Multi-Hypothesis CSV Parsing

Till Döhmen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
tilldoehmen@gmail.com

Hannes Mühleisen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
hannes@cwi.nl

Peter Boncz
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
boncz@cwi.nl

ABSTRACT

Comma Separated Value (CSV) files are commonly used to represent data. CSV is a very simple format, yet we show that it gives rise to a surprisingly large amount of ambiguities in its parsing and interpretation. We summarize the state-of-the-art in CSV parsers, which typically make a linear series of parsing and interpretation decisions, such that any wrong decision at an earlier stage can negatively affect all downstream decisions. Since computation time is much less scarce than human time, we propose to turn CSV parsing into a ranking problem. Our quality-oriented *multi-hypothesis* CSV parsing approach generates several concurrent hypotheses about dialect, table structure, etc. and ranks these hypotheses based on quality features of the resulting table. This approach makes it possible to create an advanced CSV parser that makes many different decisions, yet keeps the overall parser code a simple plug-in infrastructure. The complex interactions between these decisions are taken care of by searching the hypothesis space rather than by having to program these many interactions in code. We show that our approach leads to better parsing results than the state of the art and facilitates the parsing of large corpora of heterogeneous CSV files.

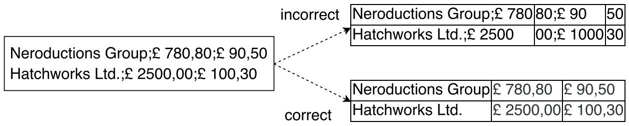
CCS CONCEPTS

• **Information systems → Inconsistent data**;

ACM Reference format:
Till Döhmen, Hannes Mühleisen, and Peter Boncz. 2017. Multi-Hypothesis CSV Parsing. In *Proceedings of SSDBM '17, Chicago, IL, USA, June 27-29, 2017,* 12 pages.
https://doi.org/http://dx.doi.org/10.1145/3085504.3085520

1 INTRODUCTION

Data scientists typically lose much time in importing and cleaning data, and large data repositories such as open government collections with tens of thousands of datasets remain under-exploited due to the high human cost of discovering, accessing and cleaning this data. CSV is the most commonly used data format in such repositories. The lack of explicit information on the CSV dialect, the table structure, and data types makes proper parsing tedious and error-prone.



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

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

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

Figure 1: Ambiguous CSV file which is at risk to be parsed incorrectly, because the number of commas and the number of semi-colons per row are the same.

Tools currently popular among data scientists, such as R and Python offer robust CSV parsing libraries, which try to address parsing of messy CSV files with a number of practical heuristics. These libraries makes a linear sequence of parsing and interpretation decisions, such that any wrong decision at an earlier stage (e.g. determining the separator character) will negatively affect all downstream decisions. Interlinking different parsing steps (backtracking on prior decisions) is not done, because if all parsing decisions affect each other, the parsing code becomes very complex (code size would need to grow quadratically in the amount of decisions or even worse).

Since CPU-cycles are currently plentiful but human time is not, this research pursues an approach where CSV parsing becomes an computerized search problem. Our quality-oriented CSV parsing approach generates several concurrent hypotheses about dialect, table structure, etc. and in the end ranks these hypotheses based on quality features of the resulting table, such that the top-1 would be the automatic parsing result, or a top-K of parsed tables could be presented to the user. A high absolute score from the quality function can also be used to automatically parse large amounts of files. Only ambigous cases would be presented to a user. This can strongly reduce human data interpretation effort.

This very practical problem touches on various areas of related work. In the extended version of this paper [6], we survey the state-of-the art on this topic, which covers areas such as computer-assisted data-cleaning (*data-wrangling*), table-interpretation (e.g. on the web), automatic list extraction and even automated semantic (web) enrichment; covered more briefly in the related work Section 5.

Outline. In Section 2 we explaine CSV parsing problem by example, and introduce our multi-hypothesis parsing framework in Section 3. We demonstrate the improved parsing quality of our approach with computed quality metrics on the full *data.gov.uk* dataset collection, as well as on a sample of this collection using human ground truth in Section 4. We summarize related work in Section 5 and describe next steps in Section 6 before concluding in Section 7.

SSDBM '17, June 27-29, 2017, Chicago, IL, USA

ACM ISBN 978-1-4503-5282-6/17/06…$15.00
https://doi.org/http://dx.doi.org/10.1145/3085504.3085520

2 COMMON CSV ISSUES

The UK Open Data portal, *data.gov.uk*, is one of the largest sources for Open Government data on the web. It contains roughly 36,000 data sets on environment, towns & cities, government, society, health, government spending, education, business & economy and transport. The most common data formats are HTML (12,392), geospatial formats (12,167), CSV (6,251), XLS (1,921), PDF (1,266), XML (839) and RDF (233) (Numbers from August 2016). CSV is the predominant tabular data format and expected to stay dominant, as departments and associated organizations are encouraged to publish data with at least *Three Stars* on the Five Star Deployment Scheme introduced by Tim Berners Lee [13], which only allows non-proprietary and easily accessible data formats such as CSV. According to RFC 4180 [20] a CSV file is supposed to contain comma-separated fields, one optional header row and succeeding data rows of the same length which are optionally quoted by double quotes. Fields which contain line breaks, delimiters or quotes have to be escaped by double quotes.

For data analysis and integration purposes it is desired to automatically extract relational tables from CSV files. Ideally, the resulting tables should only contain relational data, should have named and specifically typed columns, and should be "*tidy*" which means that every row contains one observation and every column one specific variable [24]. A random sample of 80 CSV files from the Open Government data hub *data.gov.uk* shows that a considerable amount of files is not in line with these requirements and that current CSV parsers are not sufficiently capable of automatically extracting relational tabular data from these CSV files. We will describe and categorize the most frequently encountered issues in the following:

CSV Syntax Issues. Although RFC 4180 [20] is not an official standard, it can be considered as the main reference for CSV writer and parser implementations. While a later specification of RFC 4180 [21] states that CSV files should be UTF-8 encoded, in practice CSV exists in all character encodings and meta-data on that is not part of the format. The encoding thus has to be inferred from the input file itself, which is subject to uncertainties. Figure 2b shows a table on which the encoding detection failed and the £ sign was falsely interpreted as the Polish letter Ł. 31 of 80 sampled CSVs were not UTF-8 encoded. Encoding issues are particularly critical when they affect delimiters, line endings or quotes. RFC 4180 prescribes that a CSV file should use comma separators, CRLF line endings and double quotes [20]. Other variants of the original dialect are commonly used, however. Those *dialects* use different delimiters, such as semicolons or tabs, different quotes, such as single quotes or other typographic variants, or different line endings. Meta-data on which dialect is used is not part of the CSV format, so this also has to be inferred from the input file, which can lead to ambiguous interpretations. Figure 1 shows a typical example where commas and semi-colons could both be considered as legitimate cell delimiters.

According to RFC 4180, each row is supposed to contain the same amount of fields which is in practice not always the case. The last row of the table in Figure 2c is a "ragged" row because it contains only one field. While strict parsers fail reading such files, robust parsers fill up the remaining space with empty fields. If a cell or a delimiter between two fields is missing, however, this leads to a wrong interpretation of the table.

To assess the syntactic quality of the 80 sampled CSV files we used the strict native CSV parser implementation of the statistical programming environment R [17], the robust CSV parsing package `readr` [25] and the Python library `messytables`[1] which features automatic dialect detection. The strict CSV parser failed in 24/80 cases, while the robust parser `readr` failed only in 1/80 cases. `messytables` also succeeded in the one case in which the CSV was tab-separated and hence achieved a success rate of 100% . We therefore suspect that a considerable amount of files on the *data.gov.uk* repository have syntax issues or use different CSV dialects. State-of-the-art CSV parsers do appear to be capable of dealing with syntax issues and different dialects at least to such an extent that the parsing process does not fail.

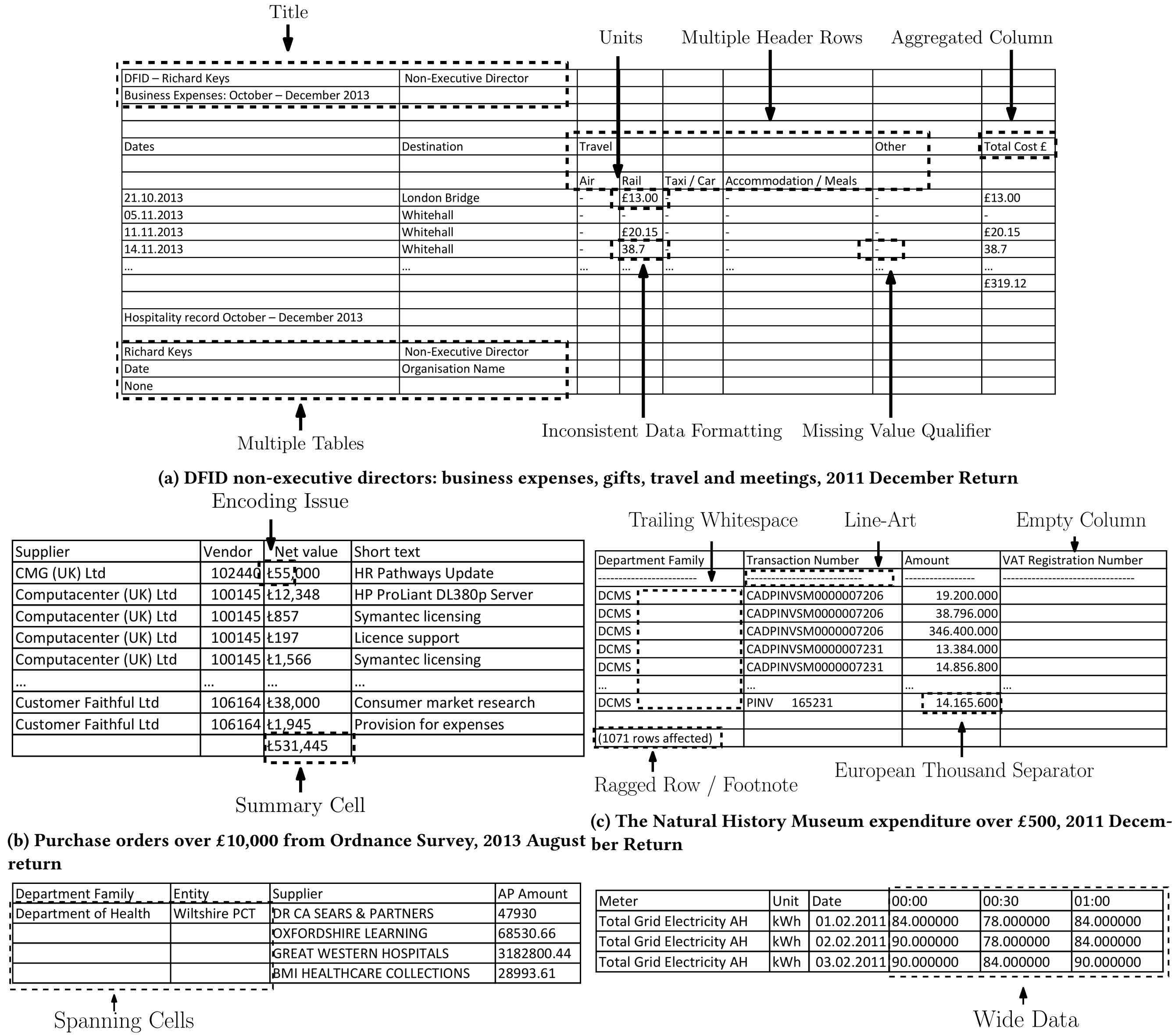
`CSVLint`[2] is a tool which checks the compliance of CSV files with the RFC 4180 beyond the mere syntax. It checks whether, e.g., the first row contains a header and if the columns contain consistent values. A regular table, written according to RFC 4180, should usually not produce any CSVLint issues. To check the quality of parsing results from `readr` and `messytables`, we wrote the parsing results back into a RFC 4180 format and checked them with `CSVLint`. For 31 out of 79 parsing results from the `readr` package, still `CSVLint` issues were observed. The *messytable* library appeared to produce better parsing results of which only 20 were still affected by issues. These results suggest that a considerable amount of files have not been parsed properly and presumably have issues which were disregarded during the parsing process. These issues will be discussed in the following paragraphs. Table 1 summarizes the results.

Parser	Success Rate	
R native	56/80	
R readr	79/80	
Python `messytables`	80/80	
Parser	**Files with `CSVLint` Issues**	
	before parsing	after parsing
`R readr`	56	31
Python `messytables`	56	20

Table 1: CSV parsing success and `CSVLint` issues

CSV File-level Issues. We observed that especially *meta-data*, such as titles, comments and footnotes, occurs very commonly in CSV files. 22 out of 80 sampled files contained some sort of meta-data (see Figure 2c). State-of-the-art parsers feature header line detection which skips meta-data rows at the beginning of the table but do not remove meta-data at the end or on the sides of the table. The table in Figure 2c contains a row which only consists of *line-art* that is supposed to improve the human readability. Such elements do not contain useful information and obstruct the data type detection. Current CSV parsers do not account for that. Due to CSV exports from spreadsheets, CSV files often contain **empty rows or columns** which surround the actual table. 18 files of our sample

[1] https://github.com/okfn/messytables
[2] http://csvlint.io/


Title
Units
Multiple Header Rows
Aggregated Column
Multiple Tables
Inconsistent Data Formatting
Missing Value Qualifier
Encoding Issue
Summary Cell
Trailing Whitespace
Line-Art
Empty Column
Ragged Row / Footnote
European Thousand Separator
Spanning Cells
Wide Data

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
05.11.2013	Whitehall	-	-	-	-	-	-
11.11.2013	Whitehall	-	£20.15	-	-	-	£20.15
14.11.2013	Whitehall	-	38.7	-	-	-	38.7
…	…	…	…	…	…	…	…
							£319.12
Hospitality record October – December 2013							
Richard Keys	Non-Executive Director						
Date	Organisation Name						
None							

(a) DFID non-executive directors: business expenses, gifts, travel and meetings, 2011 December Return

Supplier	Vendor	Net value	Short text
CMG (UK) Ltd	102440	ł55,000	HR Pathways Update
Computacenter (UK) Ltd	100145	ł12,348	HP ProLiant DL380p Server
Computacenter (UK) Ltd	100145	ł857	Symantec licensing
Computacenter (UK) Ltd	100145	ł197	Licence support
Computacenter (UK) Ltd	100145	ł1,566	Symantec licensing
…	…	…	…
Customer Faithful Ltd	106164	ł38,000	Consumer market research
Customer Faithful Ltd	106164	ł1,945	Provision for expenses
		ł531,445	

(b) Purchase orders over £10,000 from Ordnance Survey, 2013 August return

Department Family	Transaction Number	Amount	VAT Registration Number
------------------------	--------------------------	----------------	--------------------------------
DCMS	CADPINVSM0000007206	19.200.000	
DCMS	CADPINVSM0000007206	38.796.000	
DCMS	CADPINVSM0000007206	346.400.000	
DCMS	CADPINVSM0000007231	13.384.000	
DCMS	CADPINVSM0000007231	14.856.800	
…	…	…	…
DCMS	PINV 165231	14.165.600	
(1071 rows affected)			

(c) The Natural History Museum expenditure over £500, 2011 December Return

Department Family	Entity	Supplier	AP Amount
Department of Health	Wiltshire PCT	DR CA SEARS & PARTNERS	47930
		OXFORDSHIRE LEARNING	68530.66
		GREAT WESTERN HOSPITALS	3182800.44
		BMI HEALTHCARE COLLECTIONS	28993.61

(d) Spend over £25,000 in NHS Wiltshire, 2011 October Return

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

(e) DFID HQ buildings water and energy use

Figure 2: Tables from *data.gov.uk* illustrating a multitude of issues

contained entirely empty columns or rows. These rows/column do not contain useful information and should not be considered as part of the relational data. However, messytables, for example, does not remove such rows and columns. We also encountered CSV files which contained **multiple tables**. One example is shown in Figure 2a. Regular CSV parsers do not account for multiple tables in one CSV file, which leads to problems with determining header rows and column data types.

Table-level Issues. CSV is under-defined in the sense that the presence of a **header row is optional** and not explicitly known. Consequently the header row has to be inferred from the file content. Current CSV parsers use heuristics to detect the presence/non-presence of column headers which are subject to uncertainties. The RFC 4180 also prescribes that a CSV table should contain exactly one header row. Current CSV parsers build on that assumption but our sample contained at least 4 tables with **multiple header rows** (see Figure 2a). Ignoring multiple header rows either impedes column-wise data type detection or leads to omitted header information. The same holds for the **table orientation**. Usual parsers assume a vertical table orientation and detect headers and column data types based on this assumption. Another issue is *wide or narrow data*. Figure 2e contains **wide data** which means that the same variable is spread over multiple columns and that the header contains observation values, not variable names. This non-*tidy* data arrangement hampers data analysis and integration. Two tables in the sample contained wide data. Narrow data refers to a data arrangement where different variables are stores in one column. If the two variables are of a different data type, this hampers the data type detection. 13 files of the sample contained columns, rows or cells which are **aggregates** of other cells in the table. Those columns/rows/cells contain redundant information which can be easily reproduced. In addition, summary cells disturb the rectangular shape of the table as shown in Figure 2b.

Column/Cell-level Issues. CSV does not support **spanning cells**. When tables with spanning cells are exported from spreadsheets, only the first cell gets filled with values and the succeeding fields are left empty (see Figure 2d). For further data processing it would make sense to infer the original extent of the spanning cell and to duplicate the respective value to this extent. Another issue is that cells often contain **leading or trailing whitespace**, either by mistake or to visually align the cell content. The whitespace impedes data type detection, is most likely not intended and should be removed as well. Numerous files in the sample also contained **numerics with units**. Regular CSV parsers do not identify those values as numerics but as strings. This hampers possibilities for subsequent data analysis and requires additional manual preparation steps. There is also **no standard data type encoding** for CSV. Depending on the used CSV writer implementation or the system's locale, the data type formatting can differ. For the data consumer neither the data type not the format is known and has to be inferred from the data itself. Especially dates (dd-mm-yyyy vs. mm-dd-yyyy) and numerics (European vs. American thousand and decimal separators) can be ambiguously interpreted. When data was entered manually it can even be **inconsistently formatted**. Regular CSV parsers, if they detect data types, usually assume that the entire column is consistently formatted which makes the data type parsing fail on inconsistently formatted columns. **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 parsers 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 interfere with the data type detection. On the other hand, valid cell content should not be accidentally discarded by being classified as missing values.

Summary. We showed that a host of issues stand between a CSV file and a proper *tidy* relational table, and that current CSV parsers do not sufficiently address those issues. The issues touch different problem areas, reaching from solving ambiguities in broken or non-standard CSV syntax to table interpretation and normalization tasks and robust data type parsing. "Robust" CSV parsers tackle syntax issues and data type parsing to a certain extent, but miss the table interpretation/normalization aspect. Related work on table interpretation and normalization is largely focused on other input formats such as spreadsheets and HTML because they provide a larger set of hints for table interpretation tasks than CSV, such as different font types, cell formulas and spanning (see Section 5). One solution which aims at normalization of table structures in CSV is the application `DeExcelerator` [7], which does on the other hand not tackle CSV syntax issues. An existing solution which aims at integrating these different aspects does to our knowledge not exist, and this is what we set out to create.

3 MULTI-HYPOTHESIS PARSING

The parsing process of regular CSV parsers consists of a linear chain of detection and parsing steps reaching from encoding, dialect and header detection to determination of column-specific data types. Since certain properties of the input data format, such as the dialect, are not explicitly known, they have to be inferred from the data itself which is subject to uncertainties. However, "linear" CSV parsers do only pursue one single hypothesis about the file format and the correct way of parsing it. This is computationally efficient and may lead to correct parsing results but if one of the assumptions made is not correct, the parsing process is likely to lead to a wrong interpretation of the input or fails completely. As we have shown in the previous section, current CSV parsers lack important table normalization steps which prevents proper header and data type detection. Adding those additional steps, which also involve uncertainties, to a linear process chain would only increase the likelihood that the parsing result contains a false interpretation of the input. We propose to regard the problem from a more holistic point of view and provide a solution which makes it possible to integrate different solutions for specific sub-problems and on the other hand creates synergies from considering them together. The proposed *multi-hypothesis* parsing approach circumvents the problem of uncertainties in linear CSV parsing by allowing each parsing step to pursue multiple parsing hypotheses and passing all possible outcomes of the different hypotheses on to the next parsing step. In that way, a tree of parsing hypotheses and intermediate results is created, of which at least one leaf node is expected to lead to a correct interpretation of the input. This, as we will show, can be determined based on different data quality features.

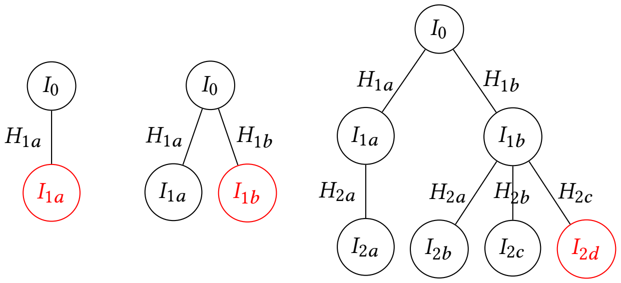


Figure 3: Three different stages of the growing hypothesis tree while it is being traversed with *breadth-first search*. Red nodes are the currently active ones.

3.1 Framework

The core of the multi-hypothesis parser is a tree data structure in which each level represents a parsing step, each node an intermediate result and each branch a parsing hypothesis. The root node is typically an input file and the successive steps are intended to gradually process the input towards the desired output. It is not prescribed which parsing steps are being implemented as long as they implement two methods: Each parsing step has to implement a `detect()`-method which creates hypotheses and respective confidence values and a `parse()`-method which processes the input according to one of the previously generated hypotheses and returns a new intermediate result. Intermediate results can be regarded as a contract between two succeeding parsing steps. An orientation detection step, e.g., promises the next step that the returned intermediate is a vertically oriented table. The succeeding parsing step can therefore focus on its specific sub-task based on a fixed assumption about the input. The previous parsing steps do, of course, not always actually deliver the intermediate result in the promised shape but because the previous steps considered all possible permutations of parsing hypotheses one can expect that at least one of the intermediate results is actually in the promised shape. The second assumption is that the path in the hypothesis tree in which all contracts are fulfilled also ultimately leads to better parsing results, which can be identified based on quality features.

The hypothesis tree is built on-the-fly while it is being traversed (see Figure 3 and Algorithm 1). Therefore, the tree traversal has to be either performed in *pre-order* or with *breadth-first search.*

Algorithm 1 Hypothesis Tree Traversal

```
 1: function $generate\_parsing\_tree(path)$
 2:     $tree \leftarrow$ CREATE_ROOT_NODE$(path)$
 3:     $hypo \leftarrow parsing\_step[1].detect(path)$
 4:     $tree$.ADD_NEW_NODE(for each $hypo$)
 5:     while not all $nodes$ evaluated, traverse $tree$ do
 6:         if $node.conf < prune\_level$ then continue
 7:         $level \leftarrow node.parent.level + 1$
 8:         $parent \leftarrow node.parent.inter$
 9:         $hypo \leftarrow node.hypo$
10:         $inter \leftarrow parsing\_steps[level].parse(parent, hypo)$
11:         if $inter$ is $null$ then continue
12:         $node$.ADD_NODE$(inter)$
13:         if $level$ is $length(parsing\_steps)$ then continue
14:         $hypo \leftarrow parsing\_steps[level + 1].detect(inter)$
15:         $node$.ADD_NEW_NODE(for each $hypo$)
16:     end while
17:     return $tree$
18: end function
```

The prune-level (see line 6 in Algorithm 1) is an optional parameter which can be set to filter our hypotheses with a very low confidence. In that way the user can control the amount of created hypotheses per step, regardless of the concrete implementation of the detection steps.

3.2 Parsing Steps

We have created a `R` package for multi-hypothesis CSV parsing, `hypoparsr`, which is available from the Comprehensive `R` Archive Network (CRAN)[3]. This section will describe the implemented parsing steps (see Figure 4).

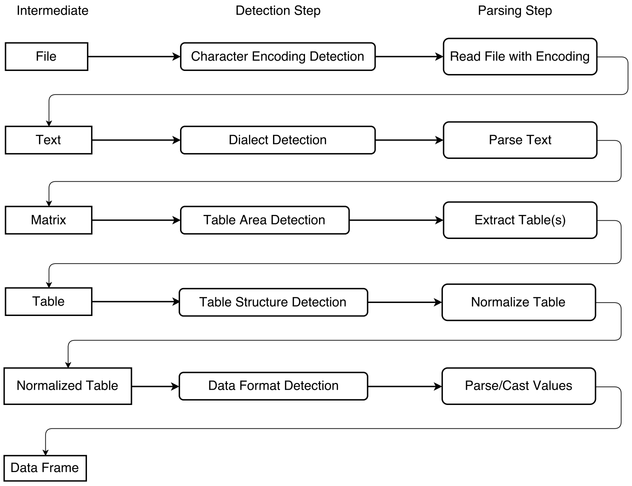


Figure 4: Parsing steps of `hypoparsr`. Table structure detection summarizes the detection of row and column functions.

These parsing steps aim at covering the most frequently occurring issues in CSV files, such as unknown file encodings, different dialects, (multiple) header rows, contained meta-data and non-standardized and inconsistent data types. The proposed detection steps use heuristics to determine a set of reasonable hypotheses and respective confidence values. Additionally, one default hypothesis always is created, which assumes that the input is conforming to RFC 4180, to assure that this possibility is never disregarded. The generated hypotheses are as explicit as possible and the parsing steps as strict as possible, in order to rule out wrong hypotheses at an early stage and confirm hypotheses most precisely. Each detection and parsing step has access to a utility function which determines the type of cell content, such as empty, numeric, date, and time, based on regular expressions. More details on the implementation are provided in [6], but we will describe the implemented parsing steps in the following:

For *encoding detection* we used the `guess_encoding()` function of the `readr` [25] package. This function provides a set of potential character encodings along with respective confidence values. The parsing step reads the file with the respective encoding and returns a text as intermediate result.

In the *dialect detection* step all combinations of 10 different delimiters, 15 different quotes in including typographic variances, 3 different line endings and two different escaping styles are checked for plausibility. The plausibility check is very simple in order to allow also unlikely interpretations to be considered for further steps. If a delimiter occurs at least once in the file it will be considered as a candidate. Quoting signs are considered as candidates if they occur

[3] https://cran.r-project.org/web/packages/hypoparsr/

at least once directly before or after a delimiter. In the parsing step the robust CSV parsing library `readr` [25] is used to parse the file. If the library encounters parsing issues it either fails, which leads to a dead branch in the parsing tree, or it emits a warning, which gets logged and taken into account by the final quality assessment.

In the *table area detection* step the outer boundaries of the actual table are determined. All empty columns left/right and empty rows above/below the table are removed in this parsing step. For future work we plan to extend this step towards detection of multiple tables. The table orientation step was inspired by the work of Pivk et al. [16]. The orientation is determined based on the column/row-wise consistency of data types. If the columns are more consistent than the rows, then a horizontal orientation hypothesis is created. A vertical orientation hypothesis gets always created by default. In the parsing step, the table either gets transposed or not.

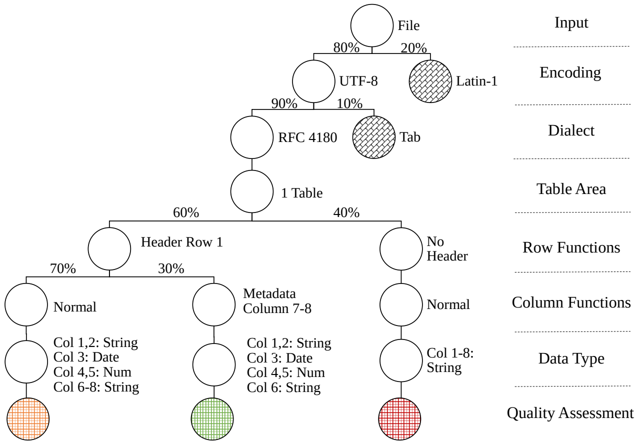


Figure 5: Exemplary Multi-Hypothesis parsing process using the proposed parsing steps.

The *table structure detection* step consists of row- and column-function detection. Row function detection is inspired by the work of Adelfio and Samet [1], which aims at classification of rows into different functional categories such as titles, headers and data. The approach of [1] is based on conditional random fields and requires a considerable amount of manually annotated training data. As this data was not available to us, we created a simpler heuristic solution which performs a similar classification and is able to produce a set of maximum 8 different hypotheses about header row location etc. In the parsing step, multiple header rows are aggregated to one, empty rows are removed and meta-data is removed from the table and saved separately.

The column function detection step uses heuristics to detect columns with spanning cells, columns with meta-data and empty columns and creates a maximum of 3 different hypotheses about their location. In the parsing step, spanning columns are expanded to their assumed original extension, empty columns are removed and meta-data is removed and saved separately.

The *data type detection* step is the last parsing step, which allows to locally optimize the parsing result. In the detection step we determine the column data types and potential data type formats, i.e. numeric separators and date formats, based on regular expressions. We support detection of numerics with different decimal and thousand separator styles and attached units, detection of dates in various formats, detection of times and logicals. In the parsing step we iteratively try to parse the column content according to the generated format hypotheses until all values were successfully parsed. If not all cells can be successfully, the column is per default considered as string column.

Figure 5 shows an exemplary hypothesis tree based on the proposed parsing steps. Theoretically, 30 different hypotheses on encoding could be created, 450 different hypotheses one dialect, one on the table area, 8 different row function hypotheses, 3 column function hypotheses and one data type hypothesis. This leads to a theoretical maximum of 324,000 parsing hypotheses:

$$30 * 450 * 1 * 8 * 3 * 1 = 324,000$$

In the *hypoparsr* implementation we defined a default pruning-level of 0.1, which means that parsing hypotheses with a confidence lower than 10% are disregarded. Because the confidence values of each detection step always add up to 100%, this means that the number of created hypotheses per step is in worst case 10. In that way the number of maximum possible parsing hypotheses can be limited to 2,400:

$$10 * 10 * 1 * 8 * 3 * 1 = 2,400$$

However, in practice we observed far lower numbers of hypotheses. We evaluated the number of created hypotheses on a set of 14,844 files from the *data.gov.uk* corpus. Table 2 shows that in median only 8 and in mean 13.4 different parsing hypotheses reached the last level of the hypothesis tree. Only in rare cases, the number of intermediates grew much larger. This was especially the case when the input contained multiple tables. In only 22 out of 14,844 cases more than 10 dialect hypotheses were created.

Variant	No. Hypotheses in Last Level			
	Min	Max	Median	Mean
Hypo	2	188	8	13.4

Table 2: Hypotheses created on *data.gov.uk* corpus.

3.3 Hypothesis Ranking

We propose a ranking approach which allows the user to not only retrieve the best hypothesis but optionally also a ranked set of parsing results. The key question for the ranking is which measurable criteria are suitable to capture the quality of the parsing result. A starting point for these considerations was the ISO/IEC 25012 Data Quality Model [11] and complementary work by Rafique et al. [18]. 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 [18], 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. We now provide 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" [11]. Because we do not know the original intent of the data, we can only assume that the original content is the intended content. The encoding and dialect steps should preferably not produce any warnings. The number of edits and moves of cells content should 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.

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 use" [11]. We cannot know if the given data is really complete, we can again only assume that the provided data is as complete as possible. 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" [11]. 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 specificness of the data type. Tables with a high number of typed cells (other than text) should be preferred.

Precision. "The degree to which data has attributes that are exact or that provide discrimination in a specific context of use" [11]. The precision of 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. 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" [11]. The understandability of a table depends strongly on whether 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 in 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" [18]. The conciseness of the table gets improved if unnecessary columns or 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 higher row-column ration should preferred.

Table 3 summarizes the proposed mapping of quality metrics to quality characteristics.

Data Quality Characteristics	Quality Metrics
Accuracy/Correctness	Parser Warnings, 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 3: Data quality characteristics mapped to quality metrics.

We used a Ranking SVM algorithm [12] with a linear kernel function to determine a ranking model using the previously described quality criteria as features. 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 parameters are easily interpretable and can be manually checked for plausibility. The training dataset for the Ranking SVM was gathered as follows: the multi-hypothesis parser was used to create different parsing hypotheses for 64 randomly sampled tables from *data.gov.uk*. We also cleaned the same tables manually and compared the result of each parsing hypothesis to the manually cleaned tables by using a string-based distance measure which is described in Section 4.1. The normalized quality features of each parsing result were used as features and the rank as target value. The feature values were scaled to a range between 0 and 1 among all permutations of the same input table. Figure 6 illustrates the training data generation process.

We used Monte Carlo cross validation [26] with 10 different randomly sampled training and test sets with a size ratio of 3/1 to tune the c parameter (0.01). The Ranking SVM ultimately lead to the feature weights shown in Table 4.

The table shows that the signs of the weights were generally in line with our expectations. 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.

However, Figure 7 shows that the success of the trained method is only marginally better than with equally weighted features. The

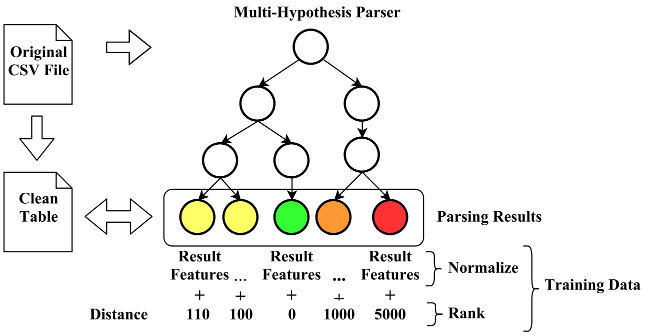


Figure 6: Schematic diagram of the training data generation process.

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 4: Quality metrics, equally weighted and weights determined by the Ranking SVM.

trained ranking method outperforms the untrained method by only 3% in terms of correctly highest ranked results. In 77% and 80% of all cases, respectively, the best parsing result is ranked highest. And in 86% and 91% of all cases, respectively, the best parsing result can be found among the three highest ranked results. The naïve method, which follows the path of highest confidence, is only about 1/3 better than the random baseline. This supports our hypothesis that parsing decisions, only based on high local confidence, do not lead to good overall parsing results.

Because the ranking with equal weights leads to good ranking decisions and the trained ranking only performs slightly better, we ultimately decided to use equal weights in the hypoparsr implementation by default. This also prevents a bias from the training data on the evaluation results (overfitting).

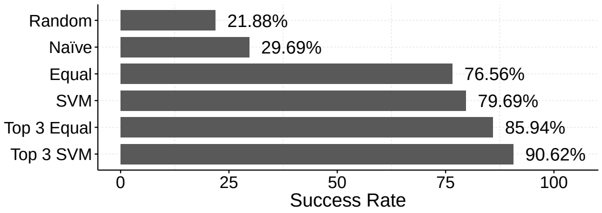


Figure 7: Comparison of different ranking methods.

4 EVALUATION

We evaluated the hypoparsr against a strict RFC 4180-conform CSV parser as implemented in the R base library [17] and the state-of-the-art CSV parser messytables which was especially designed to read messy CSV data. Furthermore, we included the application DeExcelerator [7] in the evaluation, which aims at normalizing table structures in tabular data. Because messytables and DeExcelerator have complementary features (see Tables 5), we created a pipeline which first parses the CSV files with messytables to solve CSV syntax issues and subsequently uses DeExcelerator to normalize the contained table structures and to determine data types.

Parser	Automatic Detection							
	Encoding	Dialect	Table Area	meta-data	Header	Mult. Header	Spanning Cells	Data Format
R base	-	-	-	-	-	-	-	-
messytables	x	x	-	-	x	-	-	x
DeExcelerator	-	-	x	x	x	x	x	x
hypoparsr	x	x	x	x	x	x	x	x

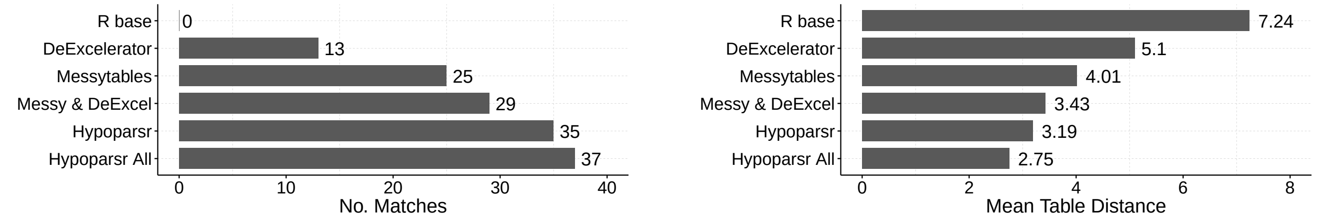
Table 5: Feature comparison of the evaluated parsers.

4.1 Ground Truth

Evaluating the correctness of parsing results is challenging because to our knowledge no common test sets or ground truths for CSV parsing evaluation exist. We therefore first established a ground truth by manually cleaning a random sample of 64 CSV files from the open government data portal *data.gov.uk*. The tables were read with a robust and manually configured CSV parser. Subsequently, header rows were manually identified, meta-data was removed, spanning cells were expanded, and NA (Not Available) markers identified and replaced with empty strings. If tables were not horizontally oriented they were transposed and if narrow or wide data was identified the tables were reshaped accordingly. Column data types for dates, times, logicals and numerics were identified and the column content was converted accordingly. If units were attached to numeric values, the unit was removed from the data cells and moved to the header row. The created ground truth is available online[4].

In order to fuzzy-match two tables and to automatically assess the deviation between two tables, we established a string-based table distance measure. To determine the distance between two tables, each table is converted into a (long) string-representation. The table is converted by appending all cells column-wise and

[4] https://github.com/tdoehmen/hypoparsr/tree/master/tests/data/cleaned



(a) Number of matches.

(b) Mean distance to cleaned tables.

Figure 8: Comparison of parsing results to ground truth with RFC 4180 conform parser, `DeExcelerator`, `messytables`, `messytables` combined with `DeExcelerator`, and the Multi-Hypothesis parser `hypoparsr`.

subsequently appending the column strings to one long string to which the header row is prepended (see Figure 9). The distance between two tables is measured based on the Levenshtein distance between the two table strings, which determines the minimum required amount of insertions, deletions and edits to turn one given string into another [14]. In that way we can assess the distance between tables in terms of missing, added and edited content. If systematic errors occur, such as not properly parsed data columns, the Levenshtein distance grows with the number of rows and over-penalizes those errors. The *Table Distance* measure is therefore the Log10-scaled Levenshtein distance. For failed parsing attempts we counted the distance from an empty string to the ground truth.

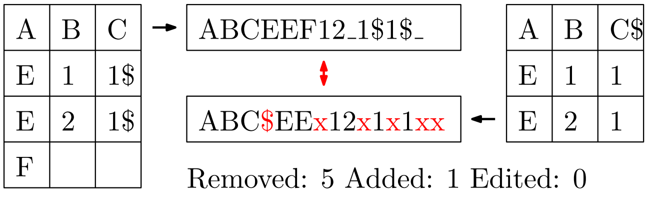


Figure 9: String-based table distance measure.

Figure 8a shows the number of matches between the ground truth and the parsing results of `hypoparsr` and the compared solutions. None of the parsing results of the RFC 4180 conform parser were in line with the manually cleaned tables. This underlines how error-prone the CSV files are and the necessity for more sophisticated CSV parsing solutions. The highest ranked result of the `hypoparsr` was in 35 out of 64 files in line with the manually cleaned tables, compared to 13, 25 and 29, when using `DeExcelerator`, `messytables` or the combination of both, respectively. The previous section showed that the ranking of the `hypoparsr` is not 100% accurate. We thus also searched for matches among all parsing hypotheses created by `hypoparsr` and not only the highest ranked (*HypoAll*). Two additional matches could be identified in non top-ranked positions, which confirms that the ranking could be optimized in future work.

Figure 8b shows the mean table distance between parsing results of the described parsers and the ground truth. The `hypoparsr` results are thus not only most often matching with the ground truth but are also in average closer to the ground truth which reflects the robustness of the solution. Even though our solution leads on the test set to better results that the state-of-the-art, the results show that there is room for improvement. Manual analysis of highly deviating parsing results showed that especially improving the normalization of complex table structures and a higher variety of row/column function hypotheses could further improve the results.

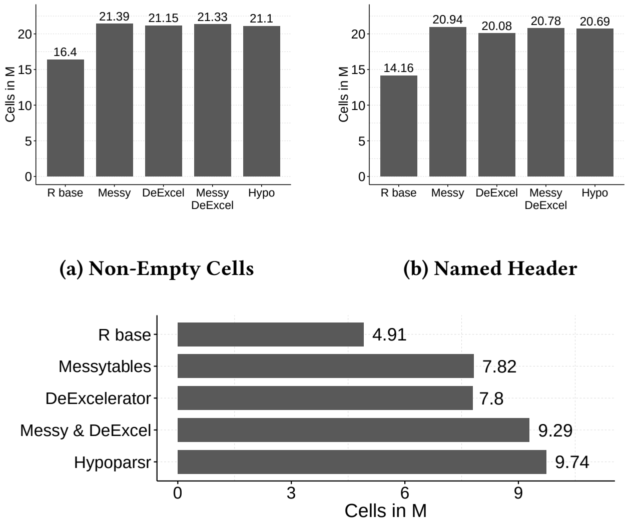
4.2 Parsing Success

We specifically aimed at creating a solution which is suited for unsupervised processing of large corpora of heterogeneous CSV data sources. We therefore evaluated the parsing success of the `hypoparsr` on a test set of 14,844 files from the open government portal, which make up approx. 90% of all CSV files on the portal *data.gov.uk* in August 2015. This corpus was reduced by only keeping files smaller than 200kB and was superficially cleaned from non-CSV and empty files. On the test set, with 99.55% the parsing success rate of `hypoparsr` was the highest of the tested systems. The remaining fraction of not successfully parsed files can furthermore mainly be attributed to non-CSV files which should have been removed from the test set. Table 6 summarizes the parsing success of `hypoparsr` and the compared systems. The low rate of the RFC 4180 conform parser shows again the poor quality of the CSV files on the data portal.

Parser	R base	Messy	DeEx	Me.DeEx	Hypo
No. Files	11,619	14,762	14,621	14,706	14,777
% Files	78.27	99.45	98.49	99.07	99.55

Table 6: Number of successfully parsed files of the Multi-Hypothesis parser, compared to others.

Not only the parsing success but also the amount of recovered cells is an important indicator for the quality of the parsing results. Recovering a high amount of data from the original files is essential for subsequent integration and analysis processes. It would not be desirable if the parsing succeeds but a large part of the input data is disregarded. We used three simple measures, namely non-empty cells, cells with column header (Named Cells), and non-string cells (Typed Cells), to evaluate the amount of recovered data from the *data.gov.uk* corpus. Figure 10 shows the aggregated amount of cells which could be recovered from the *data.gov.uk* corpus by the respective parsers. It shows that the amount of recovered non-empty cells and names cells of all solutions but the R base implementation are roughly in line. The hypoparsr recovered slightly fewer non-empty cells than messytables which can be attributed to removed meta-data, but the slightly lower amount of names cells indicates that there is room for improvement of header detection. Figure 10c reveals the true strength of the hypoparsr, which is the column-wise consistency of values and thus the ability to determine column-specific data types in which it outperforms all other systems we compared with. Since the CSV files are expected to contain tabular data, which typically have consistently typed columns, this is a strong indicator that many tables were correctly extracted from the input data.



(a) Non-Empty Cells

(b) Named Header

(c) Typed Cells

Figure 10: Amount of recovered data cells from *data.gov.uk* corpus with hypoparsr and compared solutions.

Because the amount of tables in the greatly 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 91% and accuracy of 78% in terms of files identified as affected by issues, based on the previously used sample from *data.gov.uk*. Figure 11 shows the amount and distribution of CSVLint issues in the original files from *data.gov.uk*.

By writing back the respective parsing results into a RFC 4180 conform file with UTF-8 encoding, many issues can be automatically resolved, but issues such as *Title Row*, *Inconsistent Values* and *Check Option*, which is triggered if the parsing result has only one column, will remain regardless. They indicate that either the input file did not contain a proper table or that the table was not properly parsed. Figure 12 shows the amount of issues in the results of the compared parsers. Our solution leads to the lowest amount of remaining CSVLint issues and reduced the number of issues to less then a half of the original amount of issues. The unresolved issues of hypoparsr and the combined solution of messytables and DeExcelerator have an overlap of approx. 85% of which the majority is related to inconsistent values. Some tables might simply contain columns with inconsistent values, which is not necessarily an issue which can and should be solved by CSV parsers, but rather at the data source. Remaining issues which could be solved by the compared solutions, but not by the hypoparsr, are mostly related to title rows, thus not properly detected column header, which is in line with previous observations. Of 2,936 original *Title Row* issues, messytables could solve 2,515 and hypoparsr 2,114.

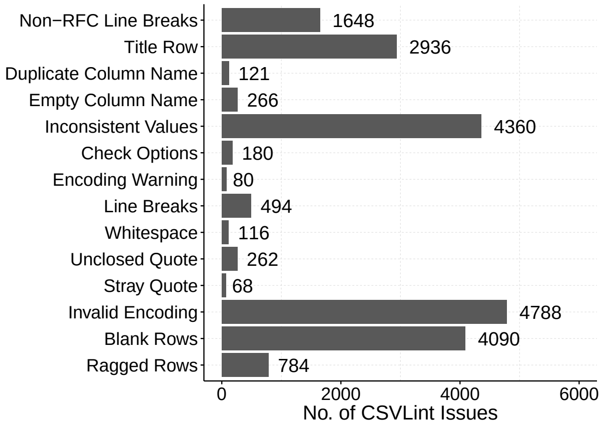


Figure 11: CSVLint issues in the *data.gov.uk* corpus.

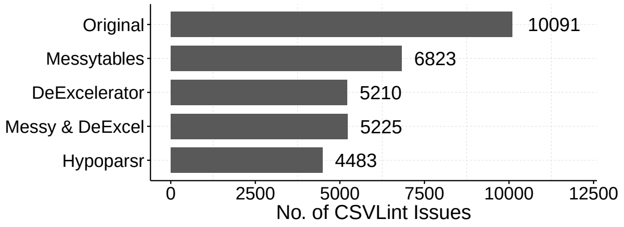


Figure 12: CSVLint issues after parsing and writing back to RFC4180-conform CSV.

5 RELATED WORK

Data-wrangling or cleaning solutions such as OpenRefine[5], Trifacta Wrangler [9] are well suited for manual cleaning of individual tables and 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. Techniques such as *predictive user interaction* [9] aim at reducing the manual effort per table but do not yield full

[5] https://github.com/OpenRefine/OpenRefine

automation which makes them unsuited for cleaning of large corpora of heterogeneous tables. Furthermore, the tabular data has to be loaded in the first place which might, given the variety of CSV dialects and syntax issues, not always succeed.

The discussed issues in CSV files are often related to non-normalized table structures. The problem of table normalization is closely related to table interpretation which lies at the intersection of document analysis, information retrieval and information extraction. However, end-to-end table interpretation solutions, as proposed by Hurst [10] and Pivk et al. [16], have a large margin of uncertainty and make use of background knowledge from the Semantic Web. As shown by Chu et al. [4], the performance of an approach using background knowledge significantly decreases for non-public data sets such as corporate datasets. For those data sets the syntactical structure of the content is more important, as also shown by Cortez et al. [5]. Seth et al. [19] proposed a reliable approach for table normalization based on sequential circuit analysis, which does not require background knowledge, but only succeeds on tables with unique access path to data cells.

Google's WebTables [3] project applied table interpretation techniques at web-scale, on a corpus of 154M tables. WebTables, however, disregards not-normalized tables and assumes that every table contains one header row and succeeding data rows. The approach of Adelfio and Samet [1], which is based on *conditional random fields*, aims at improving such processes with row-wise classification of table content into categories such as meta-data, header and data. The approach focuses on HTML and spreadsheets and makes use of features like spanning cells and font types which are not available in CSV and furthermore requires a considerable amount of manually annotated training data. Still, these approaches do not achieve human level performance. Table interpretation and normalization at human level performance is still an unsolved problem.

CSV is not often considered as input format for automatic table interpretation tasks, presumably because CSV is already considered tabular and the issues in CSV parsing and interpretation are often underestimated. The `DeExcelerator` proposed by Eberius et al. [7] was the only practical solution for table normalization we could identify which supports CSV as input format. However, Seth et al. state in a later publication that CSV contains sufficient information for their approach to work [15].

Approaches dealing with automated semantic enrichment of tabular data, amongst other open government data, often make use of CSV as input format but appear to ignore the underlying issues (see, e.g., Sharma et al [22]). Ermilov et al. [8] who worked on data from the open government data portal *publicdata.eu* recognized the necessity for a cleaning process of the CSV input, especially to remove leading and trailing meta-data which was occurred in approximately 20% of their files. However, in the end they reported: "In such cases [leading/trailing meta-data], the location of the header is currently not properly determined" [8]. The lack of suitable cleaning/normalization solutions for CSV data thus significantly reduces the amount of usable data or, if issues are ignored, leads to high noise for any kind of downstream process.

The related work mentioned so far mainly focuses on the data consumer side, because the original data lacks meta-information on its structure and content in the first place. The W3C *CSV on the Web Standard* [23] targets exactly this problem and proposes a JSON schema for CSV meta-information. The JSON contains information about the used CSV dialect, the position of the header row, contained entities and their data types. 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.

Another type of issues occurring in CSV files are inconsistent use of dialects or syntax errors. In cases of such errors our current solution still relies on the robustness of the underlying CSV parsing library `readr` [25]. List extraction solutions such as the system TEGRA by Chu et al. [4] aim at extracting tabular data from lists. The approach is able to recover tabular data from lists which contain values that are not or only inconsistently delimited by optimizing the column-wise coherence of values. The approach achieves encouraging results on lists from the Web. In future work, list extraction approaches could potentially be used to recover tabular data from CSV files which are syntactially broken.

6 FUTURE WORK

In practical terms, regarding our `hypoparsr` project, we propose to add new plug-ins and extend/improve the existing ones. Based on the observed issues in the *data.gov.uk* corpus, we propose to add support for multiple tables, table orientation detection, wide and narrow data detection, and removal of duplicated columns. Existing modules such as the row function and column function detection could be improved by creating better hypotheses, especially for header rows, which were shown to not always be correctly identified. Alternatively, more sophisticated row function classification methods, as proposed by Adelfio and Samet [1] could be considered.

The more plug-ins are added the larger the parsing tree potentially grows. In future work, different pruning methods and the effect on the result quality should be evaluated. A larger set of parsing hypotheses also calls for more reliable quality assessment. The Ranking SVM performance could be improved by repeating the described training process with a larger set of manually cleaned tables. Alternatively, a set of highest ranked solutions could be presented to the user, letting the user choose the preferred table. In that way, the user input could also be 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. In order to automatically detect and filter out unfavourable tables, an absolute measure of table quality should be established. 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.

With the current `hypoparsr`, the user receives a parsing report alongside the resulting table, stating which parsing decisions were made. Such parsing descriptions can currently not be fed back into the parsing process. Using the *CSV on the Web* standard, as formal language to describe parsing results, has the limitation that it does not cover all tasks performed by the multi-hypothesis parser, and cannot easily capture inconsistent CSV files. A new description language would be required e.g. along the lines of Arenas et al. [2]. Many different CSV files from the same source may have the same kind of issues, and the formal descriptions created by hypoparsr on large dataset collections might be clustered to group similarly structured CSV files. Further, user feedback (e.g. using a *data wrangling* user interface that starts at the initial hypoparsr output), resulting in a fine-tuned formal parsing description, might thus be applied to all CSV files in the same cluster, creating a feedback loop for bulk-parsing large CSV corpora.

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. [4] could be utilized to improve the automatic fixing procedure of syntactically broken CSV files. In our test set of very diverse files from *data.gov.uk*, serious syntactical issues which would require a re-alignment of values were, however, seldomly observed.

In our current implementation of hypoparsr, we did not focus on optimizing the runtime of the parsing process. Parsing of files from the *data.gov.uk* corpus took 45s on average (on a single core machine). In future work, the runtime performance should be further evaluated and improved, especially on larger files. We note that larger files typically are quite consistent such that taking a sample of the data likely will allow to determine a good parsing strategy.

So far, the performance of the hypoparsr 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 also be evaluated on other corpora. We hope that a variety of R users will pick up hypoparsr and contribute experiences and plug-ins that will further improve the package.

7 CONCLUSION

We have shown that CSV files in the UK open government data portal suffer from various issues which pose a major hurdle for data analysis and integration. Related work confirms that those kinds of issues are not limited to CSV's on *data.gov.uk* only. We have proposed our modular multi-hypothesis parsing framework which aims at regarding the separate issues and challenges of parsing a CSV from a more holistic and data-quality oriented point of view than current CSV parsers do. We have created an R package for multi-hypothesis CSV parsing, hypoparsr, which is available from the Comprehensive R Archive Network (CRAN) and evaluated the solution on a large sample of files from *data.gov.uk* against other CSV parsing solution. On this data set collection, our solution leads to improved parsing results than the state-of-the-art. The de-coupling of different parsing steps and the quality-based result ranking substantially reduces the implementation effort and facilitates the re-use of existing algorithm implementations for the various parsing steps.

Acknowledgments

This work was funded by the Netherlands Organisation for Scientific Research (NWO), project “Capturing the Laws of Data Nature” (Mühleisen).

REFERENCES

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

[2] M. Arenas, F. Maturana, C. Riveros, and D. Vrgoč. A framework for annotating CSV-like data. *Proceedings of the VLDB Endowment*, 9(11):876–887, 2016.

[3] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *Proceedings of the VLDB Endowment*, 1(1):538–549, 2008.

[4] X. Chu, Y. He, K. Chakrabarti, and K. 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.

[5] E. Cortez, D. Oliveira, A. S. da Silva, E. S. de Moura, and A. H. 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.

[6] T. Döhmen. Multi-Hypothesis Parsing of Tabular Data in Comma-Separated Values (CSV) Files. Master's thesis, Vrije Universiteit Amsterdam, www.cwi.nl/ boncz/msc/2016-Doehmen.pdf, 2016.

[7] J. Eberius, C. Werner, M. Thiele, K. Braunschweig, L. Dannecker, and W. 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.

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

[9] J. Heer, J. M. Hellerstein, and S. Kandel. Predictive interaction for data transformation. In *7th Biennial Conference on Innovative Data System Research, CIDR*, volume 15, 2015.

[10] M. F. Hurst. The interpretation of tables in texts. 2000.

[11] Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Data quality model. Standard, International Organization for Standardization, Geneva, CH, Mar. 2008.

[12] T. 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.

[13] J. G. Kim and M. Hausenblas. 5-star open data, 2015.

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

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

[16] A. Pivk, P. Cimiano, Y. Sure, M. Gams, V. Rajkovič, and R. Studer. Transforming arbitrary tables into logical form with TARTAR. *Data & Knowledge Engineering*, 60(3):567–595, 2007.

[17] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2016.

[18] I. Rafique, P. Lew, M. Q. Abbasi, and Z. Li. Information quality evaluation framework: Extending ISO 25012 data quality model. *World Academy of Science, Engineering and Technology*, 65:523–528, 2012.

[19] S. Seth and G. 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.

[20] Y. Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180 (Informational), Oct. 2005. Updated by RFC 7111.

[21] Y. Shafranovich. IESG CSV MIME Type, 2014.

[22] K. Sharma, U. Marjit, and U. Biswas. Automatically Converting Tabular Data to RDF : An Ontological Approach. *International Journal of Web & Semantic Technology (IJWesT)*, 6(3), 2015.

[23] J. Tennison. CSV on the Web: A Primer, W3C WG Note. Working group note, W3C, Feb. 2016. http://www.w3.org/TR/2015/REC-tabular-data-model-20151217/.

[24] H. Wickham. Tidy data. *Under review*, 2014.

[25] H. Wickham, J. Hester, and R. Francois. *readr: Read Tabular Data*, 2016. R package version 1.0.0.

[26] Q.-S. Xu and Y.-Z. Liang. Monte carlo cross validation. *Chemometrics and Intelligent Laboratory Systems*, 56(1):1–11, 2001.