MantisTable V: a novel and efficient approach to Semantic Table Interpretation

Roberto Avogadro[0000−0001−8074−7793] and
Marco Cremaschi[0000−0001−7840−6228]

University of Milano - Bicocca
{roberto.avogadro,marco.cremaschi}@unimib.it

Abstract. In this paper, we present MantisTable V, a novel unsupervised and automatic approach for the Semantic Table Interpretation. The approach is performed against DBpedia and Wikidata, and it can be easily adapted to any other Knowledge Graph. Moreover, we provide a tool (LamAPI) that allows to efficiently fetch data needed for Semantic Table Interpretation tasks from the Knowledge Graph dumps. The approach is manageable through a User Interface (tUI), a separated tool, which allows the visualisation and modification of table data and semantic annotations.

Keywords: Semantic Web · Knowledge Graph · Semantic Table Interpretation · Table Understanding · DBpedia · Wikidata · User Interface

1 Introduction

The Semantic Table Interpretation (STI) is a research field in continuous evolution with increasing interest over time, also considering the great diffusion of tabular data on the web. The input of STI is: i) a *well-formed and normalised* relational table (i.e., a table with headers and simple values, thus excluding nested and figure-like tables), as the one in Fig. 1, and ii) a *Knowledge Graph (KG)* which describes real-world entities in the domain of interest (i.e., a set of concepts, datatypes, predicates, instances, and the relations among them), as the example in Fig. 2. The output returned is a semantically annotated table, as shown in Fig. 3.

Moreover, the STI process is composed of the following main annotation steps: i) *semantic classification of columns*, which takes into account the values of a column to mark it as *Literal column (L-column)* if values are datatypes (e.g., strings, numbers, dates, etc., such as 2015, 10/04/1983, etc.), or as *Named-Entity column (NE-column)* if values are concepts (e.g., Film, Director, etc., such as Jurassic_World, Colin_Trevorrow, etc.); ii) *detection of the subject column (S-column)*, which identifies the main column (the one all the others are referring

to) among the NE-columns identified in the previous step (e.g., the Title column in Fig. 3); iii) *concept and datatype annotation*, which associates NE-columns with a concept in the KG (e.g., the column Title is associated with `Film` in Wikidata[1]), and L-columns with a datatype in the KG (e.g., the column Year is of type `date`); and iv) *predicate annotation*, which identifies the relations between the S-column and the other columns (e.g., `Film publication_date Year`).

Each of the above steps is obtained by annotating column values referring to existing KGs. For example, in Fig. 3 if the majority of entities in the Title column is associated with `Film`, these entities are of type `Film`. Similarly, `publication_date` can be identified as the predicate connecting entities in the Title column with datatypes of type `date` of the Year column.

Unfortunately, explicit situations like the ones in the example are not so common, therefore we need to set up strategies and algorithms to address several issues.

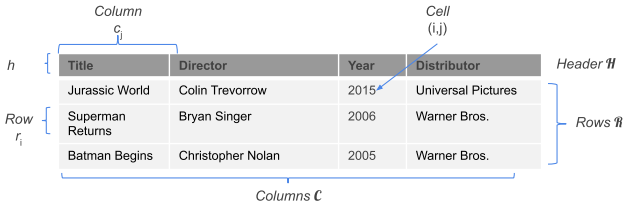

Fig. 1. Example of a well-formed relational table, with labels that are used in this paper.

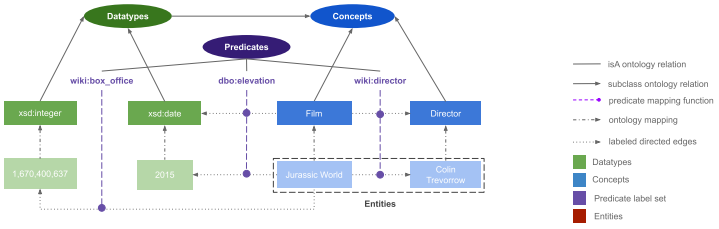

Fig. 2. A sample of Knowledge Graph.

This work is an improvement and extension of MantisTable (se*Mantics Table*) [2] and MantisTable SE [1]. We will refer to our new approach with "MantisTable V", the 5th Open Source implementation of MantisTable. Compared to MantisTable, MantisTable V is characterised by a complete refactoring due to a substantial modification of the annotation process, now no longer procedural but

[1] www.wikidata.org/wiki/Q11424

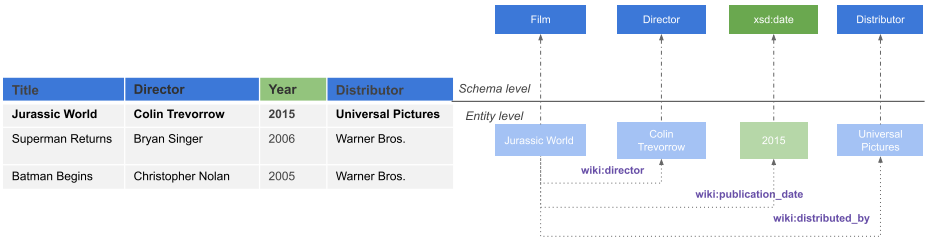

Fig. 3. Example of an annotated table.

iterative, to improve the cell disambiguation process. Compared to MantisTable SE, this new version of the approach can consider different types of tables (concerning the number of columns) and uses new algorithms to identify and classify L-columns. It also manages ambiguity in annotations optimally, as will be analysed in the following sections. Our approach uses DBpedia and Wikidata as the first matching Knowledge Graphs (KGs) because they are the richest data sources with ground truths available. MantisTable V can be easily adapted to be used with any KG, through the use of a new tool, called LamAPI (*La*bel *m*atching *API*). Thanks to LamAPI and its index systems, it is possible to efficiently search a particular entity by its ID or full-text search. Together with MantisTable V and LamAPI, we have developed a user interface (tUI) that allows the management of the tables and the annotations, as well as the update. tUI (*t*able *U*ser *I*nterface) is a fully configurable tool, which can be used with any STI approach.

The main contributions of this paper are: (i) MantisTable V, a comprehensive approach which deals with all phases of the STI process, (ii) LamAPI, an open-source tool to efficiently manage and retrieve data of KGs, (iii) tUI, a fully configurable open-source UI to manage and display and update tables and semantic annotations. All tools have been encapsulated in Docker containers to facilitate the deployment and scalability by replication.

The remainder of the paper is organised as follow: in Section 2 we describe the functionalities of the LamAPI tool while in Section 3 MantisTable V is described. Details on tUI are depicted in Section 4. Section 5 introduces the Gold Standards and discusses the evaluation results. Finally, conclusions and pointers, are presented in Section 6.

2 Data management for an efficient STI with LamAPI

As seen in Section 1, to obtain the STI of tabular data, it is required to link elements of the table with the elements of a KG. The elements in the KGs (e.g., DBpedia or Wikidata) are frequently stored in Resource Description Format (RDF) format, so to access these elements, it is necessary to query a SPARQL endpoint. For instance, the most popular way to access DBpedia dumps is by using OpenLink Virtuoso, a row-wise transaction-oriented RDBMS with a SPARQL query

engine to access to RDF graph store[2]. Wikidata instead uses Blazegraph[3] that is a high-performance graph database supporting RDF/SPARQL APIs. The issue faced with these solutions is the time required for importing the data. Wikidata 2019 dump requires some days to set up[4]. Another problem is given by the amount of information present in a KG; for instance, the Wikidata dump is about 1.1TB (uncompressed). The English version of DBpedia instead is split into multiple files of the size of 26GB, which leads to high computation times to obtain a complete STI (e.g., TableMiner+ according to the author in [8] took 13.35 hours to process the Limaye200 dataset). However, not all the information present in a KG is necessary to carry out a STI.

Therefore, in order to obtain an efficient approach, it is necessary to identify other ways for querying KG. To do that, :n the state of the art, two works can be identified, FactBase [4] and Knowledge Graph ToolKit (KGTK) [5]. FactBase index[5] introduce a manually and generic search index over Wikidata entries. FactBase index takes the cells of a table column as input and returns the top-k candidate entities for each cell. The KGTK framework [6] is used for the creation and exploitation of large KGs, such as Wikidata. However, the authors suggest using only parts of a KG. The approach described in this paper does not use SPARQL queries but queries indexes built on the *entire* DBpedia and Wikidata. These indexes are accessible through the use of four different API services.

The open-source tool that provides these APIs is called LamAPI (*La*bel *m*atching *API*)[7] and provides the following services:

1. **Lookup**: given a free text (in this case a data inside the table cell), it retrieves the entities with the greatest similarity, using the IB similarity[8] scoring algorithm of ElasticSearch which combines different search strategies (i.e., full-text search based on tokens, on n-grams and fuzzy search). The ElasticSearch contains an index of the KG entities to improve the performance. Considering a table cell containing "Jurassic World" the result returned is shown in Listing 1.1 or Listing 1.2;
2. **Concepts**: given an entity it retrieves all its concepts as shown in Listing 1.3. This service can extend automatically the list of concepts associated with a given entity through the use of vector similarity measures between the different concepts in the KG. Thanks to this functionality, it is possible to extend the candidates associated with a cell.
3. **Literals**: given an entity it retrieves all the related literals values and predicates as shown in Listing 1.3;

[2] virtuoso.openlinksw.com
[3] blazegraph.com
[4] addshore.com/2019/10/your-own-wikidata-query-service-with-no-limits-part-1/
[5] www.cs.toronto.edu/ oktie/webtables
[6] github.com/usc-isi-i2/kgtk/
[7] bitbucket.org/disco_unimib/lamapi/
[8] www.elastic.co/guide/en/elasticsearch/reference/current/index-modules-similarity.html

4. **Predicates**: given two entities it retrieves all the predicates between them; considering the entity "Jurassic_World" and the entity "Colin_Trevorrow", the list of predicates is shown in Listing 1.4;
5. **Objects**: given an entity it retrieves all the related objects and predicates; for example, with the entity "Jurassic_World" the result is the one shown in Listing 1.4.

Listing 1.1. Wikidata lookup.

```
"id": "Q3512046"
"name": "Jurassic World"
"types": {
    "id": "Q229390"
    "name": "3D film"
    },{
    "id": "Q11424"
    "name": "film"
    }
"id": "Q21877685"
"name": "Jurassic World"
"types": {
    "id": "Q3512046"
    "name": "Jurassic World"
    },{
    "id": "Q3512046"
    "name": "Jurassic World"
    }
```

Listing 1.2. DBpedia lookup.

```
"id": "Jurassic_World"
"name": "Jurassic World"
"types": {
    "id": "Film"
    "name": "Film"
    },{
    "id": "Work"
    "name": "Work"
    }
"id": "Jurassic_Park"
"name": "Jurassic World 2"
"types": {
    "id": "Film"
    "name": "Film"
    },{
    "id": "SportsTeam"
    "name": "SportsTeam"
    }
```

The data in DBpedia[9] have been preprocessed to be then integrated into LamAPI.

Listing 1.3. Query result - Concepts and Literals.

```
Concepts
"Jurassic_World":
    "rdf:type":
        "Film"
        "Work"
Literals
"Jurassic_World":
    "number":
        "dbo:budget":
            "1.5E8"
        "dbo:gross":
            "1.67E9"
        "dbo:runtime":
            "7440.0"
    "string":
        "foaf:name":
            "Jurassic World"
```

Listing 1.4. Query results - Predicate and Objects.

```
Predicate
"Jurassic_World Colin_Trevorrow":
    "dbo:director"

Object
"Jurassic_World":
    "Colin_Trevorrow":
        "dbo:director"
    "Michael_Giacchino":
        "dbo:musicComposer"
    "John_Schwartzman":
        "dbo:cinematography"
    "Kevin_Stitt":
        "dbo:editing"
    "Universal_Studios":
        "dbo:distributor"
```

Differently from DBpedia, Wikidata offers every week a new single dump file of large dimensions[10]. For Wikidata, we had to make a different design decision in order to support multi-language.

This new way to access DBpedia/Wikidata provided by LamAPI overcame the limitations of SPARQL endpoints such as:

- SPARQL endpoint response times are directly proportional to the size of the returned data. In this context, sometimes it is not even possible to get a result because of the endpoint returning a timeout;
- the volume of requests per second is limited (online endpoint) or computationally expansive (a local endpoint requires at least 64GB of ram and tons of CPU cycles);

[9] wiki.dbpedia.org/downloads-2016-10
[10] dumps.wikimedia.org/wikidatawiki/entities/

– there are some intrinsic limits in the SPARQL language expressiveness (i.e., the full-text search capability that is useful for label matching is possible to be obtained only with extremely slow "contains" or "regex" queries[11]).

LamAPI is developed using ElasticSearch, MongoDB and Python[12].

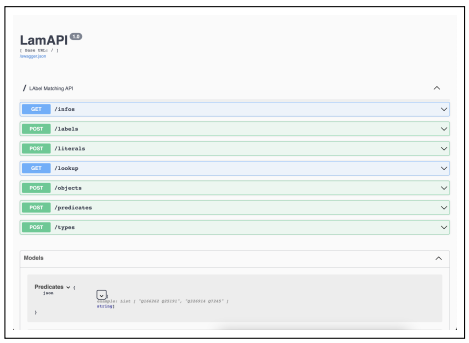

Fig. 4. LamAPI documentation page with Swagger.

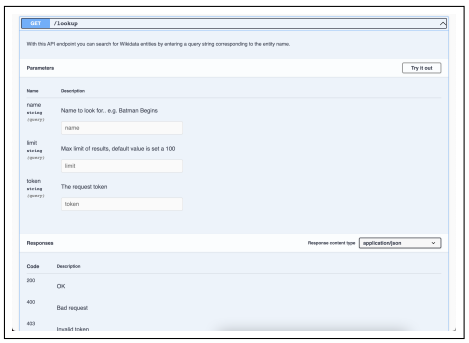

Fig. 5. Documentation of LamAPI Lookup service.

3 The MantisTable V approach

This Section will focus on the algorithmic process of MantisTable V[13]. The process is organised into eight phases as follows: i) Data Preparation and Normalisation, ii) Column Analysis and Subject Detection, iii) Cell Entity Annotation (CEA), iv) Column Predicate Annotation (CPA), v) Column Type Annotation (CTA), vi) Revision and viii) Export.

The process was designed to retrieve the candidate for a given cell only once, especially if the content of that cell is repeated across multiple tables. This allows the approach to avoid repeating queries for the same content and saves network time. Every phase must be completed for all the tables in a dataset before going to the next one. This does not preclude running the tool against only one table, but when running with multiple tables, the execution time will be sharply reduced (for the exact text in a cell, we have the same candidate entities that will be sorted considering the content of every table. This will be explained more in detail in Paragraph 3). To describe each phase of the STI approach, consider Table 1, which lists some movies with additional information, such as director and release year.

[11] docs.openlinksw.com/virtuoso/rdfsparqlrulefulltext/
[12] lamapi.ml
[13] bitbucket.org/disco_unimib/mantistable-v/

Table 1. Illustrative movies table that will be used for examples.

title	director	release year	domestic distributor	length in min	worldwide gross
Jurassic World	Colin Trevorrow	2015	Universal Pictures	124	1,670,400,637
Superman Returns	Bryan Singer	2006	Warner Bros.	154	391,081,192
Batman Begins	Christopher Nolan	2005	Warner Bros.	140	371,853,783
Avatar	James Cameron	2009	Twentieth Century Fox	162	2,744,336,793

i. Data Preparation and Normalisation. During this phase, all tables' cells are analysed using a tokeniser managing special characters and additional spaces. For each normalised cell, the candidate entities are retrieved from the **Lookup** service of LamAPI. The obtained candidates will be ranked during the next phases.

ii. Column Analysis and Subject Detection. During *Column Analysis* we identify literal columns (L-column) by using a set of Regextypes [2] (i.e., boolean, date, email, geocoords, integer, float, ISBN, URL, XPath, CSS) to identity different datatypes. If the number of occurrences of the most frequent Regextypes detected exceeds a given threshold, the column will be annotated as L-column and the most frequent Regextype will be assigned to the column under analysis; otherwise, the column will be annotated as NE-column. The subject column (S-column) can be identified between the Named Entity columns (NE-columns) thanks to content-based scores, but it will not be discussed in this approach as it would not introduce anything new from [2]. Related to the example about films (Table 1), the columns *release year*, *length in min*, and *worldwide gross* are tagged as L-column. *Director* and *domestic distributor* are NE-columns. *Title* is the S-column.

iii. Cell Entity Annotation (CEA). In the first step of this phase, the approach performs the entity-linking on NE-columns by searching the LamAPI, using the **Lookup** service, with the content of a cell $tx(i, j)$. The content of the cell $tx(i, j)$ and the candidate entities $E_{i,j} \subseteq E$ are used to disambiguate the content of the cell by considering the degree of similarity. For each cell a confidence score is calculated by computing the edit distance (Levenshtein distance) between the labels (in different languages) of candidate entity $e_{i,j} \in E_{i,j}$ and the content of the cell $tx(i, j)$:

$$1 - norm(LevenshteinDistance(tx(i, j), e_{i,j})) \tag{1}$$

All the values are normalised in [0,1] range with Divide by Maximum normalisation (for every entity).

For L-columns, the confidence score is computed as follows:

- for L-columns with numeric datatype (float and integer Regextypes): all the numeric values (object of RDF-triples) linked to candidate entities are taken using the **Literal** service of LamAPI. The confidence score is calculated as of the formula in Equation 2, where $lit(e_{i,j})$ is the numerical values associated to the candidate entity $e_{i,j}$.

$$1 - \frac{|tx(i,j) - lit(e_{i,j})|}{max(|tx(i,j)|, |lit(e_{i,j})|, 1)} \quad (2)$$

- for L-columns with string datatype: the confidence score is computed using the Jaccard distance. We change the similarity measure, particularly for the long string, because the number of edits required to change a long string into another one is not necessarily significant (Edit distance). The Jaccard instead considers n-grams.
- for L-column with date datatype, the dates are considered as sortable numeric values in the format YYYYMMDDHHmmSS. The confidence score is computed as described for the numeric datatype.

Cells with the same content in different tables start considering the same set of candidate entities, but it will be sorted differently concerning the entire table's contents in the next phases. As an example, we consider the cell containing "superman returns" in Table 1. In this case, it is referred to the movie, but if we consider Table 2, it is referred to the video game.

Table 2. Example of a table with a cell identical to the first example but different content.

videogame	publisher	release date
superman returns	electronic arts	2006
pokemon white	nintendo	2010
call of duty	activision	2003

Considering the Table 1 and the cell "Superman Returns", the candidate entities are associated to the ontology concept "film" and "video game" (Listing 1.5). In the CTA phase, where the approach extracts the types (concepts) of entities, all entities associated with the concept "video game" are penalised because the most frequent concept is "film".

Listing 1.5. Candidates for the cell Superman Returns of the movies Table.

```
"Q328695":
    "label": "Superman Returns"
    "instance_of": ["3D film", "film"]
    "confidence": 2.25
"Q655031":
    "label": "Superman Returns",
    "instance_of": ["video game"]
    "confidence": 0.2
"Q3977963":
    "label": "Superman Returns"
    "instance_of": ["album"]
    "confidence": 0.2
```

Instead, when we consider the video games in Table 2 the CPA phase, where the approach extracts the relationship between entities, allowed us to penalise all entities with the concept "film" with the same name because it has few or no relationship with the rest of the content of the row.

Listing 1.6. Candidates for the same cell of the video game table.

```
"Q655031":
    "label": "Superman Returns",
    "instance_of": ["video game"]
    "confidence": 1.0
"Q7643850":
    "label": "Superman Returns: Fortress of Solitude"
    "instance_of": ["video game"]
    "confidence": 0.41
```

Considering literal values for the cell "batman begins" with the content of the column "length in min" we are almost sure that the value is correct because after sorting all the values, we have the result shown in Listing 1.7.

Listing 1.7. Literal values for the entity batman begins (film).

```
"P4632":
    "label": "Bechdel Test Movie List ID"
    "value": 40
    "confidence": 0
"P2047":
    "label": "duration"
    "value": 140
    "confidence": 1.0
"P3110":
    "label": "ISzDb film ID"
    "value": 234
    "confidence": 0
```

iv. Column Predicate Annotation (CPA) Considering that all the necessary information is gathered in the previous phase using LamAPI, the CPA is a relatively fast process. All the predicates previously identified for each column are sorted by their relative frequency to the entire column. The predicate with the greatest frequency will be ranked first. This process allows reducing the number of candidate entities for every cell. Confidence scores for the predicates of the *director* column are shown in Listing 1.8.

When we consider the video games in Table 2, the CPA phase allowed us to penalise the "film" with the same name. This is because it does not have any relationship with the rest of the content of the table (the film does not have anything to do with "electronic arts", while the video game has a property with exact match).

Listing 1.8. Example for CPA.

```
"P57":
    "label": "director"
    "confidence": 1.0
"P58":
    "label": "screenwriter"
    "confidence": 0.75
"P162":
    "label": "producer"
    "confidence": 0.625
"P161":
    "label": "cast member"
    "confidence": 0.25
```

v. Column Type Annotation (CTA) To get the CTA annotation, we collect the types/concept of every $e_{i,j}$ resulting from the CEA. The concept with maximum frequency has been selected for the CTA annotations.

For every column, we collect the frequencies of the concepts, as shown in Listing 1.9 for Wikidata and in Listing 1.10 for DBpedia.

Listing 1.9. Example of the structure storing the most frequent concepts.

```
    "movie_table":
    "0":
        "Q229390 (3D film)": 1,
        "Q11424 (film)": 1,
        "Q25110269 (live-action/animated film)": 0.33
    "1":
        "Q5 (Human)": 1
    "3":
        "Q1762059 (film production company)": 1,
        "Q375336 (film studio)": 0.5,
        "Q1107679 (animation studio)": 0.5,
        "Q18127 (record label)": 0.5,
        "Q4830453 (business)": 0.5,
        "Q10689397 (television production company)": 0.25
```

Listing 1.10. Example of the structure storing the most frequent concepts.

```
  "movie_table":
  "0":
    "Film": 1
    "Work": 1
  "1":
    "Person": 1
  "3":
    "Company": 1
    "Organisation": 1
```

If the system returns many annotations for one column (e.g. column 0 in Listing 1.9, 1.10), the approach randomly selects one of them as the final annotation.

vi. Revision The revision phase analyses all the information gathered in the previous phases to do a final reordering of the candidates. In particular, this allows for correcting CEA entities previously selected: every entity have to be coherent with the rest of the concepts and predicates selected in every column. Moreover, predicates are also re-ranked.

vii. Export The MantisTable V approach previously described keeps the candidates coming from each phase. It is possible to apply some thresholds during the export phase to balance annotation quality and the number of annotations provided. The export threshold can have a significant role in evaluation metrics for every gold standard.

MantisTable V is developed using Python.

4 User Interface of tUI

tUI[14] is a Web application that aims to provide a visualisation tool for STI approaches; it can work with any backend that provides API endpoints to retrieve data. Endpoints are stored in a YAML configuration file: UI will display data and functionalities based on which APIs are available. tUI consists of three main parts: i) the view listing the datasets, ii) the view listing the tables contained in each dataset (Fig. 6), iii) the view showing the table data and the semantic annotations (Fig. 7). Regarding the annotated table view, tUI supports all the three main tasks of the STI (CTA, CPA, CEA): annotations can be viewed directly

[14] bitbucket.org/disco_unimib/tui/

inside the table. Concerning the CEA task, if multiple candidates are retrieved, all of them can be shown in the UI. Endpoints to retrieve the datasets list, the tables list and the table data (with or without annotations) are mandatory to ensure the basic functionality of the tool and display the data correctly. Other non-mandatory endpoints that may be provided will enable the following features: i) export of annotations in any format (Fig. 9, multiple exports supported, e.g., SemTab CSV, JSON-LD, RDF/XML, RDF/N-Triple, R2RML), ii) editing and saving annotation (Fig. 8), and iii) global search (into datasets or tables). tUI is developed using React and Typescript[15].

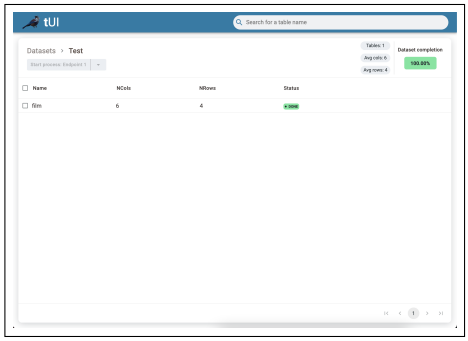

Fig. 6. Display page of the tables within a dataset.

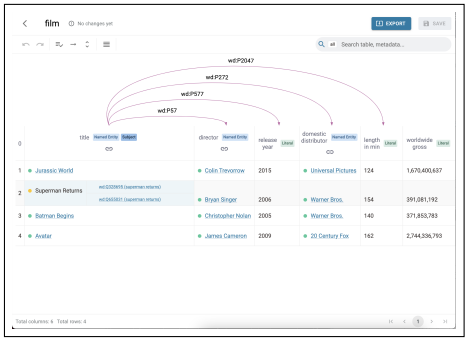

Fig. 7. Detail page of a table, with the annotation display. For each cell it is possible to see the associated entity, or the list of candidates.

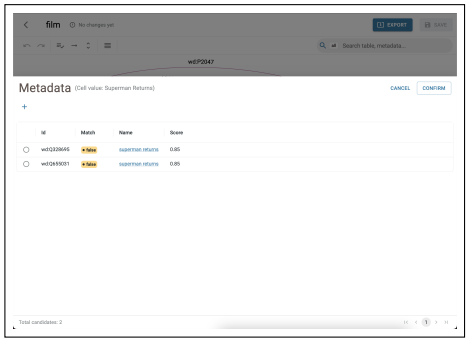

Fig. 8. Page for the analysis of candidate entities in case of uncertain annotation.

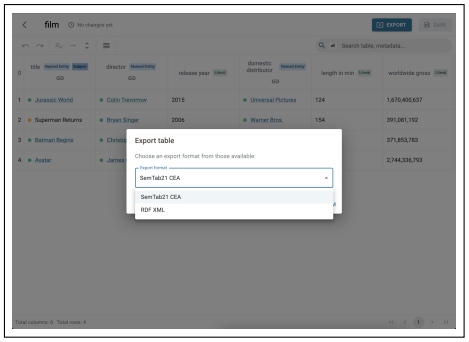

Fig. 9. Page for downloading annotations in different formats.

[15] tui-tool.ml

5 Evaluation

To evaluate our approach on a large dataset and compare it with other state-of-the-art approaches, we tested our approach against different datasets and gold standards. In particular we consider T2Dv2 Gold Standard[16] and the three most complex rounds of the various versions of the international challenge SemTab[17] [6,7]. In particular, we select the Round 3 of SemTab 2020, 2T [3], and finally the Round 2 and 3 of SemTab 2021 Hard Table. From the results shown in Table 3, in general, the accuracy of the proposed algorithm is high. It is also underlined that the approach obtained the best score in the CTA task for the SemTab2021 GitTable dataset[18]. In the Table 3 it is possible to notice differences between the datasets, which can be justified by two hypotheses: the first concerns the different complexities of the various datasets; the second concerns the use of different KGs as a target for the annotation. One possible solution is to create an additional layer to unify the different KGs, and treat them as a single dataset.

Table 3. Results on the SemTab 2020, 2021 and 2T datasets.

Tasks	**SemTab_2020**		**2T**		**SemTab2021HTR2**		**SemTab2021HTR3**	
	F1	P	F1	P	F1	P	F1	P
CEA	0.980	0.984	0.932	0.958	0.983	0.988	0.961	0.985
CTA	0.962	0.963	-	-	0.978	0.980	0.968	0.976
CPA	0.993	0.994	-	-	0.999	0.999	0.990	0.998

6 Conclusions

MantisTable V, represents the fifth version of the STI approach, MantisTable. It results from complete refactoring to substantially improve the approach, both in terms of the quality of the annotations and scalability. This second objective led to the definition and implementation of LamAPI, a system for indexing and querying KG. The current version of MantisTable also allows different managing types of tables through an improved approach to creating contexts for the disambiguation of the cells. The presence of tUI guarantees the usability of the approach, a new UI, capable of adapting concerning the services provided by the STI approaches. A limit of the current version of MantisTable V is that it performs well only on tables, including elements directly referenceable to entities in a KG (table-to-KG annotations). A challenge is to develop methods that can handle elements not present in a KG (out-of-KG annotations). Therefore, future developments on the described approach envisage the development of techniques for identifying novel entities through the use of feature-based methods and embeddings.

[16] webdatacommons.org/webtables/goldstandardV2.html
[17] www.cs.ox.ac.uk/isg/challenges/sem-tab/
[18] zenodo.org/record/5706316

References

1. Cremaschi, M., Avogadro, R., Barazzetti, A., Chieregato, D.: Mantistable se: an efficient approach for the semantic table interpretation. In: SemTab@ ISWC. pp. 75–85 (2020)
2. Cremaschi, M., De Paoli, F., Rula, A., Spahiu, B.: A fully automated approach to a complete semantic table interpretation. Future Generation Computer Systems **112**, 478 – 500 (2020)
3. Cutrona, V., Bianchi, F., Jiménez-Ruiz, E., Palmonari, M.: Tough tables: Carefully evaluating entity linking for tabular data. In: Pan, J.Z., Tamma, V., d'Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., Kagal, L. (eds.) The Semantic Web – ISWC 2020. pp. 328–343. Springer International Publishing, Cham (2020)
4. Efthymiou, V., Hassanzadeh, O., Rodriguez-Muro, M., Christophides, V.: Matching web tables with knowledge base entities: From entity lookups to entity embeddings. In: d'Amato, C., Fernández, M., Tamma, V.A.M., Lécué, F., Cudré-Mauroux, P., Sequeda, J.F., Lange, C., Heflin, J. (eds.) The Semantic Web - ISWC 2017 - 16th International Semantic Web Conference, Vienna, Austria, October 21-25, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10587, pp. 260–277. Springer (2017)
5. Ilievski, F., Garijo, D., Chalupsky, H., Divvala, N.T., Yao, Y., Rogers, C.M., Li, R., Liu, J., Singh, A., Schwabe, D., Szekely, P.A.: KGTK: A toolkit for large knowledge graph manipulation and analysis. In: Pan, J.Z., Tamma, V.A.M., d'Amato, C., Janowicz, K., Fu, B., Polleres, A., Seneviratne, O., Kagal, L. (eds.) The Semantic Web - ISWC 2020 - 19th International Semantic Web Conference, Athens, Greece, November 2-6, 2020, Proceedings, Part II. Lecture Notes in Computer Science, vol. 12507, pp. 278–293. Springer (2020)
6. Jiménez-Ruiz, E., Hassanzadeh, O., Efthymiou, V., Chen, J., Srinivas, K.: Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems. In: Harth, A., Kirrane, S., Ngonga Ngomo, A.C., Paulheim, H., Rula, A., Gentile, A.L., Haase, P., Cochez, M. (eds.) The Semantic Web. pp. 514–530. Springer International Publishing, Cham (2020)
7. Jiménez-Ruiz, E., Hassanzadeh, O., Efthymiou, V., Chen, J., Srinivas, K., Cutrona, V.: Results of semtab 2020. In: CEUR Workshop Proceedings. vol. 2775, pp. 1–8 (2020)
8. Zhang, Z.: Effective and efficient semantic table interpretation using tableminer+. Semantic Web **8**(6), 921–957 (2017)