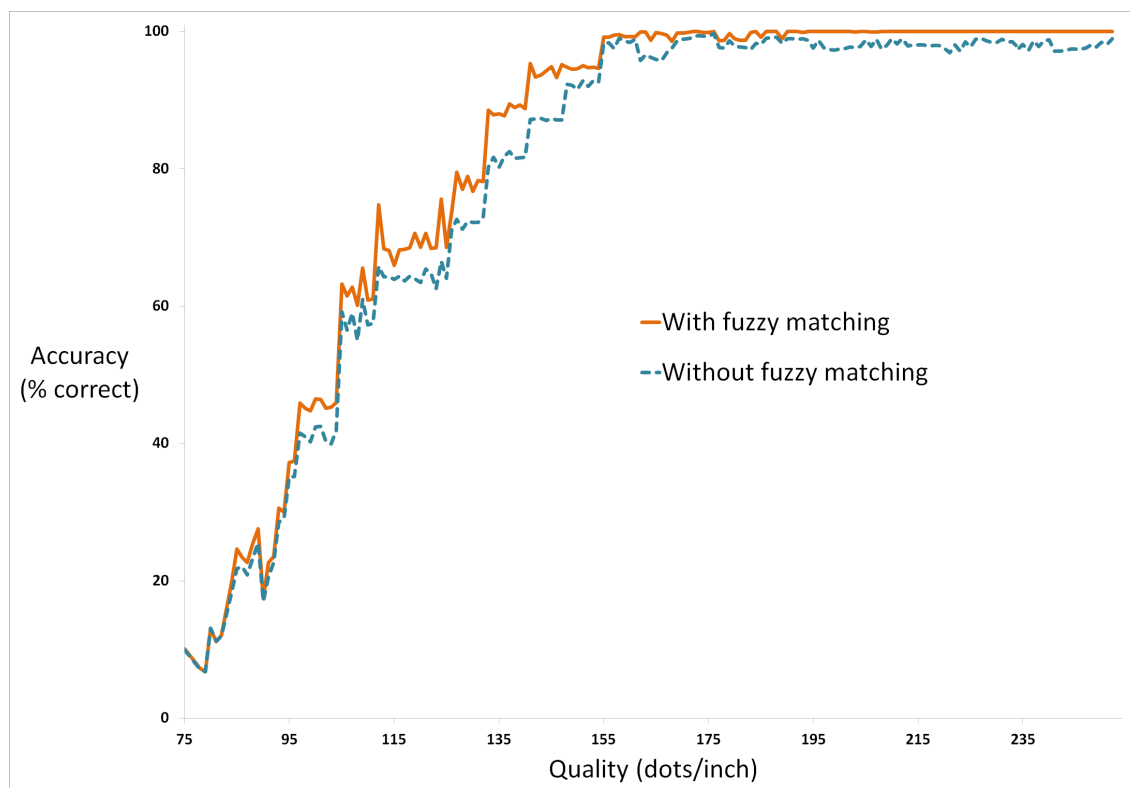


Figure 5.8: Accuracy versus Quality - with and without fuzzy matching (real world data).

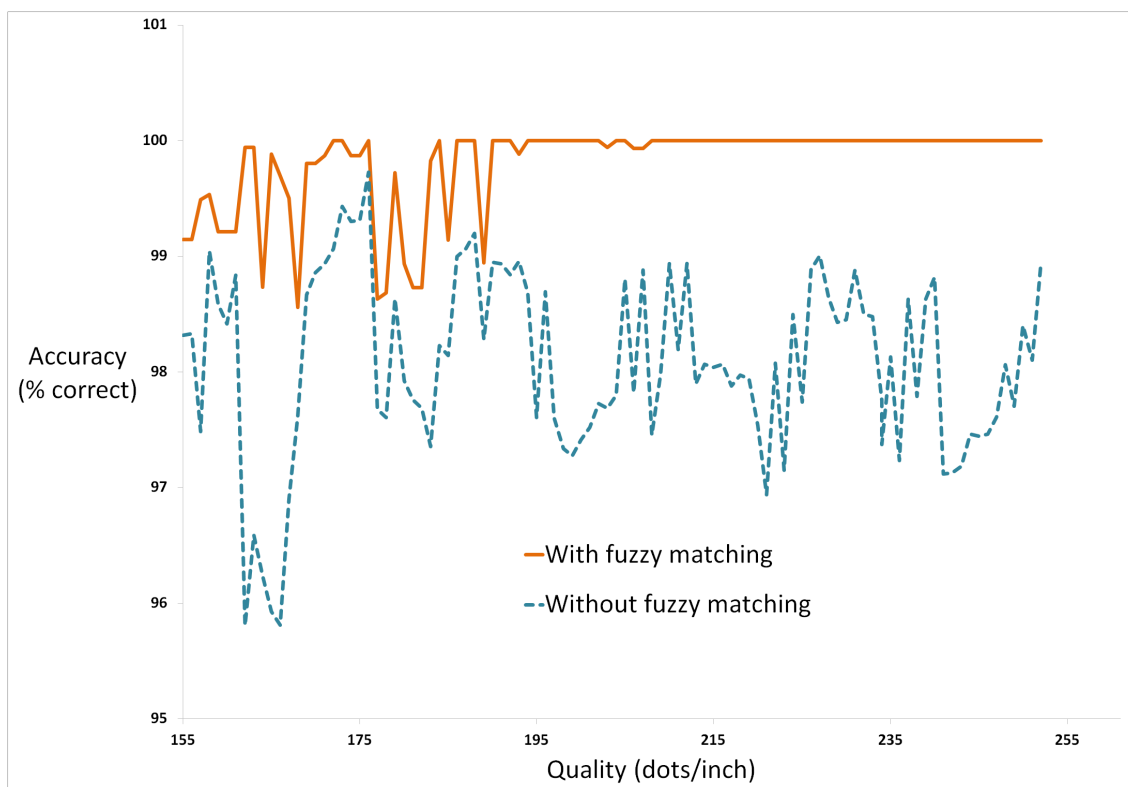


Figure 5.9: Accuracy versus (high) quality - with and without fuzzy matching (real world data).

There is consistent improvement in accuracy and reliability afforded by the fuzzy matching algorithm at all different input quality images. This testing was done on fairly consistently structured tables (such as those shown in Figure 4.2). The result of perfect accuracy at higher resolutions may be attributed to the simple structure of this table the text contained in the cells. Figure shows that image quality is not the only contributing factor to accuracy – Tesseract OCR is not perfect (and performs better with more textual input data because of the multipass nature of its OCR algorithm). With more complex table structures to consider, testing the cell division effectiveness must now occur.

5.4 Cell division effectiveness

The main portion of work being done in determining cell division points is that of determining vertical and horizontal pixel counts. The time involved in doing so, is shown in Figures 5.10 and 5.11. As with the OCR timings, the erratic nature of these timing results may also be attributed to the problem of memory allocation of the large arrays used in image processing. There is a linear relationship between the time taken and the image quality. The real world data (Figure 5.11) is seemingly more erratic than the constructed data (Figure 5.10). The real world data has more entries and thus results in a larger image (introducing complexity and causing the time taken to increase).

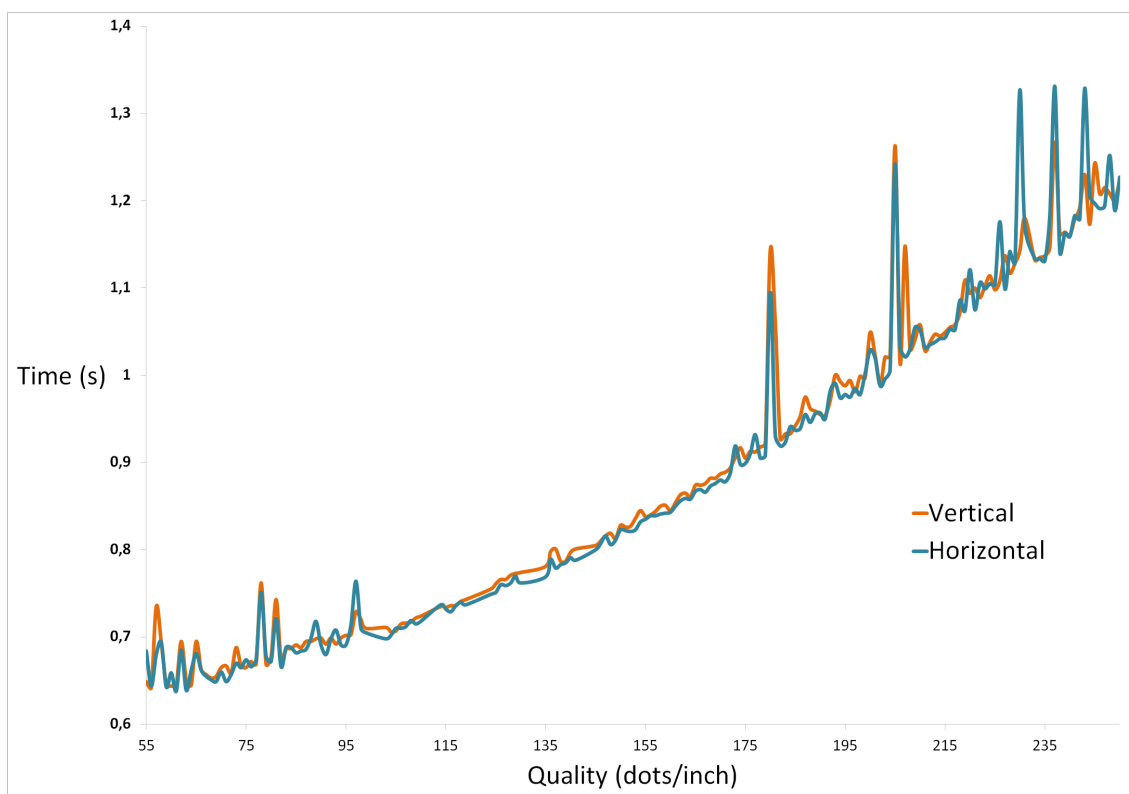


Figure 5.10: Horizontal and Vertical pixel count timings – constructed data.

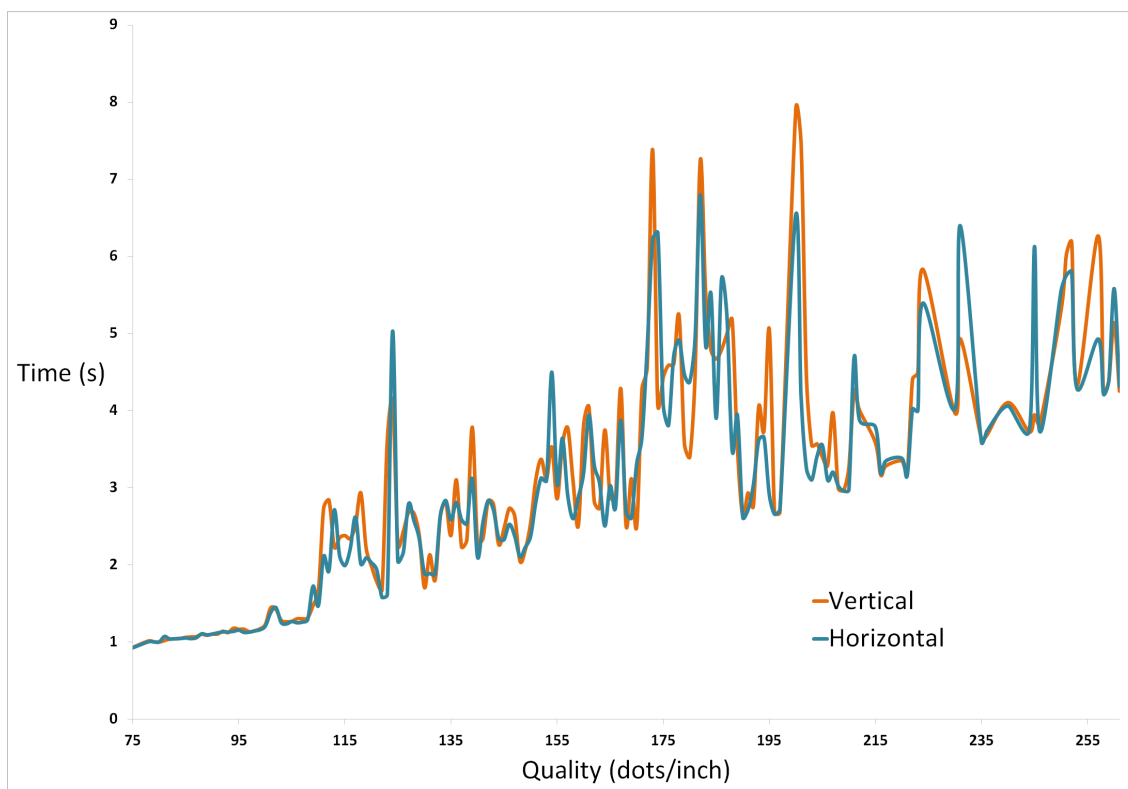


Figure 5.11: Horizontal and Vertical pixel count timings – real world data.

Determining column dimensions is a more complex task than determining row dimensions owing to the fact that cell length tends to be inconsistent while line division remains fairly consistent within a table. The accuracy of the cell division algorithm is exemplified in Figure 5.12. The centre of the gradient line shows the expected number of lines and fades outwards to represent precision loss. The remaining solid lines represent the division count for multiple sequential pages of a PDF document (constituents of the same large table). Use of the row fixing algorithm (detailed in Section 4.2.3) results in greater leeway in the precision of cell division. In this example, all the considered pages' division counts are accurate at very low pixel ratios (generally from 3.2% and downward). We have chosen to test the algorithm accuracy on simple tables where no considerable noise can interfere with pixel counts (as apposed to the table analysed in Figure 5.13 which has considerable noise in the form of a dark box above the table).

Another notable feature of these graphs is the common trend of a region with very high division counts. Figure 5.14 aids in explaining the presence of high division counts at particular pixel ratios. The cell division algorithm relies on the pixel ratio to determine gaps between columns and rows. The staggered nature of the pixel count graph results in false gaps and thus false divisions. Take note of the horizontal lines indicating the pixel threshold. When this is above the overlaid graph a gap is considered to be found (provided the gap persists for a designated length). Future work will involve finding a solution to this problem by smoothing the pixel count graph so that the staggering is resolved.

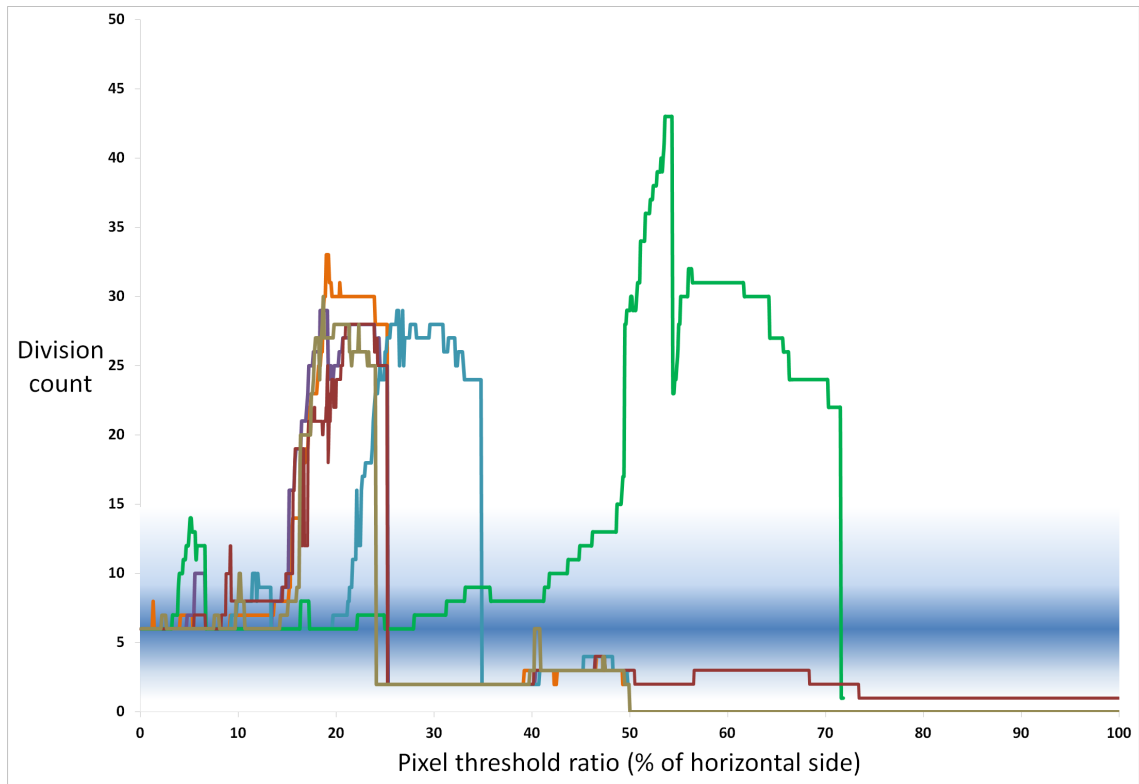


Figure 5.12: Pixel threshold ratio versus division counts for tables without noise. The solid lines represent the column division count found by our algorithm. The fuzzy line centers over the required output, with diminishing accuracy and effectiveness as it fades.

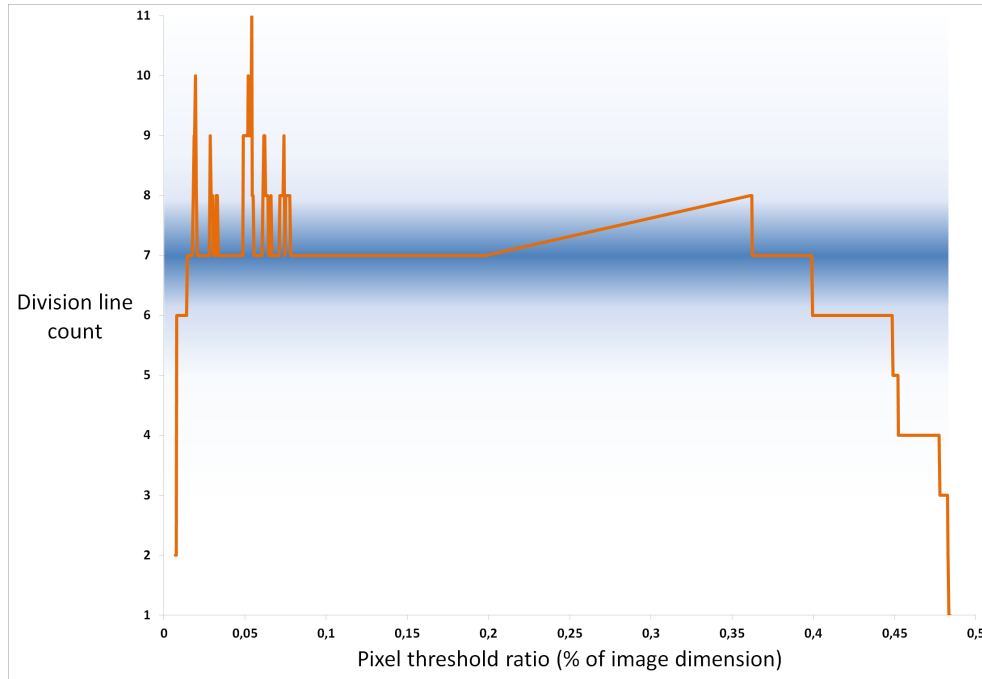


Figure 5.13: Pixel threshold ratio versus division counts for tables with a noise.

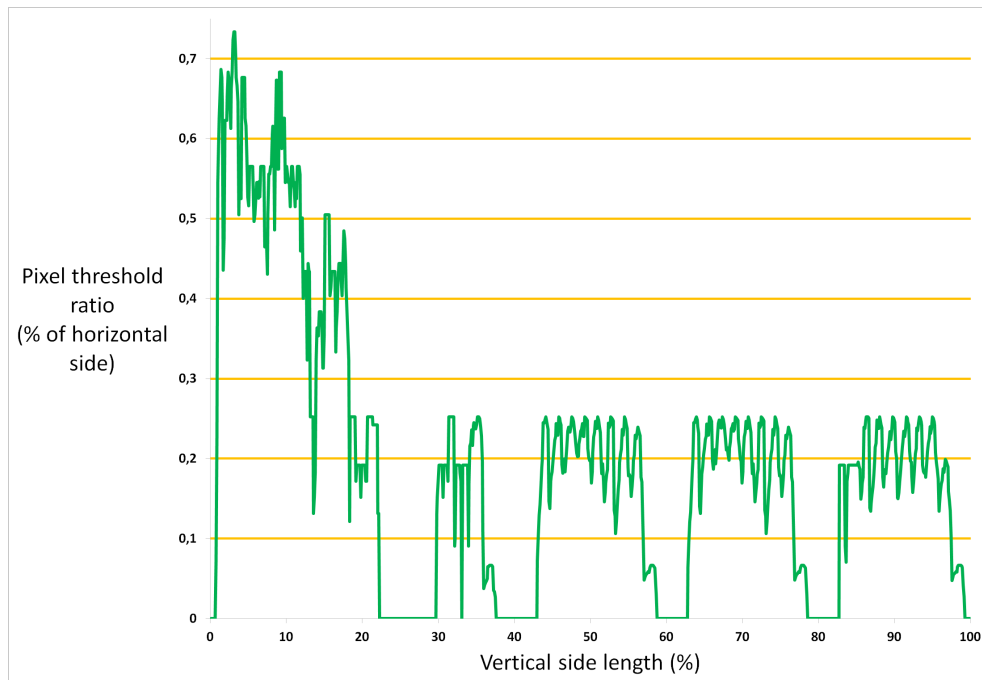


Figure 5.14: A vertical pixel count (like that as shown in Figure 4.2) scaled and overlaid on the pixel ratio. The horizontal lines exemplify how different pixel ratios can cause different algorithmic nuances to come into play.

5.5 User interface

The user interface is designed to accept an input document, a page number, and DPI (Figure 3.4) and output the matched table and an image with pixel count and line division information overlaid (Figure 3.5). The functionality of the user interface is lacking. Although a user is able to edit and modify cells and the table structure, using JavaScript and HTML tables proves constricting. Building a custom user interface will allow for improved flexibility and functionality. The current user interface simply interacts with the data extracted from the input supplementary documents and can does not change this data.

5.6 Repeatability

Keeping track of the parameters and states involved in processing a PDF is a lightweight operation. Using a dictionary data structure is sufficient to keep track of parameters. Restructuring the system to allow repetition of the same process with various parameter tweaks will be easily accomplishable in future work. As it stands presently, all parameters are kept track of and serialised to a file called *parameter_dictionary* and the state of the system is kept track of by saving pixel count information and images at various stages of the analysis (Figure 4.2). Although far from the standards of Mesirov (2010), the tracking component serves its purpose as a proof of concept and creates a platform for further work.

Chapter 6

Conclusions

6.1 Summary

The problem, as stated in Chapter 1, this research intended to solve, was the lack of an autonomous way to extract and reuse bioinformatics supplementary data (with extension to any supplementary data) in tabular form. Information extraction systems do exist but are either tailored toward a different goal or are not open source.

A variety of document formats were considered and (owing to its persistent and widespread use) PDF documents were chosen as a focal point. The chosen approach was to convert PDF pages to images and perform image analysis and OCR on these images.

The image analysis consisted of determining the table dimensions using vertical and horizontal pixel counts and thus determining ‘gap’ regions. The algorithm proved to handle tables of low complexity and low noise failed with tables of higher complexity or with considerable noise. Exploring the table dimension finding algorithm, yielded insight into how to improve said algorithm. Refinements and improvements to this algorithm will vastly improve its effectiveness. Determining cell dimensions can be done in a heuristic fashion provided the tabular data is of a consistent nature. The row fixing algorithm is designed to

allow more leeway in incorrect cell dimensioning. Once table dimensions are found, Tesseract OCR is used to extract cell contents.

It is found that the Tesseract OCR engine is not perfect and especially hindered by the short length of text present in individual cells. Countering this weakness by using high quality images and approximate string matching against a known set of words, consistently improved accuracy. Performing OCR on a large table is slow, although copying from PDFs and fixing the table structure can prove to be even more time consuming. Using OCR for extracting information from PDFs is not viable in terms of accuracy (considering the actual textual data is available to us) and efficiency.

Supplementary data is not published in a completely consistent and reusable manner. Extracting information from supplementary data published as PDF documents is a challenge as this format is designed for portability and not reusability.

Finding static supplementary data links on a web page was done using Scrapy. Extending this to scrape scientific publishing web sites in search of supplementary data corresponding to a scientific publication would be possible using Scrapy or some other web scraping platform yet is reserved for future work.

A simple user interface was developed to allow users to interact with the extracted data. With considerably more work a better user interface can be designed. The purpose of the user interface should be to allow extracted table editing and parameter tweaking. A tracking component stores these parameters, as well as pixel count and image information, yet must be integrated into the user interface.

This project set out to explore a means to provide a means of making bioinformatics supplementary data more reusable. Although the techniques explored may be lacking, much was explored regarding the viability of said techniques and further headway can now be made in this venture.

6.2 Future work

OCR is a slow and imperfect technology. As mentioned, making use of other techniques to resolve coordinates to string elements in PDFs may prove faster and more accurate. This would, however, require a streamlined and very accurate means of dividing a table correctly. The dimension finding algorithm needs improvement. A simple pixel count smoothing could yield much higher accuracy. Implementing expected gap length analysis by looking at the pixel count data, may also improve accuracy.

Text elements are stored by coordinates in PDF documents. Resolving accurate cell dimensions to text elements may be an alternative means of extracting cell information.

If OCR is still chosen as the preferred method of extracting cell information, this can be done on an entire table/page simultaneously to improve OCR accuracy. Tesseract OCR is thread safe and can therefore be run on multiple threads, which should dramatically reduce the time taken to perform OCR on all cells.

The PDF table extraction assumes the entire input page is a table. Making use of work by Liu *et al.* to identify tables within a page, will allow us to move away from this assumption.

Support vector machines (or other artificial intelligence techniques) may be used to class sections of PDFs or images as tables.

An improved user interface and a linked tracking component will also be reserved for future work.

Crowdsourcing can be used as contribution toward a repository of de-obfuscated scientific publications. If somebody has gone to the effort to prepare either manually supplementary data for reuse, make use of a system like ours to recover supplementary data, or contact the author(s) for the original data, there should be a means for storing and making this available for others to use. There exist repositories aiming to improve reusability, but none are aimed at recovering already published data.

References

- Adobe Systems Incorporated. 2006 (November). *PDF Reference*. Sixth edition edn.
- Adobe Systems Incorporated. 2013. PostScript vs. PDF. Online. Available from: <http://www.adobe.com/print/features/psvspdf/index.html>. Accessed on 23 Jul 2013.
- Allauzen, Cyril, Riley, Michael, Schalkwyk, Johan, Skut, Wojciech, & Mohri, Mehryar. 2007. OpenFst: A general and efficient weighted finite-state transducer library. *Pages 11–23 of: Implementation and Application of Automata*. Springer.
- Bailey, T. L., & Elkan, C. 1994. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proc Int Conf Intell Syst Mol Biol*, **2**, 28–36.
- Bailey, T L, & Gribskov, M. 1998. Combining evidence using p-values: application to sequence homology searches. *Bioinformatics*, **14**(1), 48–54.
- Bailey, Timothy L. 1995. *Discovering motifs in DNA and protein sequences: The approximate common substring problem*. Ph.D. thesis, University of California at San Diego.
- Bailey, Timothy L. 2011. DREME: motif discovery in transcription factor ChIP-seq data. *Bioinformatics*, **27**(12), 1653–1659.
- Bailey, Timothy L, Williams, Nadya, Misleh, Chris, & Li, Wilfred W. 2006. MEME: discovering and analyzing DNA and protein sequence motifs. *Nucleic Acids Res*, **34**(Web Server issue), W369–W373.
- Bailey, Timothy L., Boden, Mikael, Buske, Fabian A., Frith, Martin, Grant, Charles E., Clementi, Luca, Ren, Jingyuan, Li, Wilfred W., & Noble, William S. 2009. MEME

- Suite: tools for motif discovery and searching. *Nucleic Acids Research*, **37**(suppl 2), W202–W208.
- Bayat, Ardeshir. 2002. Science, Medicine, And The Future: Bioinformatics. *BMJ: British Medical Journal*, **324**(7344), pp. 1018–1022.
- Benoît, Gerald. 2005. Bioinformatics. *Annual Review of Information Science and Technology*, **39**(1), 179–218.
- Buckheit, J. B., & Donoho, D. L. 1995. WaveLab and Reproducible Research. *Pages 55–81 of: Antoniadis, Anestis, & Oppenheim, Georges (eds), Wavelets and Statistics. Lecture Notes in Statistics*, vol. 103. Springer New York.
- Crick, F. H. C., & Watson, J. D. 1954. The Complementary Structure of Deoxyribonucleic Acid. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, **223**(1152), 80–96.
- Flicek, Paul, Ahmed, Ikhlak, Amode, M. Ridwan, Barrell, Daniel, Beal, Kathryn, Brent, Simon, Carvalho-Silva, Denise, Clapham, Peter, Coates, Guy, Fairley, Susan, Fitzgerald, Stephen, Gil, Laurent, García-Girón, Carlos, Gordon, Leo, Hourlier, Thibaut, Hunt, Sarah, Juettemann, Thomas, Kähäri, Andreas K., Keenan, Stephen, Komorowska, Monika, Kulesha, Eugene, Longden, Ian, Maurel, Thomas, McLaren, William M., Muffato, Matthieu, Nag, Rishi, Overduin, Bert, Pignatelli, Miguel, Pritchard, Bethan, Pritchard, Emily, Riat, Harpreet Singh, Ritchie, Graham R. S., Ruffier, Magali, Schuster, Michael, Sheppard, Daniel, Sobral, Daniel, Taylor, Kieron, Thormann, Anja, Trevanion, Stephen, White, Simon, Wilder, Steven P., Aken, Bronwen L., Birney, Ewan, Cunningham, Fiona, Dunham, Ian, Harrow, Jennifer, Herrero, Javier, Hubbard, Tim J. P., Johnson, Nathan, Kinsella, Rhoda, Parker, Anne, Spudich, Giulietta, Yates, Andy, Zadissa, Amonida, & Searle, Stephen M. J. 2013. Ensembl 2013. *Nucleic Acids Research*, **41**(D1), D48–D55.
- Gentleman, Robert. 2004. Reproducible Research: A Bioinformatics Case Study. *Bioconductor Project Working Papers*, **Working Paper 3**(May).

- Goble, Carole A., Bhagat, Jiten, Aleksejevs, Sergejs, Cruickshank, Don, Michaelides, Darius, Newman, David, Borkum, Mark, Bechhofer, Sean, Roos, Marco, Li, Peter, & De Roure, David. 2010. myExperiment: a repository and social network for the sharing of bioinformatics workflows. *Nucleic Acids Research*, **38**(suppl 2), W677–W682.
- Goecks, Jeremy, Nekrutenko, Anton, Taylor, James, & Team, Galaxy. 2010. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. *Genome Biol*, **11**(8), R86.
- Grant, Charles E., Bailey, Timothy L., & Noble, William Stafford. 2011. FIMO: scanning for occurrences of a given motif. *Bioinformatics*, **27**(7), 1017–1018.
- Kent, W. James, Sugnet, Charles W., Furey, Terrence S., Roskin, Krishna M., Pringle, Tom H., Zahler, Alan M., Haussler, & David. 2002. The Human Genome Browser at UCSC. *Genome Research*, **12**(6), 996–1006.
- Lipman, DJ, & Pearson, WR. 1985. Rapid and sensitive protein similarity searches. *Science*, **227**(4693), 1435–1441.
- Liu, Ying, Bai, Kun, Mitra, Prasenjit, & Giles, C. Lee. 2007. TableSeer: automatic table metadata extraction and searching in digital libraries. *Pages 91–100 of: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*. JCDL '07. New York, NY, USA: ACM.
- Machanick, Philip, & Bailey, Timothy L. 2011. MEME-ChIP: motif analysis of large DNA datasets. *Bioinformatics*, **27**(12), 1696–1697.
- Mandal, Ananya. 2013. What is DNA? Online. Available from: <http://www.news-medical.net/health/What-is-DNA.aspx>. Accessed on 10 Oct 2013.
- MEME. The MEME Suite: Motif-based sequence analysis tools. Online. Available from: <http://meme.nbcr.net/meme/>. Accessed on 10 Oct 2013.
- Mesirov, Jill P. 2010. Accessible reproducible research. *Science*, **327**(5964), 415–416.
- Peng, Roger D. 2011. Reproducible research in computational science. *Science (New York, Ny)*, **334**(6060), 1226–1227.

- Ramagopalan, Sreeram V., Heger, Andreas, Berlanga, Antonio J., Maugeri, Narelle J., Lincoln, Matthew R., Burrell, Amy, Handunnetthi, Lahiru, Handel, Adam E., Disanto, Giulio, Orton, Sarah-Michelle, Watson, Corey T., Morahan, Julia M., Giovannoni, Gavin, Ponting, Chris P., Ebers, George C., & Knight, Julian C. 2010. A ChIP-seq defined genome-wide map of vitamin D receptor binding: Associations with disease and evolution. *Genome Research*, **20**(10), 1352–1360.
- Shinyama, Yusuke. 2011. PDFMiner. Online. Available from: <http://www.unixuser.org/~euske/python/pdfminer/>. Accessed on 10 Oct 2013.
- Simplistix. 2012. The xlrld Module. Online. Available from: <https://secure.simplistix.co.uk/svn/xlrld/trunk/xlrld/doc/xlrld.html?p=4966>. Accessed on 5 Oct 2013.
- Smith, Ray. 2007a. An overview of the Tesseract OCR engine. *Pages 629–633 of: Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, vol. 2. IEEE.
- Smith, Ray. 2007b. Tesseract OCR Engine: What it is, where it came from, where it is going. Online. Available from: <https://tesseract-ocr.googlecode.com/files/TesseractOSCON.pdf>. Accessed on 20 Aug 2007.
- Sproat, Richard. 2013. OpenFst Library. Online. Available from: <http://www.openfst.org/twiki/bin/view/FST/WebHome>. Accessed on 20 Oct 2013.
- Thomas-Chollier, Morgane, Sand, Olivier, Turatsinze, Jean-Valéry, Janky, Rekin's, De-france, Matthieu, Vervisch, Eric, Brohée, Sylvain, & van Helden, Jacques. 2008. RSAT: regulatory sequence analysis tools. *Nucleic Acids Research*, **36**(April), W119–W127.
- Woodford, Chris. 2010. OCR (optical character recognition). Online. Available from: <http://www.explainthatstuff.com/how-ocr-works.html>. Accessed on 10 Oct 2013.
- York, Donald G., Adelman, J., John E. Anderson, Jr., Anderson, Scott F., Annis, James, Bahcall, Neta A., Bakken, J. A., Barkhouser, Robert, Bastian, Steven, Berman, Eileen, Boroski, William N., Bracker, Steve, Briegel, Charlie, Briggs, John W., Brinkmann, J., Brunner, Robert, Burles, Scott, Carey, Larry, Carr, Michael A., Castander, Francisco J.,

Chen, Bing, Colestock, Patrick L., Connolly, A. J., Crocker, J. H., Csabai, István, Czarpata, Paul C., Davis, John Eric, Doi, Mamoru, Dombeck, Tom, Eisenstein, Daniel, Ellman, Nancy, Elms, Brian R., Evans, Michael L., Fan, Xiaohui, Federwitz, Glenn R., Fiscelli, Larry, Friedman, Scott, Frieman, Joshua A., Fukugita, Masataka, Gillespie, Bruce, Gunn, James E., Gurbani, Vijay K., de Haas, Ernst, Haldeman, Merle, Harris, Frederick H., Hayes, J., Heckman, Timothy M., Hennessy, G. S., Hindsley, Robert B., Holm, Scott, Holmgren, Donald J., hao Huang, Chi, Hull, Charles, Husby, Don, Ichikawa, Shin-Ichi, Ichikawa, Takashi, Željko Ivezić, Kent, Stephen, Kim, Rita S. J., Kinney, E., Klaene, Mark, Kleinman, A. N., Kleinman, S., Knapp, G. R., Korienek, John, Kron, Richard G., Kunszt, Peter Z., Lamb, D. Q., Lee, B., Leger, R. French, Limmongkol, Siriluk, Lindenmeyer, Carl, Long, Daniel C., Loomis, Craig, Loveday, Jon, Lucinio, Rich, Lupton, Robert H., MacKinnon, Bryan, Mannery, Edward J., Mantsch, P. M., Margon, Bruce, McGehee, Peregrine, McKay, Timothy A., Meiksin, Avery, Merelli, Aronne, Monet, David G., Munn, Jeffrey A., Narayanan, Vijay K., Nash, Thomas, Neilsen, Eric, Neswold, Rich, Newberg, Heidi Jo, Nichol, R. C., Nicinski, Tom, Nonino, Mario, Okada, Norio, Okamura, Sadanori, Ostriker, Jeremiah P., Owen, Russell, Pauls, A. George, Peoples, John, Peterson, R. L., Petravick, Donald, Pier, Jeffrey R., Pope, Adrian, Pordes, Ruth, Prosapio, Angela, Rechenmacher, Ron, Quinn, Thomas R., Richards, Gordon T., Richmond, Michael W., Rivetta, Claudio H., Rockosi, Constance M., Ruthmansdorfer, Kurt, Sandford, Dale, Schlegel, David J., Schneider, Donald P., Sekiguchi, Maki, Sergey, Gary, Shimasaku, Kazuhiro, Siegmund, Walter A., Smee, Stephen, Smith, J. Allyn, Snedden, S., Stone, R., Stoughton, Chris, Strauss, Michael A., Stubbs, Christopher, SubbaRao, Mark, Szalay, Alexander S., Szapudi, Istvan, Szokoly, Gyula P., Thakar, Anirudda R., Tremonti, Christy, Tucker, Douglas L., Uomoto, Alan, Berk, Dan Vanden, Vogeley, Michael S., Waddell, Patrick, i Wang, Shu, Watanabe, Masaru, Weinberg, David H., Yanny, Brian, & Yasuda, Naoki. 2000. The Sloan Digital Sky Survey: Technical Summary. *The Astronomical Journal*, **120**(3), 1579.