Extracting N-ary Facts from Wikipedia Table Clusters

Benno Kruit
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
kruit@cwi.nl

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

Jacopo Urbani
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
jacopo@cs.vu.nl

ABSTRACT

Tables in Wikipedia articles contain a wealth of knowledge that would be useful for many applications if it were structured in a more coherent, queryable form. An important problem is that many of such tables contain the same type of knowledge, but have different layouts and/or schemata. Moreover, some tables refer to entities that we can link to Knowledge Bases (KBs), while others do not. Finally, some tables express entity-attribute relations, while others contain more complex n-ary relations. We propose a novel knowledge extraction technique that tackles these problems. Our method first transforms and clusters similar tables into fewer unified ones to overcome the problem of table diversity. Then, the unified tables are linked to the KB so that knowledge about popular entities propagates to the unpopular ones. Finally, our method applies a technique that relies on functional dependencies to judiciously interpret the table and extract n-ary relations. Our experiments over 1.5M Wikipedia tables show that our clustering can group many semantically similar tables. This leads to the extraction of many novel n-ary relations.

ACM Reference Format:
Benno Kruit, Peter Boncz, and Jacopo Urbani. 2020. Extracting N-ary Facts from Wikipedia Table Clusters . In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3340531.3412027

1 INTRODUCTION

Motivation. Tables on the Web represent an important source of knowledge that can be used to enhance many tasks. In particular, tables in Wikipedia articles express many interesting relations that can improve tasks like web search [33], or entity disambiguation [30]. Currently, the largest repositories of knowledge on the Web are in the form of graph-like Knowledge Bases. Among the most popular KBs are the ones that were constructed from Wikipedia, in particular considering the content of infoboxes. Tables, however, are often used to state knowledge that is complementary to the knowledge contained in infoboxes. This makes tables an excellent source of additional knowledge to extend the coverage of current Wikipedia-based KBs, like Wikidata [28] or DBPedia [13].

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3412027

Problem. Unfortunately, Wikipedia tables are created to present structured knowledge for human readers, but not necessarily for machines. To process this knowledge automatically, the table must first be transformed into a coherent structure that is machine-readable. One way to reach this goal is to integrate the content of the tables into a Knowledge Base (KB), but this is challenging due to the diversity of layouts, schemata, and symbols that human contributors use to display knowledge in tabular format. For example, relations can be expressed by many different schemas, their meaning might depend on background knowledge that is expressed in the table context, columns may contain multiple attributes in list form, and many cell values are homonyms or synonyms. Secondly, many tables express knowledge about long-tail entities that are not present in KBs. Moreover, many tables on Wikipedia express n-ary relations which are challenging to interpret. In these cases, there is not a single key column that contains the entities of which the other columns express attributes, which is something that is assumed to exist by state-of-the-art table extraction systems [11, 24, 34].

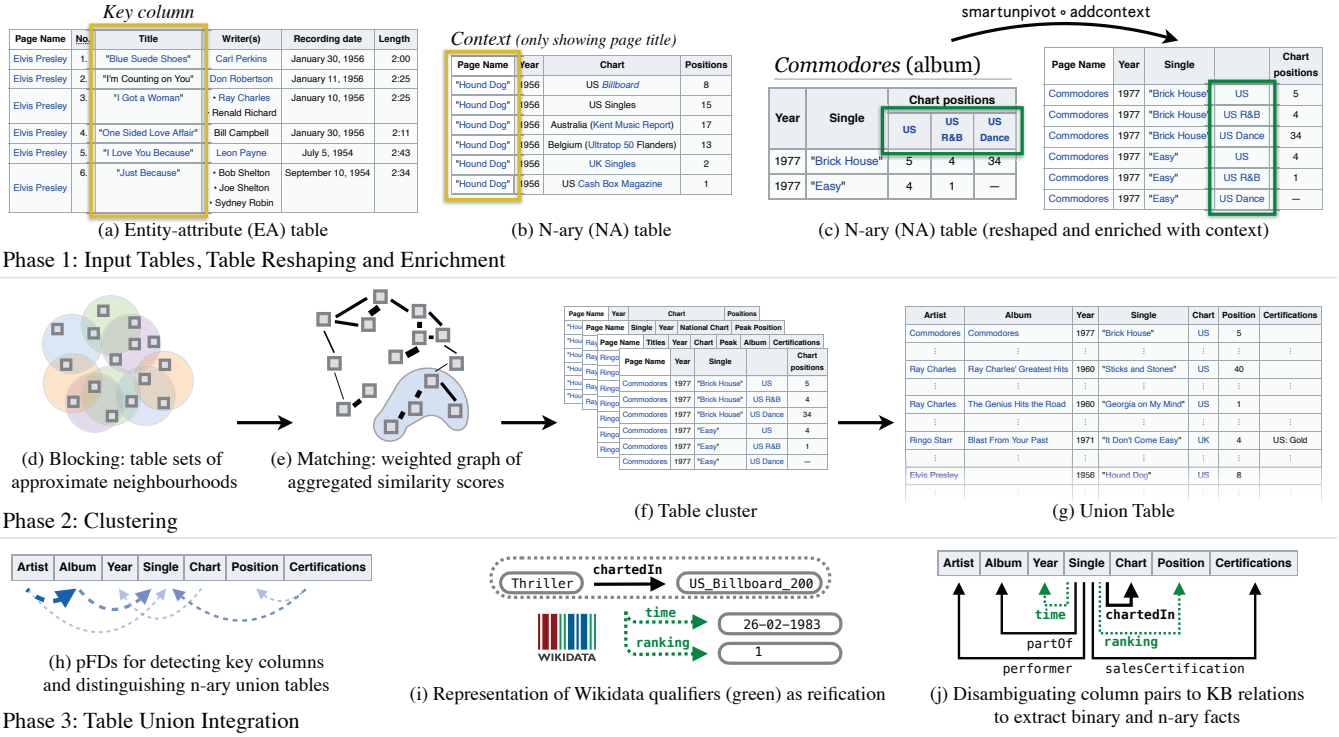
Contribution. While there are existing works that addresses these problems in isolation (we cover these in Section 5), in this paper we propose a novel method that solves them conjointly. Our method consists of a sequence of *three* main operations. First, it efficiently combines a diverse set of table corpus statistics to perform holistic schema normalization on tables that have different layouts. Then, the tables are clustered together in fewer larger tables. Finally, our method extracts both entity-attribute and n-ary facts from the clustered tables so that new facts can be added to the KB.

The novelty of our approach hinges on three components:

First, we present several techniques to normalize tables which would otherwise be ignored by current knowledge extraction approaches. These techniques rely on some heuristics which transform the tables by removing rows that span all columns, unpivoting column headers, and adding extra contextual columns.

Second, we introduce features for clustering together many similar tables into a unified collection so that knowledge about entities in one table can propagate to other tables. These features measure the potential alignment of columns using Jaccard similarities, cell embeddings, and semantic types. To scale our clustering to 1.5M tables in Wikipedia, we use set- and embedding-based approximate neighbor search to reduce the similarity search space.

Third, we describe a new method, based on probabilistic functional dependencies [29], to distinguish entity-attribute tables from the ones that describe n-ary relations and extract knowledge from both types. We show that using functional dependencies instead of simple heuristics (like picking the leftmost column with unique values [11]), as current methods do, returns a better accuracy that is beneficial to improve the downstream fact extraction. Moreover, to the best of our knowledge, ours is the first method that can extract n-ary facts from tables to a KB.



(a) Entity-attribute (EA) table (b) N-ary (NA) table (c) N-ary (NA) table (reshaped and enriched with context)

(d) Blocking: table sets of approximate neighbourhoods (e) Matching: weighted graph of aggregated similarity scores (f) Table cluster (g) Union Table

(h) pFDs for detecting key columns and distinguishing n-ary union tables (i) Representation of Wikidata qualifiers (green) as reification (j) Disambiguating column pairs to KB relations to extract binary and n-ary facts

Figure 1: Schematic overview of our pipeline. In Phase 1 (a-c), we process the set of all Wikipedia tables to clean up editorial structures (Section 3.1). In Phase 2 (d-g), we cluster them to form larger union tables (Section 3.2). In Phase 3 (h-j), we integrate them with Wikidata and extract binary and n-ary facts (Section 3.3)

We evaluate our approach on a large sample of Wikipedia tables using Wikidata [28], one of the most popular KBs, as reference KB. We report on the performance of different components of the pipeline separately, and compare to a set of strong baselines. Additionally, we extended our evaluation to a set of 1.5M tables extracted from Wikipedia, to evaluate the scalability. In this case, our system managed to extract 29.5M facts which comprise 15.8M binary facts and 6.9M more complex n-ary facts. A large percentage (approx. 77%) was novel, i.e., facts not yet in Wikidata.

Our code, annotations, and extracted data is freely available[1].

2 BACKGROUND

We start with a short recap of some well-known notions on KBs and tables, and introduce some notation used throughout the paper.

KBs. A KB $\mathcal{K}$ is a repository that contains factual statements about a set of entities $\mathcal{E}$, literals $\mathcal{L}$ and relations $\mathcal{R}$. In this work, we consider Wikidata [28], one of the most popular KBs, as $\mathcal{K}$. In Wikidata (and in other similar KBs, e.g., DBPedia [13]), the factual statements are encoded with triples of the form $\langle e, r, f \rangle$ where $e \in \mathcal{E}$, $f \in \mathcal{E} \cup \mathcal{L}$ and $r \in \mathcal{R}$. Typically, the triples express relations between entities, e.g., $\langle$`Sadiq_Khan`, `majorOf`, `London`$\rangle$, or property-value attributes, e.g., $\langle$`London`, `hasPopulation`, `8.9M`$\rangle$. Given the triple $\langle e, r, f \rangle$, we say that the pair $\langle r, f \rangle$ is an *attribute* of e, r is a *attribute relation*, and f is an *attribute value*.

A *n-ary fact* is a factual statement with n arguments. While a *binary fact* (i.e., 2-ary fact) can be naturally expressed with a triple, facts where $n > 2$ require multiple triples. Wikidata makes use of *qualifiers* to express such complex factual statements. A *qualifier* is a subordinate property-value pair assigned to a triple that annotates the fact with additional information [28]. For instance, the fact $\langle$`Sadiq_Khan`, `majorOf`, `London`$\rangle$ is annotated with the qualifier $\langle$`startTime`, "`05/09/2016`"$\rangle$. Qualifiers are represented as triples using a well-known style of reification [21], in which every fact $\langle e, p, f \rangle$ is mapped to a fresh entity $q_{e,p,f} \in \mathcal{E}$ and every qualifier $\langle r, g \rangle$ is mapped to a triple $\langle q_{e,p,f}, r, g \rangle$.

Tables. We model our collection of tables as a corpus $\mathcal{T}$ of tables, which we have extracted from the HTML of Wikipedia articles. For a given table T, we denote with $T[i][j]$ the *cell* of table T at the i^{th} row and j^{th} column. Every cell c is associated to a *cell value* $\mathrm{val}(c)$ which represents the content of a cell and is either a string or `NULL`. In the last case, we say that the cell is empty and denote it with the symbol $\boxed{\emptyset}$. Additionally, a cell may contain some links to entity-related Wikipedia pages. In Wikidata, every Wikipedia page is mapped to an entity. We denote with $\mathcal{E}_W \subseteq \mathcal{E}$ the set of such entities, and write $\mathrm{links}(c) \subseteq \mathcal{E}_W$ to refer to the entities pointed by the links in c. Some cells are marked with a *span* that extends multiple columns. We write $\mathrm{span}(c) = [i, j]$ when cell c spans columns i to j (included) in its row. If c at row k has span $[i, j]$ then $\mathrm{val}(T[k][i]) = \ldots = \mathrm{val}(T[k][j])$. However, the opposite does not hold, namely two adjacent cells can have the same value but

[1] See https://github.com/karmaresearch/takco

without an extended span. Finally, we write $\text{span}(c) \subset \text{span}(d)$ if $\text{span}(c) = [i, j]$, $\text{span}(d) = [k, l]$, $i \geq l$, $j \leq l$, and $j - i < l - k$.

We denote with $\text{cols}(T)$ and $\text{rows}(T)$ the list of all columns and rows in T respectively. We represent each column as a tuple of $|\text{rows}(T)|$ cells and each row as a tuple of $|\text{cols}(T)|$ cells. We distinguish header rows from body rows based on the table's HTML. We write $\text{head}(T)$ to refer to a list of rows in T marked as headers, while $\text{body}(T)$ refers to the remaining rows. Abusing notation, we write $\text{cols}(\text{body}(T))$ to refer to the list of columns of the table's body. We view T, $\text{cols}(T)$, $\text{rows}(T)$, $\text{body}(T)$, $\text{cols}(\text{body}(T))$, and $\text{head}(T)$ as sets if the order of the tuples does not matter, otherwise we use the suffix $[i]$ to refer to the i^{th} element in the collection (e.g., $\text{cols}(T)[1]$ is the first column of T).

Tuples are denoted with delimiters $\langle\rangle$. We introduce two auxiliary functions to operate on tuples. Function $\text{append}(a, B)$ returns a tuple where element a is appended to tuple B. Function $\cdot$, written $A \cdot B$, returns a tuple where tuple A is concatenated to tuple B. A *union table* is a table created by concatenating the body of multiple tables. In a union table, columns may be aligned into single ones or not. In the second case, empty cells are used to fill the gaps [17].

In this paper, we use the relational model to specify some operations on tables. This model views a table schema as a *relation* R with *attributes* $A_1, \ldots, A_m$, denoted as $R(A_1, \ldots, A_m)$, and calls a table with such a schema an *instance* of R. Attributes in the relational model are mapped to header cells in the table. Thus, they are different than attributes used in KBs. In the former, attributes (informally) map to the header names of the table while in the latter they are property-value pairs of entities.

We make a distinction between *Entity-Attribute (EA)* and *N-Ary (NA)* tables. EA tables contain one column with the names of entities, which we call *key column*, and every row expresses attributes of that entity in the other columns [33]. Therefore, one row can be translated into a set of attributes of an entity, and represented in $\mathcal{K}$ with triples of the form $\langle \texttt{entity}, \texttt{attribute\_relation}, \texttt{attribute\_value} \rangle$.

NA tables lack a key column and each row expresses one n-ary fact and typically $n > 2$. In this case, we say that the table expresses a *n-ary relation*. It has been shown that NA tables make up a significant portion of tables on the Web [15]. Table (a) in Figure 1 is an example of a EA table while Table (b) reports a NA table.

To improve the extraction coverage, we consider the content that we can extract from the page that contains the table. For instance, we consider the title of the article or the table caption. We represent contextual information as strings. To distinguish the various types of contextual information, we use pairs of the form $\langle X, Y \rangle$ where X is the type of information and Y is the content. For instance, the pair ⟨"Page Name", "Elvis Presley"⟩ is an example of contextual information for Table (a) in Figure 1. We refer to the set of all contextual tuples associated to table T as $\text{context}(T)$.

Table unpivoting. Tables can be categorized either as *wide* or *narrow*, depending on how they express information [31]. If they are wide (i.e., have a wide layout), then it is more likely that single attribute values are represented with dedicated columns. For instance, the header cell "US" in the third column of Table (c) in Figure 1 expresses the qualifier $\langle \texttt{chartedIn}, \texttt{US\_Billboard\_200} \rangle$ for all the triples extracted from the cells below it. For our purposes, it is more convenient if tables are in a narrow shape, i.e., header cells express attribute properties rather than attribute values. Converting a table from a wide shape to a narrow shape is known as *unpivoting*.

Wyss and Robertson [32] provide a formal definition of unpivoting, which we outline below for self-containment. This definition uses two additional relational algebra operators: δ (metadata demotion) and Δ (column deference). Given a relation schema $R(A_1, \ldots, A_m)$, let r be an instance of R with n rows and let $\texttt{y}$ be an attribute that is not in R. Then $\delta_{\texttt{y}}(r)$ appends every $A_1, \ldots, A_m$ to each row in r, returning a new instance with $|r| \times n$ rows and schema $R(A_1, \ldots, A_m, \texttt{y})$. The operator Δ is used to further process the relation. Given a relation $R(A_1, \ldots, A_m, B)$, let r be an instance of R and $\texttt{z}$ a column name that is not in R. Then, $\Delta^{\texttt{z}}_{B}(r)$ searches if an element of B at row i equals to column name at position j, and if this occurs then it copies the cell value at row i and column j in a new column with name $\texttt{z}$.

These two operators, in combination with the standard relational operators projection (Π) and selection (σ), can be used to formally define the operation of unpivoting. Let $R(A_1, \ldots, A_i, B_{i+1}, \ldots, B_m)$ be a relation, r be an instance of R, and $B_{i+1}, \ldots, B_m$ be the attributes to unpivot. Then, unpivoting r can be expressed as:

$$\text{UNPIVOT}^{\texttt{y}\rightarrow\texttt{z}}_{A_1,\ldots,A_i}(r) \coloneqq \Pi_{A_1,\ldots,A_i,\texttt{y},\texttt{z}}(\sigma_{\texttt{y}=A_1 \wedge \ldots \wedge \texttt{y}\neq A_i}(\Delta^{\texttt{z}}_{\texttt{y}}(\delta_{\texttt{y}}(r))))$$

where $\texttt{y}$ is the name of the column with the unpivoted schema and $\texttt{z}$ is the column name with the values of the unpivoted columns.

Functional Dependencies. To distinguish EA tables from NA tables, we make use of probabilistic functional dependencies (pFDs), first introduced by Wang et al. [29]. Let X, Y be two attributes of a relation R, and r be an instance of R. Then, the pFD $X \rightarrow^{p} Y$ indicates that two tuples in r that share the same value for X also share the same value for Y with probability p. To compute pFDs, we use the algorithm perTuple [29], which returns pFDs using probabilities computed on r.

3 OUR APPROACH

Our goal is to extract clean, unified, and linked n-ary facts from a large set of tables to enrich a KB with new knowledge. For example, we would like to extract from Table (c) in Figure 1 the n-ary fact that the song "Brick House" charted in the "US Billboard 200" chart at position 5 in 1977. To represent this fact, we use three triples: The triple $t = \langle \texttt{Brick\_House}, \texttt{chartedIn}, \texttt{US\_Billboard\_200} \rangle$ and the triples $\langle \texttt{q}_\texttt{t}, \texttt{pointInTime}, 1977 \rangle$ and $\langle \texttt{q}_\texttt{t}, \texttt{ranking}, 5 \rangle$, where $\texttt{q}_\texttt{t}$ is a fresh entity used to represent the qualifiers mapped to triple t.

We use Wikidata [28] as target KB because of its popularity and large coverage, and focus on Wikipedia tables since they contain a large amount of interesting factual information related to Wikidata entities. Our method, which is graphically depicted in Figure 1, can be viewed as a pipeline of three main operations: *Table Reshaping and Enrichment* (Section 3.1), *Clustering* (Section 3.2), and *KB Integration* (Section 3.3), each discussed below.

3.1 Table Reshaping and Enrichment

In Wikipedia, some tables are generated using well-defined and popular templates while others are built using modified copies of templates taken from related pages. Consequently, tables that express similar content can be very diverse from each other and this hinders a successful factual extraction.

To counter this problem, we apply a procedure to "normalize" the tables. This procedure, which we refer to as reshape($\mathcal{T}$), performs three operations on every table $T \in \mathcal{T}$. The first operation merges or removes cells that span all the columns (mergechunks(T), Section 3.1.1). The second operation unpivots some columns to transform wide tables into narrow ones (smartunpivot($T, \mathcal{U}$), Section 3.1.2). Finally, tables are further enriched with extra contextual information (addcontext(T), Section 3.1.3).

3.1.1 Merging table chunks. Sometimes, Wikipedia contributors decide to add cells that span all the columns for various purposes. For example, such cells are added below the row they belong to keep the table from becoming too wide, or at the bottom as a footnote.

These cells can confuse the interpretation procedure since they can, for instance, be recognized as separate rows with new entities. To avoid these cases, the function mergechunks, which is formally defined in Appendix A, Algorithm 3, identifies these cells and copies their content to other parts of the table. The algorithm applies three heuristics H_1, H_2, H_3 that we observed work well in practice:

- H_1 If cells that span all the columns appear at every even row of the body, i.e., at row index $i = 2, 4, \ldots$, then we assume that the cells contain extra information about the preceding row. Thus, we add an extra column with empty cells, remove the i^{th} row and copy its content in the extra column at row $i - 1$;
- H_2 If H_1 does not apply, but there are cells that span all columns as last rows in the table, then we assume that they contain a footnote. In this case, we remove the rows and add their content as contextual information of type "footnote" to the table;
- H_3 If H_1 does not apply, but there are multiple cells that span all columns that appear in the body, then we treat them as extra information about the rows below them. To this end, we add an extra column with empty cells, remove every row with index i with a cell that spans all columns and copy its content in the extra column at row $i + 1, \ldots, j$ where j is either the end of the table or the row index of the following cell that spans columns.

3.1.2 Table Unpivoting. Wide tables tend to contain columns that express attributes values rather than attribute relations. We would like to transform such tables so that the content of these columns appears in the body instead of the header. For instance, the left table in Figure 1 (c) contains the columns US, US R&B, US Dance which are the values of attributes with relation chartedIn. This table should be transformed into the right table in Figure 1 (c).

To this end, we must tackle two challenges. First, we need to define a procedure that, given an input table, detects a sequence of horizontally adjacent header cells that encode attribute values. The second challenge consists of extracting the new column header associated with these values so that we can unpivot the table.

We tackle the first challenge with a set of six boolean functions that encode some heuristics, while we rely on the content of previous headers to extract the new column header. Our procedure for unpivoting the table can be viewed as a function smartunpivot which, given in input table T and set of boolean functions $\mathcal{U}$, returns an unpivoted version of T (or T if no unpivoting was possible).

We outline the functioning of smartunpivot on table T below (the pseudocode is in Appendix A, Algorithm 4). Each boolean function $U_1, \ldots, U_6 \in \mathcal{U}$ receives in input a table cell and returns true if the

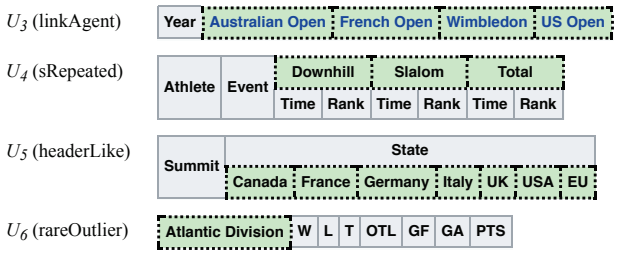


Figure 2: Examples of candidate tables headers for unpivoting. Cells in green are returned by the named heuristic.

encoded heuristics matches with the cell. First, the procedure scans the headers of T row-by-row and invokes all boolean functions in $\mathcal{U}$ with every cell in the input. An interval of adjacent cells for which a function has returned true maps to a potential set of columns with attribute values. We select the largest interval of such cells for unpivoting the table. Let us assume that this interval occurs at row i and spans columns $[j, k]$. To retrieve the new column header, we consider the cell at header row $i - 1$ (if any) and column j. If this cell has a span $[j, k]$, then we pick its value as column header, otherwise, we set the new column header with an empty cell.

Let $R(A_1, \ldots, A_{i-1}, A_i, \ldots, A_k, A_{k+1}, \ldots, A_m)$ be a relation that represents the schema of T where each attribute $A_1, \ldots, A_m$ maps to the header cell at head(T)[i]. Moreover, let y, z be two fresh attributes that will contain the unpivoted attributes and the content of the unpivoted columns respectively. We map y to $\boxed{\emptyset}$, while z maps either to a cell with the new column header or to $\boxed{\emptyset}$ if no relation was found. In the right table of Figure 1 (c), y would map to the 4^{th} column while z is the 5^{th} column.

Finally, let r be an instance of R with the body of T. We unpivot T by first executing $r' \coloneqq \text{UNPIVOT}^{y \to z}_{A_1,\ldots,A_{i-1},A_{j+1},\ldots,A_m}(r)$, and then creating a new table T' with body(T') $\coloneqq r'$ and head(T') $\coloneqq \langle\langle A_1, \ldots, A_{i-1}, A_{j+1}, \ldots, A_m, y, z\rangle\rangle$.

In the remaining, we describe the heuristics encoded by the boolean functions. Figure 2 shows examples of Wikipedia table headers for which these functions will apply.

- U_1 *(nPrefix)* Returns true if the cell starts with numeric characters;
- U_2 *(nSuffix)* Returns true if the cell ends with numeric characters;
- U_3 *(linkAgent)* Returns true if the cell contains a hyperlink to the Wikipedia page of an entity with type Agent in Wikidata, i.e.,

$$U_3(c) \coloneqq \exists e \in \text{links}(c) \text{ s.t. } \langle e, \text{isA}, \text{Agent}\rangle \in \mathcal{K} \quad (1)$$

  The underlying intuition is that entities of the type Agent, which in Wikidata includes people and organisations, are unlikely to be attribute relations but refer instead to attribute values.
- U_4 *(sRepeated)* Returns true if the cell spans an interval of columns and there is another row where the cells have equal value in the same interval. More formally, let T and r be the table and row respectively where c appears, and let $[i, j] = \text{span}(c)$.

$$U_4(c) \coloneqq \exists s \in \text{head}(T) \text{ s.t. } r \neq s \wedge \text{val}(s[i]) = \ldots = \text{val}(s[j]) \quad (2)$$

- U_5 *(headerLike)* Returns true if the cell appears in $\mathcal{T}$ more frequently either in the body or in a header cell that is spanned by

another cell. Before we define U_5, let N_t, N_b, N_s be as follows:

$$N_t(c) = |\{T \in \mathcal{T} : c \in T\}| \quad (3)$$

$$N_b(c) = |\{T \in \mathcal{T} : c \in \text{body}(T)\}| \quad (4)$$

$$N_s(c) = |\{T \in \mathcal{T} : c, d \in \text{head}(T) \wedge \text{span}(c) \subset \text{span}(d)\}| \quad (5)$$

Then, $U_5(c) \coloneqq \frac{N_b(c)+N_s(c)}{N_t(c)} > 0.5$.

- U_6 *(rareOutlier)* Returns true if the frequency of the cell in the headers of the tables in $\mathcal{T}$ is more than one standard deviation smaller than the average frequency of the cells of its header. Let T be the table where cell c appears, and $\mu(X)$ and $\sigma(X)$ be the mean and standard deviation of X. Then,

$$N_h(c) = |\{T \in \mathcal{T} : c \in \text{head}(T)\}| \quad (6)$$

$$\mu_h(T) = \mu(\{N_h(c) : c \in \text{head}(T)\}) \quad (7)$$

$$\sigma_h(T) = \sigma(\{N_h(c) : c \in \text{head}(T)\}) \quad (8)$$

and $U_6(c) \coloneqq N_h(c) < \mu_h(T) - \sigma_h(T)$.

3.1.3 Adding Contextual Information. Often, the context of a table can help the interpretation procedure to disambiguate entities more precisely. For example, a table may be located in a section whose header contains a keyword (e.g., "song") or a date that is important for disambiguating some entities. To this end, we would like to add such important contextual information to the table.

For each table $T \in \mathcal{T}$, we add to context(T) three pairs: one with the page title, one the section title, one with the table caption, provided these are available in the associated Wikipedia page. Moreover, procedure mergechunks can optionally add another set of pairs with footnotes. Procedure addcontext(T), (pseudocode in Appendix A, Algorithm 5) has the task of adding the pairs in context(T) to T. For each pair $\langle X, Y\rangle \in$ context(T), it adds an extra column with header X and cells values equal to Y. Table (b) in Figure 1 shows an example of a table modified by this procedure. Here, addcontext has added an extra column with the title of the page (Hound Dog).

3.2 Clustering

Some tables can be easily matched to the KB while others are more problematic, especially if they cover a domain that is not yet covered by the KB. For example, consider Table (b) in Figure 1. This table contains a column titled "Chart" but Wikidata does not contain any fact that involves the relation `chartedIn` in combination with the entities mentioned in this table. Because of this incompleteness, disambiguating this table on its own is hard. However, if there is another table with a similar schema but that can be matched to Wikidata, we can cluster them together so that we can propagate the knowledge obtained by matching one table to the other. Following this intuition, the next step in our pipeline consists of finding clusters of tables that express the same latent relation. To this end, we construct union tables using a set of approximate indexes followed by similarity functions designed for our specific use-case.

Due to the large size of our corpus (1.5M tables), computing our similarity scores for each pair of tables is computationally too expensive. To counter this problem, we perform a two-level clustering. First, we compute a set of table *blocks*, i.e., groups of tables that appear to be similar according to an approximate k-Nearest Neighbors procedure (k-NN) (Section 3.2.1). Then, we calculate our similarity scores on the reduced set of table pairs (Section 3.2.2). Finally, we construct a graph $\mathcal{G} = \langle \mathcal{T}, \mathcal{W}\rangle$, which has the tables in $\mathcal{T}$ as vertices, and the similarity scores defined in the sparse matrix $\mathcal{W} \in \mathbb{R}^{\mathcal{T}\times\mathcal{T}}$ as weighted edges. We partition this graph into clusters of tables, from which we construct the final set of union tables (Section 3.2.3).

3.2.1 Blocking. Since computing the similarity score between the 1.5Mx1.5M pairs of tables in $\mathcal{T}$ is computationally expensive, we construct four approximate indexes to retrieve, for a given table, the top-k similar tables using a more coarse-grained definition of similarity. We refer to the collection of similar $k+1$ tables as a *table block*. In our experiments, we use a fairly high value of k (100) to avoid that good table pairs are ignored.

The four approximate indexes $I_1, \ldots, I_4$ are constructed as follows. Two indexes I_1 and I_2 employ Locality Sensitive Hashing (LSH) with MinHash [35], which provides approximate Jaccard similarity between the cell values in two tables. We employ LSH because we have observed that two tables are more likely to express the same relation if many cell values overlap. Therefore, we construct one approximate LSH index considering the header of the table (I_1), and another considering the body of the table (I_2).

The other two indexes I_3 and I_4 perform approximate k-NN with the embeddings of header rows and body columns. We use embeddings because we noticed that sometimes table pairs might not contain exactly the same values, but words are nevertheless semantically similar. We exploit this observation considering pre-trained GloVe word embeddings [22] for every column and header row in our tables. To create cell embeddings, we simply sum the word embeddings of the words in the cell, and create header row and column embeddings by averaging the cell embeddings. Then, we index the vectors of the headers (I_3) and of the columns (I_4) and query them using approximate k-NN search offered by Facebook's library FAISS [10], one of the most scalable implementations of k-NN with embeddings.

After the indexes are computed, we retrieve the top k similar tables for each table in $\mathcal{T}$ using each index. This yields a set of $4(k+1)*|\mathcal{T}|$ blocks. We consider the union of all blocks returned by all indexes because at this stage we do not want to further sacrifice recall. For every pair of tables that appear in the same block, we compute a more fine-grained similarity score as described below.

3.2.2 Matching. The goal of our proposed similarity score between two tables is to measure to what extent pairs of columns in the two tables can be aligned. This follows the intuition that two tables express the same relation if many of their columns can be aligned.

The similarity score between two tables is an aggregation of the similarity scores between column pairs that are computed considering either the body or headers of two tables. The similarity scores between column pairs rely on a set $\mathcal{M}$ of functions, which we call *matching functions*. These functions, described below, receive in inputs two sets of cells, and compare the overlap of cell values, and the similarity with word embeddings and semantic types.

When comparing the columns of two tables A and B with m_A and m_B columns respectively, using matching function $f \in \mathcal{M}$, we compare every possible pair of columns between A and B. We create an alignment between columns in A and columns in B using

Algorithm 1 greedycolsim(A, B, f)

1: $C \coloneqq \text{body}(A)$ $D \coloneqq \text{body}(B)$
2: $m_A \coloneqq |\text{cols}(A)|$ $m_B \coloneqq |\text{cols}(B)|$
3: $E_A \coloneqq \{1, \ldots, m_A\}$ $E_B \coloneqq \{1, \ldots, m_B\}$ $M \coloneqq \emptyset$
4: **while** $|E_A| > 0$ **and** $|E_B| > 0$ **do**
5: $\langle i, j \rangle \coloneqq \underset{\langle i,j \rangle \in E_A \times E_B}{\arg\max} f(\text{cols}(C)[i], \text{cols}(D)[j])$
6: $E_A \coloneqq E_A \setminus \{i\}$ $E_B \coloneqq E_B \setminus \{j\}$ $M \coloneqq M \cup \{\langle i, j \rangle\}$
7: $S \coloneqq \sum_{\langle i,j \rangle \in M} f(\text{cols}(C)[i], \text{cols}(D)[j])$
8: **return** $\frac{1}{2}(\frac{S}{m_A} + \frac{S}{m_B})$

Algorithm 2 aggsim$(A, B, \mathcal{M}, \theta)$

9: $s_b = \max_{f \in \mathcal{M}} \text{greedycolsim}(A, B, f)$
10: $h_A = \text{head}(A)$ $h_B = \text{head}(B)$
11: $s_h = \max_{f \in \mathcal{M}} f(h_A, h_B)$
12: **return** $\theta s_h + (1 - \theta) s_b$

the greedy procedure greedycolsim(A, B, f) shown in Algorithm 1. This procedure creates $\min(m_A, m_B)$ alignments by selecting the best column matches according to f, and aggregates those similarity scores by averaging over both m_A and m_B.

The procedure greedycolsim returns a table alignment score using one matching function f. We invoke this procedure with every $f \in \mathcal{M}$. The scores are then aggregated in a manner described by procedure aggsim, Algorithm 2. The application of aggsim$(A, B, \mathcal{M}, \theta)$ on tables A and B is as follows. Generally, the aggregation of semantic matcher scores depends on whether they compute "optimistic" or "pessimistic" similarities and whether there is supervision or heuristics available [6]. In our case, we take an "optimistic" approach assuming that any of our matching functions may be the most relevant for a given pair of tables. Therefore, we take the best scored obtained by any matching function (line 9). Note that the application in line 9 considers only the body of the tables. We have observed that often correctly matching table pairs have also aligned headers. Therefore, we invoke the matching functions considering the tables' headers and optimistically max-aggregate them (line 11). Then, the final score is obtained by combining them using their weighted mean (line 12). The aggregation weight θ is found using cross-validated grid-search.

Next, we discuss our matching functions $f_j, f_e, f_d \in \mathcal{M}$.

f_j: Set Similarity. The simplest way to view of headers and columns is as a set of discrete cell values. To model whether two sets of discrete values are similar, we use their Jaccard index:

$$f_j(a, b) = \frac{|a \cap b|}{|a \cup b|} \tag{9}$$

f_e: Word Embedding Similarity. Following [20], we create word embeddings for cells by summing the word embeddings of the tokens in their values. For computing the similarity score, we use the positive cosine distance between the cell embedding, i.e.,

$$f_e(a, b) = \max(0, \frac{\bar{\boldsymbol{w}}(a) \cdot \bar{\boldsymbol{w}}(b)}{\|\bar{\boldsymbol{w}}(a)\| \|\bar{\boldsymbol{w}}(b)\|}) \tag{10}$$

where $\bar{\boldsymbol{w}}(X)$ is the mean of the embeddings of the cell values in X.

f_d: Datatype Similarity. The functions above consider only the cell values for computing the alignment score. The hyperlinks to Wikipedia pages that are present in cells can be used to create a semantic representation based on the types of entities that they link to. Additionally, we can exploit the repeated patterns in cell sets when they contain composite values involving multiple datatypes.

We proceed as follows: for every cell, we extract a number of patterns corresponding to possible semantic types. The patterns are created by detecting the named entities in the cell (we use the library Spacy (spacy.io)), and combining them with their hyperlinks. We replace each named entity in the cell with all the types of the entity in the KB. This results in patterns such as `[Football Cup] final [YEAR]`. Let a and b be two sets of cells, $N_p(a)$ be the number of unique cells in a from which pattern p is extracted, and P be the set of all patterns extracted from a and b. For every pattern extracted from a, we calculate its *overlap score* as $O_p(a) = N_p(a) \ / \ |a|$ and keep only those patterns for which $O_p(a) > \tau$ (default value $\tau = 0.5$). Our datatype similarity function is the cosine similarity between the pattern overlap vectors $\boldsymbol{O}(a), \boldsymbol{O}(b) \in [0, 1]^P$ of two cell sets:

$$f_c(a, b) = \frac{\boldsymbol{O}(a) \cdot \boldsymbol{O}(b)}{\|\boldsymbol{O}(a)\| \|\boldsymbol{O}(b)\|} \tag{11}$$

3.2.3 Clustering. Given the weighted graph $\mathcal{G}$ of table union candidate pairs, we perform clustering to find sets of unionable tables. This is equivalent to partitioning a similarity graph [16]. To this end, we employ Louvain Community Detection [2] – a state-of-the-art algorithm that scales to large graphs such as ours.

The Louvain algorithm optimizes a value known as *modularity*, which measures the density of links between of communities compared to those inside communities themselves. Recall that $\mathcal{W}$ is the matrix of weights in the edges, and let $z = \sum_{ij} \mathcal{W}_{ij}$ be the sum of its values. Given an assignment of a community c_i for each node i, the modularity is defined

$$\mathcal{Q} = \frac{1}{2z} \sum_{ij} \left[\mathcal{W}_{ij} - \frac{k_i k_j}{2z} \right] \delta(c_i, c_j) \tag{12}$$

where k_i and k_j are the sum of the weights of the edges attached to nodes i and j respectively, and δ is the Kronecker delta function. Initially each node is in its own community, after which two steps are alternated until convergence. In the first step, each node is moved to the community that maximises modularity. In the second step, the procedure constructs a new weighted graph $\mathcal{G}'$ and replaces $\mathcal{G}$ with it. In $\mathcal{G}'$, the nodes map to communities, weighted edges are an aggregated score of the edges between nodes in different communities in $\mathcal{G}$, while edges between nodes in the same community in $\mathcal{G}$ are represented by self-loops. The runtime of this procedure appears to scale with $O(n \cdot log^2 n)$ in the number of nodes [12].

After finding clusters of similar tables, we align all columns of the tables within each cluster. First, we create a matrix of max-aggregated column similarities using the matching functions in $\mathcal{M}$ for each pair of columns in the tables in the cluster. Then, we run agglomerative clustering [18] with complete linkage on this matrix to identify groups of similar columns. Agglomerative clustering iteratively combines the two clusters (i.e., two groups of columns) which are separated by the shortest distance.

Once the columns are clustered together, we create a union table with as many columns as clusters. Then, the tables are concatenated filling the gaps with empty cells. To create a header for this table, we take the most frequent header cell of each column cluster. The set of union tables will be the input of the next stage of our pipeline.

3.3 KB Integration

The last step in our pipeline consists of extracting facts from the union tables. This phase, shown at the bottom of Figure 1, determines the type of the union table and extracts the facts from it.

3.3.1 Detecting n-ary union tables. A key challenge in extracting facts from the union tables is distinguishing between EA and NA union tables. To this end, we make use of pFDs. This gives us a robust signal for union tables because their large number of rows prevents the pFDs from expressing noise, which occurs very frequently in small tables. Let $R(A_1, \ldots, A_m)$ be the relation associated to union table T and r be the instance of R with the body of T. We run perTuple on r to compute the set F_T of pFDs. Let B be the attribute of R with the highest harmonic mean of the multiset $\{p : A \rightarrow^p B \in F_T\}$ (ties are broken by taking the leftmost column). If the harmonic mean is greater than a given threshold υ (default value is 0.95) then we assume that T is a EA table and the column associated to B is the key column. Otherwise, T is an NA table.

3.3.2 Entity Disambiguation. The extraction of factual knowledge from the union table is split into two phases. First, we disambiguate the cells in T into entities in $\mathcal{K}$, regardless the type of T. We make use of the hyperlinks whenever they are available and maximise the coherence of entities if multiple matches are possible, as described in [11]. Note that here the large number of rows in the union tables is particularly helpful as it provides a clearer signal for disambiguating the entities. In the following, we denote with $\text{entity}(c) \in \mathcal{E}$ the entity associated with cell c if we found a match, otherwise $\text{entity}(c) = \texttt{NULL}$.

3.3.3 Fact Extraction. We proceed differently depending on whether T is an EA or a NA table.

If T is a EA table, we first retain all pFDs with a sufficiently high probability, i.e., greater than υ, which we call $F_T^{>\upsilon}$. For each pFD $A \rightarrow^p B \in F_T^{>\upsilon}$, we search for a relation in $\mathcal{K}$ suitable to represent the dependency between A and B. Let $Col_X \in \text{cols}(T)$ be the column of T associated with the attribute X in R. First, we compute the set of all pairs of entities mentioned in the columns, i.e., $E_{A,B} \coloneqq \{\langle a, b\rangle : \forall i.\ a = \text{entity}(Col_A[i]) \wedge b = \text{entity}(Col_B[i] \wedge a \neq \texttt{NULL} \wedge b \neq \texttt{NULL})\}$. Then, the set of matched facts for $A \rightarrow^p B$ and relation $r \in \mathcal{R}$ in $\mathcal{K}$ is $M_{A,B}(r) \coloneqq \{\langle b, r, a\rangle : \langle a, b\rangle \in E_{A,B}\} \cap \mathcal{K}$. We pick the relation $r \in R$ such that $r = \arg\max_{r \in \mathcal{R}} |M_{A,B}(r)|$, that is, the relation with the maximum overlap, like [19]. Then, we output the fact $\langle b, r, a\rangle$ for each $\langle a, b\rangle \in E_{A,B}$ so that it can be added to $\mathcal{K}$.

If T is a NA table, let $E_{A,B,C}$ be the set of tuples for attributes A, B, C defined analogously to $E_{A,B}$. For every possible pair of attributes A, B and relation $r \in \mathcal{R}$, we first identify the columns that contain entities that appear in qualifiers of facts in $M_{A,B}(r)$. To this end, we denote with $N(A, B, C, r) \coloneqq \{\langle y, c\rangle : \langle a, b, c\rangle \in E_{A,B,C} \wedge \langle q_{b,r,a}, y, c\rangle \in \mathcal{K}\}$ the set of qualifiers that could be retrieved considering the entities in Col_C, and with $Q(A, B, r) \coloneqq \{C : N(A, B, C, r) \neq \emptyset\}$ the set of attributes where some qualifiers were found. Then, we consider all A, B, r with the highest number of qualifier-matching columns $|Q(A, B, r)|$ because these are the potential n-ary relations with the largest coverage of columns. If there are multiple A, B, r with the same highest $|Q(A, B, r)|$, then we choose the one with the highest number of matched facts $|M_{A,B}(r)|$. Finally, for each $X \in Q(A, B, r)$, we identify the relation r_X that has the highest frequency in the multiset $\{y : \langle y, c\rangle \in N(A, B, X, r)\}$. This relation will be the one used to create the qualifiers with the entities in Col_X. At this point we are ready to extract the facts from T: We output the fact $\langle b, r, a\rangle$ for each $\langle a, b\rangle \in E_{A,B}$, and, for each $X \in Q(A, B, r)$, we output the triple $\langle q_{d,r,c}, r_X, e\rangle$ for each $\langle c, d, e\rangle \in E_{A,B,X}$.

Example 3.1. In Figure 1(g), we show the union table constructed from the cluster in Figure 1(f). Due to its pFDs, we classify it as an NA union table, and attempt to find matching qualifiers in $\mathcal{K}$. Let C, S, and P be the columns in this table that have the headers "Chart", "Single" and "Position", respectively. Then, we have that $\langle \texttt{US\_Billboard\_200}, \texttt{Brick\_House}, 5\rangle \in E_{C,S,P}$ (`US_Billboard_200` is the entity that matches the cell "US" in the table). Let's assume that the triple $\langle q_x, \texttt{ranking}, 5\rangle \in \mathcal{K}$ where $x = \langle \texttt{Brick\_House}, \texttt{chartedIn}, \texttt{US\_Billboard\_200}\rangle$. This means that $\langle \texttt{ranking}, 5\rangle \in N(C, S, P, \texttt{chartedIn})$ and $P \in Q(C, S, \texttt{chartedIn})$. If C, S, and `chartedIn` have the highest number of these qualifier-matching columns $Q(C, S, \texttt{chartedIn})$ and the highest number of matches $M_{C,S}(\texttt{chartedIn})$ of all column pairs and relations, we use the relation `chartedIn` to extract facts from columns C and S. Finally, if `ranking` is the most frequent relation in $N(C, S, P, \texttt{chartedIn})$, we use it for extracting qualifiers from column P. Let us assume that $E_{C,S,P}$ contains another tuple $\langle \texttt{US\_Billboard\_200}, \texttt{Thriller}, 1\rangle$. In this case, the system will output the facts $f = \langle \texttt{Thriller}, \texttt{chartedIn}, \texttt{US\_Billboard\_200}\rangle$ and $\langle \texttt{q}_\texttt{f}, \texttt{ranking}, 1\rangle$, which is graphically depicted in Figure 1(i).

4 EVALUATION

For our empirical evaluation, we considered the corpus of 1,535,332 Wikipedia tables from [1], with 1,426,303 unique tables of which 26,260 occur on more than one page. There are 330,221 unique headers, of which 247,403 (75%) occur only once. This means there are 1,287,929 tables that have a header that is shared by some other table. On average, these tables have 11 rows. The experiments here presented were performed with a Wikidata dump from Dec. 2019.

Annotations. Since there were no available gold standard to test our method, we created it using three human annotators. To this end, we developed a GUI that showed the page title, description, section title and caption, and table contents. We sampled 1000 random tables, all from different Wikipedia pages, which have 3449 columns in total. We aggregated these annotations by majority vote, with moderate agreement between annotators (Fleiss' $\kappa = 0.57$).

First, the annotators annotated the columns that should be unpivoted by selecting a horizontal sequence of cells in the header of a table. The guidelines specified that the sequence should "contain names of a *related* set of concepts that *do not* describe the content of the column below them." After annotation, the table was shown to the annotator in unpivoted form for verification. This resulted in 151 tables from the sample to unpivot.

Then, the annotators were asked to create table unions from the unpivoted tables resulting from the previous phase by iteratively merging clusters. They were presented with one query cluster and several candidate clusters, ranked according to the matchers described above. All clusters were presented as a union table (i.e. "vertical stack") of all tables in that cluster. From these candidate clusters, the annotators were asked to identify the clusters that

Heuristic	Pr.	Re.	F_1
headerLike	0.30	0.33	0.31
numPrefix	0.73	0.27	0.39
sRepeated	**0.89**	0.12	0.21
numSuffix	0.52	0.09	0.16
linkAgent	0.70	0.05	0.09
rareOutlier	0.14	0.02	0.04
All Heuristics	0.44	**0.88**	**0.58**

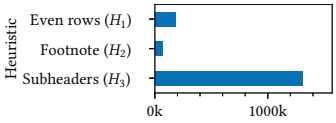
(a) Precision, Recall and F_1 of unpivot detection heuristics

Reshape	Clusters	ARI	AMI	H.	C.	V.
-	One per table	0.00	0.00	1.00	0.85	0.92
-	Same Header	0.55	0.55	1.00	0.90	0.95
Model	Same Header	0.55	0.56	1.00	0.90	0.95
Oracle	Same Header	0.56	0.57	1.00	0.90	0.95
Model	Only Header Sim.	0.83	0.77	0.97	0.95	0.96
Model	Only Body Sim.	0.69	0.67	0.99	0.93	0.96
Model	Both Sim.	**0.88**	**0.85**	0.99	0.97	0.98
Model	Oracle	0.93	0.94	1.00	0.98	0.99

(b) Clustering ablation for different configurations

	Accuracy			
Method	EA	NA	Both	EA pct.
Non-num. baseline	0.29	0.10	0.39	90%
Entity baseline	0.24	0.20	0.43	74%
Ours	**0.45**	**0.77**	**0.66**	**36%**

(c) Key column prediction accuracy for EA tables, NA table detection accuracy, both tasks combined, and the EA table percentage predicted (EA pct.). The true percentage of EA tables in our annotated sample is 35%



(d) # cells processed by heuristics in mergechunks

	Tables	Columns
Individual	173 (17%)	267 (8%)
Union	193 (**33%**)	419 (**17%**)

(e) Number of matched tables and columns in annotated sample

		Existing Facts in $\mathcal{K}$		New Facts	
	Precision	#	Context	#	Context
Binary	0.65	4,708,865	33%	15,800,524	29%
N-ary	**0.92**	2,122,518	22%	6,901,790	19%
All	0.74	**6,831,383**	30%	**22,702,314**	26%

(f) Relation matching precision, number of matched and new facts, and percentage of matched and new facts that involve the table context

Figure 3: Experimental results. The best outcomes (except the annotations-based oracle) are reported in boldface

expressed the same relation as the query table. As a proxy, the guidelines suggested that they could ask themselves: "Would it make sense to add every row in the candidate table to the query table?" Then, they identified the aligning column pairs by either selecting a query column or adding a new column to the clustered union table for every candidate column. This resulted in 577 clusters with a total of 2479 columns, in which annotators identified a key column (marking it as an EA table), or identified it as an NA table.

Finally, we let annotators evaluate the union table column disambiguations that were returned by our system. They assessed the correctness of every KB relation that was assigned to a column based on whether its semantics corresponded to that of all rows.

Although the collected annotations do not allow us to test every single component of the system, they are sufficient to evaluate the critical ones and can give us an overview of the overall performance. Below we report the results of some key experiments.

Table Reshaping. In Figure 3d, we show the number of cells that span all columns which were identified by each heuristic from Section 3.1.1. Without the application of the mergechunks procedure, each of these 1,554,692 cells would be located in the wrong columns, where they would interfere with the subsequent operations.

On the entire dataset with 1.5M tables, smartunpivot predicted that 260,528 unique headers should be unpivoted, corresponding to 933,949 different tables. A breakdown of prediction scores per heuristic is shown in Figure 3a. The heuristics are designed to be complementary because they cover a different type of header. Therefore, they are not expected to individually have high recall, but ideally the recall should be high when they are combined together. From Figure 3a, we can see that this is indeed the case with a F_1 of 0.58. Examples of false positives of these heuristics include attribute labels that occur frequently in table bodies (such as "director"). The false negatives include values for which we do not have enough statistics to result in high heuristic scores.

Table Clustering. Figure 3b reports a feature ablation study with various combination of reshaping and clustering methods. The "Oracle" reshaping strategy unpivots the tables on annotations while "Model" uses our reshape(). The "Same Header" clusters are made by grouping tables with the same header and the "Oracle" clusters are based on the gold-standard annotations. We show the performance of our clustering phase when using only the header similarities ($\theta = 1$ in Alg. 2), the body similarities ($\theta = 0$ in Alg. 2), and all similarity functions together (θ set in Alg. 2 with cross validation).

We use several scoring functions to evaluate the generated clusters. Because clusters can have very different sizes, and there are many clusters of a single element, some of these metrics are adjusted for chance to reduce the scores of random or degenerate clusterings. The *Adjusted Rand Index (ARI)* expresses the cluster and class agreement of item pairs, adjusted for chance [9]. The *Adjusted Mutual Information (AMI)* expresses the mutual information between the clusters and classes, adjusted for chance [27]. Furthermore, a clustering result satisfies *Homogeneity (H.)* if all of its clusters contain only data points which are members of a single class. A clustering result satisfies *Completeness (C.)* if all the data points that are members of a given class are elements of the same cluster. The *V-measure (V.)* is their harmonic mean (similar to the F_1 score) [26].

From Figure 3b, we see that clustering using only similarities of headers ("Only Head Sim.") outperforms the one that considers the body ("Only Body Sim."), but their aggregation gets us nearest to the performance of the oracle, which is built with human annotations.

Table Interpretation and KB Integration. In Figure 3c, we report the performance of our key-column and n-ary table detection approach, and compare it to two baselines. The "Non-numeric" baseline selects the rightmost non-numeric column that contains at least 95% unique values if it exists, and the "Entity" baseline does the same thing for columns which contain entities [25]. If such columns do not exist, they predict the table is n-ary. Note that these baselines are the most commonly used ones by state-of-the-art systems (e.g., [11]). Our approach is more accurate for both EA tables and NA tables, and closest to predict the correct rate of EA tables.

In fact, the baselines are severely biased and select EA tables in 90% and 74% of the times, incorrectly labeling many NA tables and consequently precluding the extraction of n-ary facts. In contrast, our performance is much closer to the true rate of 35%.

In Figure 3e, we compare the relation identification counts for individual tables from the annotated sample to the counts for our union tables. The percentage of (union) tables and columns for which we could identify KB relations is higher for the union tables, which indicates that creating union tables from clusters leads to more identified relations that we can use for fact extraction.

Finally, Figure 3f reports the precision for binary and n-ary facts extracted by our system on the gold standard. Interestingly, we observe that our system is more precise at extracting n-ary facts instead of binary facts. We applied our pipeline to the corpus of 1.5M tables and our system extracted 29.5M facts. Of these, 77% are novel facts, i.e., not available in Wikidata. The columns titled "Context" report the percentage of facts that involved the extra columns added by addcontext(). The large ratio indicates that including external context is beneficial as it leads to more extractions.

5 RELATED WORK

Although we are not aware of existing systems that extracts on n-ary relations from tables and integrate them with KBs, there has been a significant amount of research on similar problems to ours.

Table transformation and analysis. Pivk et al. [23] propose a comprehensive functional table model, called TARTAR, for transforming tables into logical form. The authors describe a *recapitulation* process which identifies attribute values in table headers by decomposing complex headers. In contrast to us, this is only performed when the headers have a tree-like structure, and relies on external resources. More recently, Halevy et al. [8] analyzed a large corpus of web tables, discovering structure in attribute labels to characterize their compositionality in terms of complex relations. Although it extracted many rules for open IE, the resulting records are not disambiguated nor integrated with a KB. The method by Lehmberg and Bizer [14] classifies table columns for detecting layout and list tables, distinguishing binary relations from relations with a higher arity using inter-table statistics involving fuzzy pFDs as features. However, they do not disambiguate the n-ary relations, as we do.

Table search. Much research has been done on supporting user queries over large collections of web tables. Cafarella et al. [4] described the WebTables project in which they construct a join graph for supporting table-building user queries. They make use of an Attribute Correlation Statistics Database (ACSDb), which disambiguates column headers to some extent. Similarly, Wang et al. [30] described a system for bootstrapped taxonomy population from web tables, and its application in semantic table search. The Octopus system [3] introduced search joins, where users search for relevant tables to join with a query table. Similarly, Bhagavatula et al. [1] described a relevant-join prediction approach for Wikipedia tables. Finally, the Infogather system [33] supports several user query types, and explicitly assumes that tables are entity-attribute tables with a subject column. These approaches focus on search and do not integrate the tables with a KB.

Table clustering. Ling et al. [17] introduced a method for creating union tables which makes use of the table context for constructing hidden attributes that are part of n-ary relations. In our scenario, the context is much simpler and is added to the table as extra columns, which we use in clustering. The work of Wang et al. [29] finds regularities in the schemata of a collection of tables, in order to create a mediated schema that expresses the functional dependencies between attributes to identify noisy data sources within the collection. The seminal work of Das Sarma et al. [5] describes an approach to find related tables based on their schema and contents. This results in table pair candidates for joins or unions, but does not have a way to canonicalize entities or detect duplicates. Similarly, in the recent work of Fetahu et al. [7], pairs of Wikipedia tables with equivalent or subsumed schemata are discovered with a neural network classifier, and this is useful for improving the detection of complex relations. None of these works integrate the tables with a KB, nor distinguish pivoted cells from values in the table headers.

Table integration. Several related systems exist for table integration with a KB. Both the state-of-the-art table interpretation systems T2KMatch [24] and TableMiner [34] assume that the input consists only of EA tables. More recently, the system proposed by Kruit et al. [11] has reported better performance than T2KMatch and TableMiner. This system relies on coherence to disambiguate the entities of EA tables, which we re-use in our pipeline to perform the entity disambiguation. The work of Muñoz et al. [19] is also relevant to our approach because it aims at extending DBpedia with facts extracted from Wikipedia tables. They parse the structure of the tables using TARTAR, and use hyperlinks to match cell pairs to DBpedia triples to link table columns to relations. This is followed by a post-processing step that consists of a trained triple-filtering model to boost extraction precision. However, they do not distinguish between attribute labels and values in the headers, and do not attempt to extract n-ary relations, which DBpedia does not support.

6 CONCLUSION

In this paper, we tackled the problem of automatically integrating the knowledge in Wikipedia tables into a KB. In particular, we focused on the extraction of both EA and NA tables. Our pipeline introduces new heuristics for table transformation based on, among others, unpivoting, which allows us to consider more tables. Then, our method uses clustering to alleviate the problems of KB incompleteness and table diversity. Finally, the integration step judiciously interprets the table, distinguishing EA tables from NA ones and extracts facts in a format suitable for a KB integration.

We carried out an empirical evaluation, relying on manual annotators to verify the high quality of our extractions. Our implementation is scalable as we were able to process all 1.5M tables from Wikipedia. More generally, all steps except the Louvain algorithm for clustering scale linearly with the number of tables.

Future work includes creating a model for filtering the extracted facts to boost their precision, developing a semi-supervised model for table transformation, and designing more sophisticated alignment functions for improving the quality of the table clusters. Additionally, the performance of our pipeline on non-Wikipedia tables remains an open question. In particular, our pipeline assumes that there is some overlap between the entities in the tables and in the KB and that such entities can be successfully disambiguated. It is

interesting to study whether we can relax such assumptions so that our pipeline can be applied on different table corpora.

We hope to deploy our table extraction pipeline in production for automatically augmenting Wikidata with new statements, and look forward to welcoming contributions from the research community.

REFERENCES

[1] Chandra Sekhar Bhagavatula, Thanapon Noraset, and Doug Downey. 2013. Methods for exploring and mining tables on Wikipedia. *Workshop on Interactive Data Exploration and Analytics* (2013), 18–26.

[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[3] Michael J Cafarella, Alon Halevy, and Nodira Khoussainova. 2009. Data integration for the relational web. *VLDB* 2, 1 (2009), 1090–1101.

[4] Michael J Cafarella, Alon Halevy, Daisy Zhe Wang, Eugene Wu, and Yang Zhang. 2008. WebTables: Exploring the Power of Tables on the Web. *VLDB* 1, 1 (2008), 538–549.

[5] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. 2012. Finding Related Tables. *SIGMOD* (2012), 817.

[6] Hong-Hai Do and Erhard Rahm. 2002. COMA: a system for flexible combination of schema matching approaches. *VLDB* (2002), 610–621.

[7] Besnik Fetahu, Avishek Anand, and Maria Koutraki. 2019. TableNet: An Approach for Determining Fine-grained Relations for Wikipedia Tables. *CoRR* abs/1902.0 (2019).

[8] Alon Halevy, Natalya Noy, Sunita Sarawagi, Steven Euijong Whang, and Xiao Yu. 2016. Discovering Structure in the Universe of Attribute Names. In *WWW*. ACM, 939–949.

[9] Lawrence Hubert and Phipps Arabie. 1985. Comparing partitions. *Journal of classification* 2, 1 (1985), 193–218.

[10] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734* (2017).

[11] Benno Kruit, Peter Boncz, and Jacopo Urbani. 2019. Extracting Novel Facts from Tables for Knowledge Graph Completion. In *ISWC*. Springer, 364–381.

[12] Andrea Lancichinetti and Santo Fortunato. 2009. Community detection algorithms: A comparative analysis. *Phys. Rev. E* 80 (Nov 2009), 056117. Issue 5.

[13] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and others. 2015. DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[14] Oliver Lehmberg and Christian Bizer. 2016. Web table column categorisation and profiling. In *WEBDB*.

[15] Oliver Lehmberg and Christian Bizer. 2019. Profiling the semantics of n-ary web table data. In *SBD*. 1–6.

[16] Oliver Lehmberg and Oktie Hassanzadeh. 2018. Ontology augmentation through matching with web tables. In *CEUR Workshop Proceedings*, Vol. 2288. 37–48.

[17] Xiao Ling, Alon Halevy, Fei Wu, and Cong Yu. 2013. Synthesizing union tables from the Web. *IJCAI* (2013), 2677–2683.

[18] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval*. Cambridge university press.

[19] Emir Muñoz, Aidan Hogan, and Alessandra Mileo. 2014. Using linked data to mine RDF from wikipedia's tables. In *WSDM*. ACM, 533–542.

[20] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table Union Search on Open Data. *PVLDB* 11, 7 (2018), 813–825.

[21] Natasha Noy, Alan Rector, Pat Hayes, and Chris Welty. 2006. Defining n-ary relations on the semantic web. *W3C working group note* 12, 4 (2006).

[22] Jeffrey Pennington, Richard Socher, and Christopher D Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*. ACL, 1532–1543.

[23] Aleksander Pivk, Philipp Cimiano, York Sure, Matjaz Gams, Vladislav Rajkovič, and Rudi Studer. 2007. Transforming arbitrary tables into logical form with TARTAR. *DKE* 60, 3 (2007), 567–595.

[24] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. 2015. Matching HTML Tables to DBpedia. In *WIMS*.

[25] Dominique Ritze, Oliver Lehmberg, Yaser Oulabi, and Christian Bizer. 2016. Profiling the Potential of Web Tables for Augmenting Cross-domain Knowledge Bases. In *WWW*. ACM, 251–261.

[26] Andrew Rosenberg and Julia Hirschberg. 2007. V-measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP*. ACL, 410–420.

[27] Nguyen Xuan Vinh, Julien Epps, and James Bailey. 2010. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *JMLR* 11 (2010), 2837–2854.

[28] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledge base. *Commun. ACM* 57, 10 (2014), 78–85.

[29] D.Z. Wang, Luna Dong, a.D. Sarma, M.J. Franklin, and Alon Halevy. 2009. Functional Dependency Generation and Applications in pay-as-you-go data integration systems. *WebDB* (2009), 1–6.

[30] J Wang, Bin Shao, and Haixun Wang. 2010. Understanding tables on the web. In *ER*, Vol. 1. Springer, 141–155.

[31] Hadley Wickham et al. 2014. Tidy data. *Journal of Statistical Software* 59, 10 (2014), 1–23.

[32] Catharine M. Wyss and Edward L. Robertson. 2005. A formal characterization of PIVOT/UNPIVOT. In *CIKM*. ACM, 602–608.

[33] M Yakout and Kris Ganjam. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *SIGMOD*. ACM, 97–108.

[34] Ziqi Zhang. 2017. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web* 8, 6 (2017), 921–957.

[35] Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J. Miller. 2016. LSH Ensemble: Internet-Scale Domain Search. *PVLDB* 9, 12 (2016), 1185–1196.

A ADDITIONAL PSEUDOCODE

Algorithm 3 mergechunks(T)

if $|\text{cols}(T)| = 1$ **then return** T
$B \coloneqq \text{body}(T)$ $H \coloneqq \text{head}(T)$ $n \coloneqq |B|$ $m \coloneqq |\text{cols}(T)|$
$H' \coloneqq \langle\rangle$ $B' \coloneqq \langle\rangle$ $C \coloneqq \langle\rangle$
if $\text{span}(B[2i][1]) = [1, m]$ for each $1 \le i \le \frac{n}{2}$ **then**
  for $i \coloneqq 1$ **to** $|H|$ **do** $\text{append}(H[i] \cdot \langle\boxed{\emptyset}\rangle, H')$
  for $i \coloneqq 1$ **to** $\frac{n}{2}$ **do** $\text{append}(B[2i-1] \cdot \langle B[2i][1]\rangle, B')$
else
  $i \coloneqq n$
  while $i \ge 1$ **do**
    if $\text{span}(B[i][1]) = [1, m]$ **then**
      $\text{append}(\langle \text{“}footnote\text{”}, \text{val}(B[i][1])\rangle, C)$
    else break
    $i \coloneqq i - 1$
  if $\exists j$ such that $1 \le j \le i$ and $\text{span}(B[j][1]) = [1, m]$ **then**
    $D \coloneqq \boxed{\emptyset}$
    for $i \coloneqq 1$ to $|H|$ **do** $\text{append}(\langle D\rangle \cdot H[i], H')$
    for $j \coloneqq 1$ to i **do**
      if $\text{span}(B[j][1]) = [1, m]$ **then** $D \coloneqq B[j][1]$
      else $\text{append}(\langle D\rangle \cdot B[j], B')$
  else
    $H' \coloneqq H$ $B' \coloneqq \langle B[1], \ldots, B[i]\rangle$
Let T' be a table s.t. $\text{head}(T') = H'$, $\text{body}(T') = B'$, and $\text{context}(T') = \text{context}(T) \cup C$
return T'

Algorithm 4 smartunpivot($T, \mathcal{U}$)

$H \coloneqq \text{head}(T)$ $m \coloneqq |\text{cols}(T)|$ $b \coloneqq c \coloneqq d \coloneqq 0$
for $i \coloneqq 1$ **to** $|H|$ **do**
  for all $U \in \mathcal{U}$ **do**
    Let u_l be the returned value of $U(T, H[i][l])$
    Let $[j, k]$ be the largest interval s.t. $u_j = \ldots = u_k$ = **true**
    if $(k - j) > (d - c)$ **then**
      $b \coloneqq i$ $c \coloneqq j$ $d \coloneqq k$
if $b = 0$ **then return** T
$y \coloneqq z \coloneqq \boxed{\emptyset}$
if $b > 1 \wedge \text{span}(H[b-1][c]) = [c, d]$ **then** $z \coloneqq H[b-1][c]$
Let $R(A_1, \ldots, A_{c-1}, B_c, \ldots, B_d, A_{d+1}, \ldots, A_m)$ be a relation with $A_i \coloneqq H[b][i]$ and $B_i \coloneqq H[b][i]$. Also, let r be an instance of R with $\text{body}(T)$
$s = \text{UNPIVOT}^{y \to z}_{A_1, \ldots, A_{c-1}, A_{d+1}, \ldots, A_m}(r)$
Let T' be a table where:
  ∘ $\text{head}(T') = \langle\langle A_1, \ldots, A_{c-1}, A_{d+1}, \ldots, A_m, y, z\rangle\rangle$
  ∘ $\text{body}(T') = s$
return T'

Algorithm 5 addcontext(T)

$A \coloneqq \langle\rangle$ $B \coloneqq \langle\rangle$ $H \coloneqq \langle\rangle$ $J \coloneqq \langle\rangle$
for all $C \in \text{context}(T)$ **do**
  $\text{append}(\langle C[1]\rangle, A)$ $\text{append}(\langle C[2]\rangle, B)$
for $i \coloneqq 1$ **to** $|\text{head}(T)|$ **do** $\text{append}(A \cdot \text{head}(T)[i], H)$
for $i \coloneqq 1$ **to** $|\text{body}(T)|$ **do** $\text{append}(B \cdot \text{body}(T)[i], J)$
Let T' be a table where $\text{head}(T') = H$ and $\text{body}(T') = J$
return T'