VRIJE UNIVERSITEIT AMSTERDAM

Multi-Hypothesis Parsing of Tabular Data in Comma-Separated Values (CSV) Files

by

Till Roman Döhmen

A thesis submitted in partial fulfillment for the degree of MSc Artificial Intelligence

in the
Faculty of Science
Vrije Universiteit Amsterdam

Supervisor: Prof.dr. P.A. Boncz
Second Supervisor: Dr. H.F. Mühleisen
Second Reader: Dr. Rinke Hoekstra

September 2016

VRIJE UNIVERSITEIT AMSTERDAM

Abstract

Faculty of Science
Vrije Universiteit Amsterdam

MSc Artificial Intelligence

by Till Roman Döhmen

Tabular data on the web comes in various formats and shapes. Preparing data for data analysis and integration requires manual steps which go beyond simple parsing of the data. The preparation includes steps like correct configuration of the parser, removing of meaningless rows, casting of data types and reshaping of the table structure. The goal of this thesis is the development of a robust and modular system which is able to automatically transform messy CSV data sources into a tidy tabular data structure. The highly diverse corpus of CSV files from the UK open data hub will serve as a basis for the evaluation of the system.

Preface

Going back from working life to studying and research has never felt like a step back but like an important step forward. I deeply appreciate the new experiences I have made and the new insights I have gained over the past two years. This thesis can not only be regarded as product of seven months research but also as product of an entire study and practical working experience. I especially want to thank my supervisors Peter Boncz and Hannes Mḧleisen, who made possible that I could write my thesis at such a great place as the CWI and who have always been there for me to discuss ideas and to guide me into the right direction. Furthermore I would like to thank Rinke Hoekstra, Benno Willemsem, Sara Magliacane, Alexander Boer, Davide Ceolin and Hadley Wickham for their useful input and discussions. Thank you Mark, Robin, Abe and the other table tennis enthusiasts at CWI - I enjoyed the daily matches at lot! I would also like to thank my roommates in Uilenstede, my friends and my family who supported me during my study and graciously respected my busy schedule. I am especially grateful to my parents, Claudia and Roman, who inspired and supported me in so many ways.

Contents

Abstract ii

Preface iii

Contents iv

1 Introduction 1
- 1.1 Open Government Data 2
- 1.2 Outline 3

2 Background 4
- 2.1 Comma-Separated Values 4
  - 2.1.1 RFC 4180 4
  - 2.1.2 Dialects 5
  - 2.1.3 Parsers 7
  - 2.1.4 Validation Tools 8
- 2.2 Data Frames 8
- 2.3 Tidy Data 10
- 2.4 Open Data Portal *data.gov.uk* 12

3 CSV Parsing Issues 15
- 3.1 Samples from *data.gov.uk* 15
- 3.2 Categorization of Issues 20

4 Problem Statement and Research Questions 23

5 Related Work 27
- 5.1 Data Wrangling Tools 27
- 5.2 Database Normalization 28
- 5.3 Semantic Enrichment of CSV 28
- 5.4 Automatic Table Normalization 30
  - 5.4.1 Applications 31
  - 5.4.2 Input Data Formats 32
  - 5.4.3 Background Knowledge 34
  - 5.4.4 Table Models 35
  - 5.4.5 Analysis Methods 36
  - 5.4.6 Evaluation and Limitations 39

6 Multi-Hypothesis Parser **41**
6.1 Architecture . . . 41
6.2 Parsing/Detection Steps . . . 43
6.2.1 Encoding . . . 44
6.2.2 Dialect . . . 44
6.2.3 Table Area . . . 45
6.2.4 Table Orientation . . . 47
6.2.5 Row Functions . . . 48
6.2.6 Column Functions . . . 51
6.2.7 Wide and Narrow Data . . . 53
6.2.8 Data Types . . . 55
6.3 Quality Assessment . . . 58
6.3.1 Quality Criteria . . . 59
6.3.2 Ranking . . . 61

7 Evaluation **66**
7.1 Table Quality . . . 67
7.2 CSVLint Issues . . . 74
7.3 Amount of Recovered Data . . . 76
7.4 Hypothesis Tree Size . . . 78

8 Discussion **81**
8.1 How large does the hypothesis tree grow? . . . 81
8.2 How reliable can the best parsing hypothesis be identified? . . . 82
8.3 What effect do the table normalization steps have on the table quality? . 83
8.4 Does multi-hypothesis parsing lead to better results than linear parsing? . 84
8.5 Future Work . . . 85

List of Figures **86**

List of Tables **89**

List of Algorithms **90**

A CSV Parser Comparison **91**

B Multi-Hypothesis Parser API **93**

C Multi-Hypothesis Parser Implementation Details **97**

Bibliography **101**

Declaration of Authorship **107**

Chapter 1

Introduction

Comma-Separated Values are a ubiquitous medium for data exchange across various systems and platforms. Users value its simplicity and the possibility of reading and editing it in virtually every environment. However, since the CSV file format has never been officially standardized, it is often tedious to read a CSV files from unknown sources. This is due to the fact that crucial parsing information, such as character encoding, delimiter type, presence of a header line and column data types, are not explicitly given. They either have to be defined manually or inferred from the file itself. When parsing a large corpus of CSV files from various sources, such as the 19k corpus from *data.gov.uk*, manual parser configuration is not an option. On the other hand are detection heuristics for encoding, header etc. never 100% reliable, which consequently leads to parsing failures and misinterpretations, which makes manual configuration of the parser necessary. Another aspect is that CSV files are made to contain exactly one table with an optional header row and subsequent data rows. However, many CSV files contain metadata in the first rows, multiple header rows or even multiple tables. When reading such irregular files with a regular CSV parser, manual post-processing will almost certainly be necessary.

Literature states that 80% of the effort in data analysis is spent on data cleaning and exploration [1]. Our aim is to increase the parsing success and to reduce the effort for manual post-processing and thereby increase the productivity of data analysts dealing with irregular CSV files. This shall also serve the overarching goal to integrate large corpora of CSV files from the web into a coherent, queryable database.

Regular CSV parsers try to determine one most likely parsing hypothesis and in the end the parsing either fails or succeeds. In contrast to this, we intend to generate multiple reasonable parsing hypotheses, parse the file accordingly and decide in the end, based on a quality metric, which parsing result is the best. This requires a structured procedure to generate parsing hypotheses and a method to assess the quality of the parsing result.

The goal of this thesis is to design a multi-hypothesis parsing framework which allows easy generation of parsing hypotheses, and a ranking mechanism, capable of ranking the parsing results by quality. Ultimately a prototypical multi-hypothesis CSV parser will be implemented and evaluated on the heterogeneous and issue affected corpus of CSV files from the open government data portal *data.gov.uk*

1.1 Open Government Data

Open Government Data is an initiative of various governments to open up public sector information and make it available as Open Data on the web. One prominent example is the UK open data portal *data.gov.uk* which covers various domains, such as traffic, weather, geographical information, budgeting, policies, health, education, crime and justice. In many domains public institutions are among the largest data collectors. In contrast to data collected by private companies, government data is funded by public funds and citizens demand in return to get access to the publicly funded data sets. Governments recognized that opening up data brings benefits for them as well and more and more prefer to open it up over keeping it closed or selling it to third parties. As summarized by Janssen et al. [2], the benefits can be grouped into political & social, economic & operational and technical benefits. Major political and social benefits are the increase of democratic accountability, the gained transparency, the self-empowerment of citizens and the resulting creation of trust. Economical benefits lie in stimulation of the development of new citizen services and products around Open Government Data. Additionally, Open Government Data stimulates the comparison and competitiveness between different public institutions. From the operational and technical perspective, benefits lie in the re-use of data (preventing multiple parties from making redundant efforts) and in gaining new insights from integration of different data sets across different parties. Lastly, Open Government Data establishes access to the collective problem solving and analysis capabilities of the public. This leads to new insights rooted in the curiosity and interests of the public and at the same time works as an external quality evaluation of the data itself. Nevertheless does data in most of the cases not produce direct income which makes it, especially for small institutions, harder to justify investments or assign dedicated resources to it. Nevertheless technical competences are required to publish data correctly and in appropriate data formats. Often, the lack of such competences leads to wrong use of data formats, such as CSV, and consequently to data sets which are not machine readable and require significant manual effort to make them ready for data analysis. This poses a hurdle for data consumers to make efficient use of the published data.

Janssen et al. emphasize that data consumers often focus on making use of single data sets, whereas the real value can be found in combining various data sets [2]. Information may appear irrelevant or benign when considered in isolation, but when linked and analyzed jointly it can yield important new insights. Janssen et al. point out that no standard software for processing or integration of open data is available yet.

1.2 Outline

The following chapter will provide background information on Comma-Separated Values, Data Frames and Tidy Data. Chapter 3 will demonstrate current issues of CSV parser by the example of files from *data.gov.uk* and categorize them together with other relevant issues taken from literature. The succeeding Chapter 4 picks up the stated issues, proposes a solution and the resulting research questions. Related work will be outlined in Section 5. In Chapter 6 the concept and implementation aspects of the proposed multi-hypothesis parsing framework will be explained. Chapter 7 outlines the evaluation methods and the respective results. Chapter 8 discusses the research questions and points out possible directions for future work.

Chapter 2

Background

This chapter will provide background information on Comma-Separated Values files and Data Frames, which are commonly used in statistical programming environments as well as the Tidy Data principle and the Open Government Data portal *data.gov.uk*.

2.1 Comma-Separated Values

The first use of Comma-Separated Values dates back more than a decade before the existence of personal computers. The IBM Fortran compiler under OS/360 already supported Comma-Separated Values files in 1972 [3]. Nowadays they are used for platform- and software-independent exchange of tabular data are supported by a large array of systems. As Comma-Separated Values are plain-text files, they always remain human-readable, which makes it easy to deal with them even in the absence of software support and documentation. [4]

2.1.1 RFC 4180

The CSV file format is not officially standardized. However, the IETF Request for Comments (RFC) 4180 documents "the format that seems to be followed by most implementations [in the year 2005]" [4]. Nowadays the RFC 4180 can be regarded as the main reference for CSV reader/writer implementations. In summary, it defines a CSV file as follows: "Each record is located on a separate line, delimited by a line break (CRLF). [...] There may be an optional header line. [...] Within the header and each record, there may be one or more fields, separated by commas. [...] The header will contain names corresponding to the fields in the file and should contain the same number of fields as the records in the rest of the file. [...] Each field may or may not be enclosed in double quotes" [4].

Listing 8 shows an example of a CSV file complying with RFC 4180. The file contains 5 data rows and one header row, separated by *carriage return line feed* (CRLF) line terminators. It has 3 columns, each of which have a dedicated header (*col_name_a*, *col_name_b* and *col_name_c*). The quoting of cell content is optional and only required if the cell contains CSV syntax elements such as line terminators, delimiter or double quotes. Line 4 contains an escaped double quote, which gets escaped by placing another double quote in front of it - not with an backslash ("\") character. Line 6 and 7 actually represent one data row, because the CRLF in line 6 is enclosed by double quotes.

```
1  col_name_a,col_name_b,col_name_c CRLF
2  aaa,bbb,ccc CRLF
3  "aaa","bbb","ccc" CRLF
4  "aaa","b""bb","ccc" CRLF
5  zzz,yyy,xxx CRLF
6  "aaa","b CRLF
7  bb","ccc" CRLF
```

Listing 2.1: RFC4180-conform CSV File

In the original RFC 4180 the default charset was defined as ASCII with characters x20-21, x23-2B and x2D-7E. With the introduction of RFC7111 - a MIME-type definition for CSV fragments - the default charset for CSV documents was changed to UTF-8 [5] [6].

2.1.2 Dialects

Many different variants and re-interpretations of CSV exist, which deviate from the RFC 4180. The most popular variation is the usage of other delimiters such as semicolons (";") or tabs ("\t"). Formats with other delimiters than commas are grouped under the term Delimiter-Separated Values (DSV). However, as the term DSV it is not commonly used, we will refer to Comma-Separated as well as to (Other) Delimiter-Separated-Values as CSV throughout this document. Popular possible variations of the original CSV format are summarized in the following listing:

1. The line ending can be either CRLF (Windows-typical), LF (Linux/Unix-typical) or CR (Mac OS up to version 9). For example:

   ```
   aaa,bbb,ccc LF
   ddd,eee,fff LF
   zzz,yyy,xxx
   ```

2. Not only commas (",") are used to separate fields from each other. Often, semicolons (";") or tabs ("\t") are used instead. For example:

```
aaa;bbb;ccc CRLF
zzz;yyy;xxx
```

3. Not only double quotes (") are used to quote fields, but also single quotes (') and typographic quotes (e.g. ”or ’). For example:

```
'aaa','bbb','ccc' CRLF
zzz,yyy,xxx
```

4. Fields can contain leading or trailing white spaces. For example:

```
"aaa", "bbb" , "ccc" CRLF
"zzz", "yyy" , "xxx"
```

5. The file can contain a row index column, which is headerless. For example:

```
"col_name_a","col_name_b","col_name_c" CRLF
1,"aaa","bbb","ccc" CRLF
2,"zzz","yyy","xxx"
```

6. If quotes are used to enclose fields and another quote occurs inside a field, it must be escaped by another double quote. Alternatively, it is often escaped by a backslash ("\"). For example:

```
"aaa","b\textbackslash"bb","ccc"
```

7. To escape fields which contain line breaks, quotes and delimiter, not only quotes are used. Alternatively, those characters can be escaped by backslashes ("\"). For example:

```
aaa,b\textbackslash,bb,ccc CRLF
zzz,yyy,xxx
```

8. Comments in CSV files are not uncommon. A comment line mostly starts with a hashtag (#) or dollar sign ($). For example:

```
\#this is a comment
aaa,bbb,ccc CRLF
zzz,yyy,xxx
```

CSV files can thus occur in many different variations, which is a challenge for CSV parsers. However, even if a CSV file is RFC 4180-conformant, CSV parsers are challenged to detect the presence of a header line and column data types. This information is in CSV files generally not explicitly given. Although this problem has been recognized and solutions have been proposed (See Section 5.3), the majority of CSV files on the web does not come with any explicit header or data type information.

Table 2.1: Capabilities of different CSV parsers in comparison (empty: not supported, d: fixed default, x: configurable, a: automatic detection possible)

	R			**Python**		
	native	**fread**	**readr**	**native**	**pandas**	**messytables**
Encoding	x	x	a	d	x	x
Line Terminator	a	a	a	a	a	a
Delimiter	x	a	x	x	a	a
Quotechar	x	d	x	x	a	a
Escaping	x		d	x	x	x
Comments	x		x		x	
Header	x	a	x	x	a	a
Index Col	x				x	
Empty Row	d				x	
Field Whitespace	x	x	x	x	x	x
NA Values	x	x	x		x	a
Decimal Point	x	x	x		x	a
Dates	d		x		a	a
Boolean	d	d	d		x	a

2.1.3 Parsers

Producers of CSV parsing libraries are aware of such variations and the under-definition of CSV. They try to take possible variations into account and provide possibilities to allow respective configuration of the parsing process or to detect, e.g., the dialect, header and data types automatically.

The statistical programming language R and the general purpose scripting language Python are popular among data scientists and both provide native support (*utils::read.table*[1], *csv*[2]) and additional third party libraries for CSV parsing. Popular third party libraries for R are the *data.table* library with the contained*fread*[3] implementation and *readr*[4]. Popular packages for Python are *pandas*[5] and *messytables*[6]. Table 2.1 provides an overview of the capabilities of those libraries. A more detailed overview of the available configuration parameter can be found in Appendix A.

When looking at the table it becomes obvious that there are some dissonances about possible variations of CSV files. The *messytables* library appears to be the most advanced library in terms of automation, while it is not as configurable as the *pandas* library. The Python packages *pandas* and *messytables* both use the *CSVSniffer*[7] implementation to

[1]https://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html
[2]https://docs.python.org/2/library/csv.html
[3]http://www.inside-r.org/packages/cran/data.table/docs/fread
[4]https://cran.r-project.org/web/packages/readr/readr.pdf
[5]http://pandas.pydata.org/pandas-docs/stable/io.html#io-read-csv-table
[6]https://messytables.readthedocs.io/en/latest/
[7]https://fossies.org/dox/Python-3.5.1/csv_8py_source.html

detect the dialect. This implementation is also used by the popular command line tool *CSVKit*. An alternative dialect guesser was implemented in the Python library *anycsv*[8]. Generally, the Python libraries are the most advanced and configurable, while they have weaknesses in character encoding detection. The R package *readr*, again, comes with a file encoding guesser. The biggest disagreement among different libraries appears to exist about comments and index columns. Both are not covered by the RFC 4180, and are only partly adopted by libraries in the Python and R community (R native, readr, pandas), while being ignored by others (data.table, Python native, messytables). The above mentioned parsers are using rather simple heuristics, based on RegEx pattern matches, to determine the presence of header rows and the column data types. Especially the non-native parsers are robust to unequal numbers of rows and non-closed quotation marks.

2.1.4 Validation Tools

Apart from parsers, there are also validation tools for CSV files, which are mainly meant to assist data publishers with validation of CSV files before they publish them on, e.g. data portals. The Open Data Institute developed a *ruby*-based CSV validation tool, called *CSVLint*[9]. It can, for example, detect non-UTF8 encoded files, inconsistent data types and unequal number of columns. The tool *Good Tables*[10] from the Open Knowledge Foundation is also able to validate CSV files, it is, however, more specialized towards checking the compliance with a given table schema. The R package *testdat*[11] focuses on sanity checks and automatic repair methods for R data frames, thus after e.g. a CSV has been read. It has been included because it is able to detect encoding issues, which might have been caused by false parsing of the CSV file. It also contains heuristics to detect superfluous whitespaces and data inconsistencies. Table 2.2 provides an overview of the capabilities of all three different libraries and shows that *CSVLint* covers by far most of the issues.

2.2 Data Frames

A Data Frame is a commonly used data structure for tabular data in the statistical programming language R and the Python package *pandas*. In both languages a data frame is a two-dimensional data structure consisting of a rectangular matrix of data cells,

[8] `https://github.com/sebneu/anycsv/blob/master/anycsv/csv.py`
[9] `https://github.com/theodi/csvlint.rb`
[10] `https://github.com/frictionlessdata/goodtables`
[11] `https://github.com/ropensci/testdat`

Table 2.2: Capabilities of different CSV validation tools in comparison.

	CSVLint	Good Tables	testdat
Encoding Issues	x		x
Inconsistant Line Terminator	x		
Ragged Rows	x	x	
Stray/Unclosed Quote	x		
Title Line	x		
Empty Column Name	x	x	
Duplicated Column Name	x	x	
Blank Rows	x		
Duplicated Rows		x	
Field Contains Whitespace	x		
Inconsistant Values	x		x
Outlier Values			x

accompanied by row and column headers, i.e. a frame. In R a data frame consists of a list of n equally long, named, and typed column-vectors which again contain m elements of equal data type and of a row name vector with length m. Pandas data frames are structured similarly, however in contrast to R data frames, they can contain multiple, hierarchical row and column headers. Although row headers have been partially adopted by some CSV parsers in form of row indeces (*R native*, *pandas*), they are neither widely used nor supported and cannot be considered as an integral part of CSV files. Given the overaching aim to integrate the parsing results in a relational database, which does also not employ the use of row header, is it not sensible to make use of the concept of row headers. In whatever tabular shape the input data might be, the goal is to parse it into a regular table, only consisting of a rectangular data matrix and column headers (See Figure 2.1).

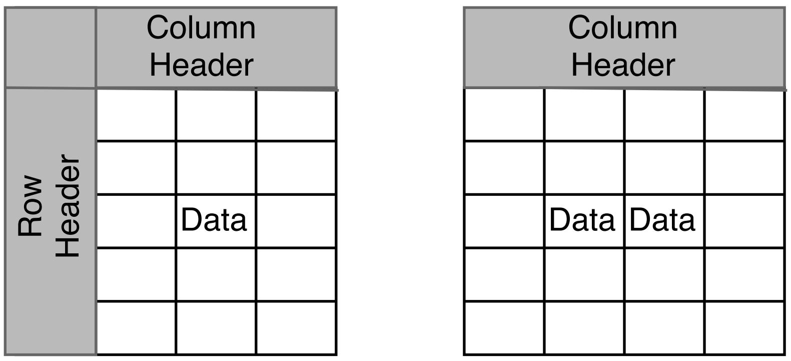


Figure 2.1: Data Frame (left) and a Regular Table (right)

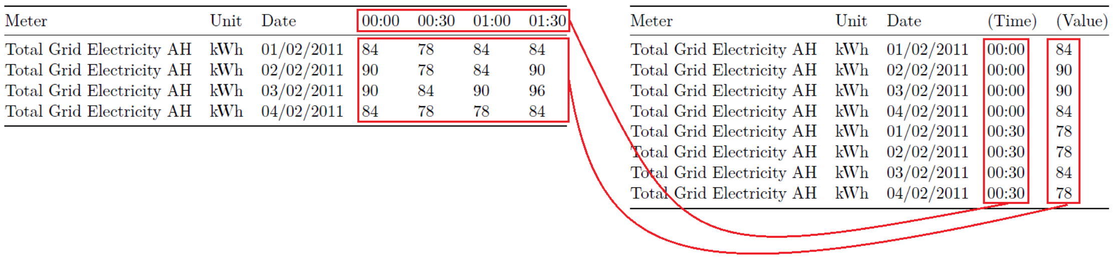
2.3 Tidy Data

Tidy Data is a principle introduced by Wickham [7] which describes the desirable arrangement of tabular data in the context of data analysis. Different software packages for data analysis require different shapes of the data and *Tidy Data* proposes an "easy and effective" arrangement of tabular data, which forms a good starting point for all kinds of analyses. Because the principle is closely related to Codd's 3rd Normal Form [8] it can also be regarded as a good starting point for data integration into a relational database. Tidy Data thus facilitates data analysis and paves the way to data integration at the same time.

The Tidy Data principles:

1. One observation per row
2. One variable per column
3. One observation unit per table

Wickham mentions common violations of the tidy data principle, which he often encountered during his work with statistical data. One of the most common violation is that column headers are actually variables. One could refer to this as (too-)wide data. See Figure 2.2 for the *too-wide* version and the tidy version of the same data set created by *melting* the respective columns and duplicating the other ones.



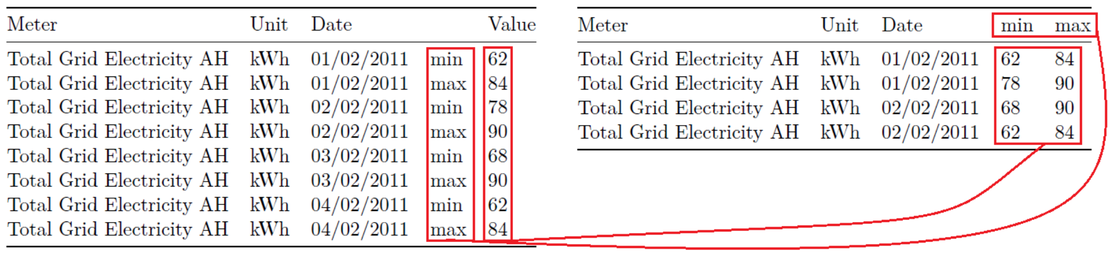
Meter	Unit	Date	00:00	00:30	01:00	01:30
Total Grid Electricity AH	kWh	01/02/2011	84	78	84	84
Total Grid Electricity AH	kWh	02/02/2011	90	78	84	90
Total Grid Electricity AH	kWh	03/02/2011	90	84	90	96
Total Grid Electricity AH	kWh	04/02/2011	84	78	78	84

Meter	Unit	Date	(Time)	(Value)
Total Grid Electricity AH	kWh	01/02/2011	00:00	84
Total Grid Electricity AH	kWh	02/02/2011	00:00	90
Total Grid Electricity AH	kWh	03/02/2011	00:00	90
Total Grid Electricity AH	kWh	04/02/2011	00:00	84
Total Grid Electricity AH	kWh	01/02/2011	00:30	78
Total Grid Electricity AH	kWh	02/02/2011	00:30	78
Total Grid Electricity AH	kWh	03/02/2011	00:30	84
Total Grid Electricity AH	kWh	04/02/2011	00:30	78

FIGURE 2.2: Gathered Wide Data

A second common issue is that multiple variables get stored in the same column. One could also refer to this as (too-)narrow data. Figure 2.3 shows the *too-narrow* version and the tidy version of the same table, which was created by *spreading* the values and moving the "min"/"max" column to the column header.

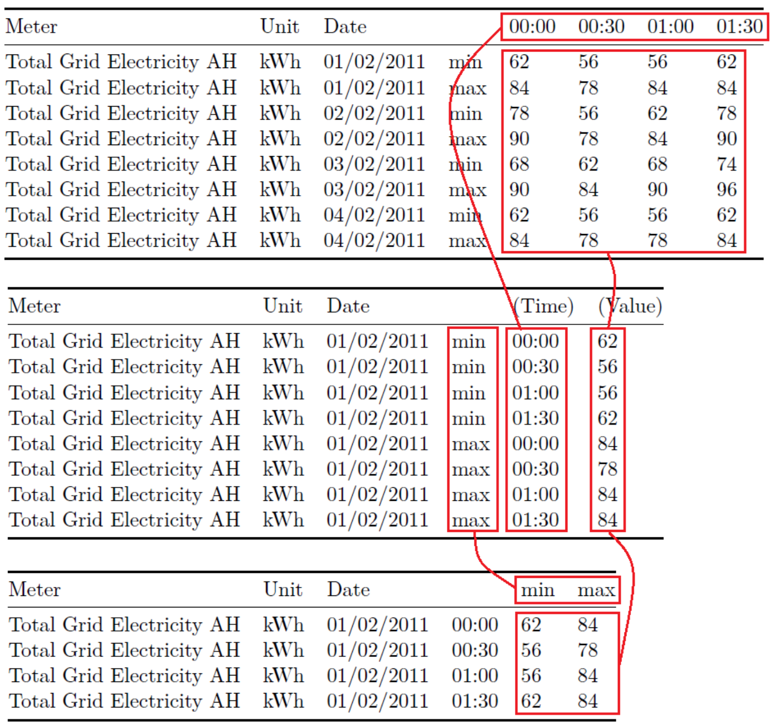
A third and special case is the combination of wide and narrow data where variable names and variables are stored in both, rows and columns, as shown in Figure 2.4. One can refer to this as *mixed* data. To turn this table into a tidy table, first values have to



Meter	Unit	Date		Value
Total Grid Electricity AH	kWh	01/02/2011	min	62
Total Grid Electricity AH	kWh	01/02/2011	max	84
Total Grid Electricity AH	kWh	02/02/2011	min	78
Total Grid Electricity AH	kWh	02/02/2011	max	90
Total Grid Electricity AH	kWh	03/02/2011	min	68
Total Grid Electricity AH	kWh	03/02/2011	max	90
Total Grid Electricity AH	kWh	04/02/2011	min	62
Total Grid Electricity AH	kWh	04/02/2011	max	84

Meter	Unit	Date	min	max
Total Grid Electricity AH	kWh	01/02/2011	62	84
Total Grid Electricity AH	kWh	01/02/2011	78	90
Total Grid Electricity AH	kWh	02/02/2011	68	90
Total Grid Electricity AH	kWh	02/02/2011	62	84

FIGURE 2.3: Spread Narrow Data



Meter	Unit	Date		00:00	00:30	01:00	01:30
Total Grid Electricity AH	kWh	01/02/2011	min	62	56	56	62
Total Grid Electricity AH	kWh	01/02/2011	max	84	78	84	84
Total Grid Electricity AH	kWh	02/02/2011	min	78	56	62	78
Total Grid Electricity AH	kWh	02/02/2011	max	90	78	84	90
Total Grid Electricity AH	kWh	03/02/2011	min	68	62	68	74
Total Grid Electricity AH	kWh	03/02/2011	max	90	84	90	96
Total Grid Electricity AH	kWh	04/02/2011	min	62	56	56	62
Total Grid Electricity AH	kWh	04/02/2011	max	84	78	78	84

Meter	Unit	Date		(Time)	(Value)
Total Grid Electricity AH	kWh	01/02/2011	min	00:00	62
Total Grid Electricity AH	kWh	01/02/2011	min	00:30	56
Total Grid Electricity AH	kWh	01/02/2011	min	01:00	56
Total Grid Electricity AH	kWh	01/02/2011	min	01:30	62
Total Grid Electricity AH	kWh	01/02/2011	max	00:00	84
Total Grid Electricity AH	kWh	01/02/2011	max	00:30	78
Total Grid Electricity AH	kWh	01/02/2011	max	01:00	84
Total Grid Electricity AH	kWh	01/02/2011	max	01:30	84

Meter	Unit	Date		min	max
Total Grid Electricity AH	kWh	01/02/2011	00:00	62	84
Total Grid Electricity AH	kWh	01/02/2011	00:30	56	78
Total Grid Electricity AH	kWh	01/02/2011	01:00	56	84
Total Grid Electricity AH	kWh	01/02/2011	01:30	62	84

FIGURE 2.4: Gathered and Spread Mixed Data

be *gathered* across multiple columns and subsequently the values not belonging to the same variable have to *spread*.

Wickham mentions more violations, like multiple variables per cell and multiple observation units per table, whose detection would most likely exceed the scope of this thesis. Our aim is to create parsing results which are as *tidy* as possible, by detecting *too-narrow*, *too-wide* and *mixed* data. In order to do that, we need to be able to detect: (a) the presence of variables in the header (b) inhomogeneous variables in columns.

Detecting variables in header names is possible insofar as the variables belong to a class of data types which are unlikely to be used as variable names, e.g., dates, times or numeric values. Beyond that, lexical and most likely contextual knowledge is required to discriminate variables (e.g. city names) from variable names. A detection method could also benefit from the fact that all values which will be *gathered* belong to the same variable and should thus have the same data type or even follow a common value distribution.

Detecting inhomogeneous variables in columns is possible as long as the different variables do not have the same data type. If this is the case, we have to check if another column of the table changes synchronously (the column containing the variable names) and if all other columns repeat (the observation names). If different variables are of the same data type (See Figure 2.3) and have been mixed in one column, detection of *too-narrow* data becomes almost impossible. We could still take a regularly repeating text column as indication for a potential variable name column, but again, we would require lexical and contextual knowledge to determine if the column contains actually contains names or just observations. This is a task for which even humans might require additional background knowledge about the purpose and meaning of the data.

It is thus very difficult or impossible to automatically generate perfect Tidy Tables, in which the column headers always contain variables names and columns only contain values belonging to the same variable (See Figure 2.5). We intent nevertheless to exploit the described possibilities, in order to make at least some specific types of tables more *tidy*.

Variable Names			
Obs1	Var1	Var2	Var3
Obs2	Var1	Var2	Var3
...	...	...	...
...	...	...	...
Obs5	Var1	Var2	Var3

FIGURE 2.5: Tidy Table

2.4 Open Data Portal *data.gov.uk*

Data.gov.uk is the official Open Data portal of the Government of the United Kingdom and one of the largest Open Government Data portals on the web. It contains data sets on Environment (8,666), Mapping (4,778), Towns & Cities (4,595), Government (2,746), Society (2,406), Health (1,656), Government Spending (1,615), Education (1,179), Transport (940) and Business & Economy (896). (Numbers from May 2016).

The Open Data Institute, which is associated with the maintenance of the data portal, specified in June 2012 that the minimum requirement for data publications on

Table 2.3: 5-Star Open Data [10]

Level	Description	Type
⋆	"make your stuff available on the Web under an open license"	PDF
⋆⋆	"make it available as structured data"	XLS
⋆⋆⋆	"make it available in a non-proprietary open format"	CSV
⋆⋆⋆⋆	"use URIs to denote things, so that people can point at your stuff"	RDF
⋆⋆⋆⋆⋆	"link your data to other data to provide context"	LOD

data.gov.uk is a *Three Star* level on the 5-start Data Openness Scale. The 5-star Data Openness Scale (See Table 2.3) was introduced by Tim Berners-Lee at the Gov 2.0 Expo 2010 [9]. It is a rating which classifies the openness of different data formats on the web, while promoting the usage of Linked Open Data as 5-Star Open Data. Non-proprietary unlinked data formats, such as CSV, are ranked as *Three Star* Open Data, while proprietary formats like Excel spreadsheets are ranked as *Two Star* Open Data.

Currently, the most common data formats on *data.gov.uk* are: HTML (11,437), CSV (5,368), WMS (4,088), WCS (3,961), XLS (1,893), WFS (1,372), PDF (1,071), XML (839), SHP (302), KML (260), RDF (233). (Numbers from May 2016). Data sets tagged with HTML are typically links to websites which require further manual steps to receive the actual data sets, e.g., by clicking through the website or by filling in a web form. WMS and WCS and WFS are geographical data formats and the amount of real Linked Open Data in form of RDF is with 233 data sets still relatively low.

The two most predominant file formats for tabular data are thus CSV and XLS(X), of which XLS(X) can be considered as legacy data because they are not compliant with the *Three Star* Open Data policy [11].

Data, which was previously published as XLS (and CSV) is now only being published as CSV. One obvious example for this is the data set *Workforce Management Information - HM Revenue & Customs and Valuation Office Agency*[12]. Although CSV is rated as 3-star open data, it does not necessarily mean that CSV data is more accessible than, for example, Excel spreadsheets. In chapter 3 some examples from *data.gov.uk* will be shown, which suggest that data publishers often simply convert Excel spreadsheets to CSV files, by using the Excel CSV export function[13]. This leads to serious misuse of the tabular data structure of CSV files and requires considerable effort on the data consumer side to make use of the data. Moreover, it appears that other ASCII data formats get simply labeled as ".CSV", even if they do not or not only contain comma-separated

[12] `https://data.gov.uk/dataset/workforce-management-information-hmrc-voa`

[13] `https://support.office.com/en-gb/article/Import-or-export-text-txt-or-csv-files-5250ac4c-663c-47ce-937b-339e391393ba`

values. A potential explanation for that is the lack of technical knowledge and time constraints on the publisher's side in combination with the 'Three Star' policy.

Chapter 3

CSV Parsing Issues

Due to flawed CSV writer implementations, manual editing of CSV files, or likewise, CSV files get corrupted. State-of-the-art CSV parsers are nevertheless often capable of parsing the input files properly as long as the errors are not leading to ambiguities. Those parsers, however, build upon the assumption that the file content is a table with one column header row and subsequent data rows. Based on this assumption they try to infer the presence of a header row and column data types. The following section will show examples of CSV files from *data.gov.uk* which contradict this expectation and pose potential hurdles, even for robust CSV parsers. Moreover, we will characterize the encountered issues and reasonably categorize those and other issues mentioned in related literature.

3.1 Samples from *data.gov.uk*

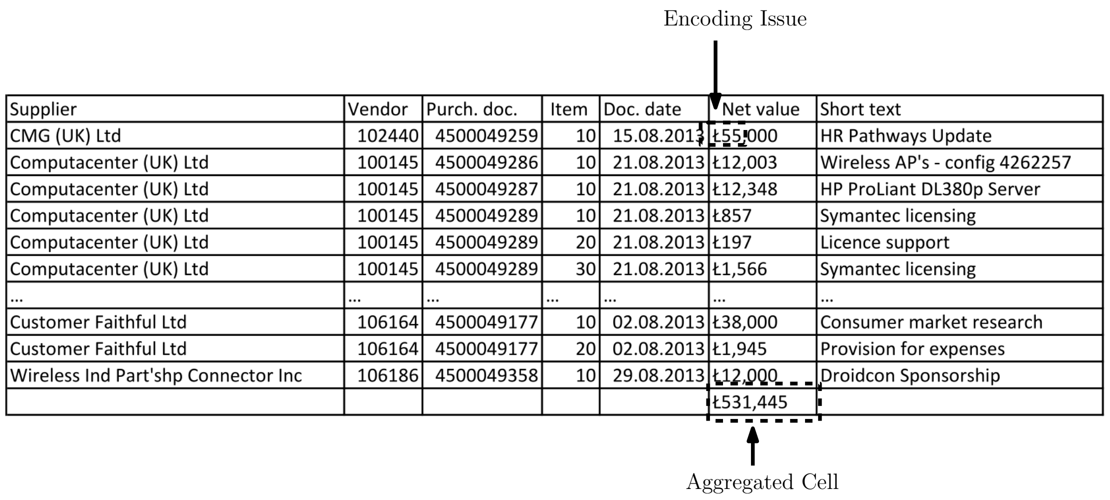
In order to get a better picture of CSV files "in the wild", we have taken a randomized sample of 100 data sets from *data.gov.uk* and analyzed them manually. We have used the *ckan API*[1] to randomly sample links to CSV files, of which 20 had contents other than CSV. Figures 3.1, 3.2, 3.3, 3.4 and 3.5 show simplified snippets of problematic CSV files we encountered in this sample.

Encoding Issue In this case, the £ sign was interpreted as the Polish letter Ł. This issue occurs because the file was encoded in ISO 8859-1, but read with ISO 8859-16 encoding. The code point which in the ISO 8859-1 encoding stands for a Pound signs, stands in the ISO 8859-16 encoding for an L with stroke [12, 13]. When the encoding is not explicitly given, it has to be inferred from the file, which can

[1] `http://docs.ckan.org/en/latest/api/index.html`

lead to false assumptions. If all files would be consistently encoded in UTF-8, this problem could be prevented.

Aggregated Cell The table has an additional row, containing the sum of a specific column. This cell is superfluous and disturbs the rectangular shape of the table. Totals or averages can easily be calculated after the parsing.


Encoding Issue
Aggregated Cell

Supplier	Vendor	Purch. doc.	Item	Doc. date	Net value	Short text
CMG (UK) Ltd	102440	4500049259	10	15.08.2013	Ł55,000	HR Pathways Update
Computacenter (UK) Ltd	100145	4500049286	10	21.08.2013	Ł12,003	Wireless AP's - config 4262257
Computacenter (UK) Ltd	100145	4500049287	10	21.08.2013	Ł12,348	HP ProLiant DL380p Server
Computacenter (UK) Ltd	100145	4500049289	10	21.08.2013	Ł857	Symantec licensing
Computacenter (UK) Ltd	100145	4500049289	20	21.08.2013	Ł197	Licence support
Computacenter (UK) Ltd	100145	4500049289	30	21.08.2013	Ł1,566	Symantec licensing
...	...	...	...	...	...	...
Customer Faithful Ltd	106164	4500049177	10	02.08.2013	Ł38,000	Consumer market research
Customer Faithful Ltd	106164	4500049177	20	02.08.2013	Ł1,945	Provision for expenses
Wireless Ind Part'shp Connector Inc	106186	4500049358	10	29.08.2013	Ł12,000	Droidcon Sponsorship
					Ł531,445	

FIGURE 3.1: Purchase orders over £10,000 from Ordnance Survey, 2013 August return[2]

Visual Table Element The table in Figure 3.2 contains an additional row, which is supposedly meant to improve the readability for humans. For CSV parser, however, it obstructs the data type detection.

Trailing Whitespace Some cells contain trailing whitespace to visually align the values. For further data processing the whitespace is of no use and should thus be stripped off.

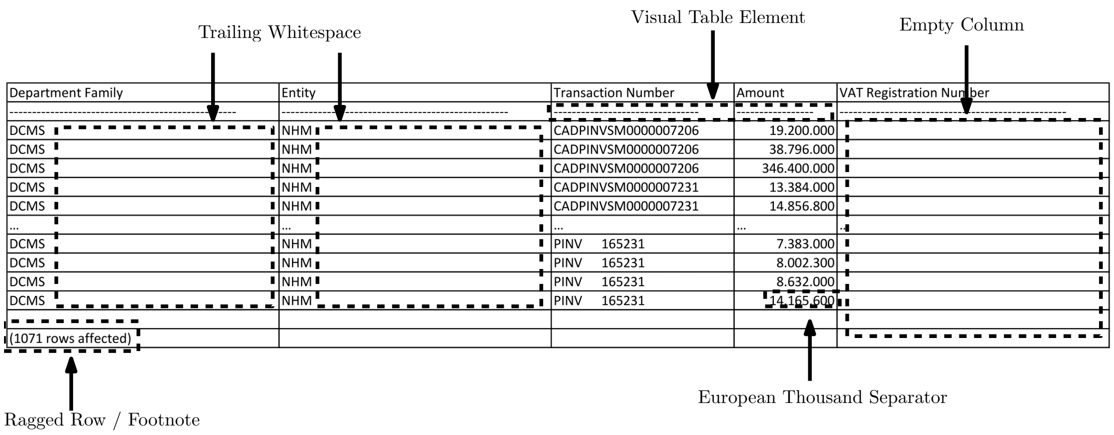
European Thousand Separator European and American thousand separators have the exact opposite meaning. While in Europe dots are used to separate thousand, they are used in America to separate the decimal places. In data processing, usually the American system is used, however, some software packages format numerics according to the locale of the system. This leads to potential ambiguities.

Empty Column Entirely empty columns are usually not useful, however, if they have a header they at least imply the information that a specific variable was not recorded. A user should have the option to either drop or keep such columns.

Footnote Footnotes can interfere with data type detection and should be removed. Nevertheless they could contain relevant information which should be preserved separately.

[2] `https://data.gov.uk/dataset/083cf440-e470-4128-b508-0a2f2e4db50f/resource/fe954daa-a609-4beb-836c-da9c6632a40f/`

Ragged Row From the table it is no longer apparent that the footnote row did not contain any delimiter. Rows which contain a different amount delimiter, i.e. cells, than the other rows, are called *ragged rows*. As we were using a robust CSV parser (*readr*) to read the file, the ragged row was repaired properly.



Department Family	Entity	Transaction Number	Amount	VAT Registration Number
--	--	------------------------------	----------------	--
DCMS	NHM	CADPINVSM0000007206	19.200.000	
DCMS	NHM	CADPINVSM0000007206	38.796.000	
DCMS	NHM	CADPINVSM0000007206	346.400.000	
DCMS	NHM	CADPINVSM0000007231	13.384.000	
DCMS	NHM	CADPINVSM0000007231	14.856.800	
...	...	...	...	..
DCMS	NHM	PINV 165231	7.383.000	
DCMS	NHM	PINV 165231	8.002.300	
DCMS	NHM	PINV 165231	8.632.000	
DCMS	NHM	PINV 165231	14.165.600	
(1071 rows affected)				

FIGURE 3.2: The Natural History Museum expenditure over £500, 2011 December Return[3]

Multiple Tables The file shown in Figure 3.3 contains multiple tables. Header and data type detection will almost certainly fail, when applied without separating the two tables.

Title Title, similar to footer, interfere with the data type detection and should be preserved separately.

Multiple Header Rows The table contains a hierarchically structured header. The header "Travel" refers to the below fields "Air", "Rail", "Taxi/Car" etc. Detecting one header row in this table would not lead to a successful outcome.

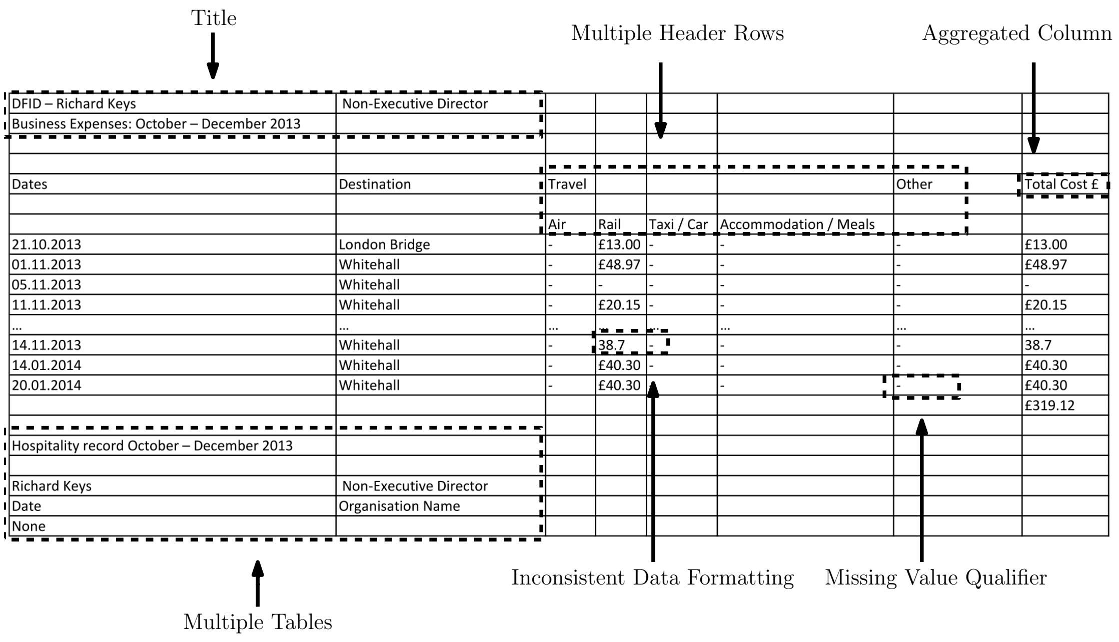
Inconsistent Data Formatting The column contains currency values, which are not consistently formatted. Currency values are usually parsed as plain text, because dealing with such inconsistencies is a hard problem. Conversion of text fields to useful numerics is tedious manual work for data consumers.

Missing Value Qualifier Missing values, i.e. NA values, are often times denoted by special characters such as "-", by expressions like "NA", "NaN", or by special numerics like "-999" or "-1". It can be challenging to distinguish them from valid cell content. CSV parser usually have a fixed or configurable set of expressions which will be interpreted as NA values. If NA values are not properly detected, they can

[3]`https://data.gov.uk/dataset/nhm-over-500/resource/32243819-31d4-49d5-b382-0e35dec92266`

interfere with the data type detection. On the other hand should valid cell content not accidentally be discarded by being classified as missing value.

Aggregated Column Content of those cells can easily be reproduced after the parsing. Because it contains redundant information, it could be discarded. However, an aggregated column might not be reproducible if it contains weighted sums or the like.


Title
Multiple Header Rows
Aggregated Column
Inconsistent Data Formatting
Missing Value Qualifier
Multiple Tables

DFID – Richard Keys	Non-Executive Director						
Business Expenses: October – December 2013							
Dates	Destination	Travel				Other	Total Cost £
		Air	Rail	Taxi / Car	Accommodation / Meals		
21.10.2013	London Bridge	-	£13.00	-	-	-	£13.00
01.11.2013	Whitehall	-	£48.97	-	-	-	£48.97
05.11.2013	Whitehall	-	-	-	-	-	-
11.11.2013	Whitehall	-	£20.15	-	-	-	£20.15
...	...	...	...	...	...	...	...
14.11.2013	Whitehall	-	38.7	-	-	-	38.7
14.01.2014	Whitehall	-	£40.30	-	-	-	£40.30
20.01.2014	Whitehall	-	£40.30		-	-	£40.30
							£319.12
Hospitality record October – December 2013							
Richard Keys	Non-Executive Director						
Date	Organisation Name						
None							

FIGURE 3.3: DFID non-executive directors: business expenses, gifts, travel and meetings, 2011 December Return[4]

Different Variables in Same Column The table in Figure 3.4 a column with different variables. Because those variables are not of the same data type, it is not possible to determine a specific column data type. With a transposed version of the table this would well be possible.

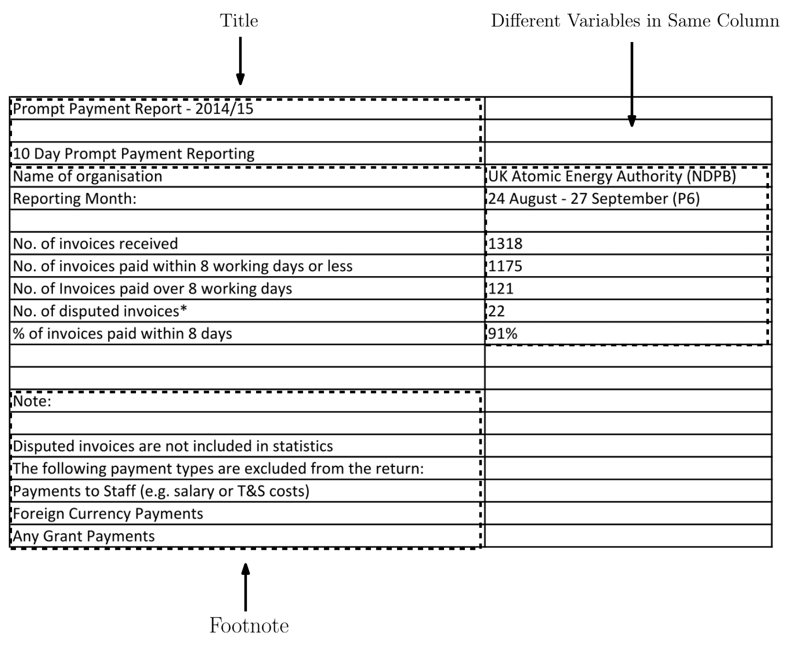
Title As many other tables, this table contains a title row.

Footnote The footnote should be removed from the table structure, but preserved.

Hierarchical Column Headers The table in Figure 3.5 contains, similar to a previously shown table (See Table 3.3), hierarchical column headers.

[4]`https://data.gov.uk/dataset/dfid-non-executive-directors-business-expenses-gifts-travel-and-meetings/resource/012b5b29-dbda-4a8a-8ba5-e858d0b10d51`

[5]`https://data.gov.uk/dataset/ukaea-prompt-payment-data/resource/9d99ff5a-a725-4537-bf15-955c077561a0`

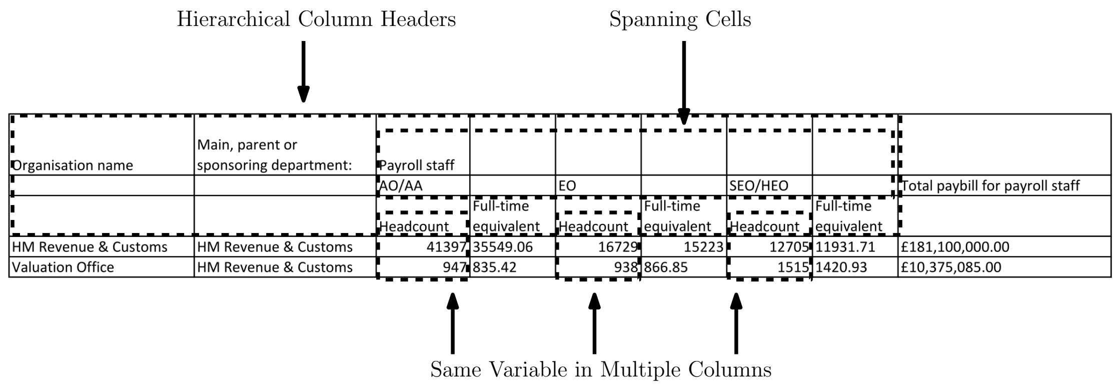

Title
Different Variables in Same Column
Footnote

Prompt Payment Report - 2014/15	
10 Day Prompt Payment Reporting	
Name of organisation	UK Atomic Energy Authority (NDPB)
Reporting Month:	24 August - 27 September (P6)
No. of invoices received	1318
No. of invoices paid within 8 working days or less	1175
No. of Invoices paid over 8 working days	121
No. of disputed invoices*	22
% of invoices paid within 8 days	91%
Note:	
Disputed invoices are not included in statistics	
The following payment types are excluded from the return:	
Payments to Staff (e.g. salary or T&S costs)	
Foreign Currency Payments	
Any Grant Payments	

FIGURE 3.4: UKAEA Prompt Payment Data, Prompt Payment 10 day - Oct 2014[5]

Spanning Cells Spanning cells in headers do often occur along with hierarchical column headers. They can, however, also occur in data fields. The spanning cells are most likely artifacts of exports from source formats which support spanning cells, e.g. Microsoft Excel. For further data processing it would be sensible to infer the original extent of the spanning cell and to duplicate the values to this extent.

Same Variable in Multiple Columns The lowest header row shows the variable names "Headcount" and "Full-time Equivalent", which are being repeated. This is not an issue which affects CSV parser directly, however, in a *tidy* version of the table all "Headcount" and "Full-time Equivalent" variables would be in only two separate columns.


Hierarchical Column Headers
Spanning Cells
Same Variable in Multiple Columns

Organisation name	Main, parent or sponsoring department:	Payroll staff						
		AO/AA		EO		SEO/HEO		Total paybill for payroll staff
		Headcount	Full-time equivalent	Headcount	Full-time equivalent	Headcount	Full-time equivalent	
HM Revenue & Customs	HM Revenue & Customs	41397	35549.06	16729	15223	12705	11931.71	£181,100,000.00
Valuation Office	HM Revenue & Customs	947	835.42	938	866.85	1515	1420.93	£10,375,085.00

FIGURE 3.5: Workforce Management Information - HM Revenue & Customs and Valuation Office Agency - October 2011[6]

[6]`https://data.gov.uk/dataset/workforce-management-information-hmrc-voa/resource/6ef7498d-bf6b-4922-a264-d8e0d3261d45`

TABLE 3.1: Issues identified in 100 random CSV data sets from *data.gov.uk*.

Issue	**Affected Files**
Not UTF-8 Encoded	31
Metadata (Titles, Footnotes, ...)	22
Empty Rows or Empty Columns	18
Aggregated Cells/Rows/Columns	13
Download Link Broken or File Empty	11
Missing Value Qualifier	11
File Not CSV (e.g. HTML, ZIP)	8
Inconsistent Data Formatting	4
Spanning Cells (Omitting Duplicates)	4
Multiple Header Rows	4
Multiple Tables per File	3
Column Headers contain Variables	3
Row Numbers in first Column	2
Visual Table Elements	2
Data Footnotes (*)	2

The tables described were not the only ones affected with issues. Table 3.1 summarizes the most frequent issues we have identified Out of the entire sample, 31 were not UTF-8 encoded and 22 contained metadata such as titles, footnotes, etc. This is largely in line with the results from Ermilov et al.[14]. Only 29 retrieved files could be considered as issue-free. Most notably, all files could be successfully read with either *messytables* or *readr*.

3.2 Categorization of Issues

Ermilov et al.[14] collected and classified issues they encountered when working with CSV files from the data.gov.uk data set. Another source for practical issues with CSV files in Open Government Data repositories can be found at the data.gov Wiki [7]. We created a more comprehensive overview of issues related to tabular data in CSV files, based on those sources, a more general overview of data quality issues by Singh et al. [15], the Tidy Data principles by Wickham, the RFC 4180 and issues we encountered (see Table 3.2).

The first category groups issue related to retrieval of the data. Linked resources have to be online and the file has to be an actual CSV file, not HTML, XLS or ZIP. Those issues are in particular mentioned in the *data.gov Wiki*, but were also encountered by us. The category "Table Parsing" refers to problems with reading the tabular data itself. Part of reading are detection of the right encoding and the dialect. The detection itself

[7]`http://data-gov.tw.rpi.edu/wiki/Current_Issues_in_data.gov`

Table 3.2: Issues Related to Tabular Data in CSV Files

Level	Issue	RFC4180 Non-conformity	CSVLint	Ermilov et al.	data.gov Wiki	Tidy Data (Wickham)	Singh et al.
Retrieval	Link Broken or File Empty		x		x		
	File Not CSV (e.g. HTML, ZIP)		x		x		
Table Parsing	Not UTF-8 Encoded	x	x		x		
	Delimiter other than ','	x	x		x		
	Quoting Character other than "	x	x				
	Inconsistent Quoting Character						
	Stray/Unclosed Quote	x	x				
	Unequal Number of Columns	x	x	x			
Table Structure	Empty Rows or Empty Columns		x	x			
	Visual Table Elements	x					
	Row Numbers in First Column	x			x		
	Metadata (Title, Footnotes, ...)	x	x	x	x		
	Multiple Tables per File	x		x			
	Spanning Cells (Omitted Duplicates)	x		x			
Header	Missing Header	x	x	x	x		
	Multiple Header Rows	x		x			
	Duplicated Header		x	x			
	Spanning Cells in Header	x		x			
	Group Header	x					
Data Arrangement	Table in Horizontal Orientation	x					
	Aggregated Cells/Rows/Columns						
	Column Headers contain Variables					x	
	Mult. Observational Units in Table					x	
	Different Variables in one Column		x			x	
	Multiple Variables per Cell					x	x
Data Formatting	Leading/Trailing Whitespace	x	x				
	European Decimal Separators						
	Inconsistent Data Types/Formatting		x				x
	Missing Value Qualifier						x
	Data Footnotes (*)	x					
Data Quality	Misspelled Data						x
	Multi-Purpose Fields						x
	Data Duplication			x			x
	Data Incompleteness						x
	Outlier Values						x
	Values Stray form Field Description						x

poses an issue, if the encoding and dialect are unknown and not in line with the RFC 4180. A second problem arises when the dialect is not consistently used or corrupted. This can lead to results with ragged rows, fields containing multiple variables or simply cause crashing of the parsing process. The next category of problems is Misuse of the Table Structure. We saw in many cases that the table contained metadata such as titles or comments, empty columns, and content which was intended to improve the readability. Issues concerning the header, like complex tables with multiple header rows, spanning cells in header and group header, were grouped in a separate category because the header is an integral part of the table. Issues belonging to the category of "Data Arrangement" deal with functional aspects of specific cells, which belong to the table, but are superfluous or incorrectly positioned. The category "Data Formatting" groups issues which complicate the data type detection of columns, such as inconsistent data formatting and value qualifier. The last category contains general data quality issues which are rooted in the data itself and are completely independent of the form in which they are stored.

Chapter 4

Problem Statement and Research Questions

If we consider the distribution of file sizes on the open data portal *data.gov.uk* (Figure 4.1), it appears that the majority of files are fairly small, while the biggest amount of data is stored in a few very large files. 92.5% of all files are smaller than 1MB and make up a small, but not neglectable, fraction of the total corpus size of 0.75%. This part of the corpus contains many loose chunks of information across various topics and institutions, which has to our knowledge never been integrated and analyzed as a whole, which could yield important new insights as emphasized in Chapter 1.

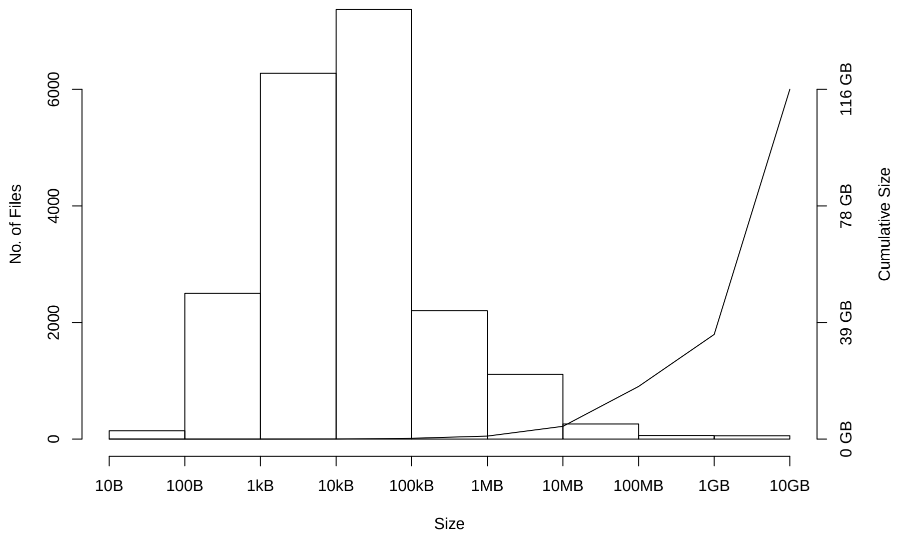


FIGURE 4.1: Size Distribution of CSV files on *data.gov.uk*

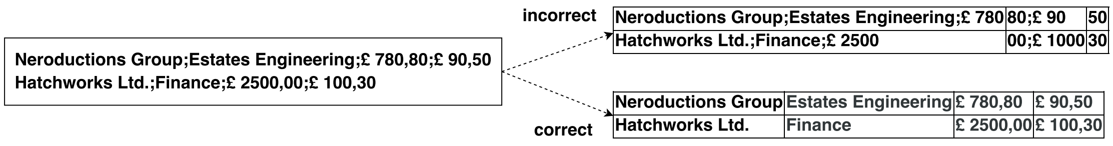
This part of the corpus is probably the most interesting, but also the most problematic one. It contains too many files to check and parse them manually on a file-by-file basis.

Table 4.1: CSV Parsing success and remaining CSVLint issues

Parser	**Success Rate**	
R native	56/80	
data.table	75/80	
readr	79/80	
messytables	80/80	
Parser	**Files with CSVLint Issues**	
	before parsing	after parsing
readr	56	31
messytables	56	20

On the other hand, the files are so heterogeneous and error-prone that current parsers are not capable of automatically and reliably reading tabular data from them. For an initial assessment of the quality of current CSV parser, we parsed the previously mentioned sample from *data.gov.uk,* with different libraries and wrote the parsing result back to a RFC 4180-conform CSV. We did not only assess the success of the parsing itself, but also the number of issues before and after the parsing with CSVLint, which detects issues, such as title rows, empty rows and inconsistent values in columns. The results in Table 4.1 show that even the most reliable parsers *messytables* leave behind unresolved issues in about 1/3rd of the cases. This suggests that although the parsers parse the CSV file without errors, they do not necessarily produce usable output.

The automatic detection mechanisms for, e.g., dialect and header rows, of current CSV parsing libraries largely rely on heuristics, which assume that the content of the CSV files are regular tables with an optional header row and succeeding data columns and potential metadata above the header row. Chapter 3 shows that this assumption does not always hold and that many CSV files also contain metadata in form of footnotes, contain multiple tables per file or tables with multiple header rows. Moreover, the reliability of the heuristics in many cases depends on the number of rows. Thus, the smaller the files, the more unreliable the heuristics become. Figure 4.2 shows a very small semicolon-delimited file, which is ambiguous, because the comma could either be a decimal separator or the cell separator. In fact, all of the tested parsers parsed this file incorrectly.



```
Neroductions Group;Estates Engineering;£ 780,80;£ 90,50
Hatchworks Ltd.;Finance;£ 2500,00;£ 100,30
```

Neroductions Group;Estates Engineering;£ 780	80;£ 90	50
Hatchworks Ltd.;Finance;£ 2500	00;£ 1000	30

Neroductions Group	Estates Engineering	£ 780,80	£ 90,50
Hatchworks Ltd.	Finance	£ 2500,00	£ 100,30

Figure 4.2: Simple Example of an ambiguous CSV File.

The parsing process in current CSV parsers is a linear chain of parsing steps (see Figure 4.3, left). If one heuristic fails and makes a wrong assumption, it will likely have a negative influence on the subsequent detection steps. The succeeding steps will fail as well or at least not make up for previous mistakes. And the more steps are added to the chain, the more likely it becomes that at least one of them fails.

One goal of this thesis is to develop a CSV parser with a broader understanding of tabular data structures (see Figure 4.3, right), which certainly requires adding more steps to the process.

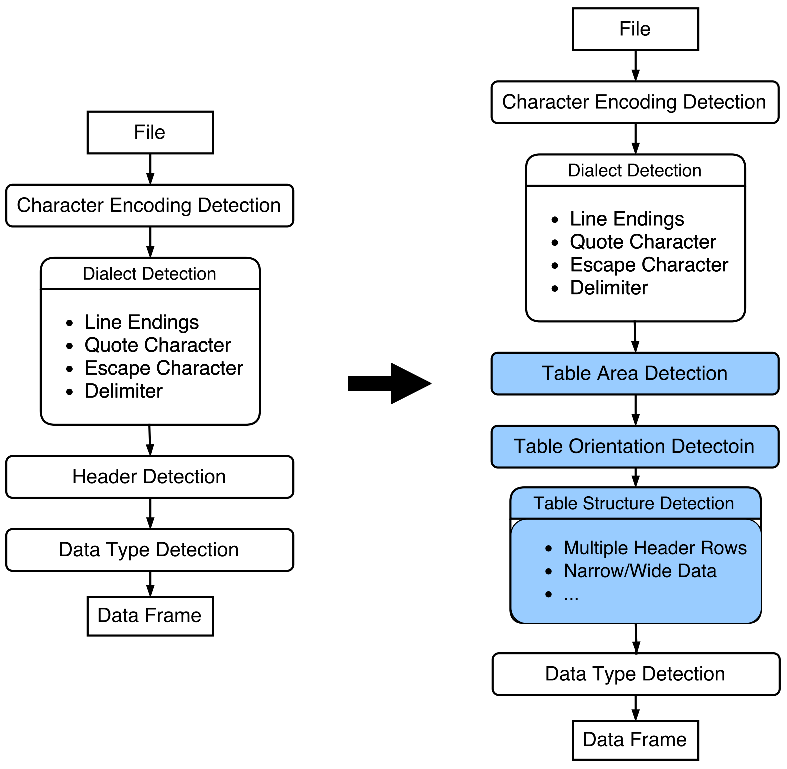


FIGURE 4.3: Parsing process without and with enhanced Table Structure Detection.

In case of the example shown in Figure 4.2 the dialect detection would require information about the data type consistency in order to make the correct decision. On the other hand, the data type detection requires proper data cells, hence a successful dialect detection is required in the first place. This leads to a circular dependency between dialect detection, data type detection and the intermediate steps along the way. Because such a look-ahead is not feasible, current CSV parsers optimize their parsing decisions only locally, which leads to previously described problems.

In order to circumvent such circular dependencies, we propose a solution which regards the parsing process from a more holistic point of view. Instead of creating a linear chain of locally optimal decisions, we aim at building a tree of possible parsing decisions, traverse it and decide in the end for the globally best parsing result (See Figure 4.4). We hypothesize that the globally best parsing result can be identified based on result features, such as the shape of the resulting table and column-wise consistency of values.

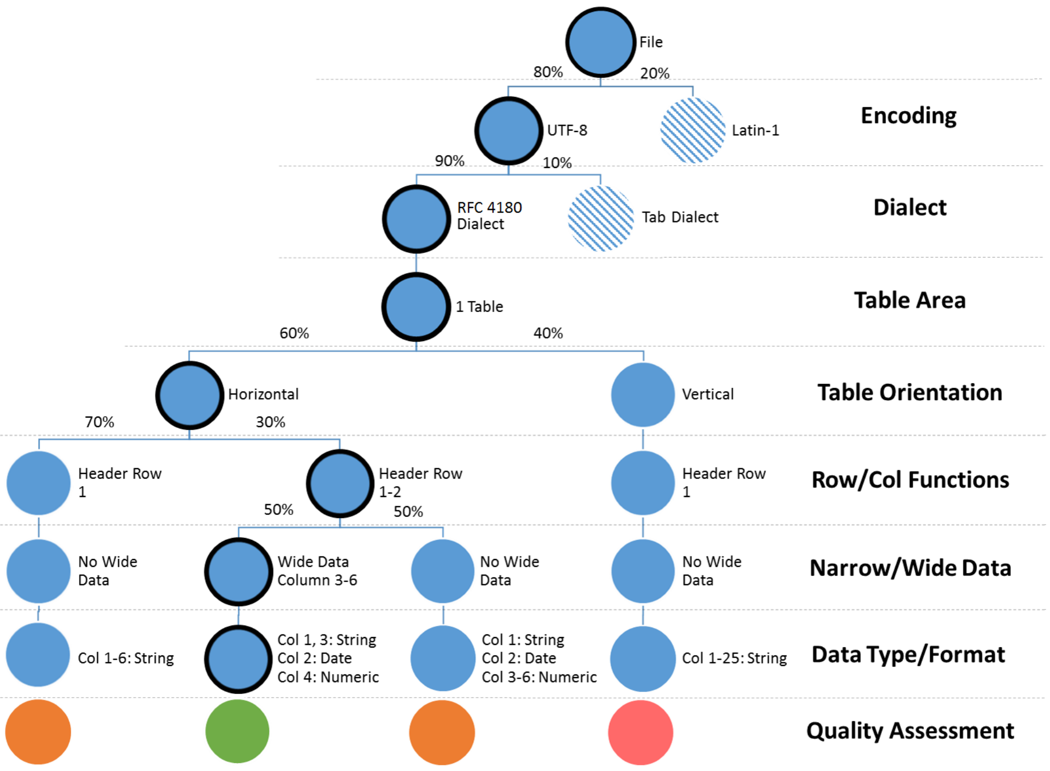


FIGURE 4.4: Multi-Hypothesis Tree

While linearly chained parsers are rather optimized towards computational efficiency, the multi-hypothesis parsing will be optimized towards result quality. However, the size of the parsing tree is subject to exponential growth, which could quickly exceed practical limitations of computing. We aim at counteracting this issue by creating only a limited amount of reasonable hypothesis per step and optionally pruning the hypothesis tree from very unlikely hypotheses.

Ultimately, following research question arise:

1. *How large does the hypothesis tree grow?*
2. *How reliable can the best parsing hypothesis be identified?*
3. *What effect do the table normalization steps have on the table quality?*
4. *Does multi-hypothesis parsing lead to better results than linear parsing?*

Chapter 5

Related Work

State-of-the-art CSV parsers have already been discussed in Chapter 2. Therefore the related work section will mainly focus on the table normalization aspect. Since normalizing table structures could be considered as a data pre-processing step, we investigated how and to what extent data wrangling tools are capable of dealing with such problems. On the other hand, parts of the problem of non-tidy data also occur in databases. We will therefore discuss if methods from the database research field could be relevant. As there are many practical solutions for creating semantically enriched data based on CSV files, which assumingly require a certain understanding of the table structure, we considered those solutions as well. However, as none of the previously mentioned directions provided satisfying solutions to the problem, we eventually focused on more general research on table interpretation and normalization.

5.1 Data Wrangling Tools

The tool *Comma-Chameleon*[1] combines *CSVLint* validation capabilities with the ability to edit the tabular content and manually fix issues in the file via a graphical user interface. This tool is helpful to fix small issues in individual files, but not practical to make structural changes to the content or to process large corpora of files. Tools like *OpenRefine*[2] provide a graphical user interface to facilitate manual reviewing and cleansing of tabular data as well. Those tools provide functionality for, e.g., merging and splitting of rows and columns, mass renaming of cell content and for reviewing of value distributions, in order to identify spelling mistakes and outlier values. *Trifacta Wrangler* takes it a step further and aims at reducing the manual effort by learning from

[1]`https://github.com/theodi/comma-chameleon`
[2]`https://github.com/OpenRefine/OpenRefine`

the users' behaviour, by using the principle of predictive user interaction [16]. It basically offers the same functionality as *OpenRefine*, but proposes useful transformations to the user. The more files of the same kind are processed, the more accurately it can predict suitable data transformations based on previous user input and the easier the wrangling process gets for the user. In summary, the focus of those tools is, however, on very generic tasks which could be used to normalize the tabular shape of input data, but are not specifically made to do that. Although, e.g., *Trifacta Wrangler* tries to support the user by learning from his behaviour, the functionality is only useful if the input data is rather homogeneous. Above that, those tools are rather focused on manually guided data processing on file-by-file basis, than on automatic processing. For the purpose of automatically normalizing a large set of heterogeneous tables they can thus not be considered as useful.

5.2 Database Normalization

The problems of table canonicalization and "tidying" [7] are generally related to database normalization, since it implies a table normalization up to Codd's 3rd Normal Form. Solutions which are able to normalize a table, as shown by Du et al. [17] and Yazici et al. [18], rely on a user-provided set of functional dependencies between columns. Based on this, they can automatically normalize a relational table up to Codd's 3rd Normal form. Since functional dependencies between columns in our data are not known, the methods in itself are not useful for us. However, algorithms for detection of functional dependencies between columns exist (see Papenbrock et al. [19]), but they create large numbers of functional dependencies and leave the open research question how to make sense out of them, with e.g. ranking methods. They can thus not yet be used in practice for detection of sensible functional dependencies. An alternative approach was presented by Cunha et al. [20], who aimed at normalizing functional dependencies in Spreadsheets, by inferring functional dependencies from cell functions, such as *sums*. Neither can this approach be considered as useful, since the CSV file format does not employ the concept of cell functions.

5.3 Semantic Enrichment of CSV

Various research has been done towards lifting tabular data, in particular CSV, to semantically richer data formats, such as RDF. The Resource Description Framework (RDF) is a method for conceptual description and modelling of information in web resources [21], which is based on the idea of expressing information in form of subject-predicate-object *triples* (i.e. entity-attribute-value) and allowing to link those concepts and relations across different data sets. By using a consistent notation for the same entities, attributes, and sometimes values, a web of linked data emerges, which facilitates data integration and allows for querying and even logical reasoning across data from various origins. *Knowledge bases*, such as Wikidata [22], contain many often-used concepts and relations and function as hub and entry point to the linked data web. Using the notation found in those knowledge bases, basically connects a data set to the whole linked data web. On the other hand, knowledge bases can serve as source for common knowledge. Wikidata, for example, contains approximately 19.5 million (August 2016)[3] items like persons, locations, events etc. The Wiki of the W3C (World Wide Web Consortium) [4] lists existing solutions which are able to convert CSV files to RDF triples. Those solutions, however, largely rely on user-defined conversion rules in domain specific languages, which define the mapping of concepts, relations and entities in a table to a knowledge base. Given this definition, the transformation can be executed automatically. Mulwad et al. [23], Sharma et al. [24] and Limaye et al. [25], again, aimed at creating those mapping automatically, by inferring the RDF concepts from the table content. These approaches use Probabilistic Graphical Models to achieve a global optimum of label assignment likelihoods. However, they only work properly if the concepts in the table are so general that they occur in public RDF knowledge bases and on the other hand are not too ambiguous. Furthermore, all of the approaches are built on the assumption that the input data is a regular table, with one header row and succeeding data rows, which, as shown in Chapter 3, does not apply to a significant amount of real-world CSV tables.

In order to achieve precision and efficiency for CSV to RDF conversion, Ermilov et al. [14] proposed a crowd sourcing approach. They developed a crowd-sourcing platform for the data portal *datacatalogs.org*, which enables people to work together on semantically enriching open data, with the long-term goal of integrating the data portal into a "distributed data warehouse". In their study, Ermilov et al. analyzed a sample of CSV files from the open government data portal *PublicData.eu* and came to the same conclusion about the poor condition of CSV files on open government data portals as we did. Ermilov et al. have been the only authors, who recognized the necessity for a cleaning process of the table, before taking any further steps. They identified a main problem, namely leading and trailing metadata, which were observed in 20% of all tables. In the end, their cleaning approach did, however, not succeed and they stated: "In such cases [leading/trailing metadata], the location of the header is currently not

[3]`https://www.wikidata.org/wiki/Wikidata:Main_Page`
[4]`https://www.w3.org/wiki/ConverterToRdf#CSV_.28Comma-Separated_Values.29`

properly determined" [14]. Also Meroño-Peñuela et al. encountered the problem of non-normalized tabular data in the context of Spreadsheet to RDF conversion of historical census data [26]. They developed a tool, called *TabLinker*[5], which facilitates the conversion process by understanding the table structures, based on coloured markups inside the Spreadsheet. Those markups, have to be manually created, which is not practical for large corpora as the one from *data.gov.uk*.

All tools mentioned so far have in common that they are typically used on the data consumer side. This is only necessary, because the original data lacks meta-information on its structure and content. The W3C *CSV on the Web Standard* [27] and the OKNF *Data Packages*[6] target exactly this problem. Both define a JSON schema for meta-information, which are added to the CSV file. The JSON shall then contain information about the used CSV dialect, the position of the header row, contained entities and their data types. The distinctive feature of *CSV on the Web* is that it allows for LOD-typical annotation of column content, which facilitates subsequent conversion from CSV to RDF. Additional meta-information on CSV files would certainly facilitate the parsing of CSV files, but for data which has already been published without meta-information, which holds for the vast majority of data in the *data.gov.uk* corpus, it is of no use.

5.4 Automatic Table Normalization

The techniques described in this section aim at reading and interpreting tables without the need of user input. The problem of table normalization is closely related to table interpretation because in order to normalize a table, the content has to be understood to a certain extent. Generally, table normalization can be considered as a side-product of table interpretation, because as soon as the content of the table is completely understood, it can be represented in any desired way. Table interpretation lies at the intersection of document analysis, information retrieval and information extraction. Hurst [28] proposes a more specific subdivision of the interpretation task into a graphical, physical, functional, structural, and a semantic level. We mainly focus on the physical, functional and structural level, which bridges the gap from semi-structured data to normalized tables, while the graphical level refers to document analysis tasks and the semantic level to ontology mapping. The physical level refers to detection of the actual shape of the table, and the breakdown into rows, columns, cells, spanning cells etc. The functional level comprises the discrimination of access cells, i.e. row/column header, from data cells. The structural level refers to detection of access paths to data cells, i.e., which access cells are indexing which data cells - or simply, how would a human read the table?

[5] `https://github.com/Data2Semantics/TabLinker`

[6] OKNF: Open Knowledge Foundation, `http://dataprotocols.org/data-packages/`

A slightly different notion of related work on extraction of tabular data from lists was also included in this section. This covers the other side of the spectrum of normalization tasks, in which the complexity of the table structure does not have to be reduced, but rather has to be recovered. Tabular data in CSV has problems from both sides of the spectrum, because on the one hand the CSV format itself can be corrupted, which leads to cases in which the proper table shape has to be recovered in a way which is closely related to table extraction from lists. On the other hand, if a CSV table can be properly read, it can still contain complex tabular table structures, which require regular table normalization procedures.

Comprehensive background information on table theory is given by Wang [29], of which relevant aspects are included in the following sections. Literature reviews from Nagy et al. [30], Silva et al. [31] and Embley et al. [32] provide an overview of research work on table interpretation, but are not up-to-date. We therefore aim at giving a more recent overview and include especially practical solutions in our review.

5.4.1 Applications

The solutions which will be discussed have different purposes, but have in common that they aim at reading tabular data which was intended to be read by humans and not by machines.

The primary purpose of the table processing system *XTable* by Wang [29] is to de-couple the logical and physical representation of a table to facilitate table manipulations, by executing them on a logical abstraction of the physical table. Although the table content gets abstracted, it does not get interpreted or automatically normalized. The approach was hence not further considered.

However, most of the reviewed solutions aim at interpreting the content of a table in order to make it queryable and integratable with other data sources. Hurst proposes the end-to-end table interpretation system *TabPro* [28], which uses a very early version of the lexical database *WordNet*, in order to achieve a semantic understanding of the table content. The table interpretation system *TARTAR*, by Pivk et al. [33], is another end-to-end table interpretation system which uses the same conceptual framework as Hurst but utilizes more recent Semantic Web technologies. Seth et al. [34] proposed the system *TANGO*, aiming at integration of heterogeneous tabular data into databases. Also, the system *Senbazuru*, by Chen et al., was created to facilitate the integration of heterogeneous tables into a "Spreadsheet Database Management System" [35]. Google

WebTables [36] was the first system to apply table interpretation technologies at web-scale, on a corpus of 154M tables. Ultimately, the system was used to answer search queries with relevant tables instead of relevant websites.

The approaches by Adelfio et al. [37] and Pinto et al. [38] do not aim at complete table interpretation, but are rather methods which could be used to improve systems such as *WebTables*. Eberius et al. [39], again, proposed a system called *DeExcelerator*, which is aimed at normalization of table structures in order to facilitate data analysis and integration tasks. This is largely in line with our aim.

Furthermore, the approaches of Elmeleegy et al. [40] (*ListExtract*) and Chu et al. [41] (*TEGRA*) were considered, which aim at extracting tabular data from lists.

5.4.2 Input Data Formats

The mentioned approaches widely rely on specific features and cues, which are not provided by every data type. Table 5.1 gives an overview of which data type provides which kind of features.

TABLE 5.1: Features of different Data Formats on table, text and cell level

Level	Feature	HTML	Spreadsheet	CSV	ASCII	List
Table	Well-defined Rows	x	x	x	x	x
	Well-defined Columns	x	x	x		
	Line-Art	x	x			
Cells	Spanning Cells	x	x			
	Style (e.g. Background Color)	x	x			
Content	Well-defined Characters	x	x	x	x	x
	Style (e.g. Font)	x	x			
	Data Type		x			
	Formulas		x			

Functional and structural information, i.e. a precise distinction of access and content cells or explicit reading paths, are provided by none of the data types. The data formats nevertheless contain the sufficient amount of information to make the content readable and interpretable for humans. The necessary information is, hence, implied in the content or the styling features of the data formats. Spreadsheets contain the largest variety of features, which could potentially be used to infer the functional and structural properties of a table. Cell functions were already shown by Cunha et al. [20] to be useful for normalization of tables in Spreadsheets. Due to the fact that cell functions are not contained in CSV files, approaches using them were disregarded. Eberius et al. and Adelfio et al. used spreadsheet input without making use of the Spreadsheet-specific features, which is why they were included in the review.

Table 5.2 provides an overview of which input formats were used by the different authors.

Table 5.2: Data Formats, used by different authors

Format	Adelfio et al. [37]	Pinto et al. [38]	TARTAR [33]	WebTables [36]	Senbazuru [35]	DeExcelerator [39]	TANGO [34]	ListExtract [40]	TEGRA [41]	TabPro [28]
HTML	x		x	x	x	x	x			x
Spreadsheet	x					x				
CSV						x				
ASCII Table		x								
List								x	x	

Only Eberius et al. also considered CSV files as input format. One possible explanation that CSV is not frequently mentioned in literature on table normalization is, that one could assume that CSV files are already in a normalized tabular shape and that sophisticated table processing of CSV is not required. However, as shown in Chapter 3, this assumption does not hold. Most of the other approaches use HTML as input format. As shown by Cafarella et al., the web contains billions of HTML tables of which an estimated fraction of 1.1% contain actual relational tables [36]. HTML tables are thus ubiquitous. The HTML format facilitates the detection of tables in documents, because HTML has a specific markup for tables. The table markup is, however, frequently misused for other purposes such as layouting, which explains why only a fraction of 1.1% of the tables on the web are actual relational tables. Regardless, the HTML format does provide a rich set of features, such as different font styles, line-art, and spanning cells, which were used by Adelfio et al. [37], Pivk et al. [33], Chen et al. [35] and Hurst [28] as cues for the interpretation of the table structure. Because those features are not provided by CSV, we have to assess to which extent those features were used and if the approaches of the respective authors would nevertheless be applicable to CSV data.

Table 5.3 provides an overview about which types of cues and features were used in different approaches. All of the approaches considered the distinction between numeric and textual cell content and, most notably, not all approaches used layout features. Seth et al. even used CSV as intermediate format, stating: "Even though [...] the format loses most appearance features (layout and cell formatting), the remaining structural and alphanumeric information is sufficient for [...] algorithmic analysis" [42]. This suggests that cell content alone could nevertheless be sufficient to infer the structure of the table. Pinto et al. [38] used ASCII tables as input, which are even less structured than

TABLE 5.3: Features of the Input Data, used in different approaches. The data type of the content was not necessarily explicitly provided by the input data, but got inferred with Regular Expressions. Background knowledge matches are used by approached which make use of background knowledge, as outlined in Section 5.4.3.

Feature	Adelfio et al. [37]	Pinto et al. [38]	TARTAR [33]	WebTables [36]	Senbazuru [35]	DeExcelerator [39]	TANGO [34]	ListExtract [40]	TEGRA [41]	TabPro [28]
Table Shape (Rows & Columns)				x	x					
Spanning Cells	x		x		x					x
Line-Art			x							x
Content Style (e.g. Font)	x				x					
Content - Text, Numeric	x	x	x	x	x	x	x	x	x	x
Content - Date, Currency, etc.	x		x			x		x	x	x
Content - Punctuation, Indentation		x						x	x	
Content - Keyword Matches	x	x			x					
Content - Backgr. Knowledge Matches			x	x				x	x	x

CSV. Elmeleegy et al. [40] and Chu et al. [41] use lists as input data format for their approaches. They do not face the challenge of detecting and normalizing complex table structures, but aim at recovering a tabular structure from lists. They neither used content style features such as different fonts or a distinction between bold and italic characters. Elmeleegy et al. [40], Chu et al. [41], Pivk et al. [33], Cafarella et al. [36] and Hurst [28] did use background knowledge in order to determine the meaning of textual content.

5.4.3 Background Knowledge

Background knowledge is used by different approaches in order to interpret the textual content of cells. Cafarella et al. [36] used web search to assess the relevance of terms on a table. The remaining approaches employ semantic web technologies and knowledge bases, as already described in section Section 5.3. Hurst emphasizes that semantic understanding of the table/cell content, i.e. the use of background knowledge, is required to recover certain implicit information and resolve ambiguities [43]. One example he mentions is the ambiguity of the stub header (see Figure 5.1), which might either refer to the cells on the right or to the cells below. In Figure 5.1 it is obvious for the reader that the stub head "Term" refers to the cells below, but we can only conclude this, because we know that the numbers "2003" and "2004" are most probably years and that "Fall", "Winter", and "Spring" are school terms. Furthermore, tables can contain cells, whose meaning is not even explicitly mentioned at all. In Figure 5.1 the information is omitted, that the "Assignment" and "Exam" columns contain "Grades". Wang refers to these implicit, but missing headers as "Virtual Headers" [29].

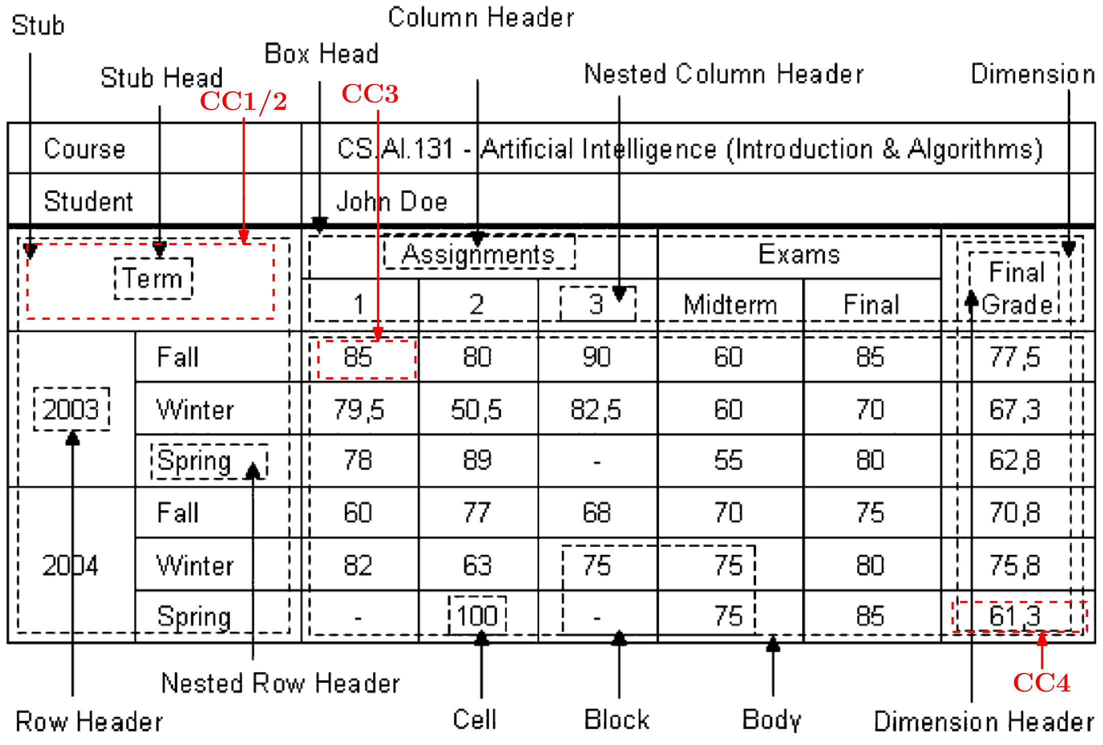
According to Adelfio et al. [37] the use of background only makes sense if the table content is globally relevant. If the table header contains internal codes of some company, global world knowledge would not be of any use. As observed by Chu et al. [41], the performance of an approach using background knowledge actually significantly decreases for corporate datasets [41]. For those data sets the syntactical structure of the content is way more important, as also shown by Cortez et al. [44]. Because our solution should work reliably, regardless of which kind of information is in the table, we do not aim at using background knowledge.

5.4.4 Table Models

Every approach makes certain assumptions about the possible structure of a table. Cafarella et al. [36] can make a very simple assumption, because they only consider tables which adhere to a structure with one header row and trailing data rows. All other variants of tables are discarded in a process, which they call *relational filtering*. Discarding tables is reasonable when dealing with table corpora on web-scale. But we aim at recovering as much tabular data as possible from the input. Discarding complete tables is thus not an option.

Adelfio et al. [37] and Pinto et al. [38] propose a higher variety of possible row functions, without explicit restrictions for their ordering. In addition to header and data rows, they propose different kinds of section- and sub-headers, title rows, aggregation rows (e.g. a total row), footnotes, and other types of non-relational metadata. Apart from row functions, Eberius et al. [39] introduce a distinction of column functions, namely data columns, empty columns, aggregation columns and columns which are not part of the data area.

Chen et al. [35] introduce the viewpoint that a table actually consists of column headers, row headers, and a data area. They make the strong assumption that the data area consists only of numeric values. This assumption does not hold in general, though. The approach of Seth et al. [34] also incorporates the concept of row headers, column headers and a data area and furthermore introduces the concept of critical cells, which mark four distinctive cells, separating the header areas from the data area. Figure 5.1 shows where these cells are located.



Course		CS.AI.131 - Artificial Intelligence (Introduction & Algorithms)					
Student		John Doe					
Term		Assignments			Exams		Final Grade
		1	2	3	Midterm	Final	
2003	Fall	85	80	90	60	85	77,5
	Winter	79,5	50,5	82,5	60	70	67,3
	Spring	78	89	-	55	80	62,8
2004	Fall	60	77	68	70	75	70,8
	Winter	82	63	75	75	80	75,8
	Spring	-	100	-	75	85	61,3

FIGURE 5.1: Table model of Hurst [28] and Pivk et al. [33] and Critical Cells (CC1-4) as introduced by Seth et al. [34]. (Adapted from Pivk et al. [33]).

The first critical cell (CC1) marks the first element in the Stub Header, CC2 the last. CC3 marks the upper left corner of the data area and CC4 the lower right corner of the data area. Seth et al. [34] make the strong assumption that the cells in the column and row header always form a unique access key to the cells in the data area and denotes tables which do not comply with that rule "Degenerate Tables" [45]. We will show in Section 6.2.6, that also quite regular tables do not necessarily comply with this assumption. Hurst [28] and Pivk et al. [33] do not make similar assumptions and further introduce the concept of nested column headers, nested row headers and dimension headers. This results in a very comprehensive table model, which is largely in line with the model of Wang [29]. Which table elements are supported by which approaches, is summarized in Table 5.4.

5.4.5 Analysis Methods

The analysis methods can be split in two categories - rule-based/heuristical methods and machine learning methods. General benefit of the rule-based and heuristical approaches is that they do not require training data.

Chen et al. [35], Pinto et al. [38] and Adelfio et al. [37] use Conditional Random Fields for categorization of row functions (e.g. header row, data row, group header row, title,

Table 5.4: Physical Table Elements used by different Authors

Level	Entity	Adelfio et al. [37]	Pinto et al. [38]	TARTAR [33]	WebTables [36]	Senbazuru [35]	DeExcelerator [39]	TANGO [34]	TabPro [28]
Physical	Table Location			x					x
	Multiple Tables (Vertically)					x	x		
	Table Orientation			x					
	Spanning Cells	x		x		x	x		x
Functional	Title/Table Caption	x	x	x		x	x	x	
	Metadata (e.g. Footnotes)	x	x			x	x	x	
	Critical Cells					x		x	
	Logical Units			x					
	Single Column Header	x	x	x	x	x	x	x	x
	Multiple Column Headers		x			x	x	x	x
	Hierarchical Column Headers		x			x		x	x
	Row Headers			x		x		x	x
	Group-/Sectionheaders	x	x			x			
	Stub Head			x				x	x
Structural	Aggregation	x					x		
	Group-/Sectionaggregate					x			
	Categories in Header			x		x		x	x

etc.). A disadvantage is that these approaches require a significant amount of manually annotated tables as training data. Adelfio et al. used 1,976 tables, which contained manual annotations of each row. The training data was upon request not provided. Cafarella et al. [36] used various classification methods such as Support Vector Machines (SVM), Naive Bayes and Logistic Regression for classification of relational and non-relational tables. The SVM classifier was performing best on this task. They used a sample of 1000 as relational/non-relational annotated tables as training data.

Seth et al. used Sequential Circuit Synthesis [46], in order to optimize a set of logical functions, representing assumptions about the table structure. This method only works if the row and column headers form a unique access path to the data cells.

DeExcelerator uses a rule-based pipeline, which processed the table through multiple heuristical detection steps, until the table is in a supposedly normalized form. The pipeline consists of empty row/column detection, interweaved header row and meta-data row detection, a refined step for empty column detection, aggregation row/column detection, data type detection and spanning column cell expansion[7].

[7]Sometimes, duplicated values are omitted in tables, when it is obvious for the human reader that the following rows contain the same value. In Spreadsheets, this can be expressed by spanning cells, which

Pivk et al. [33] also use a linear chain of detection and interpretation steps, but notes that: "There are also special cases [...], where the table consists of several 'independent' tables, and each of them may have a different orientation. In such cases, we first assign the table the orientation as described, and later (when we discover that the table actually consists of several 'independent' logical units) split the table and assign each one a respective orientation". Pivk et al., hence, re-consider previous steps.

Silva et al. [31] propose a design for an end-to-end table interpretation system which consists of linearly chained steps, but with explicit re-consideration loops. They "use the output of later steps as input to earlier ones, thus taking advantage of the extra knowledge each stage provides to improve the quality of the previous". This should counteract the effect that some steps make globally sub-optimal decisions. In this design, however, only the previous step gets re-considered. If two or three steps back a sub-optimal decision was made, the system is still stuck in a local optimum. Also Elmeleegy et al. [40] express the need for re-consideration of previous steps. In their *ListExtract* approach lines are so often splitted, merged and re-aligned, until the number of inconsistent fields converges, which was typically the case after one iteration.

In the list extraction approach of Chu et al. [41] first a big solution space is created, which is comparable to our proposed solution. Subsequently the space gets pruned and searched for optimal solutions. Chu et al. perform pruning anchor segmentation (PAS) and an A* search to minimize a certain path in the generated solution space of possible row-split and merge options. This approach significantly outperforms the comparable ListExtract system, which uses the re-consideration method. This suggests that a true global optimum can only be achieved with all possible parsing decisions on hand.

User interaction is not required by any of the systems, however, three of the approaches allow optional user interaction. Eberius et al. (*DeExcelerator*) allow user interaction to manually edit/correct every step of the parsing pipeline. Chen et al. [35] provide a graphical interface in which the user can edit the final result. Those edits trigger a re-consideration of the table structure. In order to optionally improve the performance of the system, Chu et al. utilized the "Programming by Example" principle. Their approaches allow the user to manually split a few exemplary lines of an input list and use this additional information as cues for the splitting/merging algorithm. None of the approaches uses the user feedback to generally improve future choices of the system.

span across multiple rows or columns. If a Spreadsheet with spanning cells is exported to CSV, then only the first cell gets filled with the spanning cell content and the remaining cells are left empty. In order to recover those omitted values, the content of the first cell gets copied into the remaining empty cells.

5.4.6 Evaluation and Limitations

For table interpretation tasks, general evaluation approaches have been put forward by Hu [47] and Wang [48], but with primary focus on the graphical interpretation level for detection of tables in documents. General baselines or gold standards for evaluation of end-to-end table interpretation or table normalization tasks could not be identified. Tabular results have to be either manually checked for correctness or compared to a manually created gold standard by conservative matching methods as proposed by Elmeleegy et al. [40]. We could not identify fuzzy matching methods or distance measures for tabular data.

Adelfio et al. [37] achieved a rate of 99.3% and 98.1% correctly classified rows in Spreadsheets and HTML tables, respectively, in 10 test and training runs with 1,048 relational Spreadsheets and 928 relational HTML tables. On a different test set, Pinto et al. [38] could only achieve a rate of 93.5% correctly classified rows. Because of the high background probability for data rows, the results are not necessarily as good as they seem. Pinto et al. [38] reported a recall and precision for table header detection with only 46.2% and 34%, respectively. Adelfio et al., again, achieved a recall of 91.5% and a precision of 94,5% for header row detection. The rate of entirely correct classified tables was reported as 56.3% for Spreadsheets and 84.6% for HTML tables. The WebTables [36] approach, which simply assumes the first row to be the header, could only correctly classify 24.2% and 41.2%, respectively, of the tables in the same test set. This, again, underlines the importance of proper row function detection. Pinto et al. did not report on entirely correctly classified tables. Chen et al. reached a F1 value of 0.774 for header detection and an F1 value of 0.920 and 0.811 for column header and row header hierarchy detection, respectively [49]. The produced results required, on average 1.8 and 16.2 human repair steps, for column and row headers respectively, in order to a achieve a perfect result. Without the assistance of the system 22.5 and 58.6 repair steps, respectively, would have been required. This shows that the system significantly reduces the manual effort for normalization of tables. The proposed solution can not be considered as a fully autonomous system, nevertheless.

The work of Pivk et al. [33] shows that evaluating the results of a end-to-end table interpretation systems is challenging. Manually created baselines by 14 different persons differed substantially, and were only in 2 of 21 cases identical. From a conceptual point of view, they agreed in only 60% of all cases. However, in 74.18% of all cases the interpretation of the proposed system was in line with at least one of the annotators on a conceptual level. Using a stricter evaluation, it was only in 50% of the cases in line with at least one manual annotation. In a second evaluation with 138 tables crawled from the web, and without fixed reference results, the interpretation process achieved in 85.44%

of all cases reasonable results. For further evaluation, we requested the source code of *TARTAR*. Due to missing and outdated dependencies, we could, however, not evaluate it ourselves. Hurst [28] indicated a recall and precision of 54% on unseen data for the *TabPro* system. Generally, the solutions for end-to-end table interpretation can not be regarded as reliably enough for our purposes. In most of the cases, a full interpretation of the content is also not required in order to normalize the structure of a table.

Seth et al. [34] stated a success rate of 99% for correct segmentation of the table into column header, row header and data areas. The evaluation was done on a set of 200 randomly drawn tables from websites such as Statistics Canada and the US Department of Justice. The sample, however, appears to contain a large fraction of tables which actually form unique access paths to data cells, which is a central prerequisite for their approach.

Chu et al. reported a F1 value for correct table extraction from web lists of 0.9 [41]. On the same data set, ListExtract achieved only an F1 value of 0.76. The supervised method of the approach of Chu et al. reached an F1 value of 0.97. List extraction methods can hence be considered as reliable and could potentially be used to improve the robustness of CSV parsers.

Eberius et al. [39] evaluated the results of *DeExcelerator* by human judgment with a group of 10 persons and a set of 50 tables. The human judges assessed the quality of each normalization step on a scale from 1 to 5. Approximately 90% of the metadata extraction steps were judged with the highest ranking. Table area, data type and header recognition were in approx. 80% of all cases rated with the highest rating. Additionally, the system was evaluated on a set of approx. 2000 Excel spreadsheets from the open government platform *data.gov*. For this data set, header recognition succeeded in 78% of the cases. Because *DeExcelerator* pursues the same aim as we do, and was previously evaluated on open government data sets with a good performance, we will consider the system for a comparison in Chapter 7.

Chapter 6

Multi-Hypothesis Parser

This chapter gives an overview of the architecture of the proposed Multi-Hypothesis Parser. We address each parsing step separately and provide more insights into the implemented heuristics. Finally, we describe how the best parsing hypothesis gets selected.

6.1 Architecture

The Multi-Hypothesis Parser was designed in a modular way, so that parsing steps can be easily added, removed or replaced. The interface which a parsing steps has to implement consists of a *detect*-method and *parse*-method. The *detect*-method receives an intermediate parsing result as input and provides a set of hypotheses and respective confidence values as output. A *parse*-method receives an intermediate parsing result and an associated hypothesis, and returns a new intermediate parsing result. Each parsing step has a specific position in the parsing hierarchy, which determines in which order the separate steps are executed. The core component of the parser is a tree data structure, which gets traversed in pre-order and built up on-the-fly by the respective parsing steps. When the tree is fully traversed, the parsing results of the final parsing step get collected and rated in the quality assessment module. Figure 6.1 gives an overview of the architecture and Algorithm 1 shows the tree traversal process in detail. The system was implemented in the R environment for statistical processing [50].

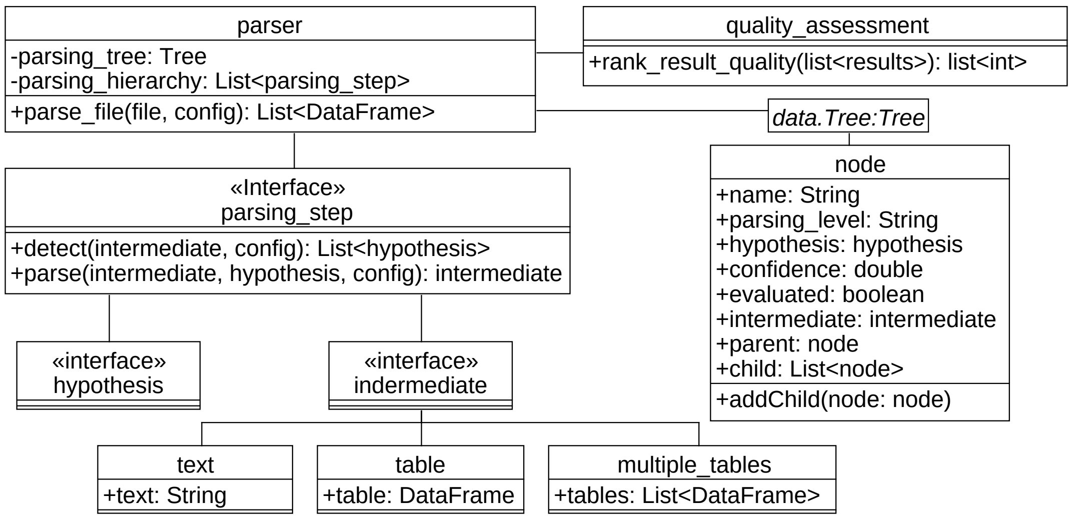


FIGURE 6.1: Architecture of the Multi-Hypothesis Parser

Algorithm 1 Hypothesis Tree Traversal

```
1: function $generate\_parsing\_tree(file\_path, config = default)$
2:     $intermediate \leftarrow file\_path$
3:     $tree \leftarrow$ CREATE_ROOT_NODE($intermediate$)
4:     $hypotheses \leftarrow parsing\_steps[1]$.DETECTOR($intermediate, config$)
5:     $tree \leftarrow$ new $childnode$ for each hypothesis in $hypotheses$
6:     while not all $nodes$ are evaluated, traverse $tree$ in $config.traversal\_order$ do
7:         if $node.confidence < config.pruning\_level$ then continue end if
8:         $level \leftarrow node.level$
9:         $parent \leftarrow node.parent.intermediate$
10:        $hypothesis \leftarrow node.hypothesis$
11:        $intermediate \leftarrow parsing\_steps[level]$.PARSER($parent, hypothesis, config$)
12:        if $intermediate$ is $null$ then continue end if
13:        $node$.ADD_PROPERTY($intermediate$)
14:        if $level$ is $count(parsing\_steps)$ then continue end if
15:        $hypotheses \leftarrow parsing\_steps[level + 1]$.DETECTOR($intermediate, config$)
16:        $node \leftarrow$ new $childnode$ for each hypothesis in $hypotheses$
17:    end while
18:    return $tree$
19: end function
```

6.2 Parsing/Detection Steps

Although the multi-hypothesis parsing framework does not generally prescribe which parsing steps have to be implemented, we propose a set of 9 parsing steps, which gradually lift the CSV file input towards a more explicit and normalized tabular form. Figure 6.2 illustrates this process. The detection steps each use heuristics to determine a set of reasonable hypotheses and respective confidence values. Regardless of the result of the heuristics, each detection step should account for the overall high background probability that the input is a perfectly regular, RFC 4180 conform table. The generated hypotheses should be as explicit as possible and the parsing steps as strict as possible, in order to rule out wrong hypotheses at early stage and confirm hypotheses most precisely.

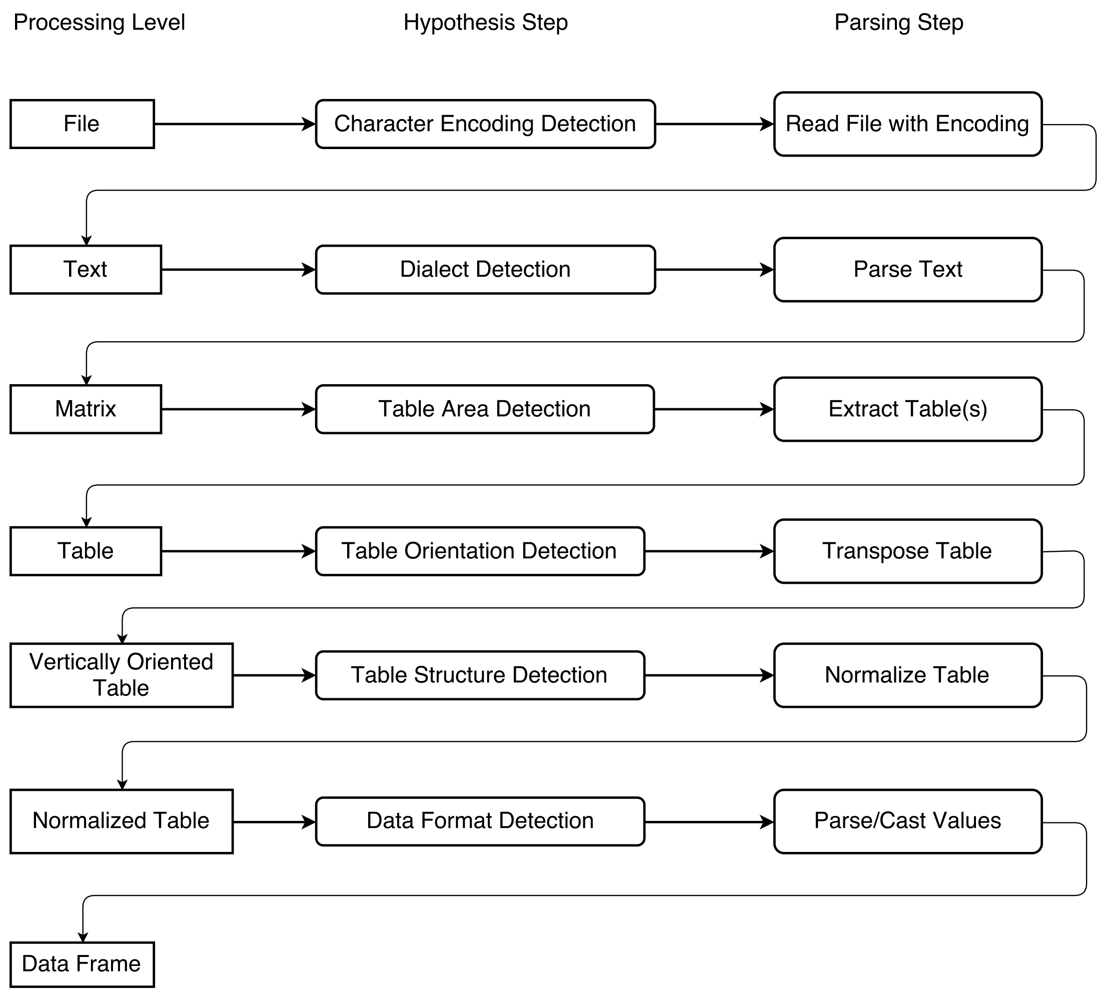


FIGURE 6.2: Parsing steps of the Multi-Hypothesis CSV Parser. Row/column function detection and narrow/wide data detection steps are in this diagram summarized under the step Table Structure Detection.

Each detection and parsing has access to a utility function, which determines the type of cell content, such as numeric, date, and time, based on regular expressions (see Listing C.1). The cell types can be considered as the algorithms "eyes" on the table content. In future work, they could be extended with more thorough regular expressions or cell content could be matched against knowledge bases (see Section 5.4.3) to determine the type of textual cell content.

Every detection step has access to a hypothesis handler, which provides auxiliary functions for collecting hypothesis. Those functions ensure that the same hypotheses is not created multiple times and that the confidence values get normalized at the end of each detection step. If no confidence value is provided, an equal confidence of all hypotheses is assumed.

6.2.1 Encoding

The web servers hosting the CSV files usually provide none or unreliable file encoding information. As a consequence, the encoding has to be inferred from the file itself, which is never 100% reliable. Since the detection of file encodings is not as straight-forward to implement, and a rather generic problem, we used an existing implementation to perform the encoding detection. For this, we considered the *guess_encoding*-function of the *readr* library and the *guess_encoding*-function of the *rvest* [51] library. The implementation of the *rvest* library always created almost the same set of hypotheses, even on very differently encoded files. The *readr* library dialect detection turned out to provide more realistic guesses. The readr *guess_encoding*-function provides a list of possible file encodings for a given file, together with respective probabilities that a certain encoding is correct. Because encoding guessing is in general rather unreliable, we ensure that a perfectly RFC 4180 compliant file with UTF-8 encoding always has the chance to be read correctly, by adding a default hypothesis for UTF-8 encoding with a likelihood of 50% relative to the encodings determined by readr. In the parsing step the file is read with a given encoding and returned as an intermediate result in form of one single string. If the reading with a certain parsing fails, a warning will be created and passed on to the next parsing steps, so that the quality assessment module can take the warning later on into account.

6.2.2 Dialect

The dialect detection is an essential parsing step in which all possible combinations of end-of-line qualifier, cell delimiter, quotation marks, and escaping styles are checked for plausibility. The plausibility criteria are on purpose very simple. In that way, different encoding hypotheses, which are locally unlikely, have the chance to prove later that they were actually correct. If a end-of-line qualifier is found at least once, it will be considered as a candidate. The same also applies for delimiter. Quotes are optional, however, if they occur it is important to detect them, because they act as escape sequence for delimiter and line endings. A quoting sign is considered as a candidate, if it occurs in front or behind a delimiter (ignoring whitespaces). Quotes inside quoted cells can be either escaped by another quotation mark of the same type (RFC 4180 conform) or by another escape character, such as a backslash character. If a quoting character with a leading backslash character is found, then the respective quoting style will be assumed, if not, the default double-quote style. Algorithm 2 shows in which way this detection routine was implemented. In total, 3 different end-of-line qualifiers, 10 different delimiters and 15 quotation marks, including foremost typographic variants, are checked. In the parsing step we use the robust parsing library *readr* to parse the string according to each hypotheses into a matrix. The parser is able to read files, even if they are not properly written and contain un-closed quotes or ragged rows. If the *readr* library encounters such an issue, it will try to fix it provisionally and returns the fixed table and a warning. This warning will be noticed and passed on to the subsequent parsing steps, together with the resulting table, so that the quality assessment module can take the warning into account later. If the parsing fails, this will lead to a dead branch in the hypotheses tree, which will be disregarded by subsequent parsing steps.

6.2.3 Table Area

The table area detection step detects the outer boundary of the actual table, inside the matrix provided by the previous parsing step. This step is necessary because CSV files can contain unnecessary whitespace around the actual table or even multiple tables. To account for the usual case in which the file contains only one table with potential surrounding whitespace, we always generate a corresponding default hypothesis, with a confidence of at least 50%. If multiple tables get detected, they will be added as alternative hypothesis, if not, only the default hypothesis will be returned with a 100% confidence. In order to actually detect multiple tables, we determine data-dense areas in the input matrix. The density again gets determined for each cell, based on non-emptiness of the cell itself and the (maximum) 8 neighbouring cells (See Figure 6.3). We make the assumption that the largest horizontal and vertical spaces in the matrix determine the outer boundaries of the table(s). Given that assumption, we determine the most dense cells in the table and extend the table area from that point on in all directions until reaching these large whitespaces in the horizontal and in the vertical or the boundaries of the table (See Algorithm 3). In the parsing step the respective

Algorithm 2 Dialect Detection

```
$eols \leftarrow$ ["$\backslash n$", "$\backslash r\backslash n$", ...]
$delims \leftarrow$ [",", ";", "$\backslash t$", ...]
$quotes \leftarrow$ ["", """, "'", ...]
$escape \leftarrow$ ["\"]
for all $eol$ in $eols$ do
    $lines \leftarrow$ SPLIT($text, eol$)
    if $count(lines) > 1$ then
        for all $delim$ in $delims$ do
            if $text$ contains $delim$ then
                for all $quote$ in $quotes$ do
                    if $text$ contains $quote$ with leading or trailing $delim$ then
                        if $text$ contains $quote$ with leading $escape$ char then
                            $hypotheses$.ADD($eol, delim, quote,$ "$escape$")
                        else
                            $hypotheses$.ADD($eol, delim, quote,$ "$double$")
                        end if
                    end if
                end for
            end if
        end for
    end if
end for
NORMALIZE_CONFIDENCE($hypotheses$)
```

tables get extracted from the original matrix and returned as a set of separate matrices. The parsing framework is generally capable of treating those tables independently until the last parsing step, where they can get collected and again joined to one set of data frames.

Algorithm 3 Table Area Detection - GET_DENSE_AREAS

```
function $get\_dense\_areas(types, density)$
    $dense\_areas \leftarrow$ LIST
    while $max(density) > 0$ do
        $i_{max}, j_{max} \leftarrow$ index of first cell with max $density$
        $i_{empty}, j_{empty} \leftarrow$ indeces of longest sequences of empty rows/columns
        $i_1, j_1 \leftarrow 1$
        $i_n, j_n \leftarrow dim(density)$
        if $any(i_{empty} \leq i_{max})$ then $i_1 \leftarrow max(i_{empty}[i_{empty} \leq i_{max}]) + 1$ end if
        if $any(i_{empty} \geq i_{max}]$ then $i_n \leftarrow min(i_{empty}[i_{empty} \geq i_{max}]) - 1$ end if
        if $any(j_{empty} \leq j_{max})$ then $j_1 \leftarrow max(j_{empty}[j_{empty} \leq j_{max}]) + 1$ end if
        if $any(j_{empty} \geq j_{max}]$ then$j_n \leftarrow min(j_{empty}[j_{empty} \geq j_{max}]) - 1$ end if
        if $i_1 == i_n$ or $j_1 == j_n$ then break end if
        $density[i_1 : i_n, j_1 : j_n] \leftarrow -1$
        $dense\_areas$.ADD($i_1, i_n, j_1, j_n$)
    end while
    return $dense\_areas$
end function
```

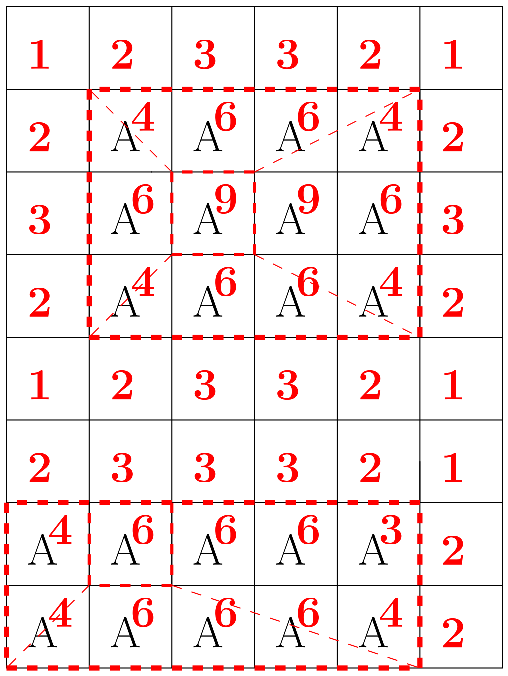

1 2 3 3 2 1
2 A^4 A^6 A^6 A^4 2
3 A^6 A^9 A^9 A^6 3
2 A^4 A^6 A^6 A^4 2
1 2 3 3 2 1
2 3 3 3 2 1
A^4 A^6 A^6 A^6 A^3 2
A^4 A^6 A^6 A^6 A^4 2

FIGURE 6.3: Table area votes for two tables.

6.2.4 Table Orientation

The table orientation detection was inspired by Pivk et al. [33]. Algorithm 4 was recreated from the original description provided by Pivk et al. The algorithm compares the last row/column of the table to all other rows/columns. The table orientation is then determined based on whether the columns or the rows are more similar to the last column or row respectively. If the columns are very similar, it is likely that the table is horizontally oriented, which is not desired. The comparison is not based on literal content, but on content categories, such as character, numeric, date etc. It turned out that this algorithm is robust only if the table does not contain sparse last rows/columns (See Figure 6.4). When this step is executed, the tables are not yet cleaned from title rows or footnotes, hence this can well be the case. To compensate for this, we improved the algorithm by taking a random sample of maximum 10 rows/columns, instead of only the last row/column, to perform the same routine (see Algorithm 5). The confidence of the generated hypothesis is derived from the row/column-wise similarity. To account for the high background probability that a table is vertically oriented, we only generate a hypothesis for horizontal orientation when the column-wise similarity is higher than the row-wise. In the parsing step, the matrix gets simply transposed in case if it is vertically oriented. Since transposing the table implies that tables content is not changed, but moved inside the table, we count one moving operation for each cell and return this value together with the parsing result. In that way the quality assessment module, can take the manipulation of the original input as quality criterion into account.

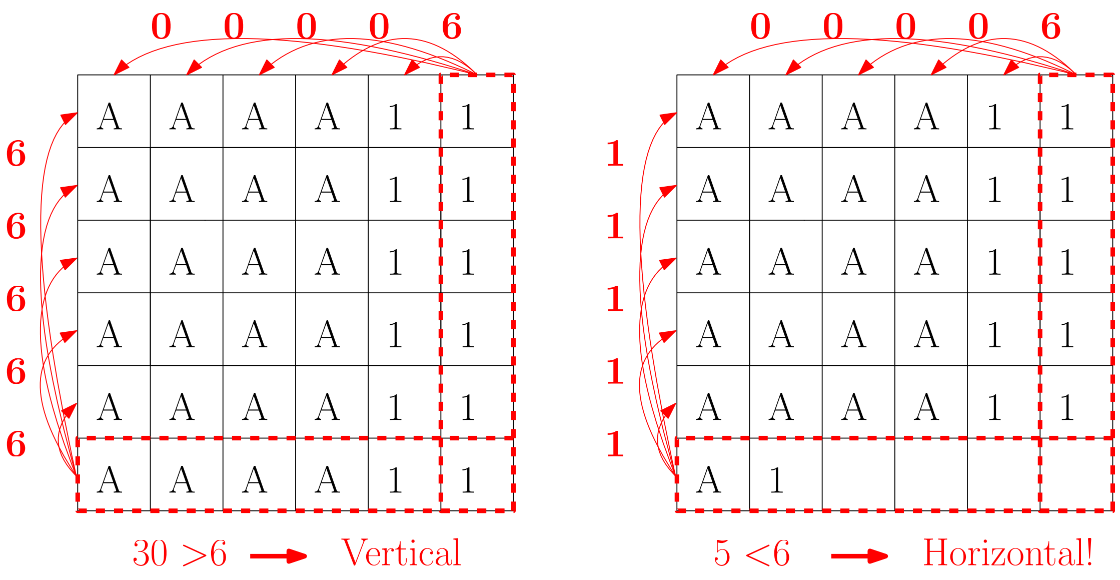


Figure 6.4: Table Orientation as proposed by Pivk et al. [33]. Left: Correct, Right: Gets falsely classified as horizontal, because the last line contains metadata.

Algorithm 4 Table Orientation - Single Reference Row/Col (Pivk et al. [33])

```
1: $types \leftarrow$ GET_CELL_TYPES($matrix$)
2: $similarity_{row}, similarity_{col} \leftarrow []$
3: for all $i$ in $1 : (nrow(types) - 1)$ do
4:     $n = nrow(types)$
5:     $consistency_{row}[i] \leftarrow$ percentage of cells in $types[i,] == types[n,]$
6: end for
7: for all $i$ in $1 : (ncol(types) - 1)$ do
8:     $n = ncol(types)$
9:     $similarity_{col}[i] \leftarrow$ percentage of cells in $types[,i] == types[,n]$
10: end for
11: if $mean(similarity_{row}) > mean(similarity_{col})$ then
12:     $hypotheses.$ADD$(conf = 1, orientation = "vertical")$
13: else
14:     $hypotheses.$ADD$(conf = 1, orientation = "horizontal")$
15: end if
```

6.2.5 Row Functions

As outlined in Chapter 5, table structure detection is a challenging task. None of the reviewed approaches is able to infer complex table structures based on the bare content of a table. The most promising approaches explicit domain-specific background knowledge, visual layout elements such as horizontal and vertical lines, or different content style features such as background colour and font types, which are only provided by richer formats such as HTML or Excel Spreadsheets. However, Adelfio et al. [37] proposed an approach for row function detection, which largely builds up on cell content features and could thus potentially be applied to CSV tables as well. The method is able to classify the function of rows into categories, such as header (including multiple header

Algorithm 5 Table Orientation - Multiple Reference Rows/Cols

```
1: $types \leftarrow$ GET__CELL__TYPES($matrix$)
2: $similarity_{row}, similarity_{col} \leftarrow []$
3: for all $r$ in random set of $min(10, nrow(types))$ samples from $1 : nrow(types)$ do
4:     for all $i$ in $1 : nrow(types)$, except for $r$ do
5:         $similarity_{row}[i] \leftarrow$ percentage of cells in $types[i, ] == types[r, ]$
6:     end for
7: end for
8: for all $r$ in random set of $min(10, ncol(types))$ samples from $1 : ncol(types)$ do
9:     for all $i$ in $1 : ncol(types)$, except for $r$ do
10:        $similarity_{col}[i] \leftarrow$ percentage of cells in $types[, i] == types[, r]$
11:    end for
12: end for
13: $hypotheses$.ADD($conf = mean(similarity_{col}), orientation =$ "$vertical$")
14: if $mean(similarity_{row}) > mean(similarity_{col})$ then
15:     $hypotheses$.ADD($conf = mean(similarity_{row}), orientation =$ "$horizontal$")
16: end if
17: $hypotheses \leftarrow$ NORMALIZE__CONFIDENCE($hypotheses$)
```

rows), title, data and non-relational rows, which is as an essential first step towards a better detection of the table structure. Because the method is based on *conditional random fields* [52], it requires a large amount of training data, which was not provided by the authors upon our request and which we were not able to create ourselves within reasonable time. However, a simple form of row function detection, for header and data rows, is performed by state-of-the-art CSV parsers as well, based on heuristics. We therefore used a heuristical approach, inspired by the approaches used in current parsers and extend it towards multiple header rows, aggregation rows and non-relational data (metadata) above and below the table. In principle, this approach could nevertheless easily be replaced by other approaches, such as the one shown by Adelfio et al.

The heuristic to detect row functions is based on *function votes*. Every row gets votes for emptiness, potential total (aggregation), metadata, header and data, as shown in Algorithm 17 in the Appendix. Each function vote gets normalized by the total of all function votes in the same row and the maximum vote for the respective function across the whole table. Subsequently, a maximum of 8 different hypotheses are created, dependent on whether the first data is suspected in the first, second or later than the second row (see Algorithm 6 and Figure 6.5). In the parsing step, empty rows and metadata rows are removed from the table. The content of metadata rows is stored separately and will be returned together with the parsing result. Multiple header rows are assumed to be spanning headers except for the last header row. This is a pattern, which was often observed in the *data.gov.uk* corpus. Those spanning header cells are collapsed column-wise, by aggregating the cell content, separated by a dot. For those cells, edits are counted and returned together with the parsing result to allow the quality assessment method to take the change of content into account. The removal of potential aggregation rows can be activated via a configuration flag and is by default deactivated.

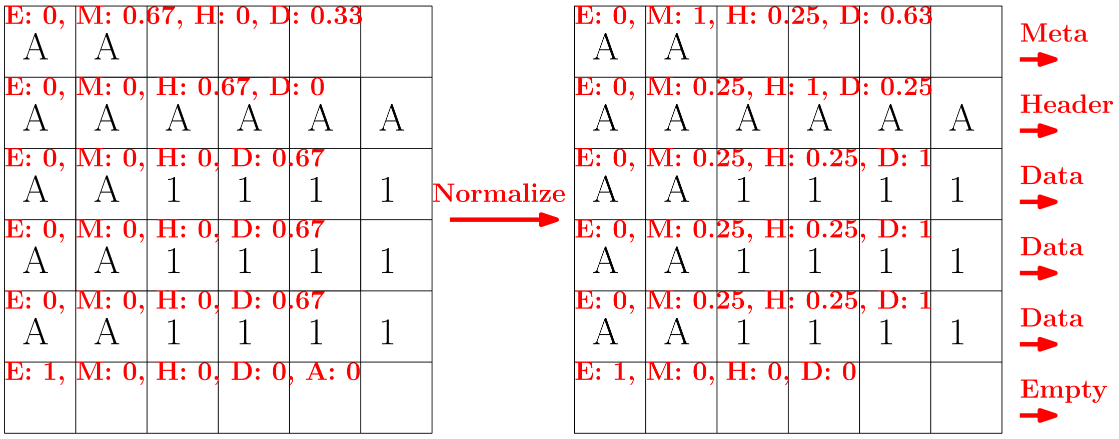

E: 0, M: 0.67, H: 0, D: 0.33
A A
E: 0, M: 0, H: 0.67, D: 0
A A A A A A
E: 0, M: 0, H: 0, D: 0.67
A A 1 1 1 1
E: 0, M: 0, H: 0, D: 0.67
A A 1 1 1 1
E: 0, M: 0, H: 0, D: 0.67
A A 1 1 1 1
E: 1, M: 0, H: 0, D: 0, A: 0
Normalize
E: 0, M: 1, H: 0.25, D: 0.63
A A
E: 0, M: 0.25, H: 1, D: 0.25
A A A A A A
E: 0, M: 0.25, H: 0.25, D: 1
A A 1 1 1 1
E: 0, M: 0.25, H: 0.25, D: 1
A A 1 1 1 1
E: 0, M: 0.25, H: 0.25, D: 1
A A 1 1 1 1
E: 1, M: 0, H: 0, D: 0
Meta
Header
Data
Data
Data
Empty

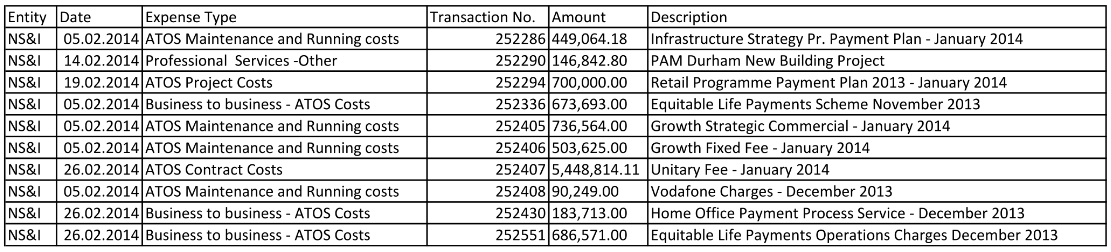
FIGURE 6.5: Row Function Votes

Algorithm 6 Row Functions

```
1: $types \leftarrow$ GET__CELL__TYPES($matrix$)
2: $votes \leftarrow$ COUNT__FUNCTION__VOTES($types$)
3: $votes \leftarrow$ NORMALIZE__FUNCTION__VOTES($votes$)
4: $functions \leftarrow$ for each row, get function with highest $votes$
5: $meta_{top} \leftarrow$ first contiguous group of indices, if $functions$ are "metadata", else []
6: $meta_{bottom} \leftarrow$ last contiguous group of indices, if $functions$ are "metadata", else []
7: $meta_{full} = [meta_{top}, meta_{bottom}]$
8: $meta_{top\_end} \leftarrow$ last index of $meta_{top}$
9: $data_{start} \leftarrow$ first row index, where $functions$ are "data"
10: $hypotheses$.ADD($header = []$)                                   ▷ hypothesis: D+
11: $hypotheses$.ADD($header = 1$)                                    ▷ hypothesis: HD+
12: if not $any(data_{start})$ then return $hypotheses$ end if
13: $hypotheses$.ADD($meta_{full}$)                                   ▷ hypothesis M*D+M*
14: $hypotheses$.ADD($header = meta_{top\_end} + 1, meta_{full}$)     ▷ hypothesis M*HD+M*
15: if $data_{start} > 1$ then
16:     $header_{full} \leftarrow 1 : (data_{start} - 1)$
17:     $hypotheses$.ADD($header_{full}$)                             ▷ add hypothesis H*D+
18:     $hypotheses$.ADD($header_{full}, meta_{full}$)                ▷ add hypothesis M*H*D+M*
19: end if
20: if $data_{start} > 2$ then
21:     $span\_header_{full} \leftarrow 1 : (data_{start} - 1)$
22:     $span\_header_{meta} \leftarrow (meta_{top\_end} + 1) : (data_{start} - 1)$
23:     $hypotheses$.ADD($span\_header_{full}$)                       ▷ add hypothesis SH*D+
24:     $hypotheses$.ADD($span\_header_{meta}, meta_{full}$)          ▷ add hypothesis M*SH*D+M*
25: end if
26: $hypotheses \leftarrow$ NORMALIZE__CONFIDENCE($hypotheses$)
```

6.2.6 Column Functions

Having a row function detection step calls for having a column function detection step as well. At first glance, it appears straight-forward to apply the previously described methods to columns instead of rows. However, to understand the implications and complications of this, we have to consider the reasoning behind the row function detection. The row function step generates hypotheses about metadata rows, header rows, etc., and in the very end of the parsing process a quality metric will judge the quality of the resulting tables and rule out the "bad hypotheses". This is expected to work, because we can make the reasonable assumption that columns of tidy tables contain a header cell and subsequent data cells store the same variable. We can therefore expect a column-wise consistency of the data types and potentially furthermore a certain uniformity of values. If metadata or header rows would get falsely classified as data they would most probably compromise this consistency. Detecting column-wise inconsistency of values could thus help to rule out wrong hypotheses. Accordingly, in order to make column function detection work properly, we would have to make a similar assumption for row content. In a tidy table each row should contain exactly one observation. We could thus make the strong assumptions that an observation always consists of a number of leading index columns, which form a unique access key to the subsequent observation variables, as proposed by Nagy et al. [42]. Or we could assume that each table contains a numeric "block" of variables on the very right and that the other columns form the index, as proposed by Chen et al. [35] [49]. However, both assumptions are not realistic and we could easily identify counter examples in the *data.gov.uk* corpus. Figure 6.6 shows a table which does not contain numeric values on the right side and where the numeric column "Transaction Number" is actually an index column rather than a variable column. Figure 6.7 again shows a table where the only potential access key columns do not form a unique access key.

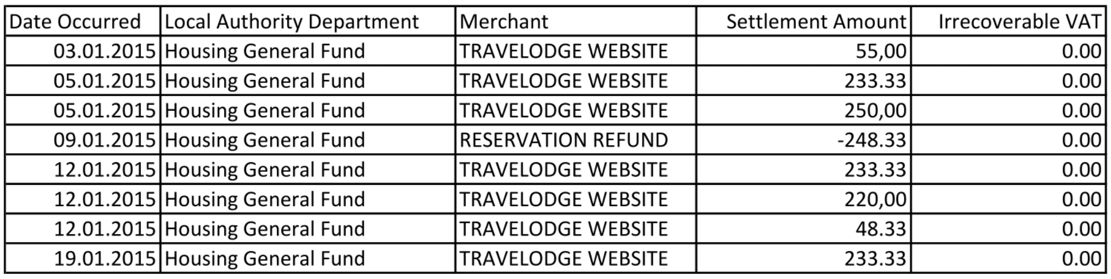


Entity	Date	Expense Type	Transaction No.	Amount	Description
NS&I	05.02.2014	ATOS Maintenance and Running costs	252286	449,064.18	Infrastructure Strategy Pr. Payment Plan - January 2014
NS&I	14.02.2014	Professional Services -Other	252290	146,842.80	PAM Durham New Building Project
NS&I	19.02.2014	ATOS Project Costs	252294	700,000.00	Retail Programme Payment Plan 2013 - January 2014
NS&I	05.02.2014	Business to business - ATOS Costs	252336	673,693.00	Equitable Life Payments Scheme November 2013
NS&I	05.02.2014	ATOS Maintenance and Running costs	252405	736,564.00	Growth Strategic Commercial - January 2014
NS&I	05.02.2014	ATOS Maintenance and Running costs	252406	503,625.00	Growth Fixed Fee - January 2014
NS&I	26.02.2014	ATOS Contract Costs	252407	5,448,814.11	Unitary Fee - January 2014
NS&I	05.02.2014	ATOS Maintenance and Running costs	252408	90,249.00	Vodafone Charges - December 2013
NS&I	26.02.2014	Business to business - ATOS Costs	252430	183,713.00	Home Office Payment Process Service - December 2013
NS&I	26.02.2014	Business to business - ATOS Costs	252551	686,571.00	Equitable Life Payments Operations Charges December 2013

FIGURE 6.6: Spend over £25,000 in National Savings & Investments, 2014 February Return[1]

[1]https://data.gov.uk/dataset/28db2fc1-134b-402c-8348-150b3dd112ae/resource/2b36ab74-7e3e-4fe6-ad6f-80f3376313a4/

[2]https://data.gov.uk/dataset/procurement-card-data/resource/1c16694d-abab-41c7-a134-fefa51f055f0

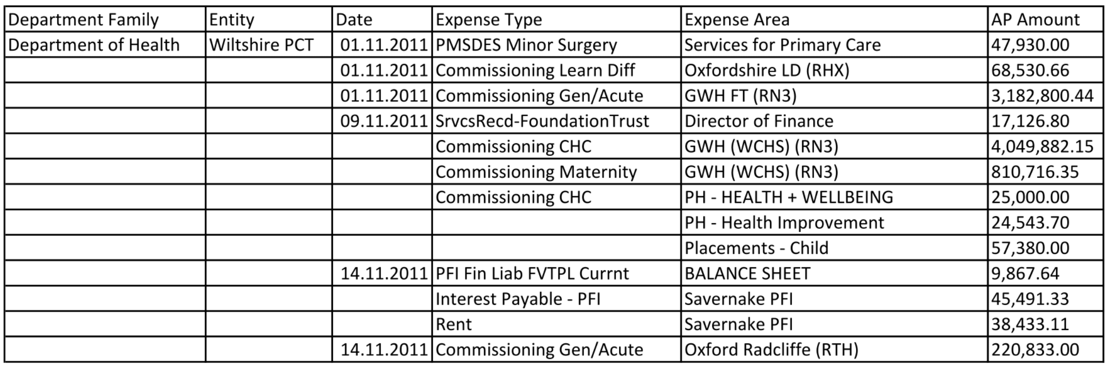


Date Occurred	Local Authority Department	Merchant	Settlement Amount	Irrecoverable VAT
03.01.2015	Housing General Fund	TRAVELODGE WEBSITE	55,00	0.00
05.01.2015	Housing General Fund	TRAVELODGE WEBSITE	233.33	0.00
05.01.2015	Housing General Fund	TRAVELODGE WEBSITE	250,00	0.00
09.01.2015	Housing General Fund	RESERVATION REFUND	-248.33	0.00
12.01.2015	Housing General Fund	TRAVELODGE WEBSITE	233.33	0.00
12.01.2015	Housing General Fund	TRAVELODGE WEBSITE	220,00	0.00
12.01.2015	Housing General Fund	TRAVELODGE WEBSITE	48.33	0.00
19.01.2015	Housing General Fund	TRAVELODGE WEBSITE	233.33	0.00

Figure 6.7: Procurement Card Data, January 2015[2]

Selecting index columns, based on functional dependencies is also not a feasible option, since state-of-the-art automatic functional dependency detection methods are not able to detect a sufficiently limited set of sensible dependencies [19]. Overall, we can draw the conclusion that making general assumptions about the data structure of observations is not practicable.

The major use case for column function detection would be the detection of non-relational data on the left or right side of the actual table, which would compromise the observation data. We did, however, not observe any table in which this was the case. What we did well observe, were spanning column cells (see Figure 6.8).



Department Family	Entity	Date	Expense Type	Expense Area	AP Amount
Department of Health	Wiltshire PCT	01.11.2011	PMSDES Minor Surgery	Services for Primary Care	47,930.00
		01.11.2011	Commissioning Learn Diff	Oxfordshire LD (RHX)	68,530.66
		01.11.2011	Commissioning Gen/Acute	GWH FT (RN3)	3,182,800.44
		09.11.2011	SrvcsRecd-FoundationTrust	Director of Finance	17,126.80
			Commissioning CHC	GWH (WCHS) (RN3)	4,049,882.15
			Commissioning Maternity	GWH (WCHS) (RN3)	810,716.35
			Commissioning CHC	PH - HEALTH + WELLBEING	25,000.00
				PH - Health Improvement	24,543.70
				Placements - Child	57,380.00
		14.11.2011	PFI Fin Liab FVTPL Currnt	BALANCE SHEET	9,867.64
			Interest Payable - PFI	Savernake PFI	45,491.33
			Rent	Savernake PFI	38,433.11
		14.11.2011	Commissioning Gen/Acute	Oxford Radcliffe (RTH)	220,833.00

Figure 6.8: Spend over £25,000 in NHS Wiltshire, 2011 October Return[3]

Eventually, we implemented a detection step for column functions (see Algorithm 7), which contains an experimental detection heuristic for spanning header columns, total columns and empty columns and empty columns with header (See Figure 6.9). The detection step creates maximum two hypotheses, which also contributes to the overall goal of limiting the computational complexity of the parsing process. However, in future work, this simple method could be easily replaced by more sophisticated methods. In the parsing step, the potentially spanning header cells get vertically expanded, empty columns get removed, and potential aggregation columns get removed optionally. Ultimately, the quality assessment method will not help to evaluate the correctness of the

[3]`https://data.gov.uk/dataset/spend-over-25000-in-nhs-wiltshire/resource/9dc939c6-24e0-4ea0-aed5-ed25100d8d2e`

spanning column cell expansion and we thus have to rely on the accuracy of the used heuristic. The only scenario in which the successful expansion of header cells could be validated by the quality assessment step is, when the expanded column cells are variable names. In that case, the narrow data detection step would pick them up, spread the table accordingly and thereby increase the column-wise data consistency, which would be preferred by the quality assessment module.

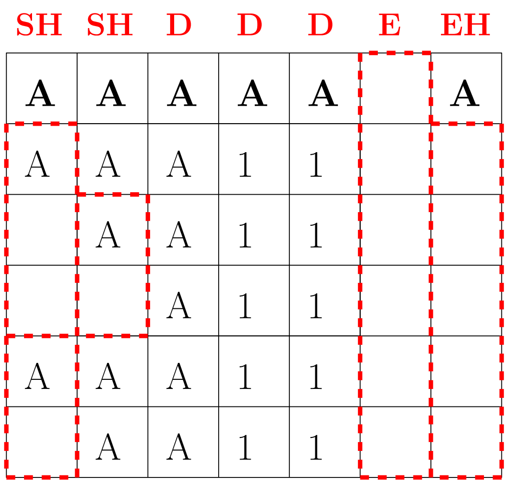


FIGURE 6.9: Column Function Detection. SH: Spanning Header, D: Data, E: Empty, EH: Empty with Header

6.2.7 Wide and Narrow Data

As outlined in Section 2.3, a full detection of wide data and narrow data is not feasible without having additional domain knowledge. We can, however, detect wide data to the extent to which variables in the header row are have specific values, which is unlikely to be variable names, such as numerics, dates or times. Furthermore, we can detect narrow data if one column contains sequentially repeating cell content, i.e. the variable names, and another column contains potentially values stemming from different variables.

We created two practical parsing steps, which employ those assumptions and are able to create reasonable hypotheses about narrow and wide data, however only for a limited set of scenarios. The steps are arranged in the parsing hierarchy so that first wide data gets detected and reshaped and then narrow data. This will allow the detection of mixed data as well, which requires first, *melting* of wide columns and subsequent *spreading* of narrow columns.

The detection heuristic for wide data checks for numeric, date and time values in header names (see Algorithm 8). See Figure 6.10. And in the parsing step the columns which supposedly belong to the same variable, are *melted*, i.e moved into one column and the former header copied into a separate column, as shown in Section 2.3.

Algorithm 7 Column Functions

```
$types \leftarrow$ GET_CELL_TYPES($table$)
$col\_functions\_basic \leftarrow []$
$col\_functions\_experimental \leftarrow []$
**for all** $j$ **in** $1 : ncol(types)$ **do**
    $col\_functions\_basic[j] \leftarrow$ "$data$"
    $col\_functions\_experimental[j] \leftarrow$ "$data$"
    **if** $all(types[,j]$ **is** "$empty$") **and** $header(types)[j]$ **is** "$empty$" **then**
        $col\_functions\_basic[j] \leftarrow$ "$empty$"
        $col\_functions\_experimental[j] \leftarrow$ "$empty$"
    **else if** $all(types[,j]$ **is** "$empty$") **and** $header(types)[j]$ **is not** "$empty$" **then**
        $col\_functions\_basic[j] \leftarrow$ "$empty\_with\_header$"
        $col\_functions\_experimental[j] \leftarrow$ "$empty\_with\_header$"
    **else if** $header(types)[j]$ **is** $total$ **and** $all(types[,j]$ **is** "$numeric$"|"$empty$") **then**
        $col\_functions\_basic[j] \leftarrow$ "$aggregate$"
        $col\_functions\_experimental[j] \leftarrow$ "$aggregate$"
    **else if** all non-empty cells of $types[,j]$ also non-empty in $types[,j+1]$) **then**
        $ratio \leftarrow$ ratio of "empty" to "non-empty" cells in $types[,j]$
        $avg\_ratio \leftarrow$ mean ratio of "empty" to "non-empty" cells in all other columns
        **if** $ratio > avg\_ratio$ **and** no previous column was classified as "data" **then**
            $col\_functions\_experimental[j] \leftarrow$ "$spanning\_header$"
        **end if**
    **end if**
**end for**
$hypotheses$.ADD($col\_functions\_basic$)
$hypotheses$.ADD($col\_functions\_experimental$)
$hypotheses \leftarrow$ NORMALIZE_CONFIDENCE($hypotheses$)
```

The detection heuristic for narrow data checks if the last but one column contains sequentially repeating values, i.e potential variable names (see Algorithm 9). See Figure 6.10. We check only the last but one column for variable names and assume that the last column contains the variables, because this was the typical structure of narrow-data tables we observed. The parsing step will *spread* the potential variable name column and the trailing multi-variable column. Thereby, the variable name column will be converted to column headers, the multiple-variable column will be split up into respective separate columns and the remaining columns are collapsed. This only works if the other columns are consistently duplicated for all variables. We use the library *reshape2* [53] to perform those transformations. If the reshaping fails, the reshape function will return an error, which leads to a dead branch in the hypotheses tree. However, a default hypotheses for non-wide/narrow always gets created, which leaves the original table untouched and always succeeds. In case the reshaping succeeds, we count one move for every cell, because they got significantly rearranged and return that count together with the parsing result. Our assumption is that correctly reshaped tables, i.e. tidy tables, will have a higher inner-column consistency of values and thus be preferred over non- or falsely reshaped tables.

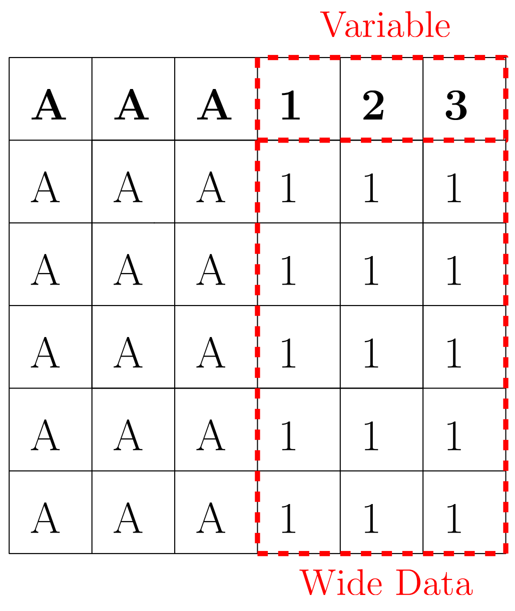


FIGURE 6.10: Wide Data detection for wide data in column 4-6

Algorithm 8 Wide Data - Detection

```
1: $types \leftarrow$ GET_CELL_TYPES($table$)
2: $hypotheses$.ADD($multi\_col\_vars = []$)
3: $pontential\_header \leftarrow$ columns with numerics, dates or times in the header
4: if $count(pontential\_header) > 1$ then
5:     $start \leftarrow min(pontential\_header)$
6:     $end \leftarrow max(pontential\_header)$
7:     $coherence \leftarrow$ percentage of equal cell types in $types[start : end]$
8:     $hypotheses$.ADD($conf = coherence, multi\_col\_vars = [start : end]$)
9: end if
10: $hypotheses \leftarrow$ NORMALIZE_CONFIDENCE($hypotheses$)
```

Algorithm 9 Narrow Data - Detection

```
1: $types \leftarrow$ GET_CELL_TYPES($table$)
2: $hypotheses$.ADD($key\_col = [], value\_col = []$)
3: if $ncol(table) > 2$ and  then
4:     $repeats \leftarrow$ determine if cell content $types[, ncol(types) - 1]$ is sequentially repeating
5:     if $repeats$ then
6:         $hypotheses$.ADD($key\_col = ncol(types) - 1, value\_col = ncol(types)$)
7:     end if
8: end if
```

6.2.8 Data Types

The (column) data types step is special, because it is the last step. We do not have to produce locally suboptimal hypotheses in order to preserve globally optimal paths in

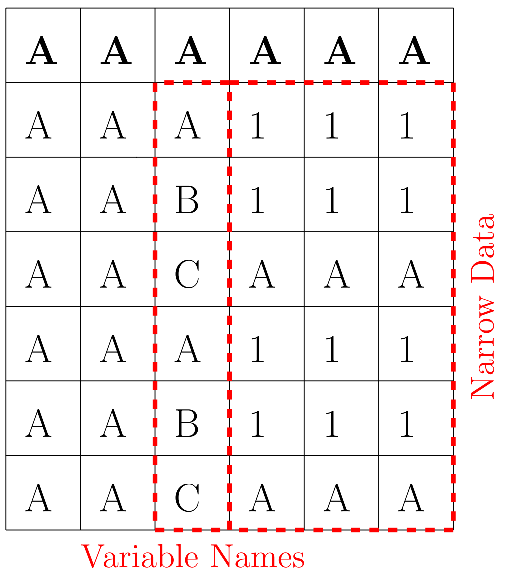


FIGURE 6.11: Narrow Data detection for narrow data in column 3-6

the hypothesis tree. We are free to locally optimize the parsing decision. Therefore we produce only one hypothesis in the detection step and optimize the final result in the parsing step. The data type detection supports 5 data types, namely numerics, logicals, dates, times and text, which are the most basic data types in databases and statistical programming languages. The data types detection (see Algorithm 10) is strict by default. This means that either all non-empty cells in a column can be associated with the same data type or the column content will be considered as text. All possible variants of NA value qualifier have already been taken into account by the cell type detection and were declared as empty cells (see Listing C.1). They do not have to comply with the data type. One important detail of the detection step is that the data type has to be consistent, but the data format, i.e. the date format, does not have to be consistent.

Based on regular expression matches[4] potential data formats are determined, ranked by number of matches and passed on to the parsing step. This helps to solve ambiguities in data formatting, as e.g., for the date "02/02/2012". The regular expressions are so strict that they take value ranges (months/days) into account. Assuming the table has only three rows and the two other dates are "10/02/2012" and "13/02/2012", then the *mm/dd/yyyy* format will get 2 votes, the *dd/mm/yyyy* format will get 3 votes. This is in line with the intuition that it is more likely that all values are in *dd/mm/yyyy* format, given the date "13/02/2012". On the other hand this approach allows for inconsistent formattings, like "02-02-2012", "10-02-2012", and "13/02/2012". In that case the formats *dd-mm-yyyy* and *mm-dd-yyyy* will get 2 votes and *dd/mm/yyyy* one vote. If two formats get the same number of votes, they will be ranked based on a pre-defined

[4]The regular expressions are due to lack of space not presented here, but can be obtained from the source code at `https://github.com/tdoehmen/hypoparsr/blob/master/R/data_type.R`

order, in this case, *dd-mm-yyyy* would be first. The detection of numeric values is robust to leading and trailing currency signs and trailing units as long as the unit does not contain other numerics. Apart from that, a thousand mark and decimal separator signs are determined. Also regular negative signs and bookkeeping style of negative numbers (leading and trailing parentheses) are distinguished. The bookkeeping style was frequently observed in the *data.gov.uk* corpus.

Algorithm 10 Data Types - Detect

```
1: $types \leftarrow$ GET_CELL_TYPES($table$)
2: $hypothesis \leftarrow$ LIST
3: for $j$ in $1 : ncol(types)$ do
4:     if $any(types[, j]$ is "$numeric$" and $all(types[, j]$ is "$numeric$"|"$empty$") then
5:         $hypothesis.types[j] \leftarrow$ "$numeric$"
6:         $hypothesis.formats[j] \leftarrow$ determine possible decimal and thousand marks
7:         $hypothesis.na\_qualifier.$ADD(determine potential NA qualifier)
8:     else if $any(types[, j]$ is "$logical$" and $all(types[, j]$ is "$logical$"|"$empty$") then
9:         $hypothesis.types[j] \leftarrow$ "$logical$"
10:        $hypothesis.formats[j] \leftarrow$ determine possible true/false qualifier
11:        $hypothesis.na\_qualifier.$ADD(determine potential NA qualifier)
12:    else if $any(types[, j]$ is "$date$") and $all(types[, j]$ is "$date$"|"$empty$") then
13:        $hypothesis.types[j] \leftarrow$ "$date$"
14:        $hypothesis.formats[j] \leftarrow$ determine possible date formats
15:        $hypothesis.na\_qualifier.$ADD(determine potential NA qualifier)
16:    else if $any(types[, j]$ is "$time$") and $all(types[, j]$ is "$time$"|"$empty$") then
17:        $hypothesis.types[j] \leftarrow$ "$time$"
18:        $hypothesis.types[j] \leftarrow$ determine possible time formats
19:        $hypothesis.na\_qualifier.$ADD(determine potential NA qualifier)
20:    else
21:        $hypothesis.types[j] \leftarrow$ "$text$"
22:    end if
23: end for
```

In the parsing step (see Algorithm 11) we attempt to convert each column into the previously determined data type by using the determined data formats. Therefore, at first, leading whitespaces, trailing whitespace and quotation marks are trimmed from each cell. Leading and trailing whitespaces in cells are a common irregularity in CSV files and quotation marks can be left at the outer boundaries of a cell if quoting signs were inconsistently used. Therefore, we considered them as artifacts from the parsing process. It has to be taken into account that this can negatively affect the parsing of CSV files which contain leading/trailing whitespaces or quotation marks on purpose and as integral part of the cell content. This would, however, be rather unusual.

Secondly, NA values which were identified in the detection step, get replaced by empty strings. Subsequently, it will be attempted to parse each cell based on the highest ranked format. Units of numeric values are stripped off and saved separately or, if only exactly one type of unit is present, it is copied to the header. If none of the numerics has fractional digits, the numerics will be saved as integers, otherwise as double values. Times, dates and logicals are simply parsed based on the given format. For every cell which could not be successfully parsed, the next highest ranked format will be used for another parsing attempt, until all potential formats have been tried. If the list of possible formats is exhausted and still not all cells have been successfully parsed, the fallback will become effective and the column will be kept as text.

Algorithm 11 Data Types - Parser

```
1: for $j$ in $1 : ncol(table)$ do
2:     $type \leftarrow hypothesis.types[j]$
3:     $formats \leftarrow hypothesis.formats[j]$
4:     $text \leftarrow table[,j]$ and trim whitespace and quotation marks
5:     $text \leftarrow text$ and replace $hypothesis.na\_qualifier$ with ""
6:     while not $all(valid)$ do
7:         $relevant \leftarrow text[!valid]$
8:         $format \leftarrow$ next $format$ from $formats$
9:         $parsed \leftarrow$ parse $relevant$ base on $format$
10:        $valid \leftarrow$ is successfully $parsed$?
11:        $table[valid, j] \leftarrow parsed$
12:    end while
13:    if not $all(valid)$ and $config.conservative\_type\_casting$ then
14:        $table[,j] = text$
15:    end if
16: end for
```

6.3 Quality Assessment

The quality assessment step is a vital final step. After the hypothesis tree is fully expanded and traversed, several potentially correct parsing results reached the last level of the parsing hierarchy. A parsing result contains the resulting table and additional information about the parsing process, such as the confidence of each parsing step, warnings emitted by parsing steps, edits and moves of cell content. Among these results, the best one has to be identified, which will ultimately be returned to the user. We have established a set of quality criteria and evaluated different methods to create a sensible ranking of the results, based on those criteria.

For later evaluation purposes, three different nuances of the original Multi-Hypothesis Parser were created, on which all ranking methods were evaluated: One minimal version, which contains only encoding detection, dialect detection, table area detection (only one table), simple row function detection (one header row), simple column function detection (only empty columns) and data type detection. One advanced version, which additionally contains the full row and column function detection capability. And one full version which also contains the table orientation detection, full column detection capability and wide/narrow data detection[5].

6.3.1 Quality Criteria

The key question is which measurable criteria are suitable to capture the quality of the parsing result. A starting point for these considerations is the ISO/IEC 25012 Data Quality Model [54] and complementary work by Rafique et al. [55] and Batini et al. [56]. Out of the 15 quality characteristics of the standard model, we considered accuracy/-correctness, completeness, consistency, precision, and understandability as characteristics which should be distinctive for a good parsing result. Additionally, we considered representational adequacy, as introduced by Rafique et al., as a desirable strength of the parsing result. The characteristics credibility, currentness/timeliness, accessibility, compliance, confidentiality, efficiency/speed, traceability, availability, portability, and recoverability were not considered as distinctive because they are not under control of the parsing process. The following will give an overview of how the different quality characteristics are defined and which respective measurements we propose:

Accuracy/Correctness "The degree to which data has attributes that correctly represent the true value of the intended attribute of a concept or event in a specific context of use" [54]. Because we do not know the original intent of the data, we can only assume that the original content is the intended content. In order to ensure that the original content of the data was not unnecessarily changed, the parsing steps should not have encountered errors. Critical errors will be disregarded, because the erroneous node will not be traversed anymore, but if less critical errors occur, in particular in the encoding and dialect parsing step, a warning is emitted. The number of warnings should be as low as possible, preferably zero. The number of edits and moves of cells content should also be as low as possible. Furthermore, we can assume that the confidence of each parsing step is a signal for the correctness of the parsing result. Results with a high underlying confidence should be preferred over results with low confidence. Additionally, the correctness of cell content could be backed of by background knowledge coming from the corpus or a knowledge base, by checking if the columns contain entities belonging to the concept described in the headers. The latter is left for future work, however.

Completeness "The degree to which subject data associated with an entity has values for all expected attributes and related entity instances in a specific context or

[5]To facilitate the evaluation against other parser, detection of multiple tables was also in the most advanced version deactivated.

use" [54]. We cannot know if the given data is really complete, we can again only assume that the provided data is as complete as possible. However, to ensure that as much as possible data from the original input is preserved, tables with higher number of total cells should be preferred.

Consistency "The degree to which data has attributes that are free from contradiction and are coherent with other data in a specific context of use" [54]. Although we cannot measure the external consistency with other sources, we can measure the internal consistency. The consistency of values within columns can be determined based on the specificy of the data type. Tables with a high number of typed cells (other than text) should be preferred. Additionally, the consistency could be determined based on coherence of the value distributions within columns, which is also left for future work.

Precision "The degree to which data has attributes that are exact or that provide discrimination in a specific context of use" [54]. The precision of a values can not be higher than provided in the original file. However, the more specifically a column is typed, the closer it potentially is to the intended precision. Also in terms of precision, tables with a high number of specifically typed cells should be preferred.

Understandability "The degree to which data has attributes that enable it to be read and interpreted by users, and are expressed in appropriate languages, symbols and units in a specific context of use" [54]. The understandability of a table depends strongly on weather column headers are given or not. Tables with a low amount of empty headers should thus be preferred. Since we often observed encoding issues, which also reduce the understandability of the table content, tables with more characters inside the latin character set should be preferred. Naturally, this only applies to tables from western origin.

Representational Adequacy "The extent to which data or information is represented in a concise, flexible and organized way with due relevancy to the users' goals to help user to achieve their specified goals" [55]. The conciseness of the table gets improved if unnecessary columns are rows are not contained in the table. Tables with a low amount of empty cells should thus be preferred. Furthermore, vertically oriented tables are easier to read for the user and thus better organized. Since vertically oriented tables tend to have more rows than columns, tables with a high row-column ration should preferred.

Table 6.1 summarizes the proposed mapping of quality metrics to quality characteristics.

Data Quality Characteristics	Quality Metrics
Accuracy/Correctness	Warnings Count, Edits, Moves, Confidence
Completeness	Total Cells
Consistency	Typed Cells
Precision	Typed Cells
Understandability	Empty Header, Non-Latin Characters
Representational Adequacy	Empty Cells, Row/Column Ratio

Table 6.1: Data quality characteristics mapped to quality metrics.

6.3.2 Ranking

In the following, we evaluated different ranking methods to properly rank the parsing results. The first and most naïve method is to traverse the hypothesis tree along the path which has the highest confidence. We will refer to this as the *naïve* method. The second method is to rank the results based on the previously identified quality metrics and to equally weigh their importance for the ranking. The third, and most sophisticated method, is to train a machine learning model in order to assess a proper weighting of the quality metrics. Ultimately, the approaches will be compared to a baseline, in which the results are randomly ranked.

We used a Ranking SVM algorithm [57] with a linear kernel function to determine a ranking model, based on the previously described quality criteria. Ranking SVM's are, e.g., used to rank query results in search engines and are well suited for the problem because they aim at creating a ranking with a minimum number of swapped pairs compared to the optimal ranking. This has the benefit over simple linear regression, that deviations from the model are only then penalized if they lead to a change in the ranking. We chose a linear kernel function, because the resulting model parameter are easily interpretable and can be manually checked for plausibility. In order to train the Ranking SVM, we gathered training data as follows: The randomly sampled tables from *data.gov.uk* (see Section 3.1) were manually sanitized [6]. Then the same sample of files was parsed with the full Multi-Hypothesis CSV parser and all parsing results were compared to the manually cleaned tables.

Because no method which is able to determine the edit distance between two tables could be identified in related work, we used a string-based distance method. To determine the

[6]The tables were read with a manually configured CSV parser. Subsequently, header rows were identified and moved to the table header and metadata was removed. Also were NA values identified and replaced by empty strings. If tables were not horizontally oriented they were transposed and if narrow or wide data was idenfied, it was reshaped accordingly. Column data types for dates, times, logicals and numerics were identified and the column content was casted accordingly. If numerics were attached with units, the unit was moved to the header row. The set of manually cleaned tables is available online: `https://github.com/tdoehmen/hypoparsr/tree/master/tests/data/cleaned`

string-based edit distance, the tables were collapsed column-wise, by appending all cells in a column to one string, and subsequently row-wise, by appending all column-strings to one string. This creates one large string, representing the whole table. The distance between two table strings was then measured using the Levenshtein distance [58]. The Levenshtein distance is a measure which determines the minimum required amount of insertions, deletions and edits to turn one given string into another. In that way we can properly assess the distance between tables in terms of removed, added and edited cell content. The method, however, over-penalizes moving of cells, as caused by reshaping or transposing of tables, which applies to only one table in the test set and will be taken into account in the evaluation. We used a memory-efficient implementation of the Levenshtein distance from the R package *RecordLinkage* [59] to compute the distances. Because the calculation of the Levenshtein distance is nevertheless computationally expensive, with a complexity of $O(mn)$, with m and n being the string lengths, we reduced the original set of 80 tables to a subset of 64 tables with a file size smaller than 200kB. The Levenshtein distances between the manually cleaned tables and the tables of the Multi-Hypothesis parser were then exported, together with the respective result features/quality metrics. Because we do not want to globally rank the parsing results, but only rank them relative to the other results in the same parsing tree, we normalized the quality metric values within a tree to a range between 0 and 1. In that way only metrics which differ within a tree have an impact on the ranking. Figure 6.12 illustrates the training data generation process.

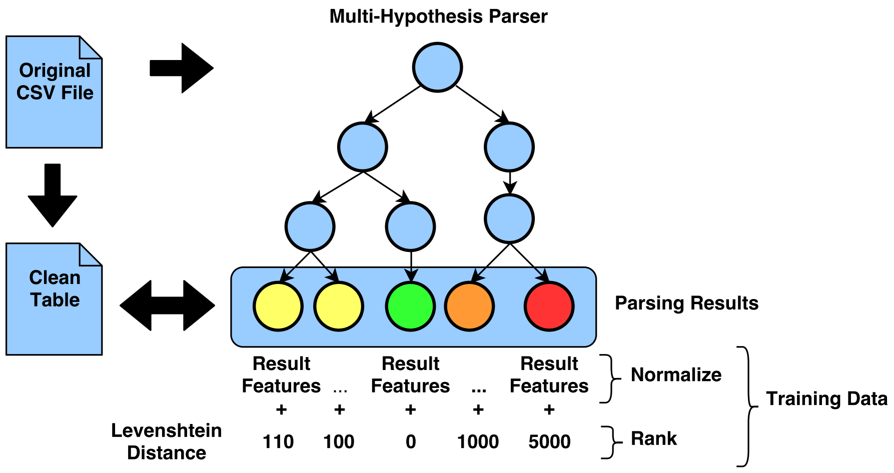


FIGURE 6.12: Schematic diagram of the training data generation process.

For a Ranking SVM it is furthermore "critical [...], that a proper value is chosen for C, the penalty factor. If it is too large, we have a high penalty for nonseparable points and we may store many support vectors and overfit. If it is too small, we may have underfitting." [60]. We performed a parameter estimation for C, with Monte Carlo cross validation [61] with 10 different randomly sampled training and test sets with a size ratio of 3/1. Ultimately, the best C parameter (0.01) was used to train the final ranking model, whose resulting weights are shown in Table 6.2, compared to the equal weights defined by us. The table shows that the sign of the weights is perfectly in line with our manually defined weights. Warnings, edits, moves, empty header, empty cells and non-latin characters get penalized and confidence, total cells, typed cells and a high row/column ratio get rewarded. Moves and warnings appear to have the highest negative impact on the ranking and empty header and non-latin characters the lowest. The number of typed cells appears to be the most distinctive positive feature.

Quality Metric	**Weighting**	
	Equally	Trained
Warnings	-1	-2.38
Edits	-1	-1.50
Moves	-1	-4.11
Confidence	1	1.57
Total Cells	1	1.42
Typed Cells	1	3.20
Empty Header	-1	-0.52
Empty Cells	-1	-1.04
Non Latin Chars	-1	-0.56
Row/Column Ratio	1	0.93

TABLE 6.2: Quality metrics, equally weighted and weights determined by the Ranking SVM. [7]

Figure 6.13 shows the success of the different ranking methods in comparison. We do not yet assess the overall quality of the parsing result, but only how accurately the best possible result is chosen from a certain set of results. The equal weighting method performs surprisingly well in comparison to the trained method. In the full, as well as in the advanced system, the trained ranking method outperforms the the untrained method by only 5% and 3% respectively, in terms of correctly highest ranked results. In the minimal system, the trained ranking even leads to a slightly lower success rate. However, they both perform substantially better than the random and the naïve ranking method. In 73-81% percent of all cases, the best parsing result is ranked highest. And in 86-98% of all cases the best parsing result can be found among the three highest ranked results. Interestingly, the naïve method, which follows the path of highest confidence, is only about 1/3 better than the random baseline, and in case of the minimal system even worse. This supports our hypotheses that parsing decisions, only based on high local confidence, do not lead to good overall parsing results. Although, it has to be taken into account that we did not gear the system towards highly reliable confidence values.

[7] In the original model the signs are inverted, because the ranking creates a descending list of tables, starting with the best one.

To assess how far off the falsely highest ranked tables actually are, we looked at the average edit distance to the best possible choice. Compared to a random sample from the same set, the ranking methods still perform better. One interesting observation was that the tables that were incorrectly ranked high were often eligible choices, which were simply not in line with how we manually cleaned tables. Section 7.1 will show an example of such a case.

We can conclude that, both, the equally weighted and the trained method are leading to satisfying ranking results. Because the ranking based on equally weighted features performs overall well and to prevent a bias from the rather small manually cleaned sample, we will use the equally weighted features for the final system, which will be evaluated in the following chapter.

However, the ranking is not 100% accurate and sometimes depends on the users' perspective. For future work, we plan to implement an option to show a set of highest ranked tables to the user and let him/her choose the preferred version of the table. Perspectively, the user input could then be used to re-train the Ranking SVM model and improve the ranking according to the users' preferences, similar to the concept of predictive user interaction [16].

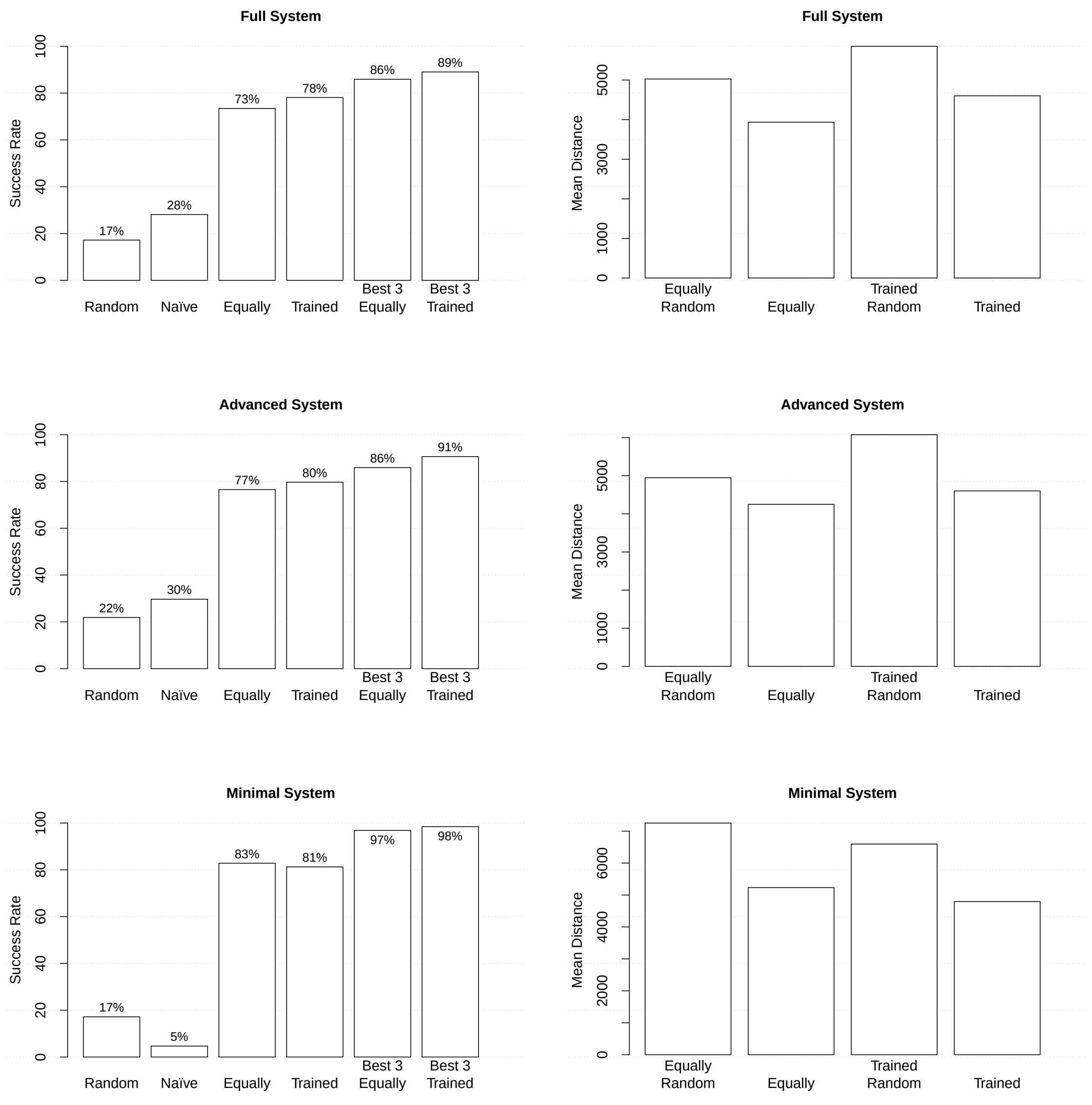


FIGURE 6.13: Full (top), advanced (middle), and minimal (bottom) system - Comparison of different ranking methods. Successfully top-ranked results (left) out of in average 17, 14 and 7 results, respectively. And the mean distance of falsely top-ranked results (right) in comparison to a random choice from the same set.

Chapter 7

Evaluation

In this chapter the proposed Multi-Hypothesis parser is evaluated. In the first section we evaluate the quality of the parsing results against a set of manually cleaned tables, which has already been described in the previous section and whose issues have been thoroughly discussed in Section 3.1. Furthermore, we evaluate the parser on all CSV files from the open government data portal *data.gov.uk*. The corpus, which was crawled in August 2015, was reduced by only keeping files than smaller than 200kB and was superficially cleaned from non-CSV and empty files. It contains 14844 files in total, which make up approx. 90% of all CSV files on the portal. In order to investigate the impact of different detection/parsing steps on the parsing result, we evaluated three different versions of the Multi-Hypothesis parser (See Figure 7.1). The minimal version (Min) contains only

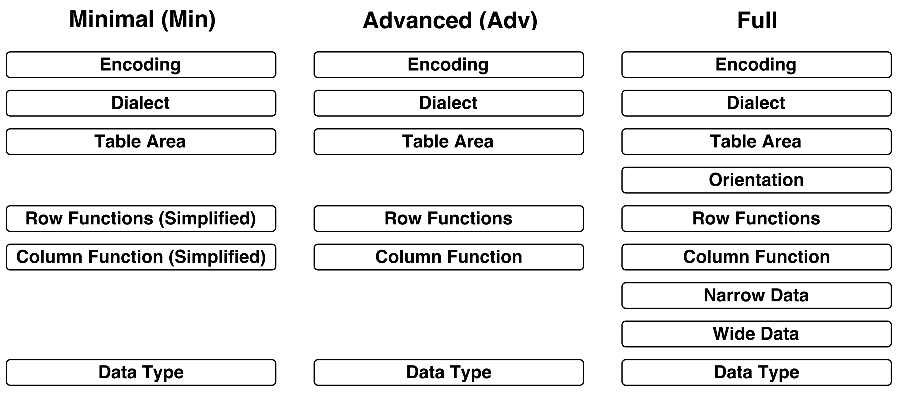


FIGURE 7.1: The three different variants of the Multi-Hypothesis parser which will be evaluated.

encoding detection, dialect detection, table area detection (only one table), simple row function detection (first row header or not), simple column function detection (only empty columns), and data type detection. The advanced version (Adv) additionally contains the full row and column function detection capability. The full version (Full) contains all parsing steps, including table orientation detection, and wide/narrow data detection .[1]

7.1 Table Quality

The performance of the Multi-Hypothesis parser was compared to a baseline, which is a strict RFC 4180 conform CSV parser, as implemented in the R base library [50]. Additionally we evaluated against the state-of-the-art CSV parser *messytables*[2], which was especially designed for reading messy tabular data (Messy). As a third comparison, we evaluated the application *DeExcelerator* [39], which has been discussed in related work (DeExel). Because *messytables* and *DeExcelerator* have complementary capabilities, we created an additional workflow, which combines both solutions (Messy.DeExcel) to a linear workflow, which has about the same scope of functionality as the proposed Multi-Hypothesis parser. We parsed the CSV files with each of the mentioned solutions and compared the result to the manually cleaned tables, using the string-distance measure described in Section 6.3.2. If a file could not be read by a respective parser, we counted the edit distance between an empty string and the cleaned table.

Figure 7.2 shows the median Levenshtein distance between parsing results and the gold standard for each solution. The figure shows that the parsing results of Multi-Hypothesis parser are in median significantly closer to the manually cleaned tables, than the results of compared systems. The advanced and full Multi-Hypothesis parser created tables, which were in median only 4.5 character-edits/insertions/deletes away from the manually cleaned table. If always the best result would have been rated highest, then the distance could have been further reduced to 3. The minimal version of the Multi-Hypothesis parser performs significantly worse than the advanced and full version, but still leads in median to better results than *messytables* and *DeExcelerator*.

Table 7.3 shows how many tables were almost exact matches with the manually cleaned table (less than 10 character-edits/insertions/deletes). The RFC 4180 conform parser did not parse one single table correctly. This underlines how error-prone the CSV files are and the necessity for more sophisticated CSV parsing solutions. The Multi-Hypothesis parser was able to parse, clean, and normalize 38 out of 64 CSV files in the almost exact way as we manually cleaned them, out of which 35 were also ranked highest. One single table could only be parsed correctly by the full version of the Multi-Hypothesis parser.

[1]To facilitate the evaluation against other parser, detection of multiple tables was also in the most advanced version deactivated.

[2]`https://github.com/okfn/messytables`

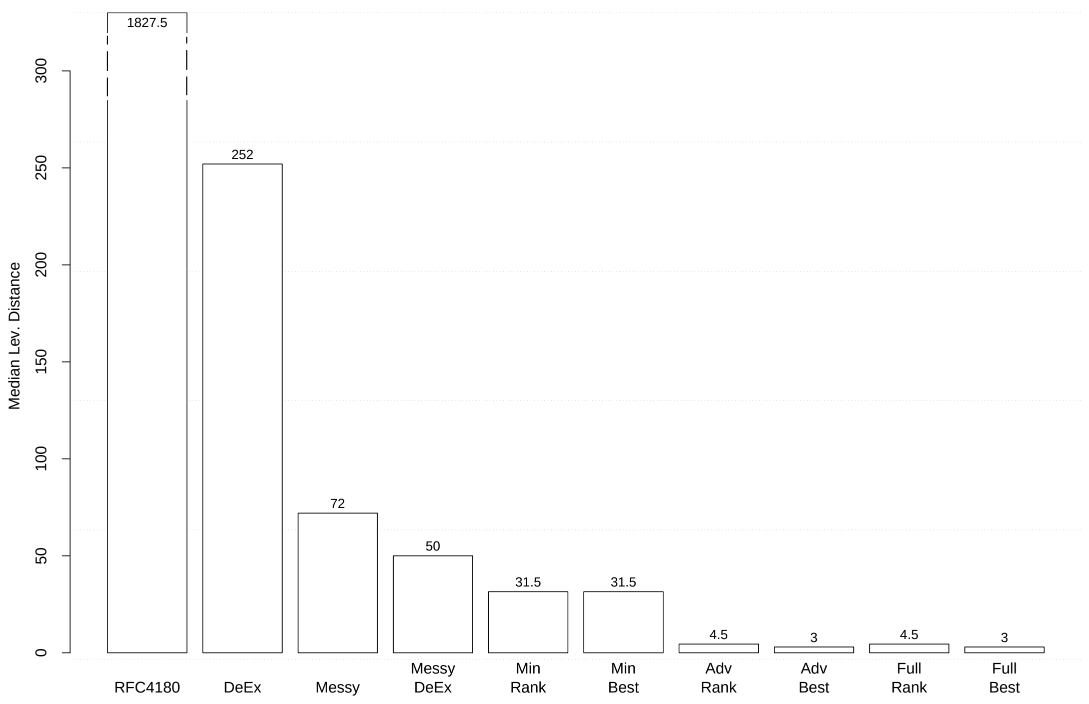


Figure 7.2: Median edit distances to manually cleaned tables, with *DeExcelerator*, *messytables*, *messytables* combined with *DeExcelerator*, and the highest ranked (Min.Rank, Adv.Rank, Full.Rank) and the best possible (Min.Best, Adv.Rank, Full.Rank) table with all versions of the Multi-Hypothesis parser.

Figure 7.4 gives a more detailed insight into the distribution of Levenshtein distances to the manually cleaned tables, for each parser. The plot shows that each parser, also the Multi-Hypothesis parser, left a certain group of not well cleaned tables behind.

To get a better understanding of which kind of issues were still present, we reviewed the deviating parsing results manually. High distances from *messytables* parsing results can be explained by the fact that *messytables* does not remove empty rows and columns from the parsing result. Additionally, it does not detect multiple header rows and metadata at the footer of the table. Furthermore, the detection of numeric values is limited to values without thousand separators and American decimal marks. Header detection and date detection are, however, performed mostly accurately. *DeExcelerator* has issues with CSV dialect detection and parsing of dates. But *DeExcelerator* detects differently formatted numeric values well, detects metadata at the head and tail of the table in many cases correctly and expands spanning cells properly. The main factors which reduce the performance of the combination of *messytables* and *DeExcelerator* are still header detection issues, metadata detection issues, limited NA value detection and a missing detection of narrow and wide data. Figure 7.5 and Figure 7.6 show correct parsing results of the Multi-Hypothesis parser, compared to false results, created by *messytables+DeExcelerator*.

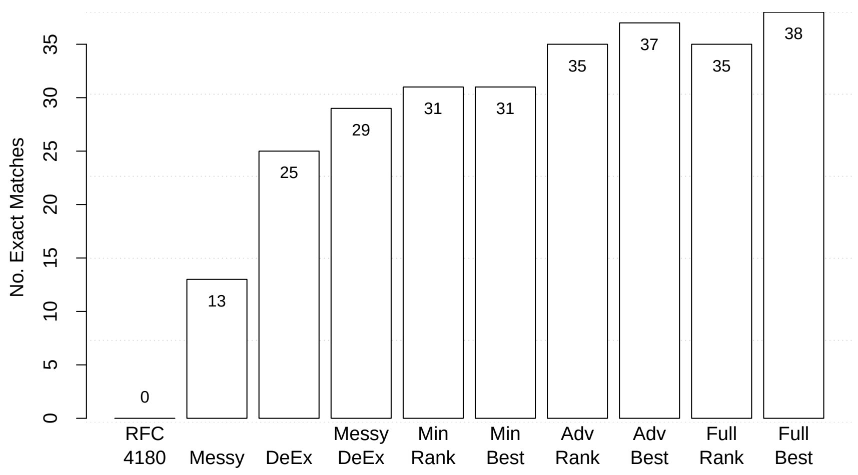


FIGURE 7.3: Number of exact matches (max off by edit distance 10) with RFC 4180 conform parser, *messytables*, *DeExcelerator*, *messytables* combined with *DeExcelerator*, and the Multi-Hypothesis parser. Results of the Multi-Hypothesis parser are shown for all three variants and for the highest ranked and the best match which could be found in the whole parsing tree, regardless of the ranking.

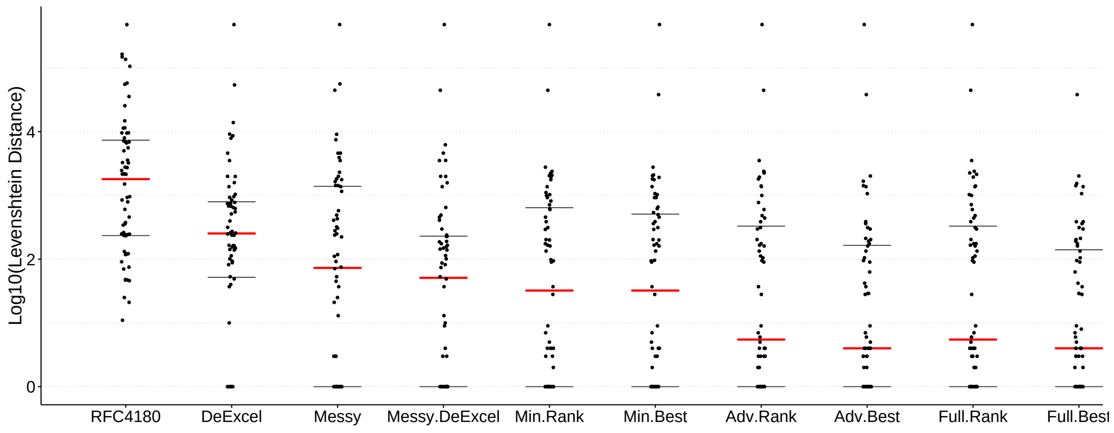
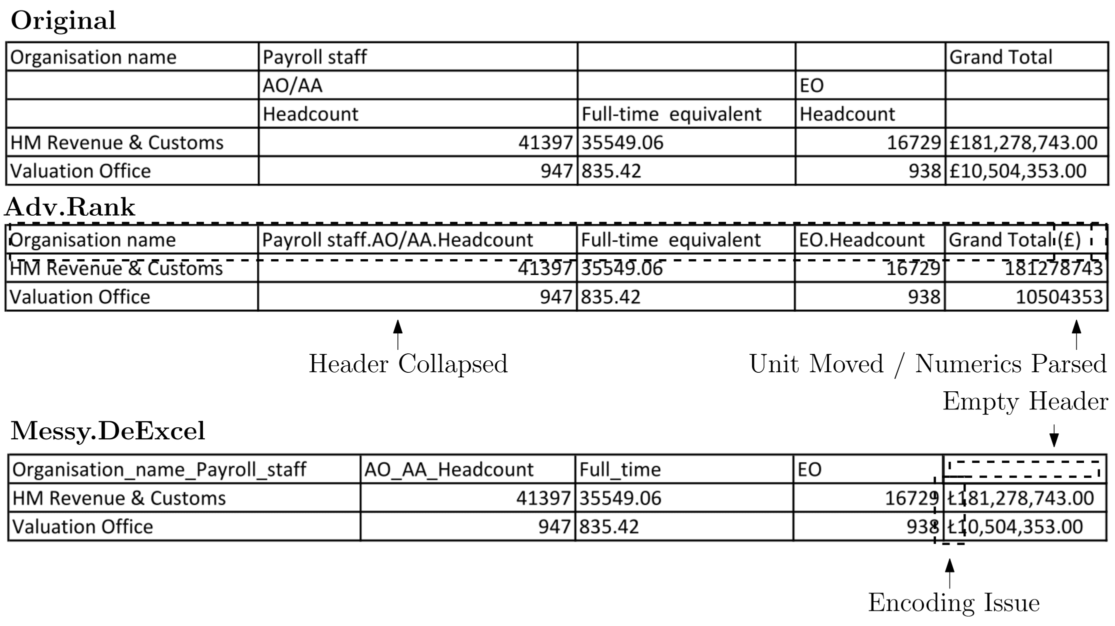


FIGURE 7.4: Edit distance to manually cleaned tables, with RFC 4180 conform parser, *messytables*, *DeExcelerator*, *messytables* combined with *DeExcelerator*, and the highest ranked (Min.Rank, Adv.Rank, Full.Rank) and the best possible (Min.Best, Adv.Rank, Full.Rank) table with all versions of the Multi-Hypothesis parser. The red bar marks the median and the black bars mark the inter-quartile range. The y-axis is logarithmically scaled.

Nevertheless, the combination of *messytables* and *DeExcelerator* leads to good overall results, which come closest to the quality of the Multi-Hypothesis parser and in some cases even lead to better results (see Figure 7.7).

Although the implementation of the Multi-Hypothesis parser performs overall better than the other systems, in 26 cases, none of the generated hypotheses led to a parsing result which was in line with the manually cleaned version of the table. The most



Original

Organisation name	Payroll staff			Grand Total
	AO/AA		EO	
	Headcount	Full-time equivalent	Headcount	
HM Revenue & Customs	41397	35549.06	16729	£181,278,743.00
Valuation Office	947	835.42	938	£10,504,353.00

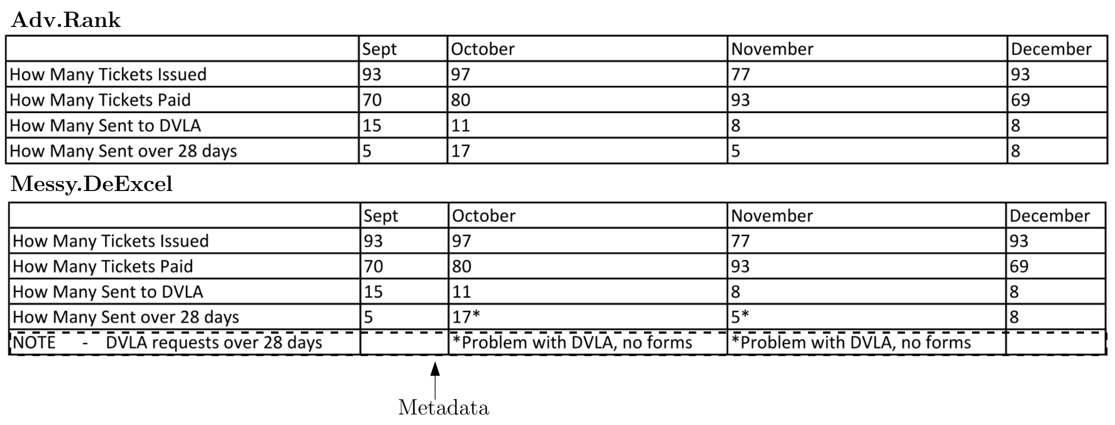
Adv.Rank

Organisation name	Payroll staff.AO/AA.Headcount	Full-time equivalent	EO.Headcount	Grand Total (£)
HM Revenue & Customs	41397	35549.06	16729	181278743
Valuation Office	947	835.42	938	10504353

Messy.DeExcel

Organisation_name_Payroll_staff	AO_AA_Headcount	Full_time	EO	
HM Revenue & Customs	41397	35549.06	16729	£181,278,743.00
Valuation Office	947	835.42	938	£10,504,353.00

FIGURE 7.5: Table with multiple header rows and units, parsed by Multi-Hypothesis parser compared to the same parsed by *messytables* and *DeExcelerator*. Issues in this table were thoroughly described in Section 3.1



Adv.Rank

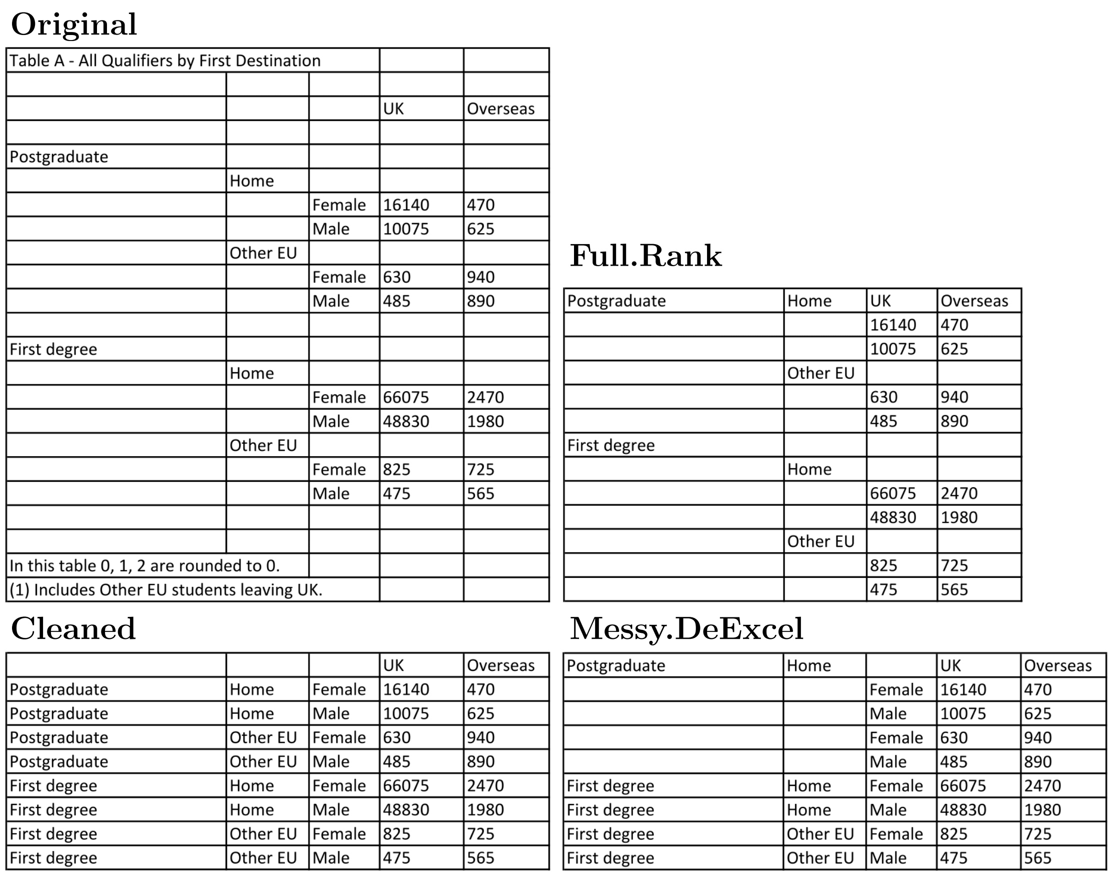
	Sept	October	November	December
How Many Tickets Issued	93	97	77	93
How Many Tickets Paid	70	80	93	69
How Many Sent to DVLA	15	11	8	8
How Many Sent over 28 days	5	17	5	8

Messy.DeExcel

	Sept	October	November	December
How Many Tickets Issued	93	97	77	93
How Many Tickets Paid	70	80	93	69
How Many Sent to DVLA	15	11	8	8
How Many Sent over 28 days	5	17*	5*	8
NOTE - DVLA requests over 28 days		*Problem with DVLA, no forms	*Problem with DVLA, no forms	

FIGURE 7.6: Table with falsely detected metadata by *messytables* and *DeExcelerator*, compared to the correctly parsed table by the Multi-Hypothesis parser.

frequently observed shortcomings are discussed in the following: In 8 out of 26 cases, dates were not parsed correctly, due to following issue: Dates, which not explicitly state "2012", but the abbreviation "12", were parsed as dates in the year 12 A.D instead of 2012. In 2 out of 26 cases, numerics with leading zeros were parsed as numbers, while they were kept as character strings (i.e. IDs) in the cleaned version. In 4 out of 26 cases, the results had issues with remaining metadata at the end of the table. This occurred only in cases in which the metadata fitted exactly to the column data types. In 4 out of 26 cases, the input had duplicated columns, which were removed in the manually cleaned version, but not tackled by the Multi-Hypothesis parser. In 2 out of 26 cases, the header detection did not succeed. This issue especially occurred in very small tables. In 2 out of 26 cases one data- or metadata row was mistakenly considered as part of the



Original

Table A - All Qualifiers by First Destination				
			UK	Overseas
Postgraduate				
	Home			
		Female	16140	470
		Male	10075	625
	Other EU			
		Female	630	940
		Male	485	890
First degree				
	Home			
		Female	66075	2470
		Male	48830	1980
	Other EU			
		Female	825	725
		Male	475	565
In this table 0, 1, 2 are rounded to 0.				
(1) Includes Other EU students leaving UK.				

Full.Rank

Postgraduate	Home	UK	Overseas
		16140	470
		10075	625
	Other EU		
		630	940
		485	890
First degree			
	Home		
		66075	2470
		48830	1980
	Other EU		
		825	725
		475	565

Cleaned

			UK	Overseas
Postgraduate	Home	Female	16140	470
Postgraduate	Home	Male	10075	625
Postgraduate	Other EU	Female	630	940
Postgraduate	Other EU	Male	485	890
First degree	Home	Female	66075	2470
First degree	Home	Male	48830	1980
First degree	Other EU	Female	825	725
First degree	Other EU	Male	475	565

Messy.DeExcel

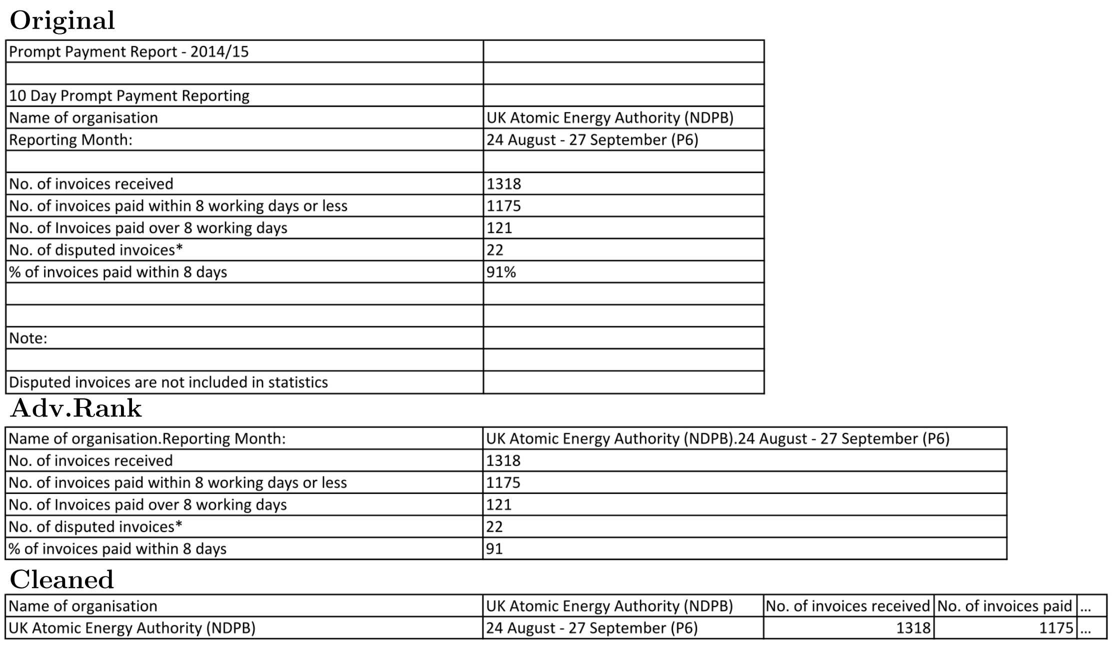
Postgraduate	Home		UK	Overseas
		Female	16140	470
		Male	10075	625
		Female	630	940
		Male	485	890
First degree	Home	Female	66075	2470
First degree	Home	Male	48830	1980
First degree	Other EU	Female	825	725
First degree	Other EU	Male	475	565

FIGURE 7.7: Table with inflated table structure and parsing results of *messytables+DeExcelerator* and the Multi-Hypothesis parser in comparison. In this case, the *messytables+DeExcelerator* lead to better parsing results than the Multi-Hypothesis parser. The first row was correctly discarded as metadata by both. The following three non-empty rows were false considered as header rows by both. Resulting from this, also the "Postgraduate" and the "Home" cells were considered as part of the header. Due to this, the Multi-Hypothesis parser did not further expand the spanning cells in column one and two. *messytables+DeExcelerator*, again, at least expanded the "First Degree", "Homes", and "Other EU" values. Because the "Male"/"Female" column did not contain a header and was sparsely filled, it was falsely considered as metadata by the Multi-Hypothesis parser. *messytables+DeExcelerator*, furthermore, removed rows which did not contain any numeric data.

header row. In 2 out of 26 cases the table should have been reshaped or transposed in a way that was not accounted for by the Multi-Hypothesis parser. See Figure 7.9 and Figure 7.8.

In three cases, the correct hypothesis was created, but not rated highest. Figure 7.10 shows a one of those cases. The highest ranked hypothesis was to read the file disregarding quotes, which turned the thousand separator into cell separator. Although the parser produced a warning which penalized this choice, it was still preferred because of the gained amount of numeric cells.

Figure 7.11 shows a case in which the highest ranked version of the table did not contain all possible data rows. Because the lower part contained empty cells, they were



Original

Prompt Payment Report - 2014/15	
10 Day Prompt Payment Reporting	
Name of organisation	UK Atomic Energy Authority (NDPB)
Reporting Month:	24 August - 27 September (P6)
No. of invoices received	1318
No. of invoices paid within 8 working days or less	1175
No. of Invoices paid over 8 working days	121
No. of disputed invoices*	22
% of invoices paid within 8 days	91%
Note:	
Disputed invoices are not included in statistics	

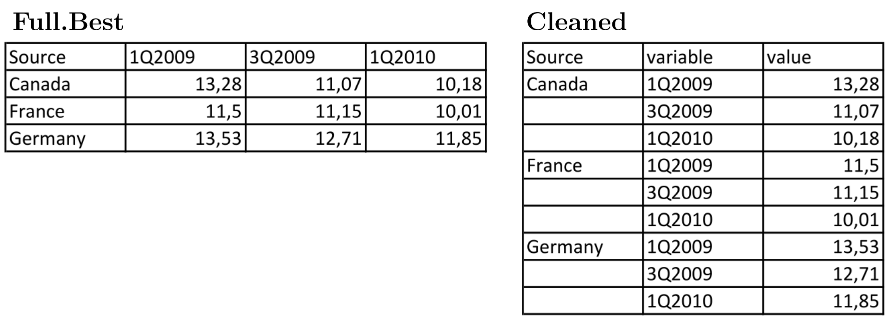
Adv.Rank

Name of organisation.Reporting Month:	UK Atomic Energy Authority (NDPB).24 August - 27 September (P6)
No. of invoices received	1318
No. of invoices paid within 8 working days or less	1175
No. of Invoices paid over 8 working days	121
No. of disputed invoices*	22
% of invoices paid within 8 days	91

Cleaned

Name of organisation	UK Atomic Energy Authority (NDPB)	No. of invoices received	No. of invoices paid	...
UK Atomic Energy Authority (NDPB)	24 August - 27 September (P6)	1318	1175	...

FIGURE 7.8: Table, which was correctly parsed, but in a different style than suggested in the cleaned tables.



Full.Best

Source	1Q2009	3Q2009	1Q2010
Canada	13,28	11,07	10,18
France	11,5	11,15	10,01
Germany	13,53	12,71	11,85

Cleaned

Source	variable	value
Canada	1Q2009	13,28
	3Q2009	11,07
	1Q2010	10,18
France	1Q2009	11,5
	3Q2009	11,15
	1Q2010	10,01
Germany	1Q2009	13,53
	3Q2009	12,71
	1Q2010	11,85

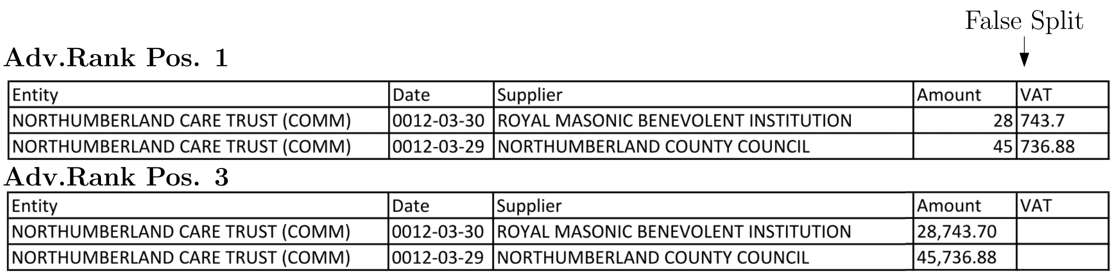
FIGURE 7.9: Table which was reshaped in cleaned version, but not by the Multi-Hypothesis parser. Variables in the header were "Q1 2013" etc., which were not detected as numerics, and thus also not as potential variables.

considered as metadata, reducing the amount of empty cells in the table. Consequently, this table was preferred over the correct parsing result which was ranked third (see Table 7.1).

Rank	Confidence	Typed Cells	Empty Cells
1	1	1	0
2	0.2	1	0
3	1	0.872	1

TABLE 7.1: Quality metrics for three highest ranked tables. Refers to Figure 7.11

Table 7.12 shows the original wide table and the correctly reshaped table created by the full versions of the Multi-Hypothesis parser, which is perfectly in line with the manually cleaned version of the table. The Multi-Hypothesis parser ranked the untouched version,


False Split

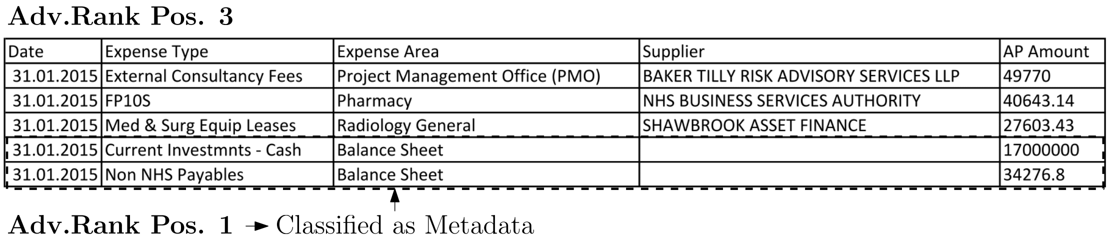
Adv.Rank Pos. 1

Entity	Date	Supplier	Amount	VAT
NORTHUMBERLAND CARE TRUST (COMM)	0012-03-30	ROYAL MASONIC BENEVOLENT INSTITUTION	28	743.7
NORTHUMBERLAND CARE TRUST (COMM)	0012-03-29	NORTHUMBERLAND COUNTY COUNCIL	45	736.88

Adv.Rank Pos. 3

Entity	Date	Supplier	Amount	VAT
NORTHUMBERLAND CARE TRUST (COMM)	0012-03-30	ROYAL MASONIC BENEVOLENT INSTITUTION	28,743.70	
NORTHUMBERLAND CARE TRUST (COMM)	0012-03-29	NORTHUMBERLAND COUNTY COUNCIL	45,736.88	

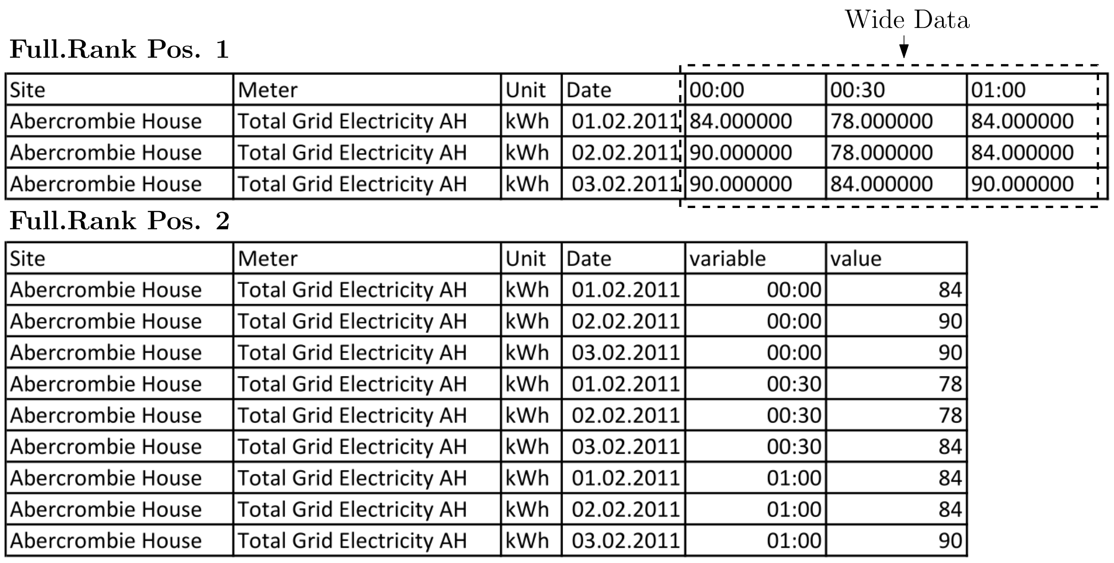
FIGURE 7.10: Table with false split.


Adv.Rank Pos. 3

Date	Expense Type	Expense Area	Supplier	AP Amount
31.01.2015	External Consultancy Fees	Project Management Office (PMO)	BAKER TILLY RISK ADVISORY SERVICES LLP	49770
31.01.2015	FP10S	Pharmacy	NHS BUSINESS SERVICES AUTHORITY	40643.14
31.01.2015	Med & Surg Equip Leases	Radiology General	SHAWBROOK ASSET FINANCE	27603.43
31.01.2015	Current Investmnts - Cash	Balance Sheet		17000000
31.01.2015	Non NHS Payables	Balance Sheet		34276.8

FIGURE 7.11: Table with mistakenly as metadata classified rows ranked highest.

however, highest and the reshaped version second. The penalty for the introduced moves slightly overruled the reward for gained typed cells (see Table 7.2). This shows that the current weighting of the quality metrics is rather risk-averse, but still creates a reasonable ranking because the "repaired" variant of the table is nevertheless ranked on high.



Full.Rank Pos. 1

Site	Meter	Unit	Date	00:00	00:30	01:00
Abercrombie House	Total Grid Electricity AH	kWh	01.02.2011	84.000000	78.000000	84.000000
Abercrombie House	Total Grid Electricity AH	kWh	02.02.2011	90.000000	78.000000	84.000000
Abercrombie House	Total Grid Electricity AH	kWh	03.02.2011	90.000000	84.000000	90.000000

Full.Rank Pos. 2

Site	Meter	Unit	Date	variable	value
Abercrombie House	Total Grid Electricity AH	kWh	01.02.2011	00:00	84
Abercrombie House	Total Grid Electricity AH	kWh	02.02.2011	00:00	90
Abercrombie House	Total Grid Electricity AH	kWh	03.02.2011	00:00	90
Abercrombie House	Total Grid Electricity AH	kWh	01.02.2011	00:30	78
Abercrombie House	Total Grid Electricity AH	kWh	02.02.2011	00:30	78
Abercrombie House	Total Grid Electricity AH	kWh	03.02.2011	00:30	84
Abercrombie House	Total Grid Electricity AH	kWh	01.02.2011	01:00	84
Abercrombie House	Total Grid Electricity AH	kWh	02.02.2011	01:00	84
Abercrombie House	Total Grid Electricity AH	kWh	03.02.2011	01:00	90

FIGURE 7.12: Table with and without reshaped wide data, ranked by the Multi-Hypothesis parser.

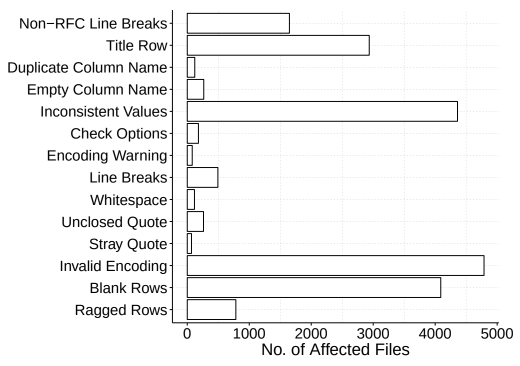
Rank	Moves	Confidence	Typed Cells
1	0	0.759	0.510
2	1	0.862	1

TABLE 7.2: Quality metrics for two highest ranked tables. Refers to Figure 7.12

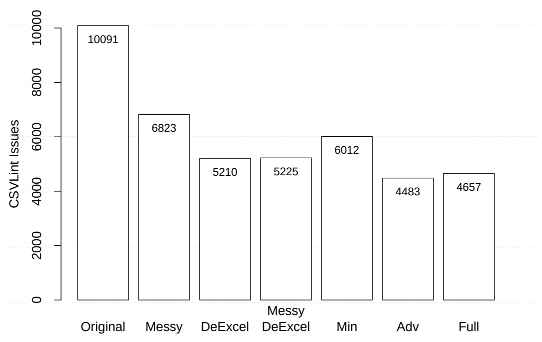
7.2 CSVLint Issues

Because the amount of tables in the whole corpus exceeds the amount we can manually assess, we used the CSV validation tool CSVLint to evaluate the quality of the parsing results. CSVLint showed to be well in line with manual evaluation, with a precision of 0.91 and accuracy of 0.78 in terms of files identified as affected by issues, based on the previously used sample from *data.gov.uk*. We evaluate the quality of parsing results by writing them back to RFC 4180 conform CSV files and letting them again check by CSVLint.

Figure 7.13a shows the variety and number of issues which were present in the original set of files from *data.gov.uk*. Invalid encoding, inconsistent values, blank rows and title rows are the most often occurring issues. In total, approximately 10000 out of 14884 files are found to have issues. Figure 7.13b shows the amount of issues in the "cleaned" versions, produced by the Multi-Hypothesis parser and the systems we compare against. Most issues could be resolved by the advanced version of the Multi-Hypothesis parser. The second best cleaning results could be achieved by the combination of *messytables* and *DeExcelerator*. The minimal and also the full version of the Multi-Hypothesis parser lead to worse results, yet, better results than *messytables* and *DeExcelerator* alone. The combination of *messytables* and *DeExcelerator* did not further reduce the amount of issues.



(A) CSVLint issues in the original files.



(B) CSVLint issues after parsing and writing back to RFC4180-conform CSV.

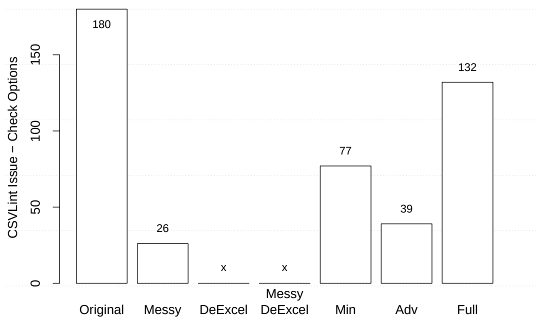
FIGURE 7.13: CSVLint issues in the *data.gov.uk* corpus.

Many issues are simply solved by reading the file and writing it back to RFC 4180-conform CSV. However, three issues which are not related to that are the *check options*, *title rows* and *inconsistent values*. The *check options* issue is triggered when the table contains only one column, which indicates that the delimiter was not properly detected. The *title rows* issue gets triggered when when the first row contains significantly more empty field than the other rows. The *inconsistent values* issue gets triggered when a column contains less then 90% of values which appear to have the same type. Checked data types are characters, numerics, dates and times. This is an indicator for various issues, such as metadata in the table structure or inconsistent value formatting.

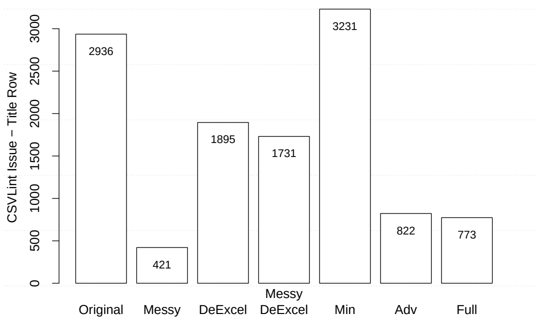
Figure 7.14a shows that the amount of tables with only one column is lowest for files parsed by *messytables*. This is an indicator that the dialect detection of *messytables* is very reliable. The advanced version of the Multi-Hypothesis parser generates slightly more one-column tables and the minimal and full version considerably more. Because all three versions use the same dialect detection, this indicates that the dialect detection itself is not the problem, but rather the table ranking. Because the quality ranking prefers tables with high row/column ration, this is a very reasonable assumption. *DeExcelerator* appears to discard one-column tables, which leads to a zero issue count for *DeExcelerator* and *messytables+DeExcelerator*, which is in this case not desirable.

Figure 7.14b shows that the amount of tables which are suspected to still have a title in the first row. Again, *messytables* yields the lowest number of issues. This indicates that the header detection of *messytables* is as well very reliable. The advanced and full versions of the Multi-Hypothesis parser also strongly reduce the number of potential title rows, although they do not reach the same performance as *messytables*. The minimal version of the Multi-Hypothesis parser, which only takes the first row of the table as potential header into account makes the original tables even worse than they initially were. This emphasized the importance of header detection beyond the first row of the table. DeExcelerator does not very successfully reduce the number of potential title rows, which affects the combination of *messytables* and *DeExcelerator* as well.

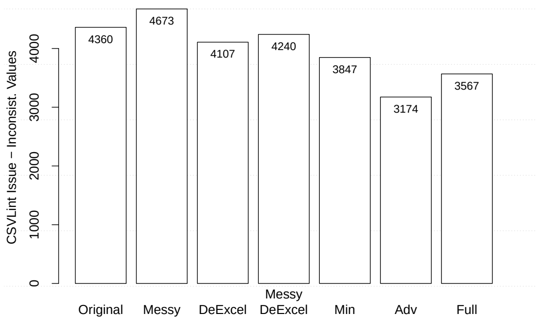
Figure 7.14c shows the amount of *inconsistent values* issues. The advanced version of the Multi-Hypothesis parser shows a strong reduction of those issues. Also the minimal and full version resolve the issue more effective than the compared solutions. This suggests that the Multi-Hypothesis parser results have a tendency towards consistently typed columns. The increase of inconsistently typed columns in the full version indicates further that the full version has issues with proper ranking of the parsing results. Because the amount of hypotheses in the full version is considerably higher than in the advanced version, it is reasonable that the quality assessment method is more disposed to sub-optimal decisions. The parsing results of *messytables* have an even higher inconsistency than the original files, which is surprising. One explanation for this is that the numeric detection of *messytables* does not detect numerics with commas as thousand separator. When writing the result back to a CSV with optional quoting, only those cells have to be quoted. CSVLint, again interprets those quoted numbers as strings. When the original version contained consistently quoted numerics, then the new version will have a higher inconsistency. However, it shows that the inconsistent values measure is a rather unreliable measure for the consistency of values. In the following section we will evaluate the parsing result quality based on more reliable measures.



(A) *Check Option* Issues



(B) *Title Row* Issues



(C) *Inconsistent Values* Issues

FIGURE 7.14: CSVLint issues after parsing the *data.gov.uk* corpus with different parsers and writing back to RFC4180-conform CSV.

7.3 Amount of Recovered Data

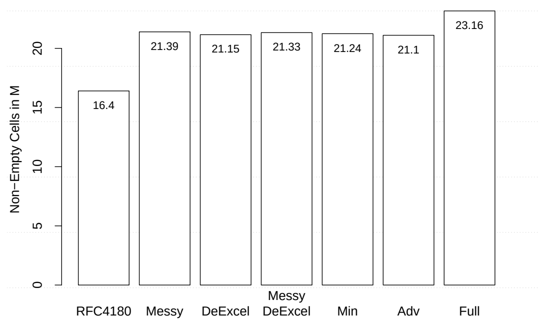
For the data consumer it is important how much of the input data can be preserved or recovered from the input files and ultimately be used for analysis or data integration. We use three simple measures to assess the amount of preserved tabular content. First, the amount of non-empty cells. The more data cells are preserved, the better. Second, the amount of names cells, i.e. cells which have an assigned column header because without a column header the data loses its interpretability. And third, the amount of typed cells, i.e. cells which are of a specific data type other than character. The more specific the data type, the easier the data can be interpreted. It has to be noted that cells are only then counted as typed, when the whole column has a consistent data type. Thus, in that way also data consistency gets properly evaluated.

The most vital requirement is, however, to be able to read the input in the first place. Table 7.3 shows the number of successfully parsed files out of 14844 files in the original corpus. The RFC 4180 conform parser fails to read the files in about 1/4 of all cases. *messytables* and the Multi-Hypothesis parser were able to read the largest amount of files with 14,762 and 14,755, respectively. *DeExcelerator* could only read approx. 140 files less. The majority of the remaining files, which could neither be read by *messytables* nor by the Multi-Hypothesis parser were non-CSV files which were not removed by the superficial cleaning of the test set. Only 22 out of the remaining 87 files were actual CSV files on which the Multi-Hypothesis parsing failed. In those 22 cases, more than 10 dialect hypotheses were created. Due to equal weighting of the dialect confidence, all hypotheses dropped below the pruning level of 0.1 and were thus disregarded. In future work, pruning in these cases should be prevented.

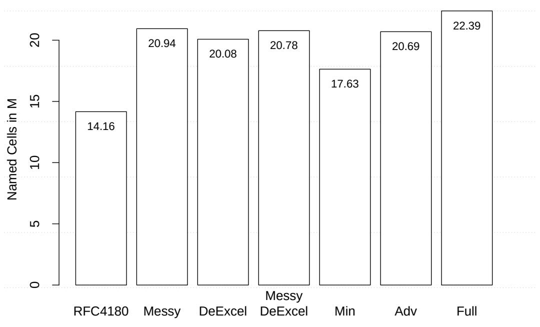
Parser	RFC4180	Messy	DeExcel	Messy.DeExcel	Min	Adv	Full
No. of Files	11,619	14,762	14,621	14,706	14,755	14,755	14755
% of Files	78.27	99.45	98.49	99.07	99.40	99.40	99.40

TABLE 7.3: Number of successfully parsed files of the Multi-Hypothesis parser, compared to others.

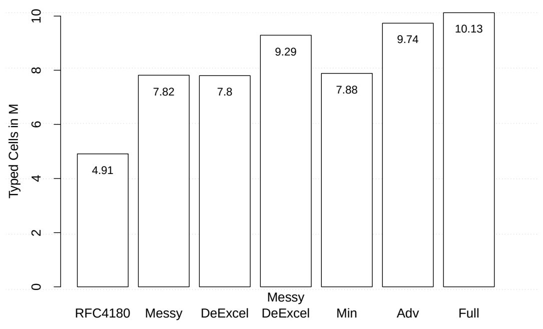
Figure 7.15a shows that the full version of the Multi-Hypothesis parser did recover the highest amount of non-empty cells from the *data.gov.uk* corpus with 23.16 million cells. However, we have to take into account that the reshaping actually adds cells to the original data. In approx. 60 cases, large tables got successfully reshaped and ranked highest, which probably explains a big part of the gained cells. The amount of recovered cells from the advanced version gives thus a better picture of the amount of cells stemming from the original input. In comparison to *messytables*, *messytables* does recover slightly more non-empty cells from the input. This is reasonable because the parsing success is high and metadata, except the metadata above the header row, does not get removed. Considering named cells, *messytables* does also recover slightly more than the Multi-Hypothesis parser (see Figure 7.15b). This again confirms the good performance of the *messytables* header detection. However, the amount of named cells in the parsing results of the Multi-Hypothesis parser is only slightly lower. Finally, the amount of typed cells uncovers the strong advantage of the Multi-Hypothesis parser (see Figure 7.15c). The parser creates about 1.9M more consistently typed cells than *messytables* or *DeExcelerator*. Only the combination of both can achieve a comparable amount of consistently typed cells.



(A) Non-Empty Cells



(B) Named Cells



(C) Typed Cells

FIGURE 7.15: Amount of recovered data cells from *data.gov.uk* corpus with the Multi-Hypothesis parser, compared to others.

7.4 Hypothesis Tree Size

The tree structure of the parsing process, was expected to be subject to exponential growth. We counteracted this by pruning of the hypothesis tree and by conservative creation on hypotheses. Theoretically, 30 different hypotheses about the encoding can be created, 450 different hypotheses about the dialect, only one about the table area, two about the orientation, 8 different row function hypotheses, two column function hypotheses, three different wide data hypotheses, two different narrow data hypotheses and one data type hypothesis. This leads to a theoretical maximum of 2,592,000 parsing hypotheses:

$$30 * 450 * 1 * 2 * 8 * 2 * 3 * 2 * 1* = 2,592,000$$

When limiting the amount of hypotheses by pruning to maximum 10, the number gets reduced to 19,200 hypotheses:

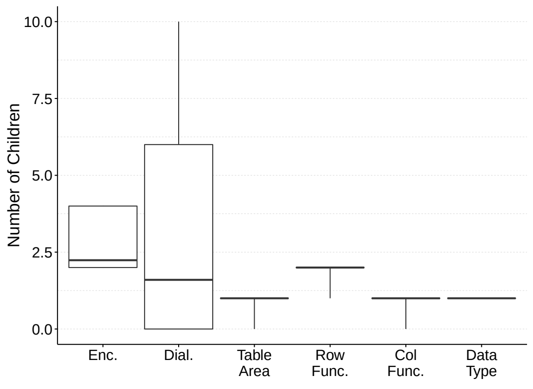
$$10 * 10 * 1 * 2 * 8 * 2 * 3 * 2 * 1 = 19,200$$

However, in practice we observed far lower numbers of hypotheses. Table 7.4 shows that in median 8 and in mean between 7 and 17 different hypotheses reach the end of the hypotheses tree. Only in rare cases, the number of intermediates grows much larger. This was especially observed when the parsing process could not make sense of the input. In one case, in which the input was an Spreadsheet export with dozens of different tables concatenated in one file, 356 hypothesis were created.

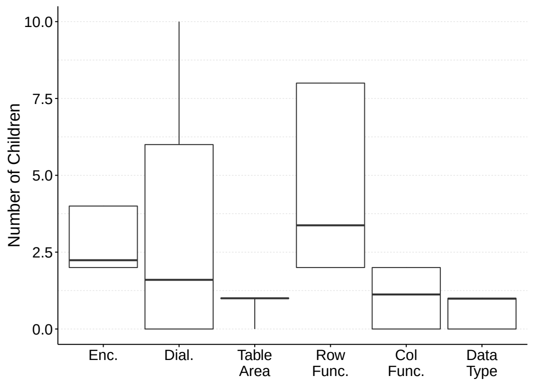
Variant	**No. Hypotheses in Last Level**					
	Min	1st. Quartile	Median	Mean	3rd. Quartile	Max
Minimal	2	4	8	7.2	8	48
Advanced	2	6	8	13.4	16	188
Full	2	8	8	17.3	20	356

Table 7.4: Number of Hypotheses in the last level of the parsing hierarchy of different variants of the Multi-Hypotheses Parser. Observed in 14755 parsing processes.

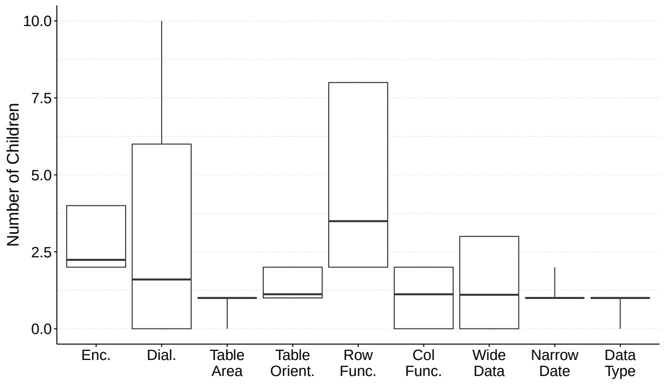
Figure 7.16 shows the amount of hypotheses per parsing step for different versions of the parser. The dialect detection and the row function detection step show the largest variety in number of created hypotheses. The chance that many different dialects pass that plausibility check is extremely low.



(A) Minimal System



(B) Advanced System



(C) Full System

FIGURE 7.16: Number of created hypotheses, per detection/parsing step. Lower and upper boundary of the boxes mark the 0.5% and 99.5% percentile, and the whiskers the minimum and maximum values.

Chapter 8

Discussion

8.1 How large does the hypothesis tree grow?

It has been shown that the growth of the hypothesis tree is modest. On the average 7.2, 13.4, and 17.3 hypotheses were create by the minimal, advanced and full version, respectively. Only in rare cases up to 356 hypotheses were created. In the current implementation the number of hypotheses equals the number of copies of the input data that have to be kept in memory. For processing of large tables on regular hardware, the amount of hypotheses could still be a potential bottleneck. Instead of explicitly creating the variations of the input table and passing them on from step to step, they could practically be applied "on-the-fly" on the original table, reducing the memory consumption to one single table. Generally, given the small size of the hypotheses tree, it would be acceptable to create a higher number of hypotheses in the detection steps or to generally add more steps, e.g. for detection of duplicated columns. The rather simplified implementations of column function-, wide data-, and narrow data detection could, in future work, be extended. Also the row function method could be extended to create more diverse hypotheses, especially for header rows, where it has been demonstrated that not always at least one correct hypothesis is created. Since memory consumption was initially suspected as main bottleneck of the process, we did not especially focus on optimizing the runtime of the parsing process. Parsing of files from the *data.gov.uk* corpus took in average 30s, 45s, and 1m per file with the minimal, advanced and full versions, respectively (on a single core machine). Because we consider computational time to be cheaper than human time, this can still be considered as an acceptable processing time. However, in future work, the runtime performance should be further evaluated, especially on larger files.

8.2 How reliable can the best parsing hypothesis be identified?

In Section 6.3.2 and Section 7.1 we have shown that the ranking based on the selected quality criteria generally leads to reasonable ranking decisions, even when the criteria are equally weighted. In 73%-83% of all cases, the best solutions were correctly ranked highest. Among the three highest ranked results, in 86-97% of all cases, the best parsing hypothesis could be found. On the manually cleaned sample, the performance could even be further improved by using a linear Ranking SVM model. But we did not use the resulting model to avoid a bias by the training data. When considering only parsing trees which also contain at least one correct solution, then in 92% (35/38) of all cases this correct solution was also ranked highest, even when using equally weighted features. Another trend which was observed is that the bigger the set of possible hypotheses gets, the worse the ranking quality gets. Therefore the full version performs on the whole *data.gov.uk* corpus worse than the advanced version.

In order to improve the ranking, either a larger training set has to be established or the ranking parameter have to be manually tuned. We have also shown that one table can have different, almost equivalent representations and that it might depend on specific user requirements which variant should be preferred. To account for this, the user should have the possibility to configure the ranking criteria manually. If e.g. for a specific kind of data set a high consistency of data values is expected or desired, the user could define a higher weight on the data type consistency. If on the other hand no data should be disregarded or moved, the edit and move penalty could be manually increased, leading to a set of tables which are less modified but potentially also less consistent. In any case the user receives a parsing report alongside the resulting table, stating which parsing decisions were made. This parsing report could, in principle, be exported to *CSV on the Web* metadata[1] (see Section 5.3), allowing the user to provide the created metadata together with the untouched original file to other data consumer. If the amount of considered files is small, then the parsing configuration could in principle be explicitly changed and fed back into the parsing process. Alternatively, also a set of highest ranked solution could be presented to user, letting him/her choose the preferred table. In that way, the user input could also used to train the internal Ranking SVM and improve future rankings according to the users' choice.

However on a large set of files, manual validation of results is not feasible. Especially, when none of the hypotheses leads to correct results, it was shown that unfavorable tables can be ranked highest, which compromises the overall data quality. In order to

[1] `https://www.w3.org/TR/tabular-metadata/#dialect-descriptions`

automatically detect and filter out unfavourable tables, an absolute measure of table quality should, in future work, be established.

8.3 What effect do the table normalization steps have on the table quality?

The table normalization steps can be split up into basic header/non-header detection (minimal version), multiple header row detection, metadata and column function detection (advanced version), and table orientation and narrow/wide data detection (full version). Considering the results of the minimal version in comparison to those of the advanced and full version, it is apparent that the detection of row and column functions are essential and improve the quality of the resulting table. The additional table orientation and narrow/wide data detection steps increase the quality of the parsing results in specific cases significantly. On the other hand those steps add a high variety to the set of possible outcomes, which complicates the ranking and leads overall to worse parsing results.

As shown in Section 7.1, in some cases the table normalization capabilities are not as good as the ones of compared systems (*DeExcelerator*). Due to the modular architecture of the Multi-Hypothesis parser, current parsing steps can be easily replaced, removed or new steps can be added. Apart from improving the heuristics of current parsing steps, we propose the following directions for future work: As already pointed out in Section 6.2.5 the *conditional random field*-based approaches of Adelfio et al. [37] should be considered as a replacement for the current row function detection. As emphasized by Hurst [28], a deeper understanding of the table content is required to solve certain ambiguities. Especially in order to distinguish variable names from values and improve the column function and wide/narrow data detection, the use of background knowledge should be considered. If a corpus contains very specific concepts, which can not be found in general knowledge bases, it could also be considered to mine background knowledge in form of header and value co-occurrences from all tables in a respective corpus. A different, background knowledge independent approach was proposed by Seth et al. [34], although this would be limited to tables in which the column and row headers form a unique access path to value cells.

8.4 Does multi-hypothesis parsing lead to better results than linear parsing?

The systems we compared to did not have the entirely same scope as the Multi-Hypothesis parser, but by combining *messytables* and *DeExcelerator* we created a linear parsing process whose capabilities are largely in line with those of the Multi-Hypothesis parser. In a one-to-one comparison to this system, the Multi-Hypothesis parsing leads to better parsing results. Out of 64 messy tables, up to 38 could be parsed, cleaned and normalized in the same ways as we did manually. The linear solution only achieved 29 correct tables. We did not only show an improvement of parsing results on the small sample, but also on the larger sample of approx. 15,000 files from *data.gov.uk*. Manual analysis of erroneous parsing results revealed that simple improvements (fixing of the date and numeric detection and detection of duplicated columns) could even further improve the quality of the parsing results. A question which arises is to which extent a linear parser would be capable of parsing the tables correctly if all the mentioned issues of the linear parsers would be fixed. In Section 6.3.2 we simulated the behaviour of linear parsers by following the path of highest confidence in the parsing tree (naïve) of the Multi-Hypothesis parser. The naïve selection of results lead to choices which were only slightly better than random choices and far off from the choices based on results quality. However, the results have to be treated with caution, because we did not focus on creating highly reliable confidence values, but the results at least show that simple heuristics can not provide sufficiently accurate confidence values to meet the right parsing decisions. That two linear parsing parsing solutions, which focused on parsing of messy tables and normalization of table structures, did not achieve better results than the simple implementation of the Multi-Hypothesis parser, suggests that it is very challenging to meet correct parsing decisions a-priori.

This leads to a second important aspect of the Multi-Hypothesis parsing approach, which is the ease of implementation. The framework uncouples different steps of the parsing process, which means that the parsing steps do not have to take other eventualities into account. They can simply build on a certain assumption about the shape of the input and process accordingly. The row function step can, e.g., always build on the assumption that the input table is horizontally oriented. If this is not the case, the parsing in this branch of the tree will eventually fail or lead to bad results, but usually one other branch will contain an actual horizontally oriented table, and there the detection will succeed and lead to results, which will ultimately rule out the other bad results. This is fundamentally different from linear parsing approaches, where all steps necessarily have to succeed or have to take previous or future steps into account. This makes it possible to "plug-in" approaches from related work, which maybe only solve a certain sub-problem, built on certain assumptions about the input table, or are generally not very reliable. Our proposed implementation of the Multi-Hypothesis parser was not yet able to solve all possible issues correctly. In future work, the currently implemented detection heuristics can be further improved or replaced by other methods, as proposed in the previous section.

8.5 Future Work

For future work we propose to first exploit the most obvious opportunities for improvements. This implies fixing the mentioned issues for date and numeric detection, and improving the row/column function and narrow/wide data detection modules by creating a higher variety of hypotheses and making more thorough plausibility checks. Because an estimated faction of 2% of all CSV files on data.gov.uk contains multiple tables, the multi-table-support should be evaluated and included in the regular parsing process. Far-reaching directions for improvements of the table normalization steps were discussed 8.3.

The higher the variety of hypotheses gets, the more reliably the result quality has to be assessed. Since data type consistency is an important quality factor, we propose a better support of different data types, such as date-times, geo-coordinates, and ZIP codes etc., as a simple way to improve the ranking accuracy. Furthermore the measure of consistency could be improved by calculating the numeric consistency on numeric, date and time columns. In order to assess the consistency of text, different text features, such as the text length could be taken into account. Also the consistency of textual values could be assessed by the amount of times certain values co-occur in different tables in a given corpus. Corpus knowledge could also be used to assess the correctness of headers by evaluating how often certain values occur under a specific header. The different quality features are currently weighted equally, but as proposed in Section 8.2, different approaches could be used to improve the weighting.

In some cases the input files could not be properly parsed, because the pruning removed all dialect hypotheses. In future work we want to prevent such cases by disabling pruning for dialect hypotheses or by creating more realistic confidence values for dialect hypotheses. In other cases, CSVLint reported a "check options" issue which indicates an unsuccessful dialect detection by the Multi-Hypothesis parser. Those cases have to be manually reviewed to assess if and why the dialect has not been properly detected. CSV files which are actually syntactically broken, are currently handled by the readr library, which provisionally fixes those issues without taking consistency of values or other criteria into account. In future work, the record alignment approach of Chu et al. [41] could be utilized to improve the automatic fixing procedure of syntactically broken CSV files. In the test set of very diverse files from data.gov.uk serious syntactical issues which would require a re-alignment of values were seldomly observed.

So far, the performance of the Multi-Hypothesis parser was only evaluated on the *data.gov.uk* corpus, which is diverse, but does certainly not contain all possible variations of CSV data. In future work the proposed solution should be evaluated on other data sets. The correct functionality of each parsing step could furthermore be ensured by testing corner cases with unit tests.

Ultimately a good way to evaluate a solution is to let people actually use it. Therefore we have published the created solution as freely available package for the statistical programming environment R [50]. The source code is available online[2].

[2]https://github.com/tdoehmen/hypoparsr

List of Figures

2.1 Data Frame . . . 9
2.2 Gathered Wide Data . . . 10
2.3 Spread Narrow Data . . . 11
2.4 Gathered and Spread Mixed Data . . . 11
2.5 Tidy Table . . . 12

3.1 *data.ogv.uk* Example 1 . . . 16
3.2 *data.ogv.uk* Example 2 . . . 17
3.3 *data.ogv.uk* Example 3 . . . 18
3.4 *data.ogv.uk* Example 4 . . . 19
3.5 *data.ogv.uk* Example 5 . . . 19

4.1 Size Distribution *data.gov.uk* . . . 23
4.2 Ambiguous CSV file . . . 24
4.3 Proposed Parsing Process Extension . . . 25
4.4 Multi-Hypothesis Tree . . . 26

5.1 Table Model Hurst . . . 36

6.1 Architecture Small . . . 42
6.2 Parsing Steps . . . 43
6.3 Table Area Detection Schema . . . 47
6.4 Table Orientation Detection Schema . . . 48
6.5 Row Function Detection Schema . . . 50
6.6 Column Function Example 1 . . . 51
6.7 Column Function Example 2 . . . 52
6.8 Column Function Example 3 . . . 52
6.9 Column Function Detection Schema . . . 53
6.10 Wide Data Detection Schema . . . 55
6.11 Narrow Data Detection Schema . . . 56
6.12 Quality Assessment Training Schema . . . 62
6.13 Ranking Evaluation . . . 65

7.1 Multi-Hypothesis Parser Variants . . . 66
7.2 Evaluation Example 1 . . . 68
7.3 Evaluation Example 2 . . . 69
7.4 Evaluation Example 3 . . . 69
7.5 Evaluation Example 4 . . . 70
7.6 Evaluation Example 5 . . . 70

7.7 Evaluation Example 6 71
7.8 Evaluation Example 7 72
7.9 Evaluation Example 8 72
7.10 Evaluation Example 9 73
7.11 Evaluation Example 10 73
7.12 Evaluation Example 11 73
7.13 CSVLint issues in the *data.gov.uk* corpus. 74
7.14 CSVLint Issues Separate Evaluation 76
7.15 Amount of Recovered Data 78
7.16 Number of Hypotheses per Step 80

B.1 Encoding Class Diagram 93
B.2 Dialect Class Diagram 93
B.3 Table Area Class Diagram 94
B.4 Table Orientation Class Diagram 94
B.5 Row Functions Class Diagram 94
B.6 Column Functions Class Diagram 94
B.7 Wide Data Class Diagram 94
B.8 Narrow Data Class Diagram 95
B.9 Data Types Class Diagram 95
B.10 CSV Parser Class Diagram 96

List of Tables

2.1 CSV Parser Capabilities 7
2.2 CSV Validation Tool Capabilities 9
2.3 5-Star Open Data 13

3.1 CSV Issues Random Sample 20
3.2 CSV Issue Categorization 21

4.1 Parsing Success Evaluation CSVLint 24

5.1 Format Features Related Work 32
5.2 Data Formats Related Work 33
5.3 Input Features Related Work 34
5.4 Table Elements Related Work 37

6.1 Data Quality Characteristics 61
6.2 Quality Metric Weights 63

7.1 Evaluation Example 10 Metrics 72
7.2 Evaluation Example 11 Metrics 73
7.3 Successfully Parsed CSV 77
7.4 Number of Hypotheses 79

A.1 Comparison of different CSV Parsers 92

List of Algorithms

1 Hypothesis Tree Traversal 42
2 Dialect Detection 46
3 Table Area Detection - GET_DENSE_AREAS 46
4 Table Orientation - Single Reference Row/Col (Pivk et al. [33]) 48
5 Table Orientation - Multiple Reference Rows/Cols 49
6 Row Functions 50
7 Column Functions 54
8 Wide Data - Detection 55
9 Narrow Data - Detection 55
10 Data Types - Detect 57
11 Data Types - Parser 58
12 Parse File 97
13 Parse File - GET_PARSING_RESULTS 98
14 Table Area Detection 98
15 Table Area Detection - GET_DATA_DENSITY 98
16 Table Orientation - EXPAND_AREAS_TOP_LEFT 99
17 Row Functions - COUNT_FUNCTION_VOTES 99
18 Row Functions - NORMALIZE_FUNCTION_VOTES 99
19 Wide Data - Parsing 100
20 Narrow Data - Parsing 100

Appendix A

CSV Parser Comparison

Table A.1 shows the configuration parameters of different CSV parsers for the statistical programming environment R and Python in comparison.

Table A.1: Comparison of different CSV Parsers

		R			**Python**	
	native	**fread**	**readr**	**native**	**pandas**	**messytables**
Encoding	fileEncoding, encoding	encoding	locale, guess_encoding	-	encoding	encoding
Line Terminator	-	auto	-	lineterminator	lineterminator	lineterminator
Delimiter	sep	sep	delim	delimiter	sep, delimiter, delim_whitespace	delimiter
Quotechar	quote	-	quote	quotechar	quotechar	quotechar
Escaping	allowEscapes	-	escape_double, escape_backslash	doublequote, escapechar	escapechar	doublequote
Comments	comment.char	-	comment	-	comment	-
Header	header	header	col_names	has_header	header	headers_guess
Index Col	row.names	-	-	-	index_col	-
Empty Row/-Col	blank.lines.skip	-	-	-	skip_blank_lines	-
Field Whitespace	strip.white	strip.white	trim_ws	skipinitialspace	skipinitialspace	skipinitialspace
NA Values	na.strings	na.strings	na	-	na_values	auto
Decimal Points	dec	dec	locale	-	decimal, thousands	auto
Dates	-	-	date_names, date_format, tz	-	parse_dates auto, dayfirst, infer datetime_format	format, type_guess
Boolean	-	-	-	-	true_values, false_values	true_values, false_values, type_guess

Appendix B

Multi-Hypothesis Parser API

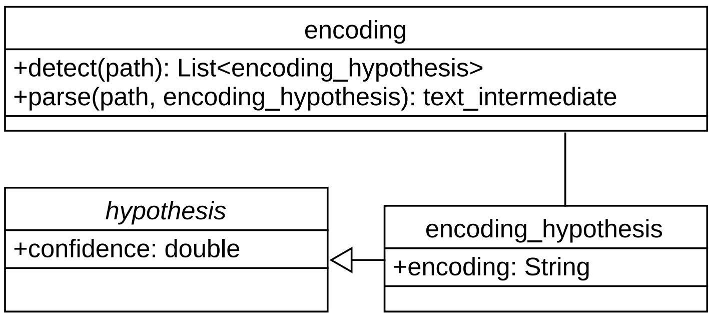

encoding
+detect(path): List<encoding_hypothesis>
+parse(path, encoding_hypothesis): text_intermediate
hypothesis
+confidence: double
encoding_hypothesis
+encoding: String

Figure B.1: Encoding Class Diagram

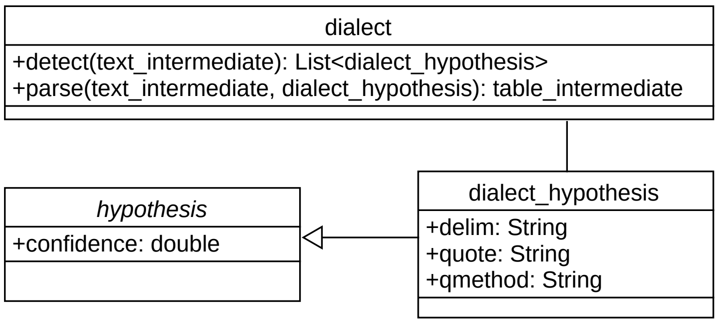

dialect
+detect(text_intermediate): List<dialect_hypothesis>
+parse(text_intermediate, dialect_hypothesis): table_intermediate
hypothesis
+confidence: double
dialect_hypothesis
+delim: String
+quote: String
+qmethod: String

Figure B.2: Dialect Class Diagram

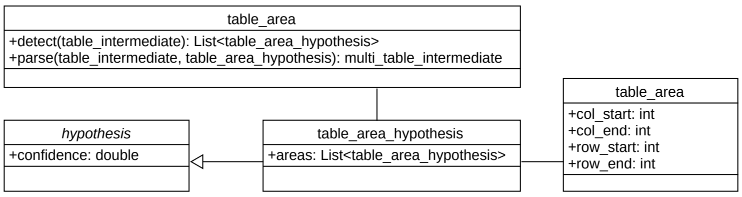

table_area
+detect(table_intermediate): List<table_area_hypothesis>
+parse(table_intermediate, table_area_hypothesis): multi_table_intermediate
hypothesis
+confidence: double
table_area_hypothesis
+areas: List<table_area_hypothesis>
table_area
+col_start: int
+col_end: int
+row_start: int
+row_end: int

FIGURE B.3: Table Area Class Diagram

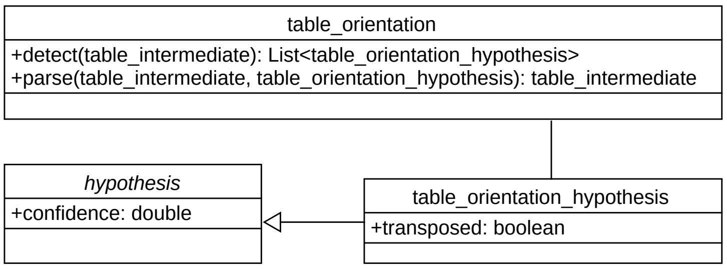

table_orientation
+detect(table_intermediate): List<table_orientation_hypothesis>
+parse(table_intermediate, table_orientation_hypothesis): table_intermediate
hypothesis
+confidence: double
table_orientation_hypothesis
+transposed: boolean

FIGURE B.4: Table Orientation Class Diagram

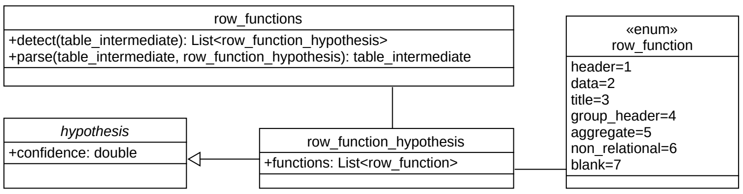

row_functions
+detect(table_intermediate): List<row_function_hypothesis>
+parse(table_intermediate, row_function_hypothesis): table_intermediate
hypothesis
+confidence: double
row_function_hypothesis
+functions: List<row_function>
«enum»
row_function
header=1
data=2
title=3
group_header=4
aggregate=5
non_relational=6
blank=7

FIGURE B.5: Row Functions Class Diagram

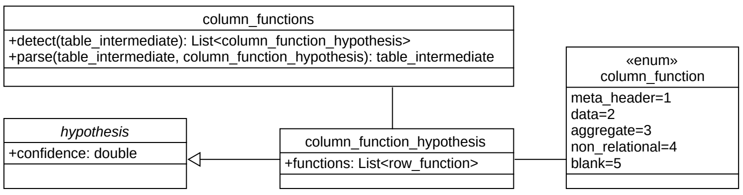

column_functions
+detect(table_intermediate): List<column_function_hypothesis>
+parse(table_intermediate, column_function_hypothesis): table_intermediate
hypothesis
+confidence: double
column_function_hypothesis
+functions: List<row_function>
«enum»
column_function
meta_header=1
data=2
aggregate=3
non_relational=4
blank=5

FIGURE B.6: Column Functions Class Diagram

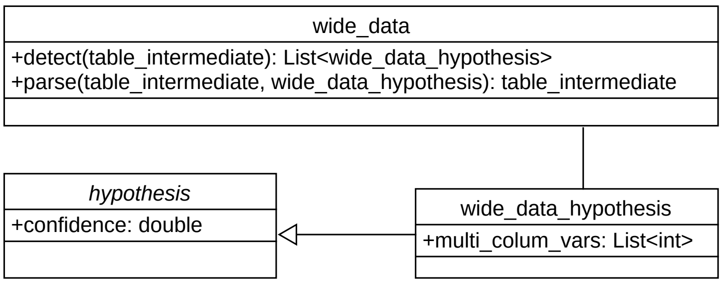

wide_data
+detect(table_intermediate): List<wide_data_hypothesis>
+parse(table_intermediate, wide_data_hypothesis): table_intermediate
hypothesis
+confidence: double
wide_data_hypothesis
+multi_colum_vars: List<int>

FIGURE B.7: Wide Data Class Diagram

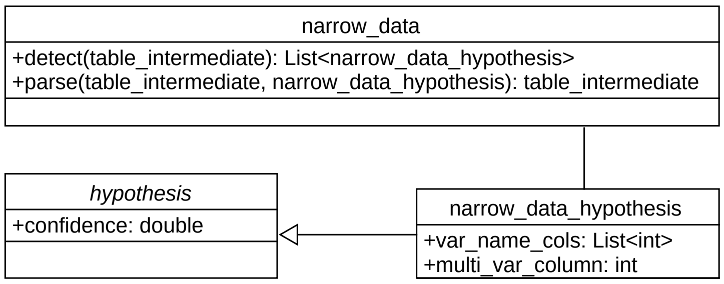

narrow_data
+detect(table_intermediate): List<narrow_data_hypothesis>
+parse(table_intermediate, narrow_data_hypothesis): table_intermediate
hypothesis
+confidence: double
narrow_data_hypothesis
+var_name_cols: List<int>
+multi_var_column: int

Figure B.8: Narrow Data Class Diagram

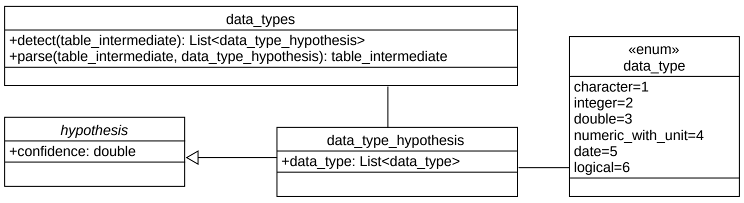

data_types
+detect(table_intermediate): List<data_type_hypothesis>
+parse(table_intermediate, data_type_hypothesis): table_intermediate
hypothesis
+confidence: double
data_type_hypothesis
+data_type: List<data_type>
«enum»
data_type
character=1
integer=2
double=3
numeric_with_unit=4
date=5
logical=6

Figure B.9: Data Types Class Diagram

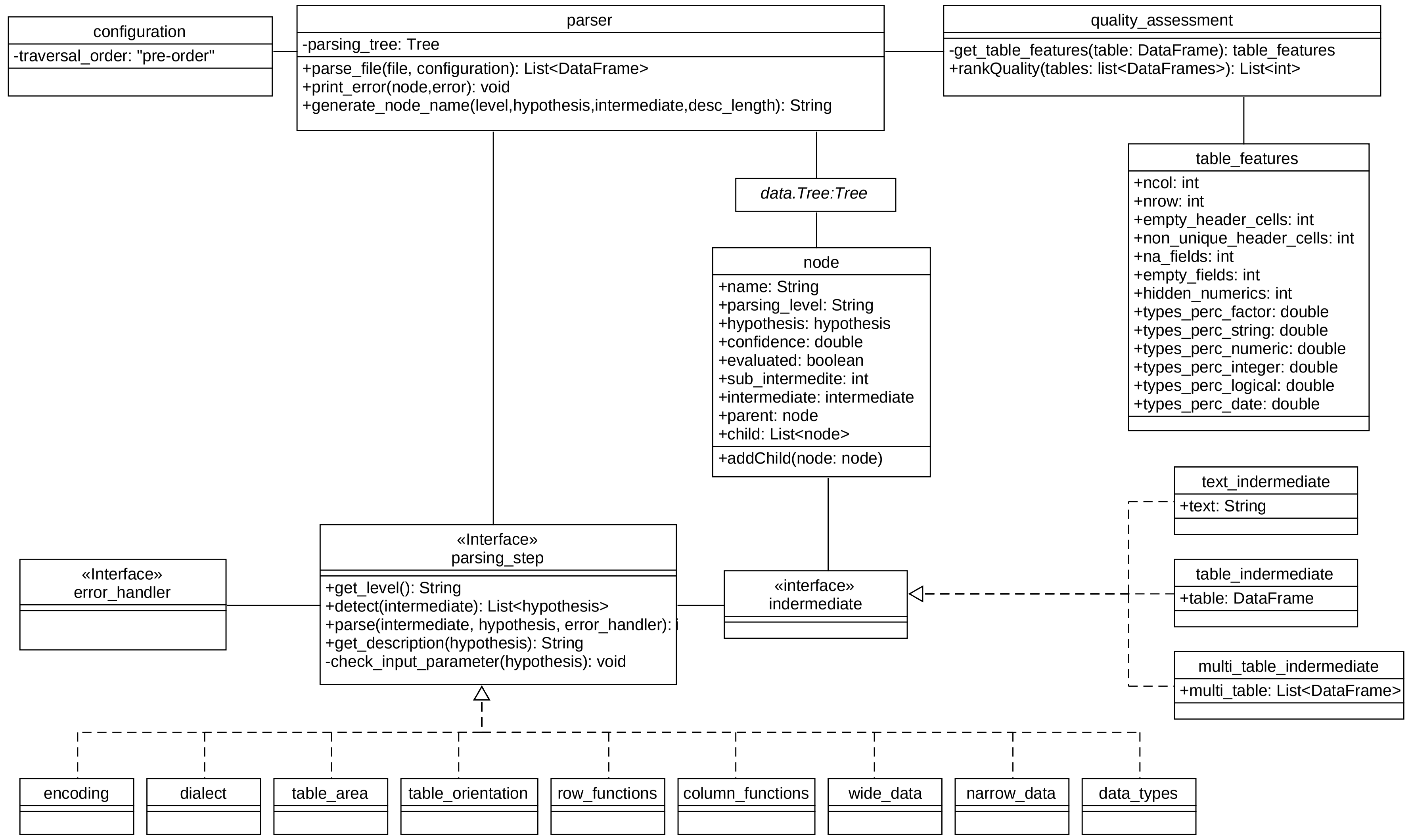

configuration
-traversal_order: "pre-order"
parser
-parsing_tree: Tree
+parse_file(file, configuration): List<DataFrame>
+print_error(node,error): void
+generate_node_name(level,hypothesis,intermediate,desc_length): String
quality_assessment
-get_table_features(table: DataFrame): table_features
+rankQuality(tables: list<DataFrames>): List<int>
table_features
+ncol: int
+nrow: int
+empty_header_cells: int
+non_unique_header_cells: int
+na_fields: int
+empty_fields: int
+hidden_numerics: int
+types_perc_factor: double
+types_perc_string: double
+types_perc_numeric: double
+types_perc_integer: double
+types_perc_logical: double
+types_perc_date: double
data.Tree:Tree
node
+name: String
+parsing_level: String
+hypothesis: hypothesis
+confidence: double
+evaluated: boolean
+sub_intermedite: int
+intermediate: intermediate
+parent: node
+child: List<node>
+addChild(node: node)
text_intermediate
+text: String
table_intermediate
+table: DataFrame
multi_table_intermediate
+multi_table: List<DataFrame>
«interface»
indermediate
«Interface»
parsing_step
+get_level(): String
+detect(intermediate): List<hypothesis>
+parse(intermediate, hypothesis, error_handler): i
+get_description(hypothesis): String
-check_input_parameter(hypothesis): void
«Interface»
error_handler
encoding
dialect
table_area
table_orientation
row_functions
column_functions
wide_data
narrow_data
data_types

FIGURE B.10: CSV Parser Class Diagram

Appendix C

Multi-Hypothesis Parser Implementation Details

```
text: ^.*
total: ^.*(total|Total|TOTAL).*
empty: ^(\s*)(["’]?)(\s*)(NULL|null|Null|NA|na|N/A|n/a|NaN|nan|#N/A|#NA|1.#IND
    |1.#QNAN|-1.#IND|-1.#IND|-NaN|-nan|)?(\s*)(["’]?)(\s*)
punctuation: ^(\s*)(["’]?)(\s*)[[:punct:]]+(\s*)(["’]?)(\s*)
id: ^(\s*)(["’]?)(\s*)(?=.*[[:upper:]].*)(?=.*[[:digit:]])[[:upper:][:digit:][:
    punct:]]+(\s*)(["’]?)(\s*)
numeric: ^(\s*)(["’]?)(\s*)(?=.*[[:digit:]])[[:digit:]\(+-.,\)\p{Sc}\%\s]+\S*(\s
    *)(["’]?)(\s*)
date: ^(\s*)(["’]?)(\s*)([[:digit:]]{2}|[[:digit:]]{4})[.\/-\s](([[:digit
    :]]{2}|[[:digit:]]{4})|[[:alpha:]]{3})[.\/-\s]([[:digit:]]{2}|[[:digit:]]{4})
    (\s*)(["’]?)(\s*)
time: ^(\s*)(["’]?)(\s*)[[:digit:]]{2}(\:)[[:digit:]]{2}((\:)[[:digit:]]{2})?(\s
    *)(["’]?)(\s*)
email: ^(\s*)(["’]?)(\s*)[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+.[a-zA-Z0-9-.]+(\s*)
    (["’]?)(\s*)
url: ^(https?:\/\/)?([\da-z.-]+).([a-z.]{2,6})([\/\w .-]*)*\/?
logical: ^(\s*)(["’]?)(\s*)(True|False|TRUE|FALSE|true|false)(\s*)(["’]?)(\s*)
```

LISTING C.1: "Regular Expressions used in GET_CELL_TYPES utility function"

Algorithm 12 Parse File

```
function $parse\_file(path, config = default)$
    $tree$ ← GENERATE_PARSING_TREE($path, config$)
    $results$ ← GET_PARSING_RESULTS($tree$)
    $ranking$ ← RANK_RESULT_QUALITY($results$)
    return $result$ with highest $ranking$
end function
```

Algorithm 13 Parse File - GET_PARSING_RESULTS

```
1: function $get\_parsing\_results(tree)$
2:     $results \leftarrow$ LIST
3:     for all $node$ in $tree$ where $node.level$ is $count(parsing\_steps)$ do
4:         $confidence \leftarrow$ traverse ancestors of $node$ and collect $confidence$ values
5:         $metadata \leftarrow$ traverse ancestors of $node$ and collect $metadata$ values
6:         $warnings \leftarrow$ traverse ancestors of $node$ and collect $warnings$ values
7:         $edits \leftarrow$ traverse ancestors of $node$ and collect $edits$ values
8:         $results$.ADD$(node.intermediate, confidence, metadata, warnings, edits)$
9:     end for
10:    return $results$
11: end function
```

Algorithm 14 Table Area Detection

```
1: $default \leftarrow$ NEW_AREA$(i_1 = 1, i_n = nrow(matrix), j_1 = 1, j_n = ncol(matrix))$
2: $default \leftarrow$ REMOVE_EMPTY_FRAME$(default)$
3: $hypotheses$.ADD$(default)$
4: if not $configuration.only\_one\_table$ then          ▷ perform real area detection
5:     $types \leftarrow$ GET_CELL_TYPES$(matrix)$
6:     $density \leftarrow$ GET_DATA_DENSITY$(types)$
7:     $dense\_areas \leftarrow$ GET_DENSE_AREAS$(types, density)$
8:     $expanded\_areas \leftarrow$ EXPAND_AREAS_TOP_LEFT$(density, dense\_areas)$
9:     $expanded\_areas \leftarrow$ REMOVE_EMPTY_FRAME$(expanded\_areas)$
10:    $hypotheses$.ADD$(expanded\_areas)$
11: end if
12: NORMALIZE_CONFIDENCE$(hypotheses)$
```

Algorithm 15 Table Area Detection - GET_DATA_DENSITY

```
1: function $get\_data\_density(types)$
2:     $density \leftarrow$ MATRIX$(size(types))$
3:     for all $i, j$ in $indices(types)$ do
4:         if $types[i, j]$ is not "$empty$" then
5:             $sub\_matrix \leftarrow [types[i, j],$ NEIGHBOUR_CELLS$(types[i, j])]$
6:             $density[i, j] \leftarrow$ count of cells in $sub\_matrix \neq$ "$empty$"
7:         else
8:             $density[i, j] \leftarrow 0$
9:         end if
10:    end for
11:    return $density$
12: end function
```

Algorithm 16 Table Orientation - EXPAND_AREAS_TOP_LEFT

```
function $expand\_areas\_top\_left(density, dense\_areas)$
    $expanded\_areas \leftarrow$ LIST
    for all $n$ in $1 : count(dense\_areas)$ do
        $i_1, i_n, j_1, j_n \leftarrow dense\_areas[n]$
        $upper\_neighbour \leftarrow$ area in $dense\_areas$ above $dense\_areas[n]$
        $left\_neighbour \leftarrow$ area in $dense\_areas$ left from $dense\_areas[n]$
        if $any(upper\_neighbour)$ then $i_1 = upper\_neighbour.i_n + 1$
        end if
        if $any(left\_neighbour)$ then $j_1 = left\_neighbour.j_n + 1$
        end if
        if $n == count(dense\_areas)$ then $i_n, j_n = \leftarrow dim(density)$ end if
        $expanded\_areas.$ADD$(i_1, i_n, j_1, j_n)$
    end for
    return $expanded\_areas$
end function
```

Algorithm 17 Row Functions - COUNT_FUNCTION_VOTES

```
function $count\_function\_votes(types)$
    $votes \leftarrow$ LIST
    for all $i$ in $1 : nrow(types)$ do
        $vote_{empty}, vote_{data}, vote_{metadata}, vote_{header}, vote_{aggregate} \leftarrow 0$
        if $all(types[i,] ==$ "empty"|"punctuation"$)$ then $vote_{empty} \leftarrow 1$ end if
        if $all(types[i,] ==$ "empty"$)$ then $vote_{empty} \leftarrow 2$ end if
        if $any(types[i,] ==$ "total"$)$ then $vote_{total} \leftarrow 1$ end if
        if $all(types[i,] ==$ "total"|"numeric"|"empty"$)$ then $vote_{total} \leftarrow 2$ end if
        if  not $all(types[i,] ==$ "empty"$)$ then
            $vote_{metadata} \leftarrow count(types[i,] ==$ "empty"$)/ncol(types)$
        end if
        $max\_check\_for\_header\_rows \leftarrow min(30, nrow(types))$
        $equal\_cells \leftarrow []$
        for all $c$ between $i + 1$ and $max\_check\_for\_header\_rows$ do
            $equal\_cells[c] \leftarrow$percentage of non-empty cells in $types[i,] \neq types[c,]$
        end for
        $vote_{header} \leftarrow mean(equal\_cells)$
        $vote_{data} \leftarrow count(types[i,] \neq$ "empty"|"punctuation"|"text"$)/ncol(types)$
        $votes[i] \leftarrow$ LIST$(vote_{empty}, vote_{data}, vote_{metadata}, vote_{header}, vote_{aggregate})$
    end for
    return $votes$
end function
```

Algorithm 18 Row Functions - NORMALIZE_FUNCTION_VOTES

```
function $normalize\_function\_votes(votes)$
    $normalized\_votes \leftarrow$ MATRIX$(dim(votes))$
    for all $i, j$ in $indeces(votes)$ do
        $normalized\_votes[i, j] \leftarrow vote[i, j] - 2/3 * max(votes[, j])$
        $normalized\_votes[i, j] \leftarrow normalized\_votes[i, j] - sum(votes[i, ])$
    end for
    return $normalized\_votes$
end function
```

Algorithm 19 Wide Data - Parsing

```
1: if $count(hypothesis.multi\_col\_vars) > 1$ then
2:     $result.intermediate \leftarrow melt(table, hypothesis.multi\_col\_vars)$
3:     $result.edits \leftarrow ncol(table) * nrow(table)$
4: else
5:     $result.intermediate = table$
6: end if
7: return $result$
```

Algorithm 20 Narrow Data - Parsing

```
1: $var\_name\_cols \leftarrow hypothesis.var\_name\_cols$
2: $multi\_var\_col \leftarrow hypothesis.multi\_var\_col$
3: if $count(var\_name\_cols) > 0$ then
4:     $result.intermediate \leftarrow spread(table, hypothesis.key\_col, hypothesis.value\_col)$
5:     $result.edits \leftarrow ncol(table) * nrow(table)$
6:     if $result.intermediate$ is $null$ then
7:         $result.error \leftarrow$"Spreading of table failed"
8:         $result.edits \leftarrow 0$
9:     end if
10: else
11:     $result.intermediate \leftarrow table$
12: end if
```

Bibliography

[1] Tamraparni Dasu and Theodore Johnson. *Exploratory data mining and data cleaning*, volume 479. John Wiley & Sons, 2003.

[2] Marijn Janssen, Yannis Charalabidis, and Anneke Zuiderwijk. Benefits, adoption barriers and myths of open data and open government. *Information Systems Management*, 29(4):258–268, 2012.

[3] IBM. IBM FORTRAN Program Products for OS and the CMS Component of VM/370 General Information. *IBM*, first ed.:p. 17, 1972.

[4] Y. Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180 (Informational), October 2005. URL `http://www.ietf.org/rfc/rfc4180.txt`. Updated by RFC 7111.

[5] Y. Shafranovich. IESG CSV MIME Type, 2014. URL `https://www.iana.org/assignments/media-types/text/csv`.

[6] Y. Shafranovich. Important note on default charset in "text/csv" (RFC 4180), 2014. URL `https://lists.w3.org/Archives/Public/public-csv-wg/2014Oct/0114.html`.

[7] Hadley Wickham. Tidy data. *Under review*, 2014.

[8] Edgar F Codd. Further normalization of the data base relational model. *Data base systems*, pages 33–64, 1972.

[9] Ed Summers. The 5 stars of open linked data, 2010. URL `http://inkdroid.org/2010/06/04/the-5-stars-of-open-linked-data/`.

[10] James G. Kim and Michael Hausenblas. 5-star open data, 2015. URL `http://5stardata.info/en/`.

[11] Open Date Institute. Report on Departmental Open Data Commitments and adherence to Public Data Principles for the period between July and September 2012. (September):1–9, 2012.

[12] ISO/IEC 8859-1:1998. Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1. Standard, International Organization for Standardization, Geneva, CH, April 1998.

[13] ISO/IEC 8859-16:2001. Information technology – 8-bit single-byte coded graphic character sets – Part 16: Latin alphabet No. 10. Standard, International Organization for Standardization, Geneva, CH, July 2001.

[14] Ivan Ermilov, Sören Auer, and Claus Stadler. User-driven semantic mapping of tabular data. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 105–112. ACM, 2013.

[15] Ranjit Singh, Kawaljeet Singh, et al. A descriptive classification of causes of data quality problems in data warehousing. *International Journal of Computer Science Issues*, 7(3):41–50, 2010.

[16] Jeffrey Heer, Joseph M Hellerstein, and Sean Kandel. Predictive interaction for data transformation. In *7th Biennial Conference on Innovative Data System Research, CIDR*, volume 15.

[17] Hongbo Du and Laurent Wery. Micro: A normalization tool for relational database designers. *Journal of Network and Computer applications*, 22(4):215–232, 1999.

[18] Ali Yazici and Ziya Karakaya. Jmathnorm: A database normalization tool using mathematica. In *Computational Science–ICCS 2007*, pages 186–193. Springer, 2007.

[19] Thorsten Papenbrock, Jens Ehrlich, Jannik Marten, Tommy Neubert, Jan-Peer Rudolph, Martin Schönberg, Jakob Zwiener, and Felix Naumann. Functional dependency discovery: An experimental evaluation of seven algorithms. *Proceedings of the VLDB Endowment*, 8(10), 2015.

[20] Jácome Cunha, João Saraiva, and Joost Visser. From spreadsheets to relational databases and back. In *Proceedings of the 2009 ACM SIGPLAN workshop on Partial evaluation and program manipulation*, pages 179–188. ACM, 2009.

[21] Dan Brickley and Ramanathan V Guha. Resource description framework (rdf) schema specification 1.0: W3c candidate recommendation 27 march 2000. 2000.

[22] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[23] Varish Mulwad, Tim Finin, and Anupam Joshi. Automatically generating government linked data from tables. 2011.

[24] Kumar Sharma, Ujjal Marjit, and Utpal Biswas. Automatically converting tabular data to rdf: An ontological approach.

[25] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347, 2010.

[26] Albert Meroño-Peñuela, Ashkan Ashkpour, Laurens Rietveld, Rinke Hoekstra, and Stefan Schlobach. Linked humanities data: The next frontier? In *A Case-study in Historical Census Data. Proceedings of the 2nd International Workshop on Linked Science*, volume 951, page 2012. Citeseer, 2012.

[27] Jenni Tennison. Csv on the web: A primer, w3c wg note. Working group note, W3C, 2016. http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/.

[28] Matthew Francis Hurst. The interpretation of tables in texts. 2000.

[29] Xinxin Wang and Derick Wood. *Tabular abstraction, editing, and formatting.* Citeseer, 1996.

[30] Daniel Lopresti and George Nagy. Automated table processing: An (opinionated) survey. In *Proceedings of the Third IAPR Workshop on Graphics Recognition*, pages 109–134, 1999.

[31] Ana Costa e Silva, Alípio M Jorge, and Luís Torgo. Design of an end-to-end method to extract information from tables. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):144–171, 2006.

[32] David W Embley, Matthew Hurst, Daniel Lopresti, and George Nagy. Table-processing paradigms: a research survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):66–86, 2006.

[33] Aleksander Pivk, Philipp Cimiano, York Sure, Matjaz Gams, Vladislav Rajkovič, and Rudi Studer. Transforming arbitrary tables into logical form with tartar. *Data & Knowledge Engineering*, 60(3):567–595, 2007.

[34] Sharad Seth and George Nagy. Segmenting tables via indexing of value cells by table headers. In *2013 12th International Conference on Document Analysis and Recognition*, pages 887–891. IEEE, 2013.

[35] Zhe Chen, Michael Cafarella, Jun Chen, Daniel Prevo, and Junfeng Zhuang. Senbazuru: a prototype spreadsheet database management system. *Proceedings of the VLDB Endowment*, 6(12):1202–1205, 2013.

[36] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[37] Marco D Adelfio and Hanan Samet. Schema extraction for tabular data on the web. *Proceedings of the VLDB Endowment*, 6(6):421–432, 2013.

[38] David Pinto, Andrew McCallum, Xing Wei, and W Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242. ACM, 2003.

[39] Julian Eberius, Christoper Werner, Maik Thiele, Katrin Braunschweig, Lars Dannecker, and Wolfgang Lehner. Deexcelerator: A framework for extracting relational data from partially structured documents. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 2477–2480. ACM, 2013.

[40] Hazem Elmeleegy, Jayant Madhavan, and Alon Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):1078–1089, 2009.

[41] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. Tegra: Table extraction by global record alignment. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1713–1728. ACM, 2015.

[42] George Nagy, David W Embley, Mukkai Krishnamoorthy, and Sharad Seth. Clustering header categories extracted from web tables. In *IS&T/SPIE Electronic Imaging*, pages 94020M–94020M. International Society for Optics and Photonics, 2015.

[43] Matthew Hurst. Towards a theory of tables. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):123–131, 2006.

[44] Eli Cortez, Daniel Oliveira, Altigran S da Silva, Edleno S de Moura, and Alberto HF Laender. Joint unsupervised structure discovery and information extraction. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 541–552. ACM, 2011.

[45] George Nagy, Sachin Seth, Dongpu Jin, David W Embley, Spencer Machado, and Mohan Krishnamoorthy. Data extraction from web tables: The devil is in the details. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 242–246. IEEE, 2011.

[46] David W Embley, Mukkai Krishnamoorthy, George Nagy, and Sharad Seth. Factoring web tables. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 253–263. Springer, 2011.

[47] Jianying Hu, Ramanujan S Kashi, Daniel Lopresti, and Gordon T Wilfong. Evaluating the performance of table processing algorithms. *International Journal on Document Analysis and Recognition*, 4(3):140–153, 2002.

[48] Yalin Wang, Ihsin T Phillips, and Robert M Haralick. Table structure understanding and its performance evaluation. *Pattern Recognition*, 37(7):1479–1497, 2004.

[49] Zhe Chen and Michael Cafarella. Automatic web spreadsheet data extraction. In *Proceedings of the 3rd International Workshop on Semantic Search over the Web*, page 1. ACM, 2013.

[50] R Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2016. URL `https://www.R-project.org/`.

[51] Hadley Wickham. *rvest: Easily Harvest (Scrape) Web Pages*, 2016. URL `https://CRAN.R-project.org/package=rvest`. R package version 0.3.2.

[52] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289, 2001.

[53] Hadley Wickham. Reshaping data with the reshape package. *Journal of Statistical Software*, 21(12):1–20, 2007. URL `http://www.jstatsoft.org/v21/i12/`.

[54] ISO/IEC 25012:2008. Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Data quality model. Standard, International Organization for Standardization, Geneva, CH, March 2008.

[55] Irfan Rafique, Philip Lew, Maissom Qanber Abbasi, and Zhang Li. Information quality evaluation framework: Extending iso 25012 data quality model. *World Academy of Science, Engineering and Technology*, 65:523–528, 2012.

[56] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR)*, 41(3):16, 2009.

[57] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2002.

[58] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. In *Soviet physics doklady*, volume 10, page 707, 1966.

[59] Andreas Borg and Murat Sariyar. *RecordLinkage: Record Linkage in R*, 2016. URL `https://CRAN.R-project.org/package=RecordLinkage`. R package version 0.4-9.

[60] Ethem Alpaydin. Introduction to machine learning (adaptive computation and machine learning). 2004.

[61] Qing-Song Xu and Yi-Zeng Liang. Monte carlo cross validation. *Chemometrics and Intelligent Laboratory Systems*, 56(1):1–11, 2001.

Declaration of Authorship

I, Till Roman Döhmen, declare that this thesis titled, 'Multi-Hypothesis Parsing of Tabular Data in Comma-Separated Values (CSV) Files' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________