ExtracTable

Extracting Tables From Plain Text

Leonardo Hübscher

Prof. Felix Naumann
Lan Jiang

28.05.2021

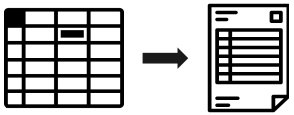
Tables in unstructured data

Data growth

100X[1]

(2013→2025)

Data is growing fast



Data format

structured → unstructured[1]

Data is stored in plain text files

Table files

>50% files[2]

(Mendeley Data)

Tables are valuable

[1] Reinsel, D., Gantz, J., & Rydning, J. (2017). Data Age 2025 : Don't Focus on Big Data; Focus on the Data That's Big
[2] Jiang, L., Vitagliano, G., & Naumann, F. (2021). Structure Detection in Verbose CSV Files

Data tables

- Often used for data sharing
- Structured
- Store information in high densities
- Interpretable by humans and machines

Example

Last name	First name	YOB
Willis	Bruce	1955
Carano	Gina	1982
Lutz	Kellan	1985

Extraction (2015)[3]

[3] https://www.imdb.com/title/tt4382872

Table formats

CSV
Dialect

```
Name,Minute,Quote
Victoria,39,
Harry,40,"\"I like complicated.\""
```

ASCII
Column boundaries

```
Name        Minute      Quote
Victoria        39
Harry           40      "I like complicated."
```

Table formats

CSV
Dialect

```
Name,Minute,Quote
Victoria,39,
Harry,40,"\"I like complicated.\""
```

Delimiter

Quotation

Escape

Required

Optional

ASCII
Column boundaries

```
Name      Minute      Quote
Victoria      39
Harry         40      "I like complicated."
```

0-8

11-17

23-43

Table format variants

```
Last name,First name,YOB
Willis,Bruce,1955
Carano,Gina,1982
Lutz,Kellan,1985
```

RFC 4180

```
"Last name"; "First name"; "YOB"
"Willis"; "Bruce"; 1955
"Carano"; "Gina"; 1982
"Lutz"; "Kellan"; 1985
```

Different dialects, intermittent spaces

```
Full Name,YOB
Bruce Willis,1955
"Carano, Gina",1982
Kellan Lutz,1985
```

Inconsistent quotes

```
Full name; YOB
Willis, Bruce;05/19/1955
Carano, Gina;04/16/1982
Lutz, Kellan;03/15/1985
```

Ambiguous CSV tables

Table format variants

```
Title: Extraction
Production year: 2015

Storyline:
Harry Turner works for the CIA like his…

[Actors]
Last name,First name,DOB
Willis,Bruce,1955
Carano,Gina,1982
Lutz,Kellan,1985
YOB = Year of birth
```

Files with preamble

```
First name   Last name      YOB
----------   ---------     ----
Bruce        Willis        1955
Gina         Carano        1982
Kellan       Lutz          1985
```

ASCII table with styling

```
First and last name          YOB
B. Willis                   1955
Gina Carano                 1982
```

Ambiguous ASCII tables

Data scientists spend up to 80% of their time on data wrangling[4]

[4] Press, G. (2016). Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says

Related Work

Hypoparsr[5]
2016

Till Döhmen
Research Assistant at
Fraunhofer FIT

Improve quality by deferring decision-making for sub-problems like encoding & dialect detection

CleverCSV[6]
2019

Gertjan van den Burg
Postdoctoral researcher at
The Alan Turing Institute

Dialect detection of CSV files based on row-patterns and cell data types

Pytheas[7]
2020

Christina Christodoulakis
PhD student at
University of Toronto

Infers the table ranges by applying a set of weighted fuzzy rules

[5] Döhmen, T. (2016) 'Multi-Hypothesis Parsing of Tabular Data in Comma-Separated Values (CSV) Files'
[6] van den Burg, G. J. J. (2019) 'Wrangling messy CSV files by detecting row and type patterns'
[7] Christodoulakis, Christina, et al. (2020) 'Pytheas: pattern-based table discovery in CSV files'

Outline

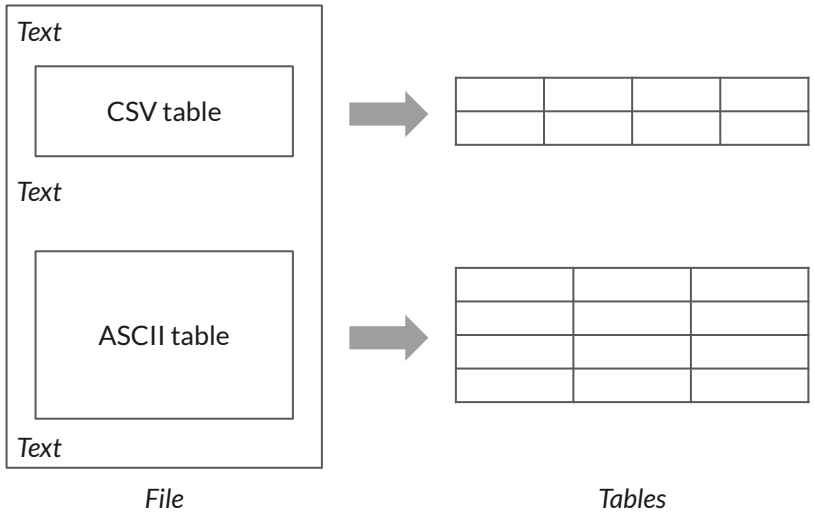
1. Introduction
2. Problem statement
3. Demo
4. ExtracTable
5. Evaluation
6. Conclusion
7. Future Work

Problem statement

We want to develop an algorithm that extracts tables from plain text files to decrease the time spend on data wrangling.

Key features

- CSV and ASCII tables
- CSV dialects deviating more from RFC 4180
- Multi-table files
- Files containing surrounding text

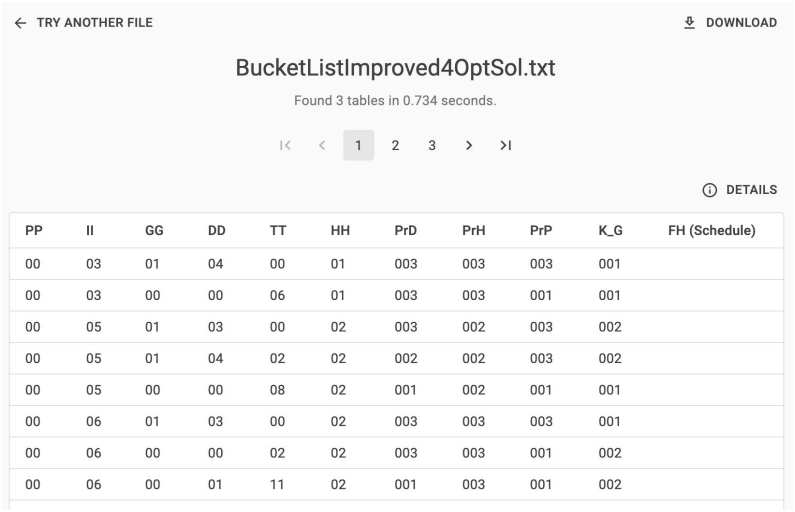


Demo

Let's see ExtracTable in action!

We want to see, how ExtracTable:

- Detects the correct dialect for CSV tables
- Works on files with surrounding text/ meta
- Works on multi-table files



PP	II	GG	DD	TT	HH	PrD	PrH	PrP	K_G	FH (Schedule)
00	03	01	04	00	01	003	003	003	001	
00	03	00	00	06	01	003	003	001	001	
00	05	01	03	00	02	003	002	003	002	
00	05	01	04	02	02	002	002	003	002	
00	05	00	00	08	02	001	002	001	001	
00	06	01	03	00	02	003	003	003	001	
00	06	00	00	02	02	003	003	001	002	
00	06	00	01	11	02	001	003	001	002	

Screenshot of demo web app

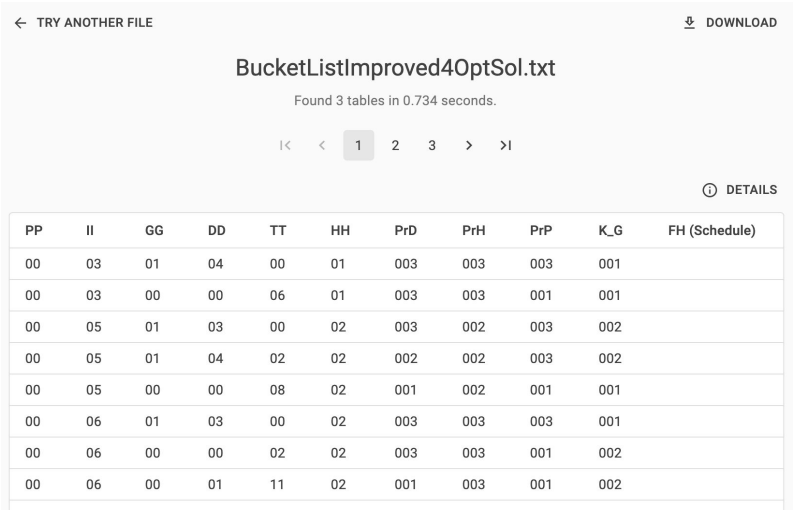
Demo

Demo

Let's see ExtracTable in action!

We want to see, how ExtracTable:

- Detects the correct dialect for CSV tables ✔
- Works on files with surrounding text/ meta ✔
- Works on multi-table files ✔



PP	II	GG	DD	TT	HH	PrD	PrH	PrP	K_G	FH (Schedule)
00	03	01	04	00	01	003	003	003	001	
00	03	00	00	06	01	003	003	001	001	
00	05	01	03	00	02	003	002	003	002	
00	05	01	04	02	02	002	002	003	002	
00	05	00	00	08	02	001	002	001	001	
00	06	01	03	00	02	003	003	003	001	
00	06	00	00	02	02	003	003	001	002	
00	06	00	01	11	02	001	003	001	002	

Screenshot of demo web app

ExtracTable

ExtracTable

Input Plain text file

Output CSV file(s) each representing a table (RFC 4180)

Main idea Exploit the data type consistency within columns to detect tables

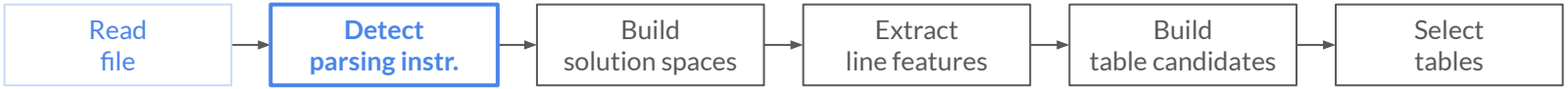
```
Full name;DOB;POB
Willis,Bruce;05/19/1955;Germany
Carano,Gina;04/16/1982;USA
Lutz,Kellan;15/03/1985;USA
```

Input

Full name	DOB	POB
Willis, Bruce	05/19/1955	Germany
Carano, Gina	04/16/1982	USA
Lutz, Kellan	15/03/1985	USA

Output

Workflow



Input Text lines

Detect valid dialects for CSV table candidates (line-based)

Detect valid column boundaries for ASCII table candidates (table-based)

Output Parsing instructions

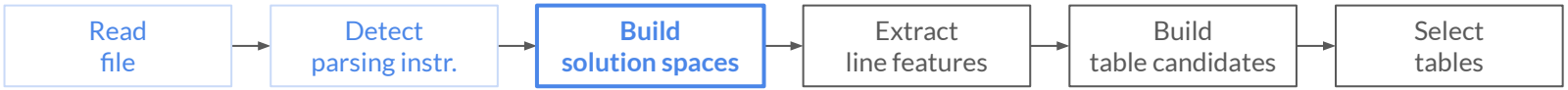
```
Willis,Bruce;05/19/1955;Germany
```

Detected parsing instructions

1. <, ε ε>
2. <; ε ε>
3. </ ε ε>

no valid column boundaries

Workflow



Input Text lines + parsing instructions

Apply parsing instructions to text lines

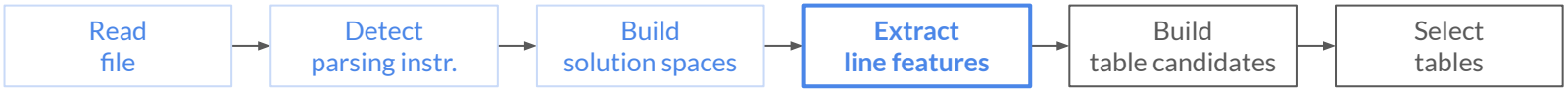
Output Interpretations

```
Willis,Bruce;05/19/1955;Germany
```

Interpretations

1	Willis	Bruce;05/19/1955;Germany	
2	Willis,Bruce	05/19/1955	Germany
3	Willis,Bruce;05	19	1955;Germany

Workflow



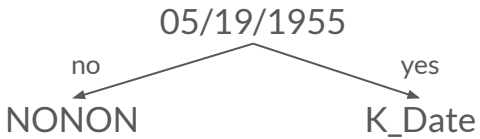
Input Interpretation

Describe cell contents by:

- Recognizing known data types (RegEx-based)
- Splitting into atomar components (String, Number, Other)

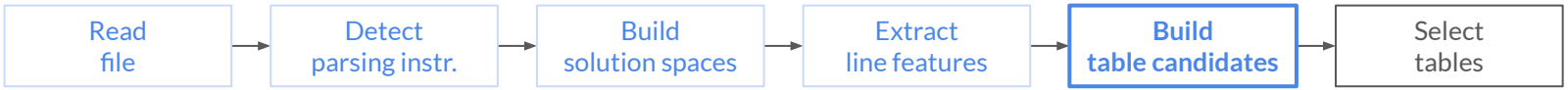
Output Enriched interpretation

Does a data type pattern match?



Number covers all kinds of numbers (int, signed number, floats, scientific)
Other matches everything that isn't a string or number component (similar to non-alphanumeric)

Workflow



Input Enriched interpretations

Group interpretations of subsequent lines by parsing instruction and column count

Divide bins into compatibility blocks using consistency measure

Output Table candidates

Workflow

Group interpretations of subsequent lines by parsing instruction an column count

```
Full name;DOB;POB
Willis,Bruce;05/19/1955;Germany
Carano,Gina;04/16/1982;USA
Lutz,Kellan;15/03/1985;USA
```

Bin 1 2 columns, CSV, [delimiter=,]		**Bin 2** 3 columns, CSV, [delimiter=;]			**Bin 3** 3 columns, CSV, [delimiter=/]		
		Full name	DOB	POB			
Willis	Bruce;05/19/1955;Germany	Willis,Bruce	05/19/1955	Germany	Willis,Bruce;05	19	1955;Germany
Carano	Gina;04/16/1982;USA	Carano,Gina	04/16/1982	USA	Carano,Gina;04	16	1982;USA
Lutz	Kellan;15/03/1985;USA	Lutz,Kellan	15/03/1985	USA	Lutz,Kellan;15	03	1985;USA

Bin 4: 2 columns, CSV, [delimiter=space] not shown for clarity reasons

Workflow

Bin 1 2 columns, CSV, [delimiter=,]		Bin 2 3 columns, CSV, [delimiter=;]			Bin 3 3 columns, CSV, [delimiter=/]		
		Full name	DOB	POB			
Willis	Bruce;05/19/1955;Germany	Willis,Bruce	05/19/1955	Germany	Willis,Bruce;05	19	1955;Germany
Carano	Gina;04/16/1982;USA	Carano,Gina	04/16/1982	USA	Carano,Gina;04	16	1982;USA
Lutz	Kellan;15/03/1985;USA	Lutz,Kellan	15/03/1985	USA	Lutz,Kellan;15	03	1985;USA

Bin 1 2 columns, CSV, [delimiter=,]		Bin 2 3 columns, CSV, [delimiter=;]			Bin 3 3 columns, CSV, [delimiter=/]		
		K_TEXT	K_TEXT	K_TEXT			
K_TEXT	SONNONONOS	K_TEXT	K_DATE	K_TEXT	SOSONN	N	SON
K_TEXT	SONNONONOS	K_TEXT	K_DATE	K_TEXT	SOSONN	N	SON
K_TEXT	SONONNONOS	K_TEXT	K_DATE	K_TEXT	SOSON	NN	SON

Bin 4: 2 columns, CSV, [delimiter=space] not shown for clarity reasons

Workflow

Divide bins into compatibility blocks using consistency measure

Bin 1 2 columns, CSV, [delimiter=,]		Bin 2 3 columns, CSV, [delimiter=;]			Bin 3 3 columns, CSV, [delimiter=/]		
		K_TEXT	K_TEXT	K_TEXT			
K_TEXT	SONNONONOS	K_TEXT	K_DATE	K_TEXT	SOSONN	N	SON
K_TEXT	SONNONONOS	K_TEXT	K_DATE	K_TEXT	SOSONN	N	SON
K_TEXT	SONONNONOS	K_TEXT	K_DATE	K_TEXT	SOSON	NN	SON

Bin 4: 2 columns, CSV, [delimiter=space] not shown for clarity reasons

Workflow

Bin 1 2 columns, CSV, [delimiter=,]	Bin 2 3 columns, CSV, [delimiter=;]	Bin 3 3 columns, CSV, [delimiter=/]

K_TEXT	SONNONONOS
K_TEXT	SONNONONOS

K_TEXT	SONONNONOS

K_TEXT	SONNONONOS
K_TEXT	SONNONONOS
K_TEXT	SONONNONOS

K_TEXT	K_TEXT	K_TEXT

K_TEXT	K_DATE	K_TEXT
K_TEXT	K_DATE	K_TEXT
K_TEXT	K_DATE	K_TEXT

K_TEXT	K_TEXT	K_TEXT
K_TEXT	K_DATE	K_TEXT
K_TEXT	K_DATE	K_TEXT
K_TEXT	K_DATE	K_TEXT

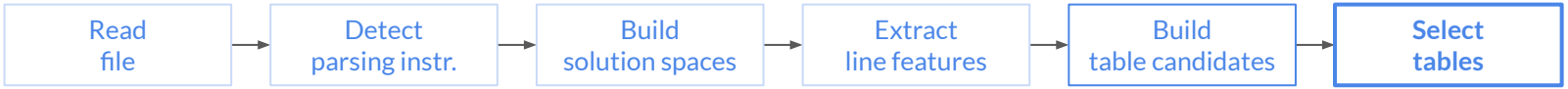
SOSONN	N	SON
SOSONN	N	SON

SOSON	NN	SON

SOSONN	N	SON
SOSONN	N	SON
SOSON	NN	SON

Bin 4: 2 columns, CSV, [delimiter=space] not shown for clarity reasons

Workflow



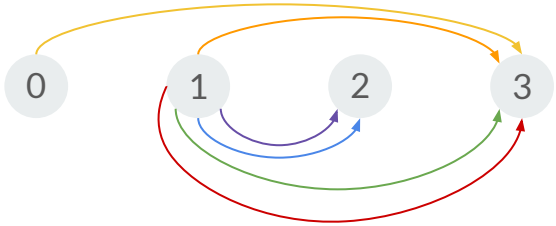
Input Table candidates

Map table candidates to multi-edge DAG

Vertexes	line indexes
Edges	table candidates
Distance	based on consistency and row count

Optimal solution = shortest path

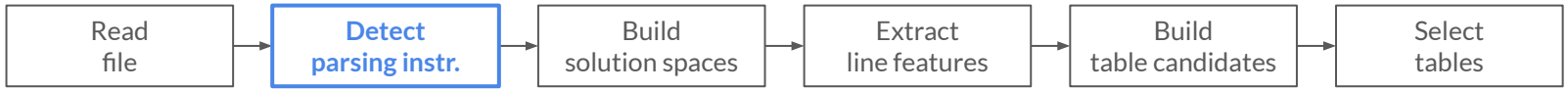
Output Selected tables



Distances

Table 1	-4.0	Table 4	-7.2
Table 2	-4.0	Table 5	-8.0
Table 3	-7.0	Table 6	-16.0

Insight: Dialect detection



Task Detect valid dialects for CSV table candidates (line-based)

Input Text line

Output Valid dialects

```
"Willis, Bruce";05/19/1955;"Germany"
```

1. <␣ ε ε>
2. <" ε ε>
3. <" , ε>
4. <, ε ε>
5. </ ε ε>
6. <; ε ε>
7. <; " ε>

We show only dialects having components represented as single character for readability reasons

Naive dialect detection

Assumption Dialect components can be only non-alphanumeric

Steps

1. Find all non-alphanumeric characters

```
"Willis, Bruce";05/19/1955;"Germany"
```

Non-alphanumeric character:

- double quote (")
- comma (,)
- space (␣)
- semi-colon (;)
- slash (/)

} 5

Naive dialect detection

Assumption Dialect components can be only non-alphanumeric

Steps

1. Find all non-alphanumeric characters
2. Build all dialect combinations

Remember

- Delimiter is required and must be different from quotation/ escape
- Quotation and escape are optional

```
"Willis, Bruce";05/19/1955;"Germany"
```

Number of dialect combinations

=	5 options
*	5 - 1 (*delimiter*) + 1 (*epsilon*)
*	5 - 1 (*delimiter*) + 1 (*epsilon*)
=	125 dialects

Naive dialect detection

Assumption Dialect components can be only non-alphanumeric

Steps

1. Find all non-alphanumeric characters
2. Build all dialect combinations
3. Test dialects and return valid ones

Remember

- Delimiter is required and must be different from quotation/ escape
- Quotation and escape are optional

```
"Willis, Bruce";05/19/1955;"Germany"
```

Number of dialect combinations

=	5 options
*	5 - 1 (*delimiter*) + 1 (*epsilon*)
*	5 - 1 (*delimiter*) + 1 (*epsilon*)
=	125 dialects

Only 7 dialects are valid

Smart dialect detection

Ideas

- Reduce number of tests by pruning impossible dialect configurations
- Abort parsing on first invalidity
- Re-use parsing results

Steps

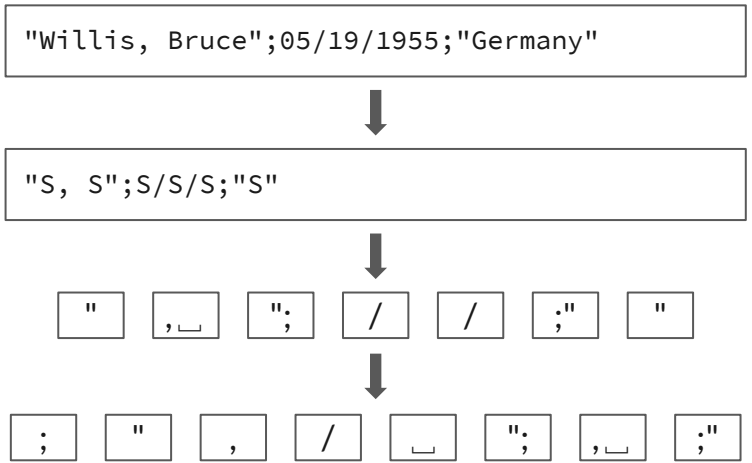
1. Detect delimiter sequences
2. Detect quotation and escape on the fly

Smart dialect detection

Goal Detect delimiter sequences

Steps

1. Replace consecutive alphanumeric characters with placeholder
2. Split by placeholder
3. Build substring combinations

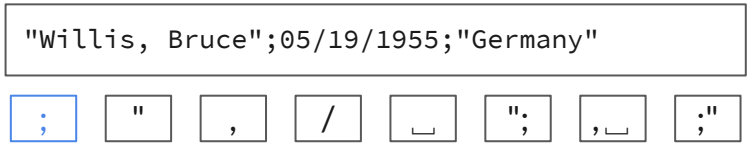


Smart dialect detection

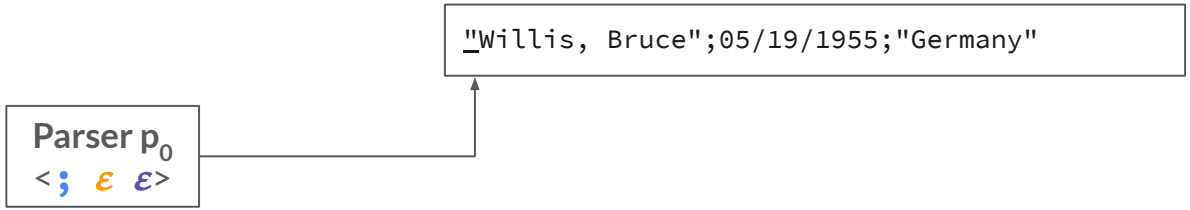
Goal Detect quotation and escape on the fly

Steps

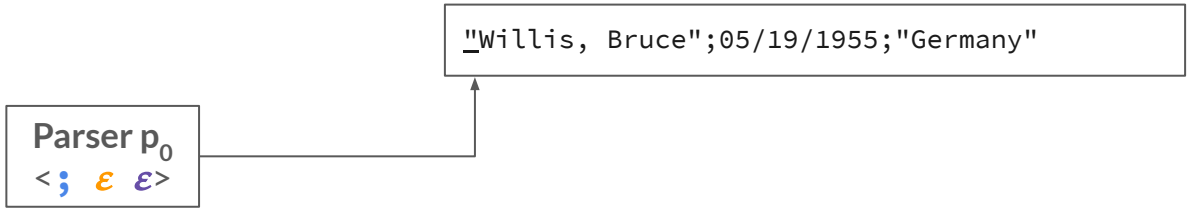
1. Start parser with dialect
   a. d_0=<; ε ε>
   b. d_1=<" ε ε>
   c. …



Smart dialect detection



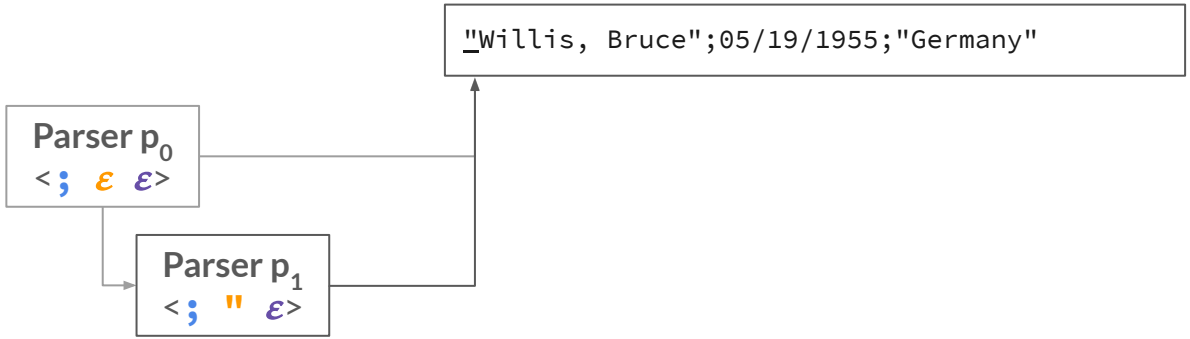
Smart dialect detection



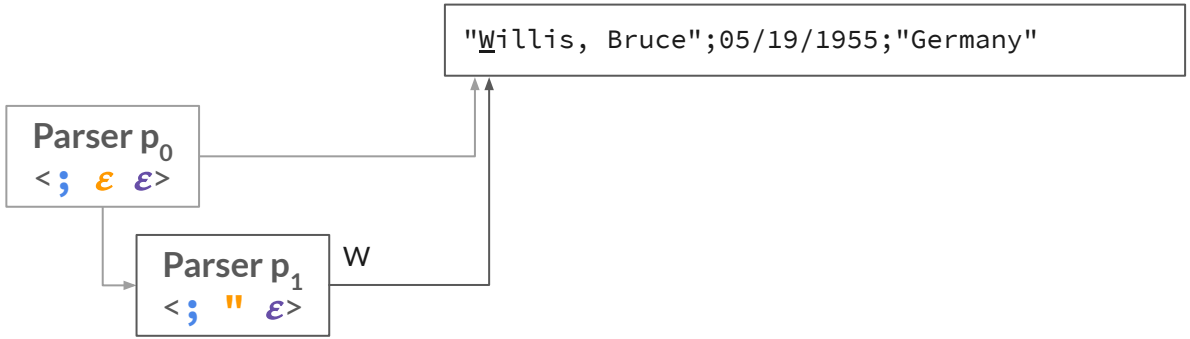
Start new parser, if:

- ✓ remaining substring does not start with dialect component
- ✓ character is non-alphanumeric
- ✓ parser does not exist already

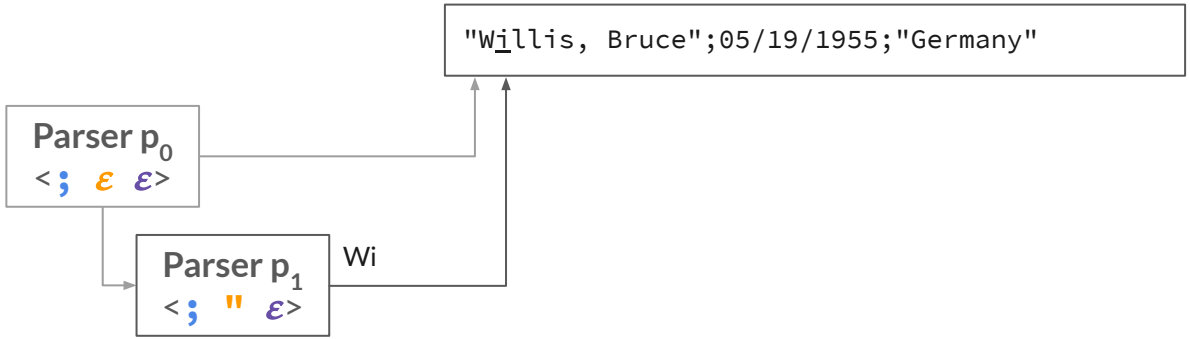
Smart dialect detection



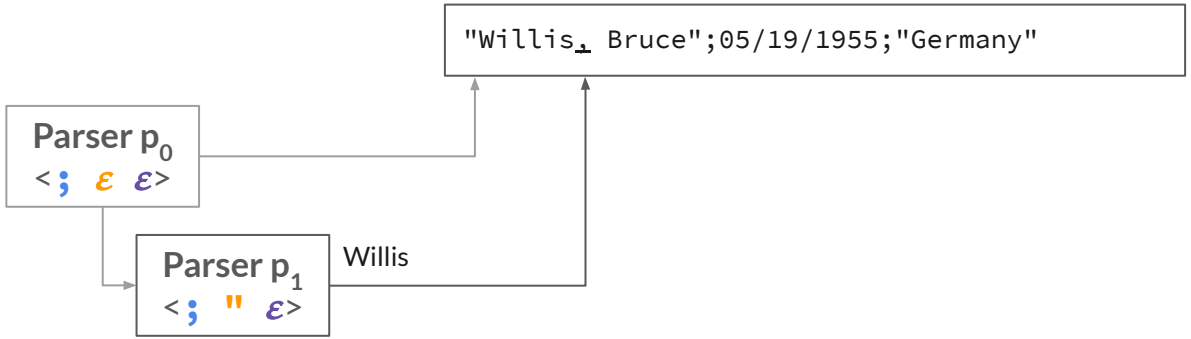
Smart dialect detection



Smart dialect detection



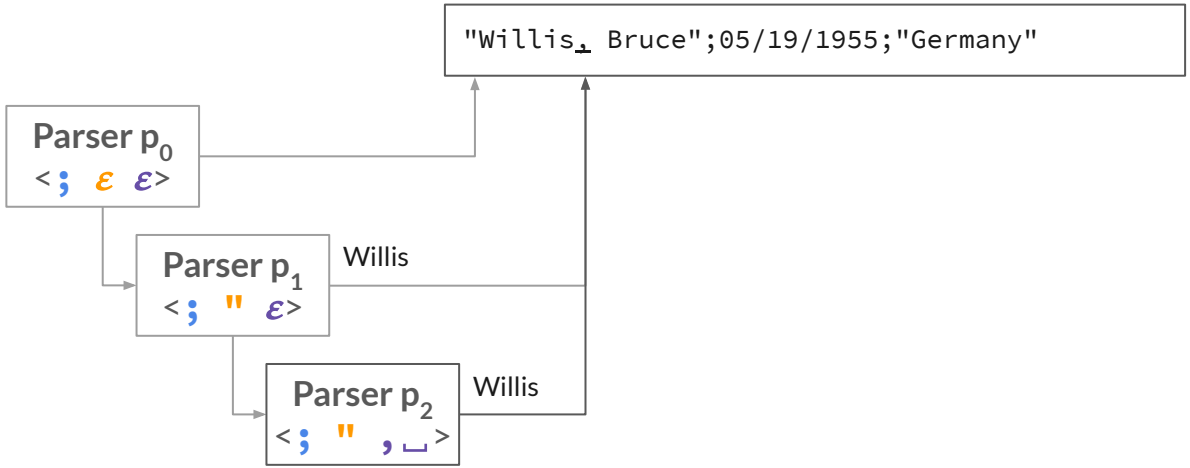
Smart dialect detection



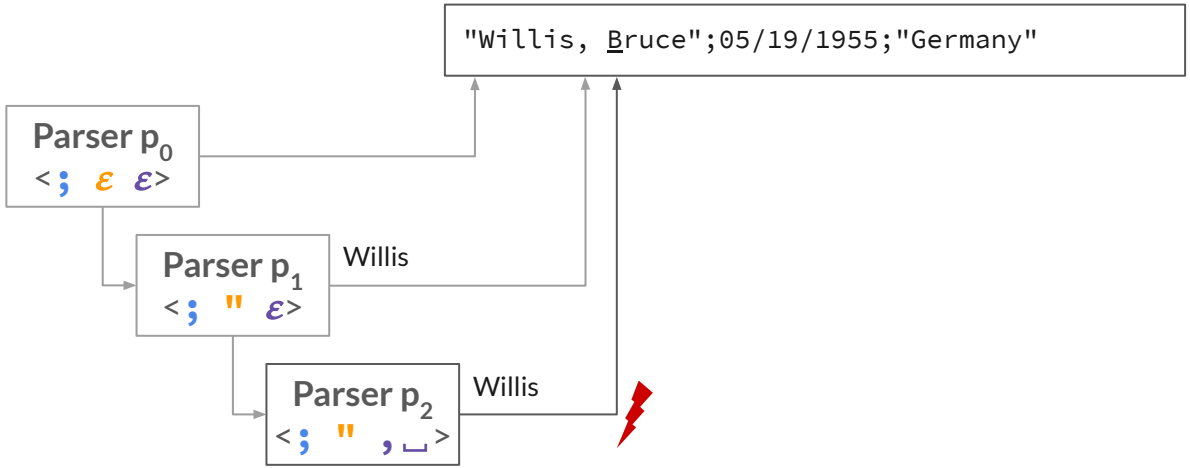
Start new parser, if:

- ✓ remaining substring does not start with dialect component
- ✓ character is non-alphanumeric
- ✓ parser does not exist already

Smart dialect detection



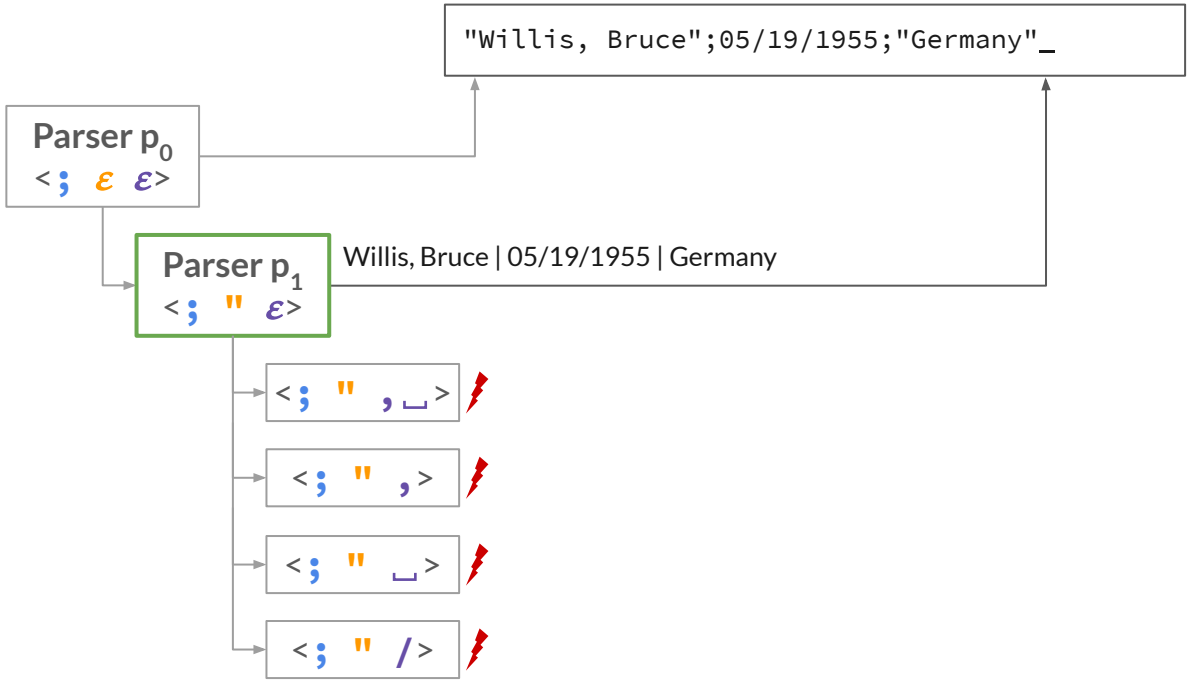
Smart dialect detection



Stop a parser, if either:

- x Escaping content
- Missing ending quotation
- Quotation not followed by delimiter or newline
- Trailing escape at line end

Smart dialect detection



A valid dialect is found, if:

✓ The parser reaches the end of line without invalidities

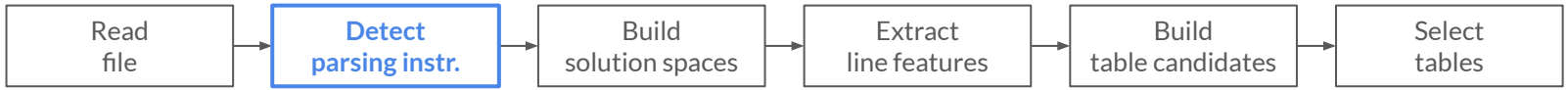
Smart dialect detection

```
"Willis, Bruce";05/19/1955;"Germany"
```

Valid dialects = parsers that processed the whole input

1. <␣ ε ε>
2. <" ε ε>
3. <" , ε>
4. <, ε ε>
5. </ ε ε>
6. <; ε ε>
7. <; " ε> ✔

Insight: Dialect detection



Naive

+ Easy to implement
− Tests all dialect combinations

Smart

- Uses custom parser to detect dialects on the fly
+ Aborts testing a configuration as early as possible
+ Caches parsed results

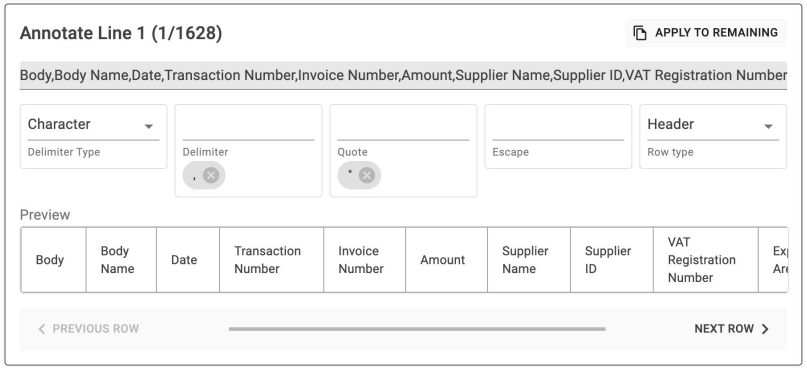
Evaluation

Experimental setup

Dataset 1,000 annotated files from Mendeley Data, UKdata, GitHub

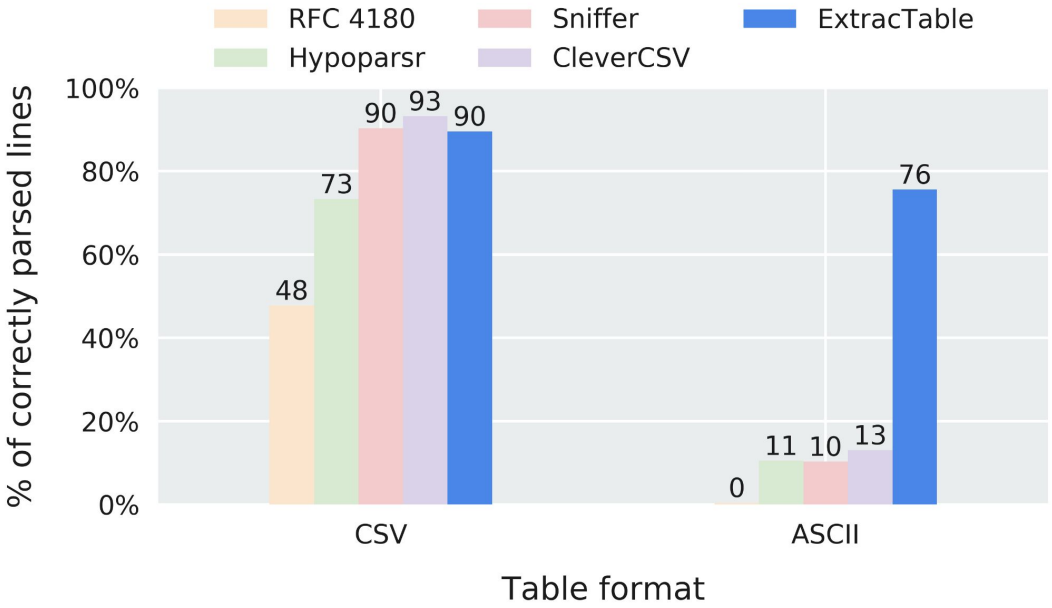
Annotations

- Parsing instructions (line-level)
- Row types
- Table ranges



Screenshot of line annotation

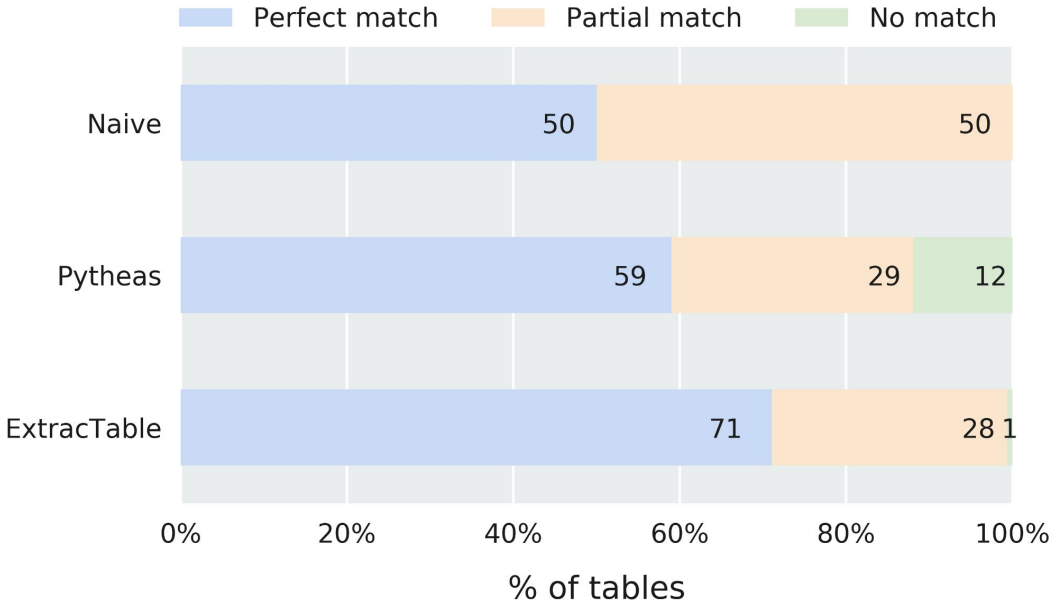
Line parsing accuracy



RFC 4180
Always returns dialect of RFC 4180 without reading the file

Sniffer
Python CSV package
Based on heuristics

Table range selection



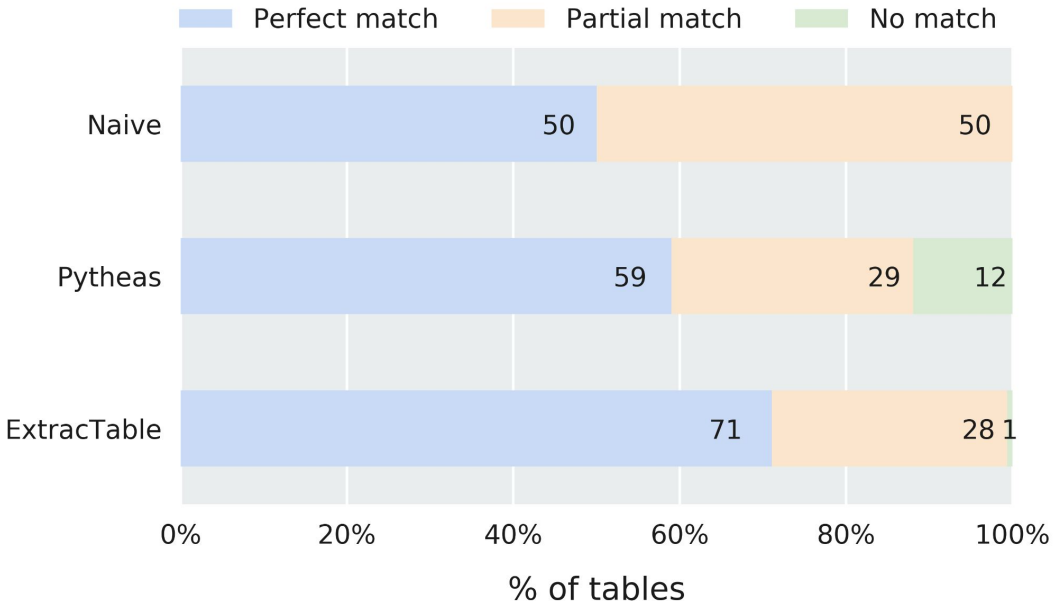
Naive
All lines of a file belong to a single table

Measure Jaccard

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

Perfect match J = 1.0
Partial match 0.0 < J < 1.0
No match J = 0.0

Table range selection

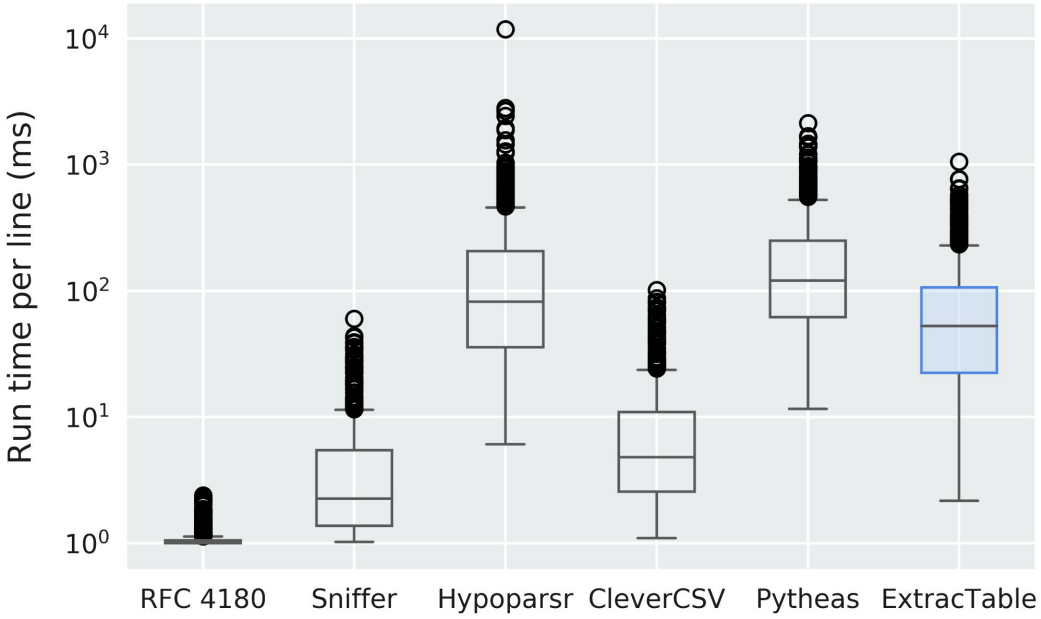


Eager match

Returned table has no corresponding table in ground truth

Naive:	0%
Pytheas:	6%
ExtracTable:	16%

Run time

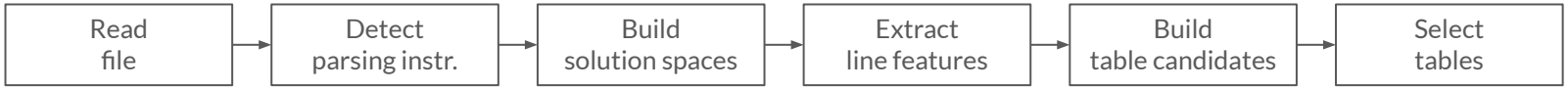


Time measurement
Linux system call `getrusage`

Data
Files that have been finished by all parsers

Conclusion

We want to develop an algorithm that extracts tables from plain text files to decrease the time spend on data wrangling.



+ ExtracTable covers more tables than existing work
+ ExtracTable outperforms/ works similar to existing approaches
+ Data scientists spend less time on data wrangling

– Depending on the context, data scientists need to select tables manually

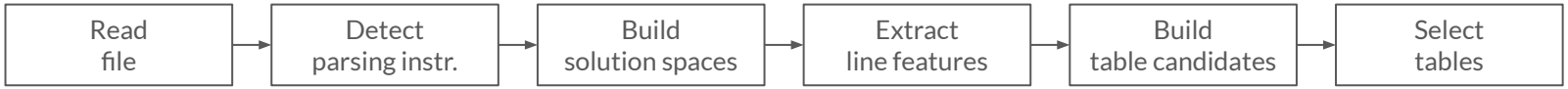
Future Work

- Support multi-row cells
- Support spanning rows and columns
- Support different newline-characters

Data Parsing → Data Integration → Data Transformation → ...[8]

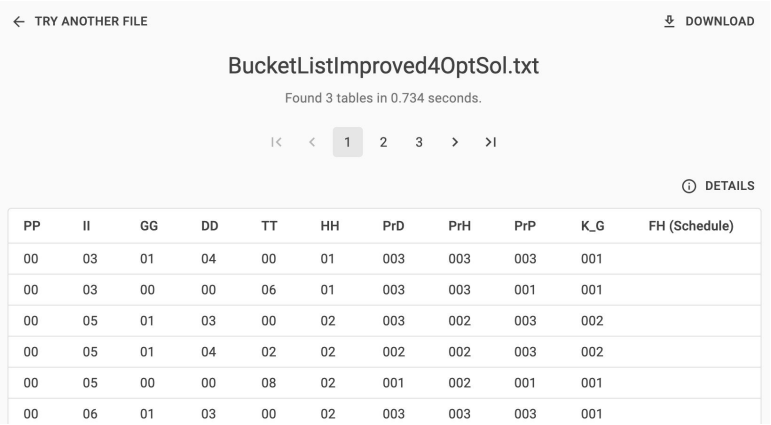
[8] Nazabal, Alfredo, et al. (2020) "Data engineering for data analytics: a classification of the issues, and case studies."

Overview



Key features

- CSV and ASCII tables
- CSV dialects deviating more from RFC 4180
- Multi-table files
- Files containing surrounding text



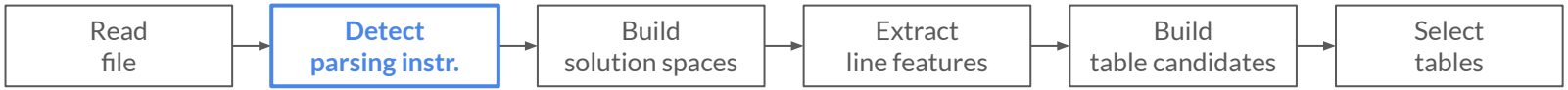
PP	II	GG	DD	TT	HH	PrD	PrH	PrP	K_G	FH (Schedule)
00	03	01	04	00	01	003	003	003	001	
00	03	00	00	06	01	003	003	001	001	
00	05	01	03	00	02	003	002	003	002	
00	05	01	04	02	02	002	002	003	002	
00	05	00	00	08	02	001	002	001	001	
00	06	01	03	00	02	003	003	003	001	

Demo web app
http://172.20.11.15:3000

BACKUP SLIDES

Workflow

Detect column boundaries

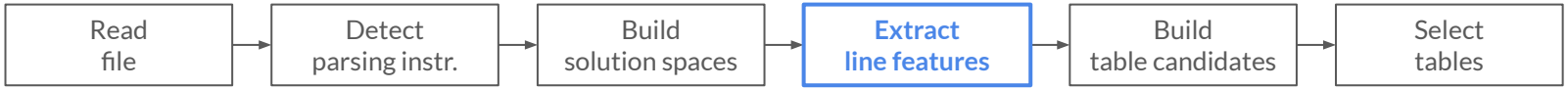


Main idea Detect vertical lines of whitespace in subsequent lines

```
1: Topic                                                 Student             Supervision
2: Extracting Plain Tables from Text                     Leonardo Hübscher   Felix Naumann, Lan Jiang
3: Distributed Duplicate Detection on Streaming-Data     Jakob Köhler        Thorsten Papenbrock
4: Multi-Aspect Embeddings for Fiction Novels            Lasse Kohlmeyer     Ralf Krestel, Tim Repke
5: Generating Rap Lyrics with Flow and Rhythm            Noel Danz           Ralf Krestel, Tim Repke
```

```
1: 00000111111111111111111111111111111111111111111111111110000000111111111111000000000001111111111111
2: 00000000001000001000000100001000011111111111111111111110000000010000000011000001000000001000100000
3: 00000000000100000000010000000001001000000000000011110000010000001111111100000000100000000000111111
4: 00000000000010000000000100010000000100000011111111111000001000000000111110000100000000100010000001
5: 00000000001000100000010000100001000100000011111111110000100001111111111000010000000010001000001
```

Extract line features



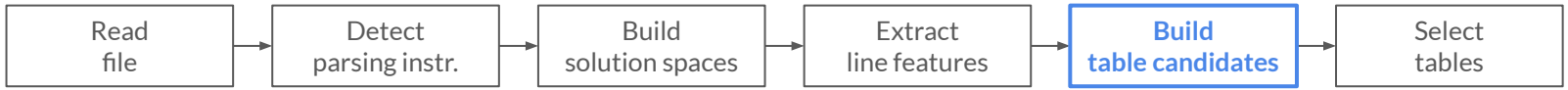
Main idea Disjoint data type RegEx

Collected from

- Data-driven
- RegEx-Libraries
- Custom

Name	Description
Boolean	true or false (ignoring case)
Brackets	Anything in-between a pair of round, box, curly, or angle brackets
Currency	Number preceeded or subseeded by a currency character
Date	Date and/ or time in various formats Date d/m/yy, dd/mm/yyyy, m/d/y,mm/dd/yyyy, yy-m-d, yyyy-mm-dd Date-separator can be one of space, slash, dot, minus Time: hh:mm:ss Time-separator can be any non-word character Partially taken from RegexBuddy
Domain name	Taken from RegexBuddy
E-Mail address	Taken from RegexBuddy
Empty	Empty string or values that represent a missing value.
File path	Matching absolute and relative paths using / or \ as delimiter
Hash	A string consisting of upper or lower case characters, digits, and underscore At least one number and English letter required.
IP address	IP address in IPv4 format - taken from RegexBuddy
Number	Signed, unsigned floats and integers. Supports scientific notation
Percentage	Number followed by the percentage sign
String	Sequence of English letters (ignoring case) - minimum length: 1
Text	Multiple strings separated by ␣.:&%?!—'"/()[]
URL	Created by diegoperini[1]

Consistency score



Main ideas

- Use homogeneity function from existing work
- Distinguish between data type consistency and value uniformity

Table consistency more than log2(c) columns are consistent with c = table column count

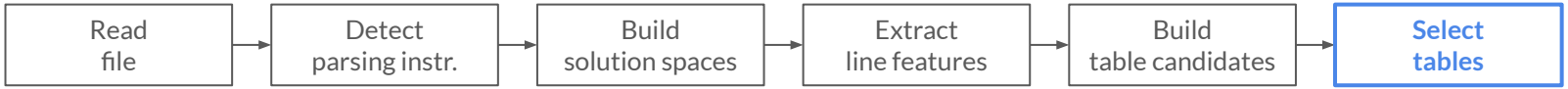
Column consistency pattern homogeneity exceeds threshold

H_c is the **homogeneity** of column c, the sum of squares of the proportions of each data type $Type$ present in that column:

$$H_c = \sum_{Type} \left(\frac{|i \in R : c_i \in Type|}{|R|} \right)^2 \quad (2)$$

"If 75% of values in a column are numbers and 25% are dates, then the column's homogeneity is $(0.75)^2 + (0.25)^2 = 0.625$"

Consistency score



$$
\begin{aligned}
dist : tc \rightarrow & - score(data(C_{tc}, h_{tc})) * (m_{tc} - h_{tc})^2 \\
& - score(header(C_{tc}, h_{tc})) * (m_{tc}^2 - (m_{tc} - h_{tc})^2) \\
& - 0.0001 * \mathrm{sgn}(h_{tc})
\end{aligned}
$$

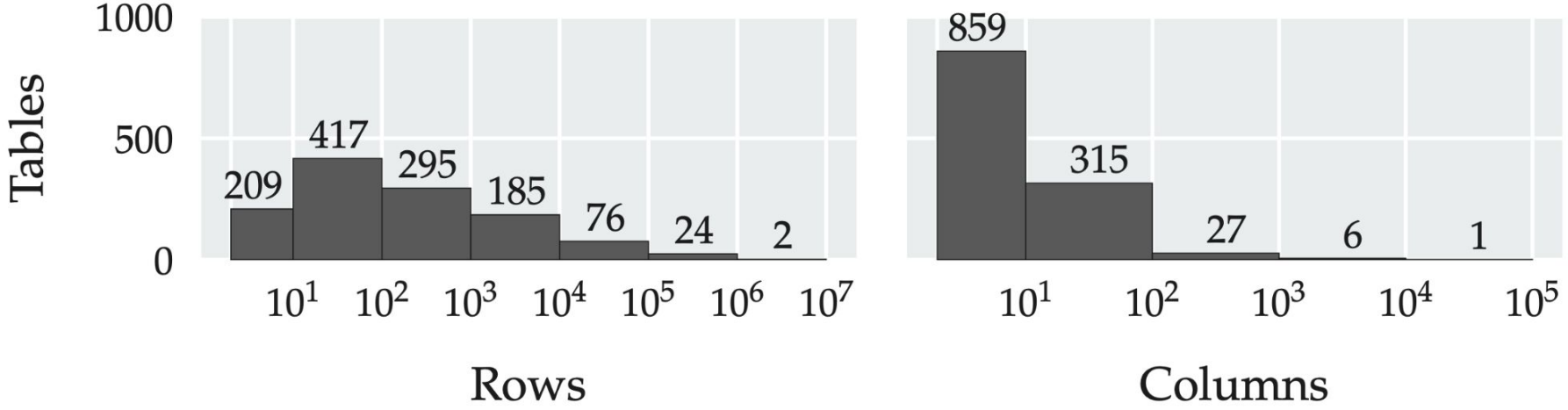
Main ideas

- favor bigger tables over smaller tables
- favor more consistent tables
- favor tables with header

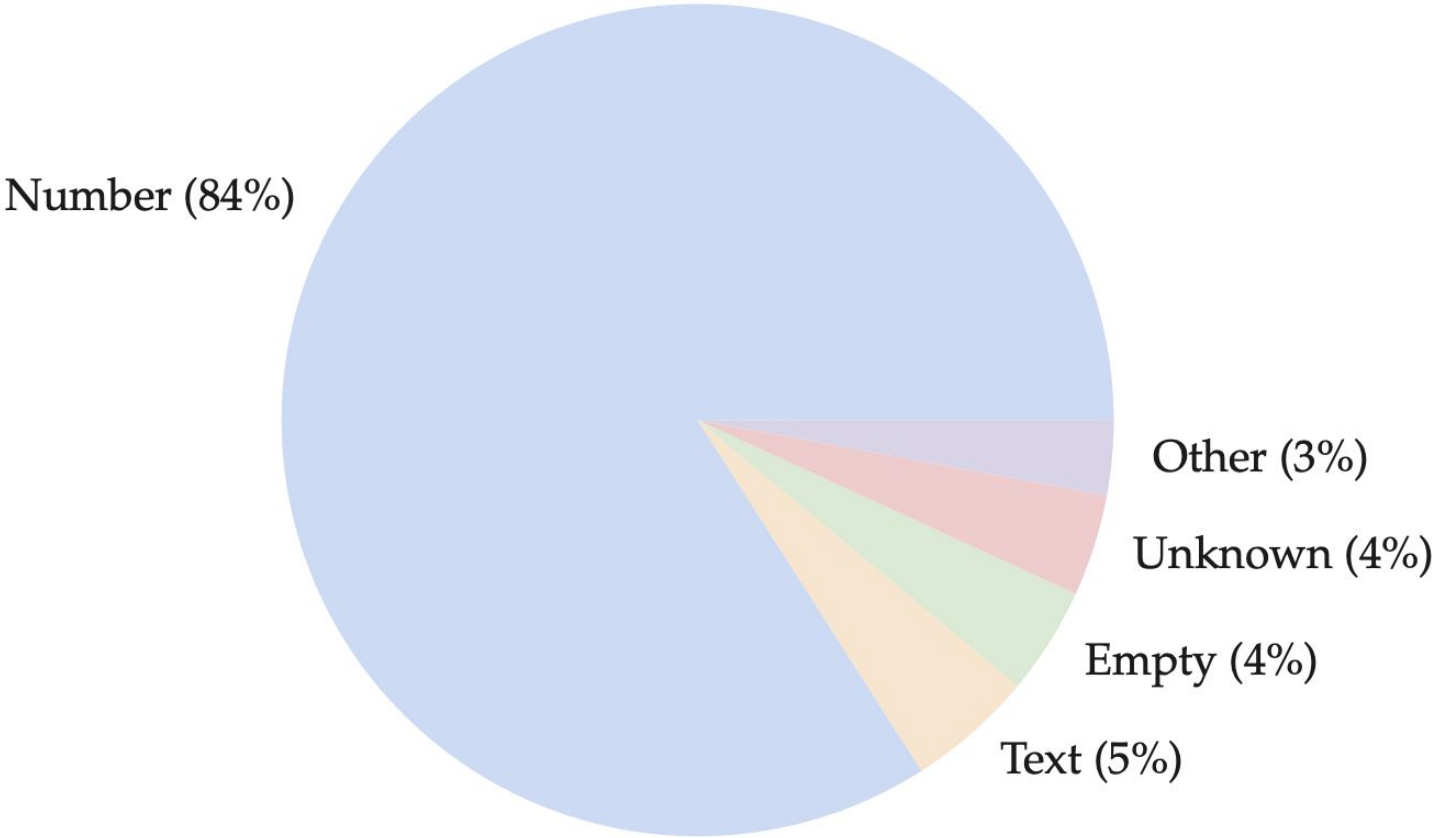
Tie-breaking

- ratio of recognized cells (higher)
- number of pattern components per field (lower)
- column count (higher)

Statistics

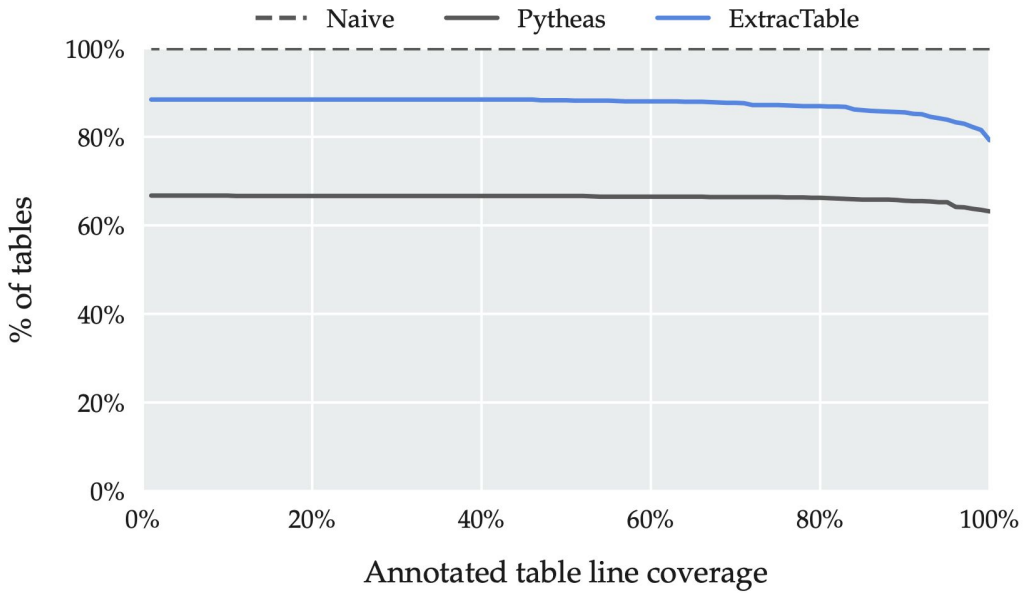


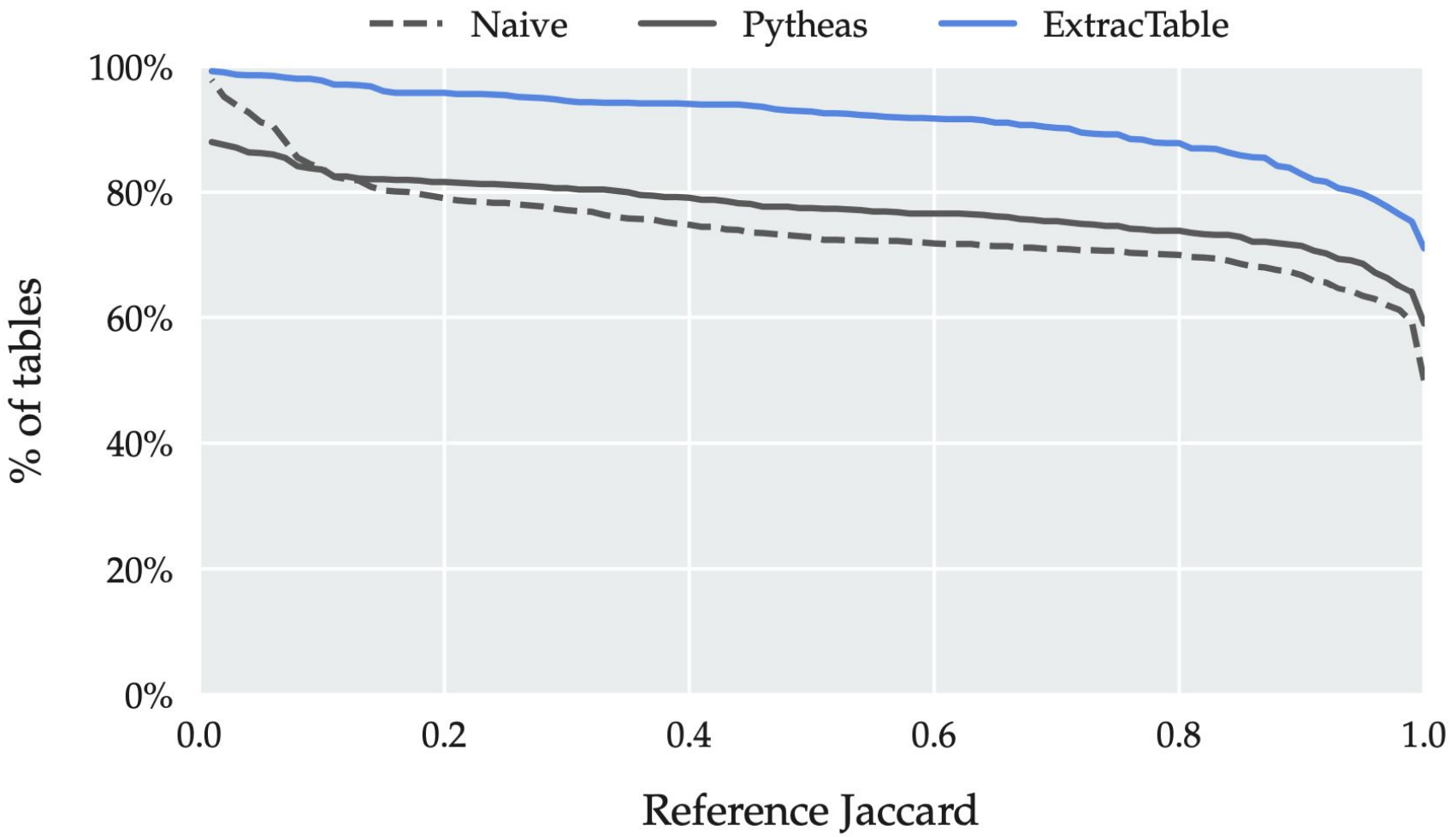
UKdata (%)	GitHub (%)	Mendeley Data (%)
95.4	68.0	22.9


Number (84%)
Other (3%)
Unknown (4%)
Empty (4%)
Text (5%)

Evaluation

	Simple-single		Complex-single		Complex-multi	
	F1	bal. Acc.	F1	bal. Acc.	F1	bal. Acc.
Naive	0.999	0.500	0.927	0.500	0.945	0.500
Pytheas	**1.000**	0.944	0.949	0.909	0.828	0.637
ExtracTable	0.999	**0.999**	**0.997**	**0.987**	**0.973**	**0.796**




Naive
Pytheas
ExtracTable
100%
80%
60%
40%
20%
0%
% of tables
0.0
0.2
0.4
0.6
0.8
1.0
Reference Jaccard