BKGFT
Building Knowledge Graph from Tables*

Sibar Soumi
sibar@mail.uni-paderborn.de

Data Science Group, Paderborn University, Paderborn, Germany
https://github.com/sibarsoumi/bkgft

Abstract. An approach to automatically extract RDF triples based on tables in given wikipedia articles.

Keywords: Data science · Semantic web · Relation extraction · Knowledge graph · Wikipedia tables.

* Supervision: Prof. Dr. Axel Ngonga, Dr. Ricardo Usbeck

Table of Contents

1 Introduction

In this paper an approach for building a knowledge graph from tables in wikipedia articles will be demonstrated, discussed and evaluated. The input is a dump of wikipedia articles. The output is a knowledge graph as RDF statements in a file in turtle format [1].

This takes place in two main steps; 1) Reading and Preprocessing 2.1, 2) Raltion Extraction 2.2, which will be described in the following.

2 Approach

2.1 Reading and Preprocessing

Reading the tables directly from the given wikipedia dump is not possible due to the absence of tables' annotations, i.e. a table such as that in Figure 1 is found in the dump as demonstrated in Figure 2

Rank	County	Population (2010 Census)	Seat	Largest city
1	Jefferson	658,466	Birmingham	Birmingham
2	Mobile	412,992	Mobile	Mobile
3	Madison	334,811	Huntsville	Huntsville
4	Montgomery	229,363	Montgomery	Montgomery
5	Shelby	195,085	Columbiana	Hoover (part) Alabaster
6	Tuscaloosa	194,656	Tuscaloosa	Tuscaloosa
7	Baldwin	182,265	Bay Minette	Daphne
8	Lee	140,247	Opelika	Auburn
9	Morgan	119,490	Decatur	Decatur
10	Calhoun	118,572	Anniston	Anniston
11	Etowah	104,303	Gadsden	Gadsden
12	Houston	101,547	Dothan	Dothan
13	Marshall	93,019	Guntersville	Albertville
14	Lauderdale	92,709	Florence	Florence
15	St. Clair	83,593	Ashville & Pell City	Pell City

Fig. 1. Table from: https://en.wikipedia.org/wiki/Alabama/

```
Alabama Alcoholic Beverage Control Board. Rank !! County !! Population(2010 Census) !! Seat !! Largest city
1 Jefferson 658,466 Birmingham Birmingham 2 Mobile 412,992 Mobile Mobile 3 Madison 334,811 Huntsville
Huntsville 4 Montgomery 229,363 Montgomery Montgomery 5 Shelby 195,085 Columbiana Hoover (part)Alabaster 6
Tuscaloosa 194,656 Tuscaloosa Tuscaloosa 7 Baldwin 182,265 Bay Minette Daphne 8 Lee 140,247 Opelika Auburn
9 Morgan 119,490 Decatur Decatur 10 Calhoun 118,572 Anniston Anniston 11 Etowah 104,303 Gadsden Gadsden 12
Houston 101,547 Dothan Dothan 13 Marshall 93,019 Guntersville Albertville 14 Lauderdale 92,709 Florence
Florence 15 St. Clair 83,593 Ashville &Pell City Pell City Politics During Reconstruction following the
```

Fig. 2. Representation of the table in Figure 1 in the dump

To get the tables with a kind of annotation that enables recognizing the structure of the table, the reading approach stated in Algorithm 1 is used.

It is iterated in line 3 over the lines of the given dump, because the articles in the given dump are separated by line separator, i.e. each line represents an article. The procedure getLink, in line 4, detects the URL at the biginning of the given line and gives it back, because according to the convention used in

[1] https://www.w3.org/TR/turtle/

Algorithm 1 Read&Preprocess(*Dump*)

```
1: ListOfURLS ← new empty list
2: ListOfTables ← new empty list
3: for each line ∈ Dump do
4:     add(ListOfURLS, getLink(line))
5: for each url ∈ ListOfURLS do
6:     Page ← fetchOverHttp(url)
7:     for each table ∈ Page do
8:         splitMerged(table)
9:         if getCountOfRows(table) ≥ 2 then
10:            add(ListOfTables, convertToJSON(table))
11: return ListOfTables
```

the dump, the URL of each article is found at the beginning of the its line. The procedure splitMerged, in line 8, resplits the merged cells in the given table by repeating the tags <tr> and <td> that have the attributes rowspan and colspan as many times as the given value of the attribute. This is outlined on an example in Figure 3.

The open-source Java library jsoup [2] is used to parse the html code of the fetched pages and to detect and navigate in the tables in the page. The open-source Java library Gson [3] is used to encode the tables as JSON objects (line 10) and to decode them later in Algorithm 2.

P.S. Theoretically the tables need not be encoded as JSON Strings and then decoded again, because the output of Algorithm 1 is the input of Algorithm 2. But due to practical reasons, both algorithms have been executed separately. Therefore, the tables had to be stored locally and then read again by the second algorithm.

Year	Film	Art director(s)
1927/28	The Dove	William Cameron Menzies
1927/28	Tempest	William Cameron Menzies
1927/28	7th Heaven	Harry Oliver
1927/28	Sunrise: A Song of Two Humans	Rochus Gliese
1928/29	The Bridge of San Luis Rey	Cedric Gibbons
1928/29	Alibi	William Cameron Menzies
1928/29	The Awakening	William Cameron Menzies
1928/29	Dynamite	Mitchell Leisen
1928/29	The Patriot	Hans Dreier
1928/29	Street Angel	Harry Oliver

⇨

Year	Film	Art director(s)
1927/28	The Dove	William Cameron Menzies
	Tempest	
	7th Heaven	Harry Oliver
	Sunrise: A Song of Two Humans	Rochus Gliese
1928/29	The Bridge of San Luis Rey	Cedric Gibbons
	Alibi	William Cameron Menzies
	The Awakening	
	Dynamite	Mitchell Leisen
	The Patriot	Hans Dreier
	Street Angel	Harry Oliver

Fig. 3. Demonstration of procedure splitMerged

2.2 Relation Extraction

The approach of extracting relations based on the tables returned by Algorithm 1 is shown in Algorithm 2

[2] https://jsoup.org/
[3] https://github.com/google/gson/

Algorithm 2 ExtractRelations(*ListOfTables*,*ns*)

Input: *ListOfTables:* List of tables, *ns:* Own namespace
Output: List of RDF triples

ListOfTriples ← new empty list
for each *table* ∈ *ListOfTables* **do**
    String *Table*[*row*][*col*] ← convertFromJSON(*table*)
    for i=2 to *row* **do**
        for j=1 to *col* **do**
            URI ← disambiduate(*Table*[*i*][*j*])
            if *URI* is valid DBPedia URI **then** *Table*[*i*][*j*]←*URI*
    i=1
    while (*Table*[2][*i*] is Date or Number) **and** (i≤*col*) **do**
        i ← *i*+1
    if *i*≤*col* **then**
        j←*i*
        for i=1 to *row* **do**
            for k=1 to *col* **do**
                if *j*≠*k* **then**
                    if *Table[i][k]* and *Table[i][j]* are URIs **then**
                        $JK \leftarrow$ checkEntityEntity($T[i][j], T[1][k], T[i][k]$)
                        $KJ \leftarrow$ checkEntityEntity($T[i][k], T[1][j], T[i][j]$)
                        if $JK \neq \emptyset$ **then**
                            for each $p \in JK$ **do**
                                addTrip(*ListOfTriples*, $T[i][j]$,p,$T[i][k]$)
                        else
                            addTrip(*ListOfTriples*, $T[i][j]$,*ns*,$T[1][k]$,$T[i][k]$)
                        if $KJ \neq \emptyset$ **then**
                            for each $p \in KJ$ **do**
                                addTrip(*ListOfTriples*, $T[i][k]$,p,$T[i][j]$)
                        else
                            addTrip(*ListOfTriples*, $T[i][k]$,*ns*,$T[1][j]$,$T[i][j]$)
                    else if *Table[i][k]* is not URI and *Table[i][j]* is URI **then**
                        $JK \leftarrow$ checkEntityLiteral($T[i][j], T[1][k]$)
                        if $JK \neq \emptyset$ **then**
                            for each $p \in JK$ **do**
                                addTrip(*ListOfTriples*, $T[i][j]$,p,$T[i][k]$)
                        else
                            addTrip(*ListOfTriples*, $T[i][j]$,*ns*,$T[1][k]$,$T[i][k]$)
return *ListOfTriples*

The algorithm iterates over all the tables and tries to extract knowledge by performing three main steps; **1)** Disambiguating 2.2.1 (lines 4-7) **2)** Identifying the Subject/Object column 2.2.2 (lines 8-10) **3)** Extracting relations 2.2.3 (lines 11-35).

The tables are decoded from JSON String to a 2-dimensional array, in line 3, using the open-source Java library Gson [4].

The open-source Java framework RDF4J [5] is used to process the generated RDF data.

Our own namespace is set to: `"http://bkgft.upb.de"`, i.e Algorithm 2 has been started with *ns*=`"http://bkgft.upb.de"`

2.2.1 Disambiguating The cells are disambiguated and linked to DBPedia URIs, in line 6, using the API of the Open Source Named Entity Disambiguation Framework AGDISTIS [6]. This is shown in the example in Table 1.

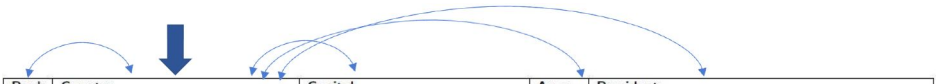
World rank	Country	Capital	Area	President
17	Germany	Berlin	357386	Frank-Walter Steinmeier
34	France	Paris	640679	Emmanuel Macron
52	Lebanon	Beirut	10452	Michel Aoun
56	Ecuador	Quito	283561	Lenín Moreno

⇓

Rank	Country	Capital	Area	President
17	http://dbpedia.org/resource/Germany	http://dbpedia.org/resource/Berlin	357386	http://dbpedia.org/resource/Frank-Walter_Steinmeier
34	http://dbpedia.org/resource/France	http://dbpedia.org/resource/Paris	640679	http://dbpedia.org/resource/Emmanuel_Macron
52	http://dbpedia.org/resource/Lebanon	http://dbpedia.org/resource/Beirut	10452	Michel Aoun
56	http://dbpedia.org/resource/Ecuador	http://dbpedia.org/resource/Quito	283561	Lenín Moreno

Table 1. Disambiguation

2.2.2 Identifying the Subject/Object column Subject/Object column refers to the main column of the table so that either the subject or the object in each extracted triple must be from it. For example, the subject/object column of Table 1 is the column Country because each extracted triple would represent a fact about a country. Therefore, either the subject or the object of each triple extracted from this table has to be from the column Country. In other words, there are pairwise relations between France and Paris, France and 640679, France and Emmanuel Macron, but there are no relations between Paris and Emmanuel Macron or between Paris and 640679. This is outlined in Figure 4.



Rank	Country	Capital	Area	President
17	http://dbpedia.org/resource/Germany	http://dbpedia.org/resource/Berlin	357386	http://dbpedia.org/resource/Frank-Walter_Steinmeier
34	http://dbpedia.org/resource/France	http://dbpedia.org/resource/Paris	640679	http://dbpedia.org/resource/Emmanuel_Macron
52	http://dbpedia.org/resource/Lebanon	http://dbpedia.org/resource/Beirut	10452	Michel Aoun
56	http://dbpedia.org/resource/Ecuador	http://dbpedia.org/resource/Quito	283561	Lenín Moreno

Fig. 4. Subject/Object column

[4] https://github.com/google/gson/
[5] http://rdf4j.org
[6] http://aksw.org/Projects/AGDISTIS.html

A simple rule-based approach, picking the left-most column which is not a number or a date, is used (lines 8-10). According to [2], this approach achieves an accuracy of 83%.

2.2.3 Extracting relations Our approach of extracting relations (lines 11-35) considers the table header to be the predicate and the cells to be the subjects and objects. Thus, for each row, it is attempted to find a relation pairwise between the cell from the Subject/Object column and each other cell from the other columns.

For example, in row 2 in Table 1 the following pairs of cells are checked: (France, 34), (France, Paris), (France, 640679), (France, Emmanuel Macron). It is checked at the beginning whether the relation can be linked to URIs in DBPedia. If possible, triples using the found DBPedia URIs as predicates are added to the graph. Otherwise, triples using the labels from the table header (under our namespace) are added to the graph.

With regard to this, we differentiate two types of checking; 1) Entity-Entity check (when both cells are linked to URIs), 2) Entity-Literal check (when one of the cells is a literal or could not be linked to a URI).

In the following this will be explained on Table 1 as an example.

2.2.4 Entity-Entity Check As an example of an Entity-Entity check the pair (France, Paris) will be considered, as shown below:

Rank	Country	Capital	Area	President
17	http://dbpedia.org/resource/Germany	http://dbpedia.org/resource/Berlin	357386	http://dbpedia.org/resource/Frank-Walter_Steinmeier
34	http://dbpedia.org/resource/France	http://dbpedia.org/resource/Paris	640679	http://dbpedia.org/resource/Emmanuel_Macron
52	http://dbpedia.org/resource/Lebanon	http://dbpedia.org/resource/Beirut	10452	Michel Aoun
56	http://dbpedia.org/resource/Ecuador	http://dbpedia.org/resource/Quito	283561	Lenín Moreno

The following SPARQL Query is sent to DBPedia:

```
SELECT ?relation WHERE
{<http://dbpedia.org/resource/France> a ?typeOfSubject.
?relation rdfs:domain ?typeOfSubject.
?relation rdfs:range ?typeOfObject.
<http://dbpedia.org/resource/Paris> a ?typeOfObject.
?relation rdfs:label ?lab.
FILTER(lang(?lab)="en").
FILTER regex(?lab, "^Capital$", "i" ).}
```

The types of `dbr:France` and `dbr:Paris` are found and it is checked whether an ontology exists whose domain a type of `dbr:France` is, whose range a type of `dbr:Paris` is, and whose label matches with the label in the table header. This is outlined in Figure 5 If such an ontology exists, assume `dbo:capital`, then the following triple is added to the graph:

```
dbr:France dbo:capital dbr:Paris.
```

If such an ontology does not exist in DBPedia, then capital will be added as a predicate under our own namespace (`http://bkgft.upb.de`). Thus, the following triples are added:

```
dbr:France <http://bkgft.upb.de/capital> dbr:Paris.
<http://bkgft.upb.de/capital> rdfs:label "Capital"@en.
```

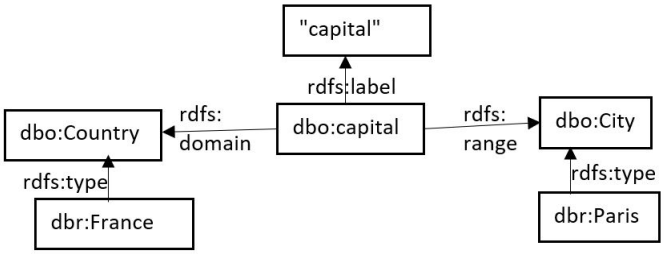


Fig. 5. Outline of Entity-Entity DBPedia Check (a)

Analogically, the same is performed once again considering Country as a predicate, `dbr:Paris` as a subject and `dbr:France` as an object, as shown below:

Rank	Country	Capital	Area	President
17	http://dbpedia.org/resource/Germany	http://dbpedia.org/resource/Berlin	357386	http://dbpedia.org/resource/Frank-Walter_Steinmeier
34	http://dbpedia.org/resource/France	http://dbpedia.org/resource/Paris	640679	http://dbpedia.org/resource/Emmanuel_Macron
52	http://dbpedia.org/resource/Lebanon	http://dbpedia.org/resource/Beirut	10452	Michel Aoun
56	http://dbpedia.org/resource/Ecuador	http://dbpedia.org/resource/Quito	283561	Lenín Moreno

The following SPARQL Query is sent to DBPedia:

```
SELECT ?relation WHERE
{<http://dbpedia.org/resource/Paris> a ?typeOfSubject.
?relation rdfs:domain ?typeOfSubject.
?relation rdfs:range ?typeOfObject.
<http://dbpedia.org/resource/France> a ?typeOfObject.
?relation rdfs:label ?lab.
FILTER(lang(?lab)="en").
FILTER regex(?lab, "^Country$", "i" ).}
```

This is outlined in Figure 6

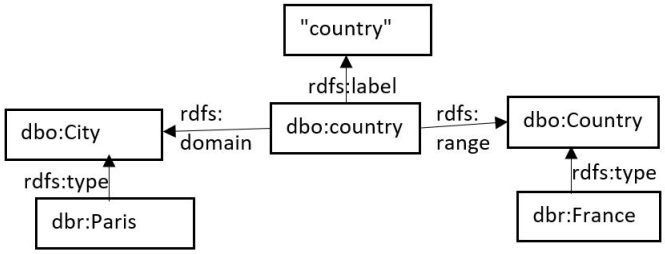


Fig. 6. Outline of Entity-Entity DBPedia Check (b)

In this case, either the following triple will be added:

```
dbr:Paris dbo:country dbr:France.
```

Or the following:

```
dbr:Paris <http://bkgft.upb.de/country> dbr:France.
<http://bkgft.upb.de/country> rdfs:label "Country"@en.
```

2.2.5 Entity-Literal Check As an example of an Entity-Literal check the pair (France, 640679) will be considered, as shown below:

Rank	Country	Capital	Area	President
17	http://dbpedia.org/resource/Germany	http://dbpedia.org/resource/Berlin	357386	http://dbpedia.org/resource/Frank-Walter_Steinmeier
34	http://dbpedia.org/resource/France	http://dbpedia.org/resource/Paris	640679	http://dbpedia.org/resource/Emmanuel_Macron
52	http://dbpedia.org/resource/Lebanon	http://dbpedia.org/resource/Beirut	10452	Michel Aoun
56	http://dbpedia.org/resource/Ecuador	http://dbpedia.org/resource/Quito	283561	Lenín Moreno

The following SPARQL Query is sent to DBPedia:

```
SELECT ?relation WHERE
{<http://dbpedia.org/resource/France> a ?typeOfSubject.
?relation rdfs:domain ?typeOfSubject.
?relation rdfs:range ?typeOfObject.
?typeOfObject a rdfs:Datatype.
?relation rdfs:label ?lab.
FILTER(lang(?lab)="en").
FILTER regex(?lab, "^Area$", "i" ).}
```

The type of `dbr:France` is found and it is checked whether an ontology exists whose domain a type of `dbr:France` is, whose range a datatype is, and whose label matches with the label in the table header. This is outlined in Figure 7

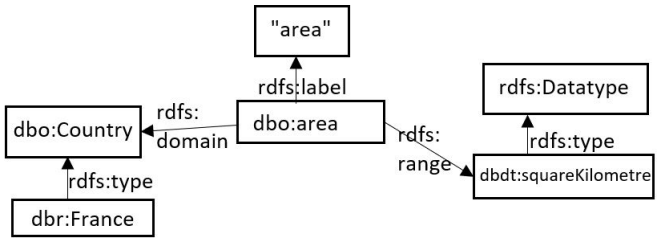


Fig. 7. Outline of Entity-Literal DBPedia Check

If such an ontology exists, assume `dbo:area`, then the following triple is added to the graph:

```
dbr:France dbo:area "640679".
```

If such an ontology does not exist in DBPedia, then area will be added as a predicate under our own namespace (`http://bkgft.upb.de`). Thus, the following triples are added:

```
dbr:France <http://bkgft.upb.de/area> "640679".
<http://bkgft.upb.de/area> rdfs:label "Area"@en.
```

3 Evaluation

By running the algorithm for about 4 hours, 95644 RDF statements have been in total generated. For the evaluation of the outcome, 200 triples have been randomly chosen and manually checked. The 200 triples have been chosen only from the subset of triples that do not have the predicate `rdfs:label` because such triples are always correct.

Definition: Triples that have not been linked correctly to DBPedia but they had to be linked.

Example1: `dbr:Tomáš_Jelínek <http://bkgft.upb.de/nhl-team> dbr:Ottawa_Senators .`

had to be linked to: `dbr:Tomáš_Jelínek dbo:formerTeam dbr:Ottawa_Senators .`

Example2: `dbr:Sonu_Nigam <http://bkgft.upb.de/year> "1992" .` had to be linked to: `dbr:Sonu_Nigam dbp:presentYear "1992" .`

Precision:

– When triples not linked correctly to DBPedia but had to be linked (defined above) are considered as true:

$$P = \frac{\#True}{\#Total} = \frac{36}{200} = 0.18$$

– When triples not linked correctly to DBPedia but had to be linked (defined above) are considered as false:

$$P = \frac{\#True}{\#Total} = \frac{19}{200} = 0.095$$

4 Runtime

- Worst case runtime of Algorithm 1 (Read&Process) = $\mathcal{O}(n\,t\,p(r,c))$, so that:
n: Number of articles in the given Dump.
t: Maximum number of tables in a single article.
r, c: Dimensions of biggest table in the whole dump.
$p(x, y)$: Runtime of procedure splitMerged given a table with dimensions x,y.
- Worst case runtime of Algorithm 2 (ExtractRelations) = $\mathcal{O}(n\,r\,c\,q)$), so that:
n: Number of tables in the given ListOfTables.
r, c: Dimensions of biggest table in ListOfTables.
q: max {Runtime of disambiguate, Runtime of checkEntityEntity, Runtime of checkEntityLiteral}

It is quite difficult to compute the runtime more accurately than above because the runtime of procedures: disambiguate, checkEntityEntity and checkEntityLiteral varies depending on the words being queried and on the network connection. Moreover, both Algorithm 1 and Algorithm 2 are in practice multithreaded to improve the performance with regard to the runtime.

The three procedures mentioned above are the most costly part in our appraoch in terms of time, due to the latency over the internet.

5 Analysis of weakness points

In the following many weakness points of our approach are presented that resulted in incorrectness by the outcome.

5.1 Incorrect entity linking

Due to the absence of context by step 2.2.1, many entities could not be correctly disambiguated and have been linked to incorrect URIs. For example:

- `dbr:Titanic_(District_Electoral_Area) <http://bkgft.upb.de/studios> "Paramount / Fox" .`
  Titanic is linked to the city, not to the movie.
- `dbr:Newark_and_Sherwood <http://bkgft.upb.de/new-route> dbr:U.S._Route_1/9_Truck .`
  Newark is linked to Newark in the UK, not to Newark in New Jersey.
- `dbr:The_Grotesque_(novel) <http://bkgft.upb.de/year> "1995" .`
  The Grotesque is linked to `dbr:The_Grotesque_(novel)`, not `dbr:The_Grotesque_(film)`.
- `dbr:Bikstok_Røgsystem <http://bkgft.upb.de/atomic-number> "92" .`
  92 is the atomic number of the chemical element Uranium, but the subject has been linked to the Danish dancehall band that has an album named Uranium!

5.2 Mistakes by dissolving merged cells

Dissolving colspans and rowspans naively (procedure splitMerged in Algorithm 1) by resplitting the cells results in some mistakes: For example:

Year	Film	Art director(s)	Interior decorator(s)
1945	**Black-and-White**		
	Blood on the Sun	Wiard Ihnen	A. Roland Fields
	Experiment Perilous	Albert S. D'Agostino and Jack Okey	Darrell Silvera and Claude E. Carpenter
	The Keys of the Kingdom	James Basevi and William S. Darling	Thomas Little and Frank E. Hughes
	Love Letters	Hans Dreier and Roland Anderson	Samuel M. Comer and Ray Moyer
	Color		
	Frenchman's Creek	Hans Dreier and Ernst Fegté	Samuel M. Comer
	Leave Her to Heaven	Lyle R. Wheeler and Maurice Ransford	Thomas Little
	San Antonio	Ted Smith	Jack McConaghy
	A Thousand and One Nights	Stephen Goosson and Rudolph Sternad	Frank Tuttle

⇨

Year	Film	Art director(s)	Interior decorator(s)
1945	**Black-and-White**	**Black-and-White**	**Black-and-White**
1945	*Blood on the Sun*	Wiard Ihnen	A. Roland Fields
1945	*Experiment Perilous*	Albert S. D'Agostino and Jack Okey	Darrell Silvera and Claude E. Carpenter
1945	*The Keys of the Kingdom*	James Basevi and William S. Darling	Thomas Little and Frank E. Hughes
1945	*Love Letters*	Hans Dreier and Roland Anderson	Samuel M. Comer and Ray Moyer
1945	**Color**	**Color**	**Color**
1945	*Frenchman's Creek*	Hans Dreier and Ernst Fegté	Samuel M. Comer
1945	*Leave Her to Heaven*	Lyle R. Wheeler and Maurice Ransford	Thomas Little
1945	*San Antonio*	Ted Smith	Jack McConaghy
1945	*A Thousand and One Nights*	Stephen Goosson and Rudolph Sternad	Frank Tuttle

Fig. 8. Dissolving merged cells

This leads to meaningless triples such as:

```
Black-and-White Art-director(s) Black-and-White .
Color Year "1945" .
```

5.3 "1 Cell = 1 Entity" Problem

Assuming that every cell contains only one entity results in some inaccuracies. For example, from the table shown in Figure 8 above, the following triple would be extracted:

```
dbr:A_Thousand_and_One_Nights_(1945_film) dbo:director "Stephen
Goosson and Rudolph Sternad".
```

Whereas the following would be more correct:

```
dbr:A_Thousand_and_One_Nights_(1945_film) dbo:director
dbr:Stephen_Goosson.
dbr:A_Thousand_and_One_Nights_(1945_film) dbo:director
dbr:Rudolph_Sternad.
```

Statistic[29]	Brazil	Germany
Goals scored	1	7
Total shots	18	14
Shots on target	8	10
Ball possession	52%	48%
Corner kicks	7	5
Fouls committed	11	14
Offsides	3	0
Yellow cards	1	0
Red cards	0	0

Fig. 9. Table from https://en.wikipedia.org/wiki/Brazil_v_Germany_(2014_FIFA_World_Cup)

5.4 Diversified table structures

The structures of the tables are not always so homogeneous that a subject/object column exists, the table header represents the predicates and each row represents facts about the entity in the cell of subject/object column. For example:

Using our approach, meaningless triples such as the following could be extracted from the table in Figure 9 :

```
goals-scored Brazil "1".
total-shots Germany "14".
... etc
```

5.5 Simple label matching

Simple label matching from table headers is not a good idea for many reasons. Examples:

– Looking for ontologies in DBPedia that represent the extracted relations is carried out by means of a simple non-case-sensitive label matching from the table header, which fails in many cases, although such ontologies do exist, due to different words that refer to similar concepts. This is clearly shown

Fig. 10. Simple Matching problem

in Figure 10. The ontology `dbo:director` could not be found, because its label "film director" does not match the table header "Art director(s)". This leads to introduce "Art director(s)" as a new ontoloy under our namespace instead of using `dbo:director`.

– Sometimes the table header contains only abbreviations such as the table in Figure 11

Season-by-season record [edit]

Main article: List of Minnesota North Stars seasons
See also: List of Dallas Stars seasons

The team had 17 playoff appearances, a 77-82 playoff record, 2 Norris Division championships, and 2 Campbell Conference chan

Note: *GP = Games played, W = Wins, L = Losses, T = Ties, Pts = Points, GF = Goals for, GA = Goals against, PIM = Penalties in*

Season	GP	W	L	T	Pts	GF	GA	PIM	Finish	Playoffs
1967–68	74	27	32	15	69	191	226	738	fourth, West	**Won** Quarterfinals (Kings) 4-3 Lost Semifinals (Blues) 4-3
1968–69	76	18	43	15	51	189	270	862	sixth, West	*Out of playoffs*
1969–70	76	19	35	22	60	224	257	1,008	third, West	Lost Quarterfinals (Blues) 4-2
1970–71	78	28	34	16	72	191	223	898	fourth, West	**Won** Quarterfinals (Blues) 4-2 Lost Semifinals (Canadiens) 4-2
1971–72	78	37	29	12	86	212	191	853	second, West	Lost Quarterfinals (Blues) 4-3

Fig. 11. Abbreviations in table header

– The semantics of the table content is sometimes strongly related to the context of the article and no meaningful relations can be figured out by depending only on the table itself and omitting the article.
For example, the table on the left-hand-side in Figure 12 comes from the wikipedia article Dollar and the one on the right-hand-side comes from 2018 FIFA World Cup Final.
The following triple would be extracted from the left table:

`Canada established 1858.`

However, it makes no sense because without considerung the article, it can not be known, what established Canada in 1858.
Likewise, no meaningful knowledge can be extracted from the right table isolated from the context of its article, because the information in the table refers to this exact football match, i.e. the info without the match that it describes is meaningless.

Economies that use a dollar [edit]

Countries	Currency	ISO 4217 code	Established	Preceding currency
Antigua and Barbuda	East Caribbean dollar	XCD	1965	West Indies dollar
Australia and its territories	Australian dollar	AUD	1966-02-14	Australian pound 1910-1966 Pound sterling 1825-1910
Bahamas	Bahamian dollar	BSD	1966	Bahamian pound
Barbados	Barbadian dollar	BBD	1935	
Belize	Belize dollar	BZD/USD	1973	British Honduran Dollar
Bermuda	Bermuda dollar	BMD	1970	Pound sterling
Brunei	Brunei dollar (Alongside the Singapore dollar)	BND (SGD)	1967	Malaya and British Borneo dollar
Canada	Canadian dollar	CAD	1858	Canadian pound 1841-1858 Spanish dollar pre-1841 Newfoundland dollar, 1865 – 1949 in the Dominion of Newfoundland

Statistics

First half[101]

Statistic	France	Croatia
Goals scored	2	1
Total shots	1	7
Shots on target	1	1
Saves	0	0
Ball possession	39%	61%
Corner kicks	1	4
Fouls committed	8	7
Offsides	1	0
Yellow cards	2	0
Red cards	0	0

Fig. 12. Tables with no meaning without the embedding articles

References

1. P. Venetis, A. Halevy, J. Madhavan, M. Pa¸sca, W. Shen, F. Wu, G. Miao, and C. Wu. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, 4(9):528–538, 2011.
2. A. Ngonga Ngomo and I. Ermilov. TAIPAN: Automatic Property Mapping for Tabular Data. `https://svn.aksw.org/papers/2016/EKAW_Taipan/public.pdf`