LinkingPark: An Automatic Semantic Table Interpretation System

Shuang Chen[a,1], Alperen Karaoglu[b], Carina Negreanu[b], Tingting Ma[a], Jin-Ge Yao[c], Jack Williams[b], Feng Jiang[a,*], Andy Gordon[b], Chin-Yew Lin[c]

[a]*School of Computer Science, Harbin Institute of Technology, 150001 Harbin, China*
[b]*Microsoft Research Cambridge, Cambridge CB1 2FB, UK*
[c]*Microsoft Research Asia, 100080 Beijing, China*

Abstract

In this paper, we present LinkingPark, an automatic semantic annotation system for tabular data to knowledge graph matching. LinkingPark is designed as a modular framework which can handle Cell-Entity Annotation (CEA), Column-Type Annotation (CTA), and Columns-Property Annotation (CPA) altogether. It is built upon our previous SemTab 2020 system, which won the 2nd prize among 28 different teams after four rounds of evaluations. Moreover, the system is unsupervised, stand-alone, and flexible for multilingual support. Its backend offers an efficient RESTful API for programmatic access, as well as an Excel Add-in for ease of use. Users can interact with LinkingPark in near real-time, further demonstrating its efficiency.

Keywords: Semantic Table Interpretation, Entity Linking, Tabular data, Knowledge Graph
2010 MSC: 00-01, 99-00

1. Introduction

Tables are commonly used to organize information. The ability to automatically extract semantics in tables can empower many downstream applications, including but not limited to (i) knowledge base population [1, 2], i.e., extracting knowledge from tables; (ii) intelligent spreadsheet program assistance, e.g., automatically populating rows or columns [3], auto-completing data cells [4]; (iii) supporting semantic table search [5, 6]; (iv) data cleaning [7] and integration [8]. These applications typically require semantic table interpretation (STI) to annotate a string in a table cell with its corresponding entity in a reference knowledge graph, to predict the entity type of a column, and to detect the relationships between a pair of columns (Fig. 1).

This paper presents LinkingPark[2], an automatic semantic table interpretation system that matches tabular elements to knowledge graphs. LinkingPark is designed as a modular framework where each component can be optimized independently to address the requirements of semantic table interpretation. As shown in Fig. 2, LinkingPark includes a column analysis module, an entity linking module, a property linking module, a type inference module, and a knowledge graph module. LinkingPark has a number of desirable properties, including stand-alone architecture, flexibility for multilingual support, no dependence on labeled data, etc.

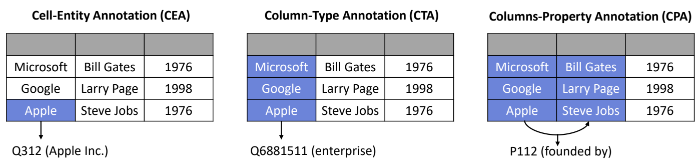


Figure 1: Annotation requirements for semantic table interpretation. The target knowledge graph in this example is Wikidata.

This article presents a re-designed version of our work [9] for the SemTab 2020 challenge to improve system stability and efficiency. In particular, we make the system more self-contained and stable by removing the dependency on the external MediaWiki API[3]. Moreover, to improve efficiency, we replace some sophisticated and heavyweight modules with negligible regression in performance. Details are described in Sec 6.3. We re-conduct all experiments on the SemTab 2020 benchmarks [10] and the Tough Tables dataset [11], add speed measurements and ablation studies over key components, offer a RESTful API for programmatic access, and create an Excel Add-in for easy

*Corresponding author
Email addresses: `hitercs@gmail.com` (Shuang Chen), `t-alkara@microsoft.com` (Alperen Karaoglu), `cnegreanu@microsoft.com` (Carina Negreanu), `v-tinma@microsoft.com` (Tingting Ma), `jinge.yao@microsoft.com` (Jin-Ge Yao), `jack.williams@microsoft.com` (Jack Williams), `fjiang@hit.edu.cn` (Feng Jiang), `adg@microsoft.com` (Andy Gordon), `cyl@microsoft.com` (Chin-Yew Lin)

[1]This work was conducted during Shuang and Tingting's internship at Microsoft Research Asia and Alperen's AI residency program at Microsoft Research Cambridge.

[2]All of the code, data, video, shared services, and system documentation are available at: `https://aka.ms/XLKG`

[3]`https://www.wikidata.org/w/api.php?action=help&modules=wbsearchentities`

user interaction. LinkingPark has been released as an open-source project [12].

2. Task formulation

In this work, we focus on processing *relational tables* with a single *subject column* as most previous work [2, 4, 6, 13, 14, 15] in the literature.[4] Relational tables describe a set of entities in the subject column along with their attributes in the remaining columns (or relational columns) [16]. A *relational table* $T = \{\{t_{11}, \ldots, t_{1n}\}, \ldots, \{t_{m1}, \ldots, t_{mn}\}\}$ contains a matrix of m rows and n columns of cells, where $r_i = \{t_{i1}, \ldots, t_{in}\}$ denotes the i-th row and $c_j = \{t_{1j}, \ldots, t_{mj}\}$ denotes the j-th column [17]. The content of each cell, denoted as t_{ij}, can be either a textual mention of a named entity described in a reference knowledge graph (called an *entity mention*) or a literal value (e.g., numerical values). We do not assume the existence of any metadata (e.g., column headers or table names) to make our approach more generally applicable.

The knowledge graph (KG) is denoted as $(\mathcal{E}, \mathcal{T}, \mathcal{P}, \mathcal{F})$, where $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{E}|}\}$ is the set of all *entities* in the KG, $\mathcal{T} = \{\tau_1, \ldots, \tau_{|\mathcal{T}|}\} \subseteq \mathcal{E}$ is the set of *types* in the KG. A *type ontology* consists of the types connected with the `subclass_of` relation. $\mathcal{P} = \{p_1, \ldots, p_{|\mathcal{P}|}\}$ is the set of possible *properties* to describe key attributes of an entity. $\mathcal{F}$ is the set of *facts* which consists of a set of RDF triples $\langle s, p, o\rangle$, where s denotes a *subject* (an entity $e \in \mathcal{E}$), $p \in \mathcal{P}$ is a *property* (also known as *predicate* or *relation*) and o denotes an *object* (an entity e, or a data value). The target knowledge graph used in this paper is Wikidata.[5]

Following SemTab [17] which contains the most relevant shared tasks of this study, the three annotation tasks in table interpretation are (see also Fig. 1):

- **CEA (Cell-Entity Annotation)**: to link each entity mention string t_{ij} in table T to its referent entity in $\mathcal{E}$.
- **CTA (Column-Type Annotation)**: to associate a table column c_j with an entity type $t \in \mathcal{T}$.
- **CPA (Columns-Property Annotation)**: to associate a pair of columns, c_s and c_t with a property $p \in \mathcal{P}$.

3. Design principles

LinkingPark is designed with the following principles:

- **Weak dependence on labeled data** Recent representation learning-based semantic table interpretation methods [18, 19, 20] required task-specific labeled data. In contrast, our system adopts an unsupervised approach which makes it easier to adapt to new knowledge graphs.

[4]Our system could be easily modified for other tables, see Sec. 6.6.
[5]`http://wikidata.org/`

- **Stand-alone** In contrast to existing tools, such as bbw [21] and JenTab [22], LinkingPark does not rely on external search services. It is self-contained and therefore does not depend on blackbox third-party components whose availability, stability, or efficiency cannot be guaranteed.
- **Effectiveness** We tailor our system for real-world scenarios to handle noisy strings with typos. The disambiguation algorithm considers multiple factors beyond surface form (e.g., entity popularity information, column-wise type consistency, and row-wise property relatedness).
- **Efficiency** We design our system with the aim for real-time applications while retaining a modest memory cost, which is often overlooked in other recent work [21, 22, 23].
- **Ease of use** The system exposes a RESTful API for easy integration, as well as a demo interface in the form of an Excel Add-in for direct user interaction. We also open-source the code to benefit the research community.
- **Multilingual support** By leveraging multilingual resources, LinkingPark can be easily extended to process multilingual tables and link them to a multilingual knowledge graph such as Wikidata [24].

4. System architecture

Fig. 2 illustrates the architecture of the LinkingPark system. A table is passed to the backend to generate semantic annotations at entity, type, and property levels.

The base annotation workflow starts by processing the input table through a *column analysis module* which classifies the basic data type of cells and detects the *subject column*. The table, along with the initial annotations, is then passed to the core table interpretation engine. The *entity linking module* first generates candidate entities through a *candidate generation sub-module*, which then become the input for both the *entity disambiguation sub-module* and the *property linking module*. The property linking module outputs property annotations between column pairs, which also provides important features for the entity disambiguation sub-module. The entity disambiguation sub-module then adopts an iterative algorithm to generate entity annotations. Finally, the type inference module outputs type annotations based on the linked entities.

To make LinkingPark more directly accessible for end users, we also build a demo interface in the form of an Excel Add-in. Due to the space limitation, this demo interface is described in Appendix A. Next, we describe each core module in detail.

4.1. Column analysis module

This module conducts column analysis by classifying the data type of a cell and detecting the subject column. We consider four kinds of basic data types: integers (`int`), floating numbers (`float`), datetime, and strings. A cell

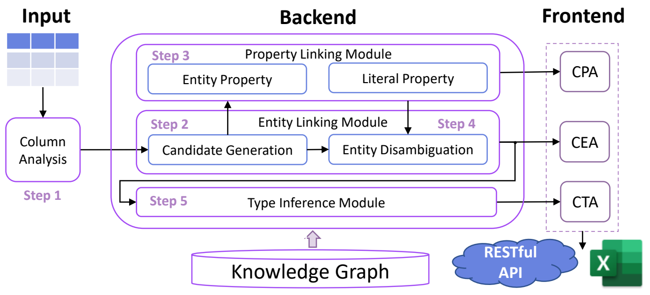


Figure 2: LinkingPark System Architecture.

content is classified into `int`, `float`, or `datetime` if it can be converted into that type using standard type conversion functions, otherwise it is treated as `string`. The second step is to detect the subject column (with its index denoted as ξ) of the table. Based on the assumption that important information appears first [25], we implement a simple heuristic that assigns the leftmost column that contains cells of the string data type as the subject column.[6]

4.2. Candidate generation sub-module

Given an entity mention t_{ij}, we want to generate its corresponding candidate entities $E_{ij} = (e_{ij1}, \ldots, e_{ijk})$ in the target KG. In order to deal with the potential misspellings in t_{ij}, we design a cascade pipeline to balance the effectiveness and efficiency. This module includes an offline step for alias map mining and two core online steps:

- **Alias map mining:** In practice, an entity can be referred by not only its canonical name but also its aliases (e.g., nicknames and abbreviations). In order to handle the naming variation, we construct a large alias map table[7] by mining entity aliases from multiple sources including the multilingual labels of an entity, aliases in Wikidata, and anchor texts in Wikipedia, similar to the strategies used in some previous studies [26].

- **Search via dictionary lookup:** The next step after alias map mining is to conduct dictionary lookup over the alias map. However, this could not handle mention strings with spelling errors (e.g., "Michael Jackson" $\rightarrow$ "Michael Jackso**n**n"). Following the design principles of a typical spelling corrector[8], we implement a tailored mention spelling corrector for better candidate retrieval. Specifically, our corrector checks all strings within one[9] edit distance to the original mention string, then retains the strings in the alias map as candidates. The spelling corrector is triggered only if the exact dictionary lookup fails to generate candidates, which is efficient in practice as the dictionary lookup happens in memory.

- **Search via fine-grained fuzzy matching:** The above step is not intended for mention strings with multiple spelling errors due to computational complexity. Therefore, we build a fine-grained index to conduct fuzzy string matching for mention strings that can not acquire candidates in the above step. Specifically, this index is built using all the observed aliases in the mined alias map.[10] During prediction time, search uses a weighted combination of word-based BM25 score and character trigram-based[11] BM25 score to do fuzzy matching[12]. This step improves the recall of candidate generation, but may also return more false positive candidates compared with the previous step.

4.3. Property linking module

Given the subject column (whose index is denoted as ξ) and a non-subject column (say the j-th column), the property linking module aims to associate a pair of columns, c_ξ and c_j with a property $p \in \mathcal{P}$. The property p can be classified into two categories according to the data type of the arguments: *entity property* (e.g., `place of birth` (P19)) or *literal property* (e.g., `date of birth` (P569)), which our system handles separately.

For property linking, we utilize an approach similar to [27]. The pseudo-code of the property linking algorithm is shown in Algorithm 1. For each non-subject j-index column, the algorithm outputs its entity property distribution $P_{\text{entity},j}(p)$ and literal property distribution $P_{\text{literal},j}(p)$.

The entity property distribution $P_{\text{entity},j}(p)$ is computed by the proportion of cells in j-index column with entity property p (Line 7). The possible entity properties between the cell $t_{\xi j}$ in subject column and the cell t_{ij} in j-index column, $\text{prop}_{\text{entity}}(i, j)$, are computed via $f_{\text{entity}}(E_{i\xi}, E_{ij})$ which is formally defined by:

$$f_{\text{entity}}(E_{i\xi}, E_{ij}) = \{p | s \in E_{i\xi}, o \in E_{ij}, \langle s, p, o\rangle \in \mathcal{F}\}. \quad (1)$$

Specifically, it finds all possible properties p whose subject lies in $E_{i\xi}$ and object lies in E_{ij}.

As for the literal property distribution $P_{\text{literal},j}(p)$, it is similarly computed by accumulating row-wise statistics (Line 8). Similar to the above, $\text{prop}_{\text{literal}}(i, j)$ contains the set of possible literal properties between the cell $t_{\xi j}$ in the subject column and the cell t_{ij} in j-index column. Different from the entity property, the calculation of $\text{prop}_{\text{literal}}(i, j)$ requires a matching process between the property's value and the cell contents t_{ij}, which is shown in the sub-procedure $f_{\text{literal}}(E_{i\xi}, t_{ij})$ of Algorithm 1. Basically,

[6]In practice, we found this simple heuristic can reach a high coverage on a sample of web tables from Common Crawl.

[7]Aliases are the keys and their potential entities are the values.

[8]`https://norvig.com/spell-correct.html`

[9]Setting the edit distance threshold larger than one is computationally expensive as the number of candidate corrections grows exponentially with the length of edit distance.

[10]The details for the index construction are in Appendix B.

[11]The character n-gram index is used to balance precision/recall and searching efficiency. In practice, we found that the trigram-based index worked well. We did not explore much on this parameter.

[12]Indexed by the Elasticsearch engine - `https://www.elastic.co/`

Algorithm 1 Property linking algorithm

Input: Table T with candidate lists $\{E_{ij}\}$, subject column index ξ, matching confidence weights $\{\omega_1, \omega_2\}$

Output: Entity property distribution $\{P_{\text{entity},j}(p)\}$ and literal property distribution $\{P_{\text{literal},j}(p)\}$ for each non-subject j-index column

```
1: for $j = 1$ to n do
2:     if $j \neq \xi$ then
3:         for $i = 1$ to m do
4:             $\text{prop}_{\text{entity}}(i,j) = f_{\text{entity}}(E_{i\xi}, E_{ij})$
5:             $\text{prop}_{\text{literal}}(i,j), w_{i,j} = f_{\text{literal}}(E_{i\xi}, t_{ij})$
6:         end for
7:         $P_{\text{entity},j}(p) = \frac{|\{i|p \in \text{prop}_{\text{entity}}(i,j), i \in \{1,2,\ldots,m\}\}|}{m}$
8:         $P_{\text{literal},j}(p) = \frac{\sum_{i=1;p\in\text{prop}_{\text{literal}}(i,j)}^{m} \omega_{i,j}}{m}$
9:     end if
10: end for
11: return $\{P_{\text{entity},j}(p)\}$, $\{P_{\text{literal},j}(p)\}$
```

Sub-procedure $f_{\text{literal}}(E_{i\xi}, t_{ij})$

```
1: if $f_{\text{literal\_exact}}(E_{i\xi}, t_{ij}) \neq \{\}$ then    ▷ Try exact match
2:     return $f_{\text{literal\_exact}}(E_{i\xi}, t_{ij})$, $\omega_1$
3: else
4:     return $f_{\text{literal\_fuzzy}}(E_{i\xi}, t_{ij})$, $\omega_2$    ▷ Fuzzy match
5: end if
```

it first tries to find exact matches between property's value and cell contents. If it succeeds, it will return the matched properties along with a confidence weight w_1. Otherwise, it resorts to finding a fuzzy match and will return a relative lower confidence weight w_2. Due to space limitations, the detailed matching function implementation is described in Appendix C.

After acquiring entity property distribution $P_{\text{entity},j}(p)$ and literal property distribution $P_{\text{literal},j}(p)$, the linked property $\hat{p}_j$ for j-index column is defined as:

$$\hat{p}_j = \underset{p \in \mathcal{P}}{\text{argmax}} \left(\max(P_{\text{entity},j}(p), P_{\text{literal},j}(p)) \right). \quad (2)$$

4.4. Entity disambiguation sub-module

Given an entity mention t_{ij} along with its candidate list $E_{ij} = (e_{ij1}, \ldots, e_{ijk})$, the entity disambiguation stage aims to select the correct entity $\hat{e}_{ij} \in E_{ij}$ from its candidate list based on their contextual information.

Formally, given a table $T = \{\{t_{11}, \ldots, t_{1n}\}, \ldots, \{t_{m1}, \ldots, t_{mn}\}\}$, the objective of entity disambiguation is to find the most compatible entity assignments for each cell t_{ij}:

$$\underset{e_{11}, e_{12}, \ldots, e_{mn} \in E_{11} \times E_{12} \times \cdots \times E_{mn}}{\text{argmax}} g(e_{11}, e_{12}, \ldots, e_{mn}|T). \quad (3)$$

where the function $g(e_{11}, e_{12}, \ldots, e_{mn}|T)$ measures the compatibility score among different entities $\{e_{11}, e_{12}, \ldots, e_{mn}\}$ linked by the cells of table T. For example, MICROSOFT (Q2283) is more compatible with the candidate entity - APPLE INC (Q312) than APPLE (Q89) for the mention "Apple" in Fig. 1.

Since the exact inference of the above objective is NP-hard, we adopt the framework of an Iterative Classification Algorithm (ICA) [28] for approximate inference. ICA is an iterative local search method which greedily re-assigns each cell to the entity that maximises the probability conditioned on the current entity assignments of other cells.

The main assumption behind the design of the disambiguation model is to characterise: (1) *type consistency* along each column of entities, and (2) *property relatedness* within each row of attribute values. In other words, entities mentioned in the same column should have compatible types, while entities or values mentioned in the same row (henceforth describing the same entity) should be related via relational facts and satisfy lexical constraints.

The disambiguation procedure is shown in Algorithm 2, which can be described as the following two steps:

1. **Initialization** (line 1): Let e_{ij}^t be the entity assignment of cell t_{ij} at iteration t. Initially, the entity assignments for all cells are independently set by maximising scores for each specific cell (line 1). The score is a weighted combination of the lexical similarity lexical_sim(e, t_{ij}) and a row support score $s_{ij}^{\text{row}}(e)$. For tie-breaking, the initial assignment of e_{ij}^0 prefers the candidate entity which has the larger number of incoming links in the knowledge graph (here, Wikidata). The number of incoming links[13] can capture the entity prominence which serves as a proxy for entity popularity. The lexical similarity is computed by calculating the maximum string similarity based on edit distance between the cell text and each of the entity names in Wikidata. The row support score $s_{ij}^{\text{row}}(e)$ is calculated by extracting the property features at both entity and literal level. This feature characterises the property relatedness between the current candidate entity and the remaining cells in the same row. The pseudo-code of the calculation is shown in sub-procedure $s_{ij}^{\text{row}}(e)$ of Algorithm 2.

2. **Iterative refinement phase** (lines 2-5): During the iterative phase, besides lexical similarity and row support score as in previous phase, we also consider one extra feature - column score $s_{ij}^{\text{col}}(e)$ to characterize type consistency along the column. The column score $s_{ij}^{\text{col}}(e)$ is calculated by averaging the entity similarity between the current candidate entity and the entity assignments of the other cells in the same column after the previous iteration (e_{kj}^{t-1}). Specifically, we represent each entity as a sparse feature vector where each property and

[13]We have considered the page views data (`https://dumps.wikimedia.org/other/pageview_complete/readme.html`) in Wikidata and the prior statistics computed from Wikipedia anchor links to estimate the entity popularity. However, both methods suffered from limited coverage problems and were less applicable to other KGs. Furthermore, DBpedia Lookup Service (`https://wiki.dbpedia.org/lookup`) and Wikidata Lookup Service (`https://www.mediawiki.org/wiki/API:Search`) also include the number of incoming links as a kind of entity popularity.

Algorithm 2 Iterative entity disambiguation algorithm

```
Input: Table $T$ with candidate lists $\{E_{ij}\}$, subject column index $\xi$, property distributions $\{P_{\text{entity},j}(p)\}$, $\{P_{\text{literal},j}(p)\}$, parameters $\{\alpha, \beta\}$, maximum number of iterations $N$
Output: Entity assignments $\{\hat{e}_{ij}\}$ for each cell $t_{ij}$
1: $e^0_{ij} = \text{argmax}_{e \in E_{ij}}\ \alpha \cdot s^{\text{row}}_{ij}(e) + \beta \cdot \text{lexical\_sim}(e, t_{ij})$    ▷ Initialization
2: while $t < N$ and any entity assignment has changed do    ▷ Refinement
3:     $s^{\text{col}}_{ij}(e) = \frac{1}{m-1} \sum_{k=1;k\neq i}^{m} \text{ent\_sim}(e, e^{t-1}_{kj})$    ▷ Column score calculation
4:     $e^t_{kj} = \text{argmax}_{e \in E_{ij}}\ \alpha \cdot s^{\text{row}}_{ij}(e) + \beta \cdot \text{lexical\_sim}(e, t_{ij}) + (1 - \alpha - \beta) \cdot s^{\text{col}}_{ij}(e)$
5: end while
6: return $\{e^t_{ij}\}$
Sub-procedure $s^{\text{row}}_{ij}(e)$    ▷ Row score calculation
1: if $j = \xi$ then    ▷ Subject column
2:     $s^{\text{entity}}_{ik}(e) = \max_{p \in f_{\text{entity}}(\{e\}, E_{ik})} (P_{\text{entity},j}(p))$    ▷ Entity property support score from column $k$
3:     $s^{\text{literal}}_{ik}(e) = \max_{(p,w) \in f_{\text{literal}}(\{e\}, t_{ik})} (P_{\text{literal},j}(p) \cdot w)$    ▷ Literal property support score from column $k$
4:     return $\frac{1}{n-1} \sum_{k=1;k\neq\xi}^{n} \max(s^{\text{entity}}_{ik}(e), s^{\text{literal}}_{ik}(e))$    ▷ Average property support score from each relational column
5: else
6:     return $\max_{p \in f_{\text{entity}}(E_{i\xi}, \{e\})} (P_{\text{entity},j}(p))$    ▷ Property support score with the subject column
7: end if
```

the value of `instance of` (P31) / `subclass of` (P279) properties serve as one feature dimension. Our basic assumption is that the properties of an entity are also a proxy of its type, besides the explicit type annotation in the KG. The ent_sim$(\cdot, \cdot)$ function is implemented by calculating the cosine similarity of the above sparse feature vectors.

Finally, after the maximum number of iterations is reached, or all entity assignments remain unchanged, the linked entity $\hat{e}_{ij}$ for entity mention t_{ij} is set as e^t_{ij}.

4.5. Type inference module

Our type inference algorithm is a heuristic voting method that is fully dependent on the entity linking results. Given an entity e, we retrieve its entity types $\mathcal{T}(e)$ from the Wikidata knowledge graph.[14] To predict the type of the j-th column, we first find the most common type shared by most of the linked entities via majority voting. However, the most common type may not be unique. In this case, we want to prioritise the most specific type. However, the type ontology in Wikidata is noisy, thus it is hard to determine the specificity of a certain type [9]. In this work, we design a set of criteria for tie-breaking. The first criterion named AverageLevel(t) characterises the specificity of a type t based on the type ontology:

$$\text{AverageLevel}(t) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[t \in \mathcal{T}(\hat{e}_{ij})] \cdot l(\hat{e}_{ij}, t) \qquad (4)$$

[14]Implemented by tracing the `instance of` (P31) property of the entity followed by at most two `subclass of` (P279) properties in the type ontology, treating the values of `instance of` (P31) and `subclass of` (P279) as the *types*. Further increasing the number of hops will increase the time complexity and retrieve too generic and not useful types (e.g., metaclass (Q19478619) and concept (Q151885)).

Here $l(\hat{e}_{ij}, t)$ denotes the distance between the entity $\hat{e}_{ij}$ and type t in the type hierarchy. Since a lower distance with respect to the entity nodes indicates a more specific type, we select the type with the minimum AverageLevel(t) to break the above ties. However, this method does not guarantee uniqueness. In practice, we found the following scheme works well on the SemTab datasets for tie-breaking. For the subject column, we select the type with minimum Population(t) on Wikidata:

$$\text{Population}(t) = \sum_{e \in \mathcal{E}} \mathbb{1}[t \in \mathcal{T}(e)] \qquad (5)$$

For relational columns, we select the type with the minimum InstanceRank(t):

$$\text{InstanceRank}(t) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}[t \in \mathcal{T}(\hat{e}_{ij})] \cdot r(\hat{e}_{ij}, t) \qquad (6)$$

$r(\hat{e}_{ij}, t)$ is the position of the type t among the statement group of the `instance of` (P31) property for entity $\hat{e}_{ij}$.

4.6. Knowledge graph module

LinkingPark requires frequent access to the knowledge graph to retrieve specific information (e.g., entity name, entity types, and property values, etc.) for a given entity. However, accessing information in a large-scale knowledge graph via SPARQL queries is inefficient. In order to meet realistic requirements for efficient access and flexibility, we expose a key-value interface to query the KG, which maps item id to a serialized binary object for the referenced entity. This structure and the overall system make no assumptions about the ontology in use to accommodate noisy ontologies such as Wikidata.

Moreover, to allow flexibility in deployment environments and usage scenarios, this KG access layer can be configured in two modes: 1) Memory-based: an in-memory key-value data store[15], for scenarios where low latency is critical; and 2) Disk-backed: a persistent disk-mapped key-value store[16], where memory/disk balance can be better controlled. The details are described in Appendix D.

5. Related work

Semantic table interpretation (STI) is important for applications in semantic web and database communities. It has attracted substantial research attention over the years. Starting from the earliest work [29, 30] which studies this task in a domain-specific fashion, the later research efforts have been focusing on matching tabular elements to general domain knowledge graphs (e.g., DBpedia and Freebase). Limaye et al. [5] build a probabilistic graphical model to capture the interdependence relationship between the three subtasks in STI. However, conducting probabilistic inference over their method is very computationally expensive. This approach was later improved by Mulwad et al. [31] with semantic message passing. These two works did not consider non-entity columns, thus failing to give a complete semantic table annotation. Later work such as T2K [32] and TableMiner$^+$ [16] considered literal columns. Most of the above methods are targeted at annotating web tables, where sufficient context (e.g., column headers, table names, surrounding paragraphs) are used as the matching features. In contrast, our system only assumes the existence of tabular contents, which is typical in the spreadsheet program scenario.

As one of the most popular KGs, Wikidata was not used as the target KG for semantic table interpretation before the SemTab 2020 challenge [10]. Compared with previous popular KGs (e.g., DBpedia) used in the STI literature, Wikidata brings additional challenges due to its large scale[17] and noisy type ontology [9]. Among the core participants in SemTab 2020, MTab4Wikidata [23] leverages a symmetric delete spelling correction algorithm[18] to handle spelling errors by pre-calculating a large index. LinkingPark@SemTab20 [9] includes a cascade pipeline using the MediaWiki API for candidate generation and an iterative coarse-to-fine entity disambiguation algorithm. DAGOBAH [33] designs an entity lookup method based on regular expressions and the Levenshtein distance via the Spark framework, using 150 machines. bbw [21] designs a meta-lookup method searching over 80 engines (e.g., Wikidata, Wikipedia, Bing, etc). JenTab [22] uses a Create, Filter, and Select approach leveraging the publicly available endpoints of Wikidata. SSL [34] generates conceptual subgraphs by creating SPARQL queries over the public Wikidata SPARQL endpoint. MantisTable SE [35], an extension of MantiesTable [36] on Wikidata, provides a tool to support querying the knowledge graph. LinkingPark differs from the systems mentioned above as it has (1) a stand-alone architecture vs. LinkingPark@SemTab20, bbw, JenTab, and SSL, etc; (2) an efficient and stable infrastructure for KG access and lookup vs. DAGOBAH and MantisTable SE; (3) significantly reduced disk resource consumption for the candidate generation module[19] vs. MTab4Wikidata.

STI approaches need to address the challenges of noise and incompleteness in knowledge graphs. In this work, we leverage the properties of an entity as a proxy of entity type to address these problems. Other viable ways include completing the type annotation first via neural networks based methods [37, 38] for predicting the missing types in noisy KGs.

6. Evaluation

6.1. Data statistics

This work uses Wikidata[20] as the target knowledge graph. We use the JSON dump (version 20200525) downloaded from Wikimedia[21]. The statistics about the knowledge graph are shown in Table 1. For evaluation, we use the benchmark datasets provided by SemTab 2020 [10]. It consists of four evaluation rounds with datasets coming from two sources: automatically generated (AG) datasets relying on an automatic table generator [17] and the Tough Table (2T) dataset [11] which consists of real tables with controlled noise from multiple resources. Table 2 shows the statistics for these datasets. These large-scale manually- or automatically-curated tables with multiple noisy levels pose various challenges (e.g., name variation, ambiguity, and efficiency, etc.) for the task of table interpretation. Therefore, they can be considered a comprehensive testbed for matching/linking systems.

Table 1: Statistics of the Wikidata knowledge graph.

Entities #	Types #	Properties #
86,494,417	165,119	7,567

Table 2: Statistics of the SemTab 2020 datasets used for evaluation.

	Automatically Generated (AG)				2T
	Round1	Round2	Round3	Round4	Round4
Tables #	34,294	12,173	62,614	22,207	180
Avg. Rows #	7.3	7.0	6.3	21.4	1,080.2
Avg. Cols #	5.0	4.6	3.7	3.5	4.5
CEA Targets #	985,110	283,447	768,325	994,921	667,244
CTA Targets #	34,294	26,727	97,586	31,922	540
CPA Targets #	135,774	43,753	166,633	56,476	–

[15]Implemented over Redis - `https://redis.io/`.

[16]Utilizing RocksDB - `http://rocksdb.org/` as engine.

[17]The latest Wikidata contains about 100 million entities, which is more than 15 times larger than DBpedia.

[18]`https://github.com/wolfgarbe/SymSpell`

[19]30GB vs. 1.5TB SSD resource.

[20]`https://www.wikidata.org/`

[21]`https://dumps.wikimedia.org/wikidatawiki/entities/`

6.2. Setup and evaluation metrics

For this evaluation, our system was hosted in a Linux virtual machine on Azure[22] with 640GB RAM memory and 80 CPU cores with EPYC^{TM} 7551 Series processors. We empirically set α to be 0.50, β to be 0.30, N to be 10, w_1 to be 1.0, w_2 to be 0.8, and retain at most 1000 candidate entities per mention from the candidate generation module.[23] We slice the tables into multiple mini tables, each with 20 rows, with respect to the natural occurrence order (top-down) in the table and process them separately for tables in larger sizes, in a parallel fashion.[24] For simplicity, we extract the type and property annotations only based on the first 20 rows. Our system at most consumes 165GB RAM memory and 350GB disk resources, with 10 CPUs at 25% usage, which are requirements easily met in modern enterprise computer servers.

As for the experiments, we follow the SemTab evaluation setup where the annotation targets (cells, columns, column pairs) are pre-specified and extra annotations are disregarded.[25] We use the official evaluation tools[26] provided by SemTab 2020 [10]. The evaluation metrics for CEA and CPA tasks are the standard Precision, Recall, and F1-score. While, for the evaluation of CTA task, SemTab 2020 designs approximate metrics for Precision (APrecision), recall (ARecall), and F1-score (AF1) by considering partially correct annotations based on the type ontology of Wikidata.

6.3. Compared systems

We compare our system with other systems participating in SemTab 2020 [10], which provides a common testbed to facilitate unbiased comparisons. We do not compare with other systems, such as T2K [32] and TableMiner$^+$ [16], due to the non-trivial cost to adapt them to the specific task settings in our experiments and porting to other KGs.

LinkingPark@SemTab20 [9] is the earlier version of our system for the SemTab 2020 challenge. It includes a cascade pipeline including MediaWiki API for candidate generation, an iterative coarse-to-fine entity disambiguation algorithm with dynamic TF-IDF weighting scheme, a multi-pass property linking algorithm that makes use of knowledge graph embeddings, and a type inference algorithm tackling the loose ontology in Wikidata. It also adopts the ranking information returned from MediaWiki as a proxy for entity popularity for both the candidate list shortlist and as the prior feature in the entity disambiguation algorithm. It achieved the overall second prize among 28 teams participating in SemTab 2020 [10].

LinkingPark is the latest version of our system presented in this paper. Compared with our earlier system, LinkingPark: i) removes the dependency of the external MediaWiki API for candidate generation to improve system stability (Sec 4.2); ii) reduces the computational cost of the previous system by removing heavyweight designs, including dynamic TF-IDF weighing scheme for column-wise entity similarity calculation and property linking methods based on KG embeddings (Sec 4.3 and 4.4). Due to the reduced computational cost, LinkingPark can handle a much larger size of entity candidates with maximum of 1,000 instead of 50 as in LinkingPark@SemTab20.

Other SemTab 2020 participants such as bbw, JenTab, SSL, DAGOBAH, MTab4Wikidata, and MantisTable SE, described earlier in Sec. 5, are included in our comparison and we directly quote their official results from the SemTab 2020 competition.

6.4. Performance

6.4.1. Comparison with the earlier system at SemTab 2020

We compare the performance between the new LinkingPark system presented in this paper and its prototype at SemTab 2020 in Table 3. We can observe that LinkingPark@SemTab20 performs quite well on Rounds 1-3 and is even ranked at top1 at certain task/rounds. Round 4 introduces a subset of the Tough Tables (2T) dataset with an average number of $\approx 1,080$ rows per table. This additional complexity makes it almost infeasible to use the original candidate generation scheme, with the number of queries significantly increased. Without the ranking information for entity candidates returned from the MediaWiki API, which we creates our entity prior scoring and candidate list shortlist, the performance of our earlier system drops on Round 4, especially on the 2T subset.

Relying on MediaWiki API makes it difficult to provide stable and low-latency service and is not suited for real world applications. Therefore, one of the key goals in designing the new version of LinkingPark is to remove dependency without loss of quality. We can observe that the re-architected LinkingPark achieves comparable performance[27] with LinkingPark@SemTab20 on Round 1-3 and improves significantly on Round 4 and the Tough Tables datasets due to the more scalable popularity features and increased size of candidate lists for better entity recall.

[22] `Standard L80s_v2 series VM size`

[23] Since there is no development set for SemTab 2020 benchmarks [10], we randomly sample 500 tables (about 1.5% of the data) from each round except the relatively small Tough Tables [11] for analysis. We set the parameters based on the observed patterns in the development set.

[24] Implemented with the `multiprocessing` package in Python.

[25] In practice, LinkingPark also supports *NIL detection* (i.e., decide when not to output predictions) by score thresholding as implemented in our codebase.

[26] `https://github.com/sem-tab-challenge/aicrowd-evaluator`

[27] We suspect the key reasons for the slight performance drop in Round 1 and Round 2 are due to (1) we remove the dependency to the external MediaWiki API for candidate generation, which usually returns a smaller number of candidates (less than 50), thus potentially better for precision. Our candidate generation method is recall-oriented, which sometimes returns a much larger number of candidates, which may confuse the downstream disambiguation model; (2) as introduced in Sec. 6.3, we replace some sophisticated and heavyweight modules of our earlier solution, which may also affect performance.

Table 3: Performance of LinkingPark@SemTab20 and LinkingPark (this paper). Bold scores denote LinkingPark (this paper) outperforms the LinkingPark@SemTab20. Underlined scores denote LinkingPark (this paper) marginally underperforms the LinkingPark@SemTab20.

System	Dataset	CEA			CTA			CPA		
		F1	Precision	#	AF1	APrecision	#	F1	Precision	#
LinkingPark @SemTab20	Round1	0.987	0.988	**1**	0.926	0.926	**1**	0.967	0.978	2
	Round2	0.993	0.993	2	0.984	0.985	**1**	0.993	0.994	2
	Round3	0.986	0.986	2	0.978	0.978	**1**	0.985	0.988	3
	Round4	0.985	0.985	2	0.953	0.953	4	0.985	0.988	5
	2T	0.810	0.811	3	0.686	0.687	3	-	-	-
LinkingPark (this paper)	Round1	0.987	0.987	-	**0.938**	**0.938**	-	0.967	0.970	-
	Round2	0.993	0.993	-	0.984	0.985	-	0.992	0.993	-
	Round3	**0.987**	**0.987**	-	**0.979**	**0.979**	-	**0.993**	**0.994**	-
	Round4	**0.988**	**0.988**	-	**0.972**	**0.972**	-	**0.995**	**0.995**	-
	2T	**0.908**	**0.908**	-	**0.784**	**0.784**	-	-	-	-

6.4.2. Comparison with other systems

Table 4 shows a comparison against other top systems participating in SemTab 2020 [10]. We can observe that most of the top systems perform quite well on the automatically generated datasets. In contrast, the performances on the Tough Tables dataset are much lower, where only MTab4Wikidata, LinkingPark, and bbw managed to maintain F1 scores over 0.85 for the CEA task. It is worth mentioning that MantisTable SE and DAGOBAH achieve relatively high performance in terms of precision but suffer from low recall. The results for the CTA task on the Tough Tables dataset are similar to the CEA task with the exception that bbw achieves a much lower AF1 on the CTA task. Most of the systems except MTab4Wikidata and LinkingPark struggle on Tough Tables since they either rely on external search services or are not equipped with stable infrastructure to conduct efficient candidate lookup. LinkingPark is second only to MTab4Wikidata on the automated generated dataset, but achieves better performance on the Tough Tables dataset. We suspect that the candidate generation step causes the slight performance gap in the automatically generated data, since MTab4Wikidata builds a resource-heavy hashing table which exhaustively stores all possible spelling errors and can better handle data noise. Furthermore, our system both handles single column scenarios and fully explores tabular context due to the column feature, while MTab4Wikidata only considers row-wise matching features. Due to the space limitations, we show an ablation study over the key components in our system in Appendix E.

6.5. Latency

The running times of LinkingPark over the SemTab 2020 datasets are shown in Table 5. In this work, we thoroughly measure the speed of our system on each dataset. First, as noted in our earlier work [9], LinkingPark@SemTab20 consumed 2-3 days (roughly 6.572 seconds/table) just for entity candidate generation by calling the MediaWiki API. In contrast, LinkingPark (Memory-based) only takes 0.825 seconds to annotate a table. We can also observe that LinkingPark (Memory-based), as expected, is faster than LinkingPark (Disk-backed) due to the unconstrained KG access. Finally, the time consumption in our system highly depends on the size of the tables, so it can be further sped up via multi-processing.[28]

6.6. Limitations

Although LinkingPark has already achieved excellent performance in comparison with other existing semantic table interpretation tools, we identify a few improvement opportunities.

Type of tables: the current system is targeting relational tables with a single subject column in line with most existing works [2, 4, 6, 13, 14, 15] in the semantic table interpretation literature. However, there are other types of tables (e.g., non-relational tables or denormalized tables with multiple subject columns) that are not covered. We plan to integrate the existing works on table type classification [40, 41, 42] as a pre-processing step to filter the high-quality relational table and adopt the method [43] to decompose multi-subject tables into smaller single-subject tables. Furthermore, we also provide an extended algorithm to cope with tables with multiple subject columns in Appendix H.

Subject column detection: the current system includes a simple heuristic for detecting the subject column. However, subject column detection on denormalized tables can be complex. We leave the improvement of the subject column detection algorithm as future work.

Initialization of the disambiguation algorithm: the disambiguation algorithm described in Sec. 4.4 includes an initialization phase, where we initially use a weighted combination of the lexical similarity and a row support score as the primary criterion, then the number of incoming links in the knowledge graph for tie-breaking. This design might not be effective in some scenarios, especially when

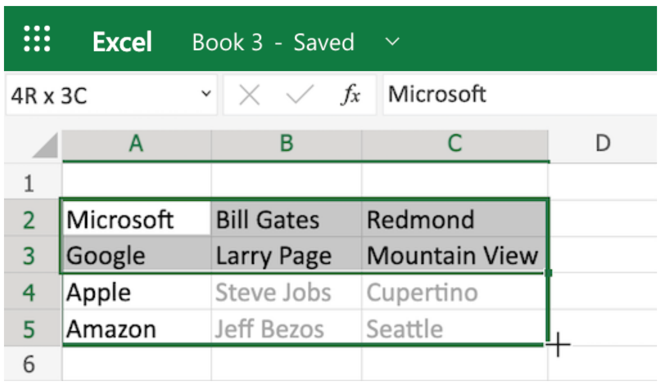
[28]Previous systems rarely report the speed of their system thoroughly, re-running their systems is difficult, even impossible due to the unavailability of open-sourced implementation. To the best of our knowledge, only three systems (MTab [39], JenTab [22], and DAGOBAH [33]) in SemTab 2020 provide latency results for their systems in their papers or extension work. We summarize their speed measurements in Appendix G for reference. These results are not strictly comparable due to different hardware and setup.

Table 4: Comparison with other top systems in SemTab 2020. F1/AF1 scores are used as the primary metric in SemTab 2020.

System	Automatically Generated Dataset (Round4)						Tough Tables			
	CEA		CTA		CPA		CEA		CTA	
	F1	Pr	AF1	APr	F1	Pr	F1	Pr	AF1	APr
MTab4Wikidata	**0.993**	**0.993**	**0.981**	0.982	**0.997**	**0.997**	0.907	0.907	0.728	0.730
DAGOBAH	0.984	0.985	0.972	0.972	0.995	0.995	0.412	0.749	0.718	0.747
bbw	0.978	0.984	0.980	0.980	0.995	0.996	0.863	**0.927**	0.516	**0.789**
JenTab	0.973	0.975	0.930	0.930	0.994	0.994	0.374	0.541	0.624	0.669
SSL	0.833	0.833	0.946	0.946	0.924	0.924	0.198	0.198	0.363	0.363
MantisTable SE	0.812	0.985	0.725	**0.989**	0.803	0.988	0.400	0.804	0.474	0.639
LinkingPark@SemTab20	0.985	0.985	0.953	0.953	0.985	0.988	0.810	0.811	0.686	0.687
LinkingPark (this paper)	0.988	0.988	0.972	0.972	0.995	0.995	**0.908**	0.908	**0.784**	0.784

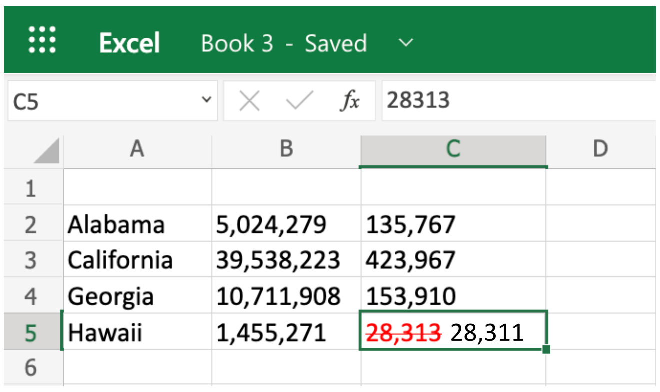
Table 5: Average running times (seconds) per table for LinkingPark at different setup over the SemTab 2020 datasets.

System	Round1	Round2	Round3	Round4	2T	Avg.
LinkingPark (Memory-based)	0.439	0.595	0.537	1.682	84.339	0.825
LinkingPark (Disk-backed)	0.461	0.607	0.569	2.131	101.579	0.947



	A	B	C	D
1				
2	Microsoft	Bill Gates	Redmond	
3	Google	Larry Page	Mountain View	
4	Apple	Steve Jobs	Cupertino	
5	Amazon	Jeff Bezos	Seattle	
6				

Figure 3: Use case: semantic flash fill.



	A	B	C	D
1				
2	Alabama	5,024,279	135,767	
3	California	39,538,223	423,967	
4	Georgia	10,711,908	153,910	
5	Hawaii	1,455,271	~~28,313~~ 28,311	
6				

Figure 4: Use case: table fact checking.

handling abbreviations with limited context as we observed. We plan to investigate a better initialization strategy in the future.

Cross-lingual matching: the current multilingual table annotation is supported by the multilingual resources in the KG. However, it could not handle the cross-lingual matching cases [44], which could be implemented by utilizing multilingual embeddings (e.g., multilingual BERT [45]) as a future research direction.

Multiple KGs: the system currently only supports linking to the Wikidata KG. However, as there are many KGs in the world, we deem the extension of LinkingPark to the other KGs as important future work.

7. Use cases

In this section, we describe two potential use cases of LinkingPark in Excel to improve tabular data productivity.

7.1. Semantic flash fill

As shown in Fig. 3, assume a user inputs a couple of rows in Excel, then LinkingPark can automatically detect named entities (e.g., MICROSOFT (Q2283)) and infer the relationship between the pair of columns (e.g., the relationship between column A and column B is FOUNDED BY (P112), the relationship between column A and column C is HEADQUARTERS LOCATION (P159)). After the user continues to type a list of cells in the left-most column, LinkingPark can automatically link these cells to their corresponding entities (e.g., APPLE INC. (Q312)) based on the partial tabular context and fill the content of the missing cells by retrieving corresponding property information from the KG (e.g., Wikidata). This functionality can speed up the table creation process for Excel users.

7.2. Table fact checking

As shown in Fig. 4, assume a user inputs or pastes a table, then LinkingPark can analyze that this table is describing a list of US states (e.g., ALABAMA (Q173)) along with their population (P1082) and areas (P2046). After that, the system can automatically detect the inconsistency between the data in the table and the KG, then inform the user. For example, the area of Hawaii[29] is 28,311 square kilometers in Wikidata. In this way, LinkingPark can power a table fact-checking functionality to detect data mismatches and fix potential data errors.

[29]`https://www.wikidata.org/wiki/Q782`

8. Conclusion

We present LinkingPark, an automatic semantic table interpretation system. Built upon our previous design on SemTab 2020, LinkingPark achieves highly competitive results on the SemTab 2020 and Tough Tables benchmarks, while enabling some real world table intelligence scenarios (exemplified in the available Excel Add-in interface). We believe LinkingPark can contribute significantly to future research efforts towards a better understanding of tabular data. For the ease of future usage and extensions, we are standardizing the specification of the knowledge graph resources and automating the migration process for other knowledge graphs that follows this specification. In this way, new knowledge graphs can be easily consumed by LinkingPark as long as they can be converted into our specified data interface, then the system can act as a hub of annotation services targeting various knowledge graphs, fitting the vision of Semantic Web.

Acknowledgements

We would like to thank Börje F. Karlsson, Xi Chen, and Qian Liu for their constructive suggestions, and all the anonymous reviewers for their helpful feedback.

References

[1] E. Muñoz, A. Hogan, A. Mileo, Using linked data to mine rdf from wikipedia's tables, in: Proceedings of the 7th ACM international conference on Web search and data mining, 2014, pp. 533–542.

[2] D. Ritze, O. Lehmberg, Y. Oulabi, C. Bizer, Profiling the potential of web tables for augmenting cross-domain knowledge bases, in: Proceedings of the 25th international conference on world wide web, 2016, pp. 251–261.

[3] S. Zhang, Smarttable: equipping spreadsheets with intelligent assistance functionalities, in: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, 2018, pp. 1447–1447.

[4] S. Zhang, K. Balog, Auto-completion for data cells in relational tables, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, 2019, pp. 761–770.

[5] G. Limaye, S. Sarawagi, S. Chakrabarti, Annotating and searching web tables using entities, types and relationships, Proceedings of the VLDB Endowment 3 (1-2) (2010) 1338–1347.

[6] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, Recovering semantics of tables on the web, in: Proceedings of the VLDB Endowment, 2011, pp. 528–538.

[7] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, Y. Ye, Katara: A data cleaning system powered by knowledge bases and crowdsourcing, in: Proceedings of the 2015 ACM SIGMOD international conference on management of data, 2015, pp. 1247–1261.

[8] S. Gupta, P. Szekely, C. A. Knoblock, A. Goel, M. Taheriyan, M. Muslea, Karma: A system for mapping structured sources into the semantic web, in: Extended Semantic Web Conference, Springer, 2012, pp. 430–434.

[9] S. Chen, A. Karaoglu, C. Negreanu, T. Ma, J.-G. Yao, J. Williams, A. Gordon, C.-Y. Lin, Linkingpark: An integrated approach for semantic table interpretation., in: SemTab@ ISWC, 2020, pp. 65–74.

[10] E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, V. Cutrona, Results of semtab 2020, in: CEUR Workshop Proceedings, Vol. 2775, 2020, pp. 1–8.

[11] V. Cutrona, F. Bianchi, E. Jiménez-Ruiz, M. Palmonari, Tough tables: Carefully evaluating entity linking for tabular data, in: International Semantic Web Conference, Springer, 2020, pp. 328–343.

[12] S. Chen, A. Karaoglu, C. Negreanu, B. F. Karlsson, T. Ma, J.-G. Yao, J. Williams, F. Jiang, A. Gordon, C.-Y. Lin, LinkingPark: Automatic Semantic Table Interpretation Software (Apr. 2022). `doi:10.5281/zenodo.6496662`.

[13] F. Chirigati, J. Liu, F. Korn, Y. Wu, C. Yu, H. Zhang, Knowledge exploration using tables on the web, Proceedings of the VLDB Endowment 10 (3) (2016) 193–204.

[14] S. Zhang, K. Balog, Ad hoc table retrieval using semantic similarity, in: Proceedings of the 2018 world wide web conference, 2018, pp. 1553–1562.

[15] S. Zhang, E. Meij, K. Balog, R. Reinanda, Novel entity discovery from web tables, in: Proceedings of The Web Conference 2020, 2020, pp. 1298–1308.

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

[17] E. Jiménez-Ruiz, O. Hassanzadeh, V. Efthymiou, J. Chen, K. Srinivas, Semtab 2019: Resources to benchmark tabular data to knowledge graph matching systems, in: European Semantic Web Conference, Springer, 2020, pp. 514–530.

[18] J. Chen, E. Jiménez-Ruiz, I. Horrocks, C. Sutton, Colnet: Embedding the semantics of web tables for column type prediction, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 29–36.

[19] X. Deng, H. Sun, A. Lees, Y. Wu, C. Yu, Turl: table understanding through representation learning, Proceedings of the VLDB Endowment 14 (3) (2020) 307–319.

[20] D. Wang, P. Shiralkar, C. Lockard, B. Huang, X. L. Dong, M. Jiang, Tcn: Table convolutional network for web table interpretation, in: Proceedings of the Web Conference 2021, 2021, pp. 4020–4032.

[21] R. Shigapov, P. Zumstein, J. Kamlah, L. Oberländer, J. Mechnich, I. Schumm, bbw: Matching csv to wikidata via meta-lookup, in: CEUR Workshop Proceedings, Vol. 2775, RWTH, 2020, pp. 17–26.

[22] N. Abdelmageed, S. Schindler, Jentab: A toolkit for semantic table annotations, in: Second International Workshop On Knowledge Graph Construction Co-located with the ESWC, 2021.

[23] P. Nguyen, I. Yamada, N. Kertkeidkachorn, R. Ichise, H. Takeda, Mtab4wikidata at semtab 2020: Tabular data annotation with wikidata., in: SemTab@ ISWC, 2020, pp. 86–95.

[24] D. Vrandečić, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Communications of the ACM 57 (10) (2014) 78–85.

[25] Wang, Xinxin, Tabular abstraction, editing, and formatting, in: Thesis of University of Waterloo, UWSpace, 2016.

[26] V. Efthymiou, O. Hassanzadeh, M. Rodriguez-Muro, V. Christophides, Matching web tables with knowledge base entities: from entity lookups to entity embeddings, in: International Semantic Web Conference, Springer, 2017, pp. 260–277.

[27] A. Karaoglu, C. Negreanu, S. Chen, J. Williams, D. Fabian, A. Gordon, C.-Y. Lin, Wiki2row - the in's and out's or row suggestion with a large scale knowledge base, Tech. Rep. MSR-TR-2020-37, Microsoft (October 2020).

[28] L. Getoor, Link-based classification, in: Advanced methods for knowledge discovery from complex data, Springer, 2005, pp. 189–207.

[29] G. Hignette, P. Buche, J. Dibie-Barthélemy, O. Haemmerlé, An ontology-driven annotation of data tables, in: International Conference on Web Information Systems Engineering, Springer, 2007, pp. 29–40.

[30] G. Hignette, P. Buche, J. Dibie-Barthélemy, O. Haemmerlé, Fuzzy annotation of web data tables driven by a domain ontology, in: European Semantic Web Conference, Springer, 2009, pp. 638–

653.

[31] V. Mulwad, T. Finin, A. Joshi, Semantic message passing for generating linked data from tables, in: International Semantic Web Conference, Springer, 2013, pp. 363–378.

[32] D. Ritze, O. Lehmberg, C. Bizer, Matching html tables to dbpedia, in: Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, 2015, pp. 1–6.

[33] V.-P. Huynh, J. Liu, Y. Chabot, T. Labbé, P. Monnin, R. Troncy, Dagobah: Enhanced scoring algorithms for scalable annotations of tabular data., in: SemTab@ ISWC, 2020, pp. 27–39.

[34] D. Kim, H. Park, J. K. Lee, W. Kim, Generating conceptual subgraph from tabular data for knowledge graph matching., in: SemTab@ ISWC, 2020, pp. 96–103.

[35] M. Cremaschi, R. Avogadro, A. Barazzetti, D. Chieregato, Mantistable se: an efficient approach for the semantic table interpretation., in: SemTab@ ISWC, 2020, pp. 75–85.

[36] M. Cremaschi, F. De Paoli, A. Rula, B. Spahiu, A fully automated approach to a complete semantic table interpretation, Future Generation Computer Systems 112 (2020) 478–500.

[37] A. Melo, J. Völker, H. Paulheim, Type prediction in noisy rdf knowledge bases using hierarchical multilabel classification with graph and latent features, International Journal on Artificial Intelligence Tools 26 (02) (2017) 1760011.

[38] V. Cutrona, G. Puleri, F. Bianchi, M. Palmonari, Nest: Neural soft type constraints to improve entity linking in tables, in: Further with Knowledge Graphs, IOS Press, 2021, pp. 29–43.

[39] P. Nguyen, I. Yamada, N. Kertkeidkachorn, R. Ichise, H. Takeda, Demonstration of mtab: Tabular data annotation with knowledge graphs, in: International Semantic Web Conference, Posters, Demos, and Industry Tracks, 2021.

[40] Y. Wang, J. Hu, A machine learning based approach for table detection on the web, in: Proceedings of the 11th international conference on World Wide Web, 2002, pp. 242–250.

[41] O. Lehmberg, D. Ritze, R. Meusel, C. Bizer, A large public corpus of web tables containing time and context metadata, in: Proceedings of the 25th International Conference Companion on World Wide Web, 2016, pp. 75–76.

[42] J. Eberius, K. Braunschweig, M. Hentsch, M. Thiele, A. Ahmadov, W. Lehner, Building the dresden web table corpus: A classification approach, in: 2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC), IEEE, 2015, pp. 41–50.

[43] K. Braunschweig, M. Thiele, W. Lehner, From web tables to concepts: A semantic normalization approach, in: International Conference on Conceptual Modeling, Springer, 2015, pp. 247–260.

[44] A. Sil, G. Kundu, R. Florian, W. Hamza, Neural cross-lingual entity linking, in: Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[45] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: NAACL, ACL, Minneapolis, Minnesota, 2019, pp. 4171–4186.

[46] P. Nguyen, I. Yamada, H. Takeda, Mtabes: Entity search with keyword search, fuzzy search, and entity popularities, in: The 35th Annual Conference of the Japanese Society for Artificial Intelligence, 2021, pp. 1N4IS1a02–1N4IS1a02.

[47] V. Cutrona, J. Chen, V. Efthymiou, O. Hassanzadeh, J.-R. Ernesto, J. Sequeda, K. Srinivas, N. Abdelmageed, M. Hulsebos, D. Oliveira, C. Pesquita, Results of semtab 2021, in: CEUR Workshop Proceedings, Vol. 3103, 2021, pp. 1–12.

[48] N. Abdelmageed, S. Schindler, B. König-Ries, Biodivtab: A table annotation benchmark based on biodiversity research data, in: CEUR Workshop Proceedings, Vol. 3103, 2021, pp. 13–18.

[49] R. Boeddinghaus, S. Marhan, D. Berner, S. Boch, M. Fischer, J. Kattge, V. Klaus, T. Kleinebecker, Y. Oelmann, D. Prati, D. Schäfer, I. Schöning, M. Schrumpf, E. Sorkau, E. Kandeler, P. Manning, E. Kandeler, Plant functional trait shifts explain concurrent changes in the structure and function of grassland soil microbial communities (2017).

[50] M. Fischer, T. Nauss, M. Tschapka, W. Weisser, J. Müller, Aggregated species richness and habitat heterogeneity variables for testing the habitat-heterogeneity hypothesis, 2006-2018 (2020).

[51] S. Seibold, M. Gosňer, N. Simons, N. Blüthgen, J. Müller, D. Ambarli, C. Ammer, J. Bauhus, M. Fischer, C. Fürstenau, J. C. Habel, K. E. Linsenmair, T. Nauss, A. Ostrowski, C. Penone, D. Prati, P. Schall, E.-D. Schulze, J. Vogt, S. Wöllauer, W. Weisser, Arthropod data from 150 grassland plots, 2008-2017, and 140 forest plots, 2008-2016, used in "Arthropod decline in grasslands and forests is associated with drivers at landscape level", Nature (2019).

[52] S. Leonhardt, B. Peters, A. Keller, Trap nesting solitary bee species measured on all grassland VIPs 2017-2018 (2020).

[53] S. Leonhardt, B. Peters, A. Keller, Fatty acids in pollen of Osmia bicornis larval provisions 2017-2018 (2020).

[54] S. Leonhardt, B. Peters, A. Keller, Amino acids in pollen of Osmia bicornis larval provisions 2017-2018 (2020).

[55] M. Staab, A. Schuldt, T. Assmann, H. Bruelheide, A. Klein, Ant community structure during forest succession in a subtropical forest in South-East China (2014) 32–40.

[56] T. Wubet, Y. Wu, F. Buscot, Soil Fungal metagenome from 12 CSPs based on the fungal ITS rDNA pyrotags (2013).

[57] K. Nadrowski, Deviations from stem breaking probabilities at species level (2013).

[58] H. Bruelheide, D. Eichenberg, W. Kröber, M. Böhnke, C. Ristok, Main Experiment: Leaf traits and chemicals from individual trees in the Main Experiment (Site A & B) (2012).

[59] V.-P. Huynh, J. Liu, Y. Chabot, F. Deuzé, T. Labbé, P. Monnin, R. Troncy, Dagobah: Table and graph contexts for efficient semantic annotation of tabular data, in: CEUR Workshop Proceedings, Vol. 3103, 2021, pp. 19–31.

[60] N. Abdelmageed, S. Schindler, Jentab meets semtab 2021's new challenges, in: CEUR Workshop Proceedings, Vol. 3103, 2021, pp. 42–53.

Appendix A. User interface

To allow users to interact directly with the system and better understand its capabilities, we offer a direct user interface - shown in Fig. A.5. Using Microsoft Excel, users can input data and select a range of cells to trigger the LinkingPark annotation service by simply clicking the "Detect" button in the right-side taskpane. Resulting linked entities, types, and properties are shown in this taskpane. And links are added in-place in the spreadsheet, which users can follow for more information about each cell entity.

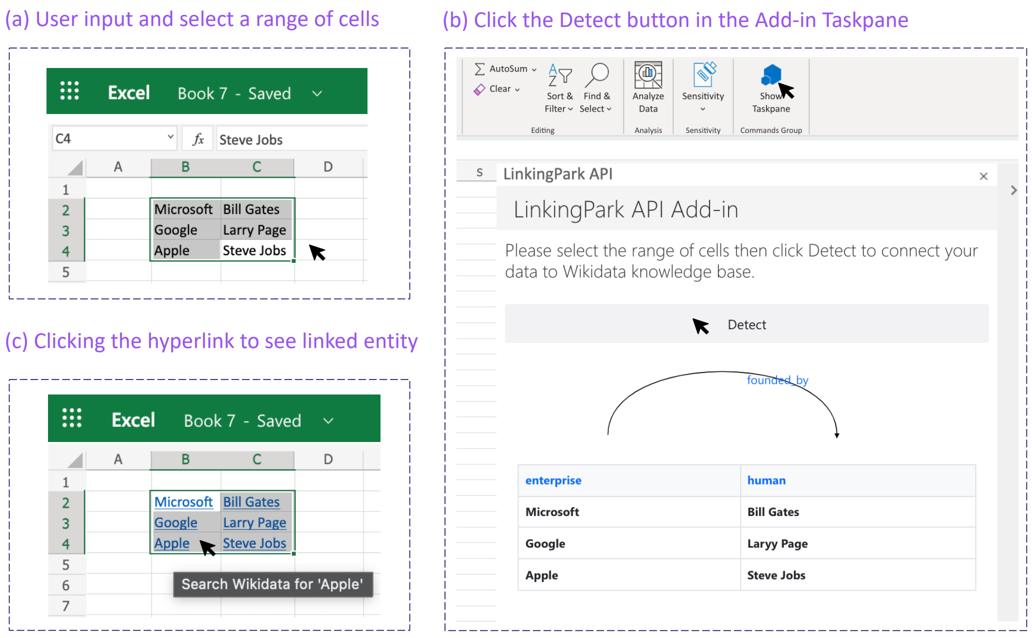
Appendix B. Index construction

This index is built to facilitate fuzzy string matching over the alias map. Specifically, the index contains two properties, "title" which is an indexed field storing the aliases in the mined alias map, and "qid" which is an unindexed field storing the associated Wikidata IDs with such an alias.

As for the "title" field, its main field uses a standard analyzer in Elasticsearch and a character-based sub-field with a character trigram-based analyzer. During the querying phrase, the scoring function is "2×word-based BM 25 score+ character-based BM 25 score".

Appendix C. Matching procedure

As for the matching procedure $f_{\text{literal\_exact}}(E_{i\xi}, t_{ij})$ and $f_{\text{literal\_fuzzy}}(E_{i\xi}, t_{ij})$, it tries to find all possible properties



Microsoft	Bill Gates
Google	Larry Page
Apple	Steve Jobs

Please select the range of cells then click Detect to connect your data to Wikidata knowledge base.

enterprise	human
Microsoft	Bill Gates
Google	Laryy Page
Apple	Steve Jobs

Figure A.5: Snippets of the direct user interface of LinkingPark in Microsoft Excel.

whose subject lies in $E_{i\xi}$, whose object matches t_{ij} under a given matching function. The strict matching function is implemented based on the basic data type of each property: exact match is used for strings, match within a threshold[30] is applied for numerical quantities and the exact match under the ISO datetime format for dates and times. As for the fuzzy match, we implement unit conversion functions for numerical properties including length, area, volume, frequency, mass, temperature, etc[31], and match its value with the cell value under unit conversion; For datetime entries, the fuzzy match allows at most a difference of ten days.

Appendix D. Data access layer

In our current implementation, the extra data access layer consists of three *dictionaries*: 1) Entity name map: which maps an item id to its entity names, which is used in the lexical similarity calculation. For fast lookup it can be fully stored in memory and consumes 26.1 GB RAM; 2) Entity property values map: which maps an item id to its all the properties along with values; 3) Entity type information map: which maps an item id to its entity types. As discussed in 4.6, these last two stores can be configured into either Memory-only or Disk-backed modes depending on the requirements of specific scenarios or deployment environment constraints.

[30] In practice, we choose $1e^{-2}$.
[31] https://www.wikidata.org/wiki/Wikidata:Units

Appendix E. Ablation studies

Appendix E.1. Feature ablation

We conduct an ablation study for the different features used in the entity disambiguation algorithm with results displayed in Table E.6. By removing the row score, the entity linking system F1-score drops 0.04 and 0.17 on automatically generated tables of Round 4 and Tough Tables respectively. The column score, which characterizes type consistency along the column, takes little effects on the automatically generated tables, but improves 0.012 F1 on Tough Tables. This is due to the automatic generation process [17] producing tables based on existing properties in the KG, therefore utilizing row score to characterize property relatedness is already sufficient to resolve the ambiguity. However, the Tough Tables dataset contains real tables with cells which may not have correspondence in the target KG, or the table might even only have one single column as context which could only benefit from the column score feature. Similar trends can be observed by removing the lexical or popularity features. In order to make the system robust in real-world tables (which can be extremely varied), we should consider all available aspects of information.

Table E.6: Feature ablation of entity disambiguation on Round4 dataset of SemTab 2020.

Setting	Round 4 (AG)		Tough Tables	
	F1	Pr	F1	Pr
LinkingPark (full)	0.988	0.988	0.908	0.908
LinkingPark (w/o row score)	0.948	0.948	0.738	0.738
LinkingPark (w/o col. score)	0.987	0.987	0.896	0.896
LinkingPark (w/o lexical)	0.986	0.986	0.898	0.898
LinkingPark (w/o popularity)	0.988	0.988	0.883	0.883

Table E.7: Entity linking performance of Round4 dataset of SemTab 2020 under different candidate generation methods, DS denotes Dictionary Search, FS denotes Fuzzy Search.

Setting	Round 4 (AG)		Tough Tables	
	F1	Pr	F1	Pr
LinkingPark (DS+FS)	0.988	0.988	0.908	0.908
LinkingPark (DS-only)	0.985	0.989	0.809	0.897

Appendix E.2. Candidate generation method ablation

Table E.7 shows the entity linking performance under different candidate generation methods. Specifically, we compare the full candidate generation method, LinkingPark (DS+FS), which combines the dictionary search and fuzzy search, with LinkingPark (DS-only), which only adopts the dictionary search method[32]. As we can see from Table E.7, LinkingPark (DS+FS) improves over LinkingPark (DS) 0.003 F1 on the automatically generated dataset of Round4. It demonstrates the dictionary search which can handle 1 edit typo can handle most of the cases in the automatically generated dataset. However, LinkingPark (DS+FS) significantly improves LinkingPark (DS) over 0.099 F1 points on Tough Tables due to the significantly improved recall. The above results show that candidate generation based on fuzzy search is complementary to the candidate generation methods based on dictionary search, which is especially helpful on Tough Table and potentially in real-world scenarios with noisier data.

Appendix F. Latency of other systems

We summarize the detailed measurements of other systems as reported in their original papers in Table F.8.

Appendix G. Results on SemTab 2021 datasets

We also report the performance of LinkingPark on SemTab 2021 [47] datasets with Wikidata as the target KG in Table G.9. We make the following changes to migrate LinkingPark to SemTab 2021.

1. Re-index the KG with Wikidata JSON dump (version 20210830), which is the closest downloadable dump to the specified version (20210828) in SemTab 2021.[33]

2. For the BioTable dataset whose CPA task consists of multi-subjects, we use the target files for the CPA task as input to predict the property annotation. We believe this strategy has also been adopted by other systems.

3. For the BiodivTab dataset [48][34] which includes column headers[35], we output the CTA annotation matched with the column headers which performs better than voting from cell contents.

Except for the above-mentioned changes, all of the model designs including parameter settings are the same as the presented system in the main paper. As you can see, our system still maintains highly competitive performance over SemTab 2021 datasets compared with other top systems. The results support the claim that our system generalises well.

Appendix H. Multi-subject extension

Although we focus on processing tables with a single subject column, we provide a simple extension of our algorithms for scenarios with no assumption on the number of subject columns. Specifically, we modify Algorithm 1 and Algorithm 2 with no pre-specified subject column index as the input. The adapted algorithms are shown in Algorithm 3 and Algorithm 4 respectively. As for Algorithm 3, instead of only matching with the single subject column as in Algorithm 1, we check all the columns pairs to calculate both the entity property distribution $\{P_{\text{entity},i,j}(p)\}$ and literal property distribution $\{P_{\text{literal},i,j}(p)\}$ between the i-index column and j-index column. Then the final property distribution between the i-index column and j-index column becomes:

$$P_{i,j}(p) = \max(P_{\text{entity},i,j}(p), P_{\text{literal},i,j}(p)). \tag{H.1}$$

Finally, we can output the property with the highest confidence and set a confidence threshold δ to prune predictions with low confidence.

As for Algorithm 4, we also maintain the iterative entity disambiguation framework as in Algorithm 2, with only the modification for the row score calculation. Specifically, the row score is now calculated by considering all the possible properties from the other columns. We treat the j-index column as both the subject column and object column when measuring the support score with the remaining columns, and then select the larger one (Line 19 in Algorithm 4).

To verify the effectiveness of the extended algorithms, we conduct experiments on the BioTable, whose CPA task contains multiple subject column targets.[36] As you can see from Table G.10, the extended version indeed significantly improves the performance of LinkingPark on the BioTable dataset. However, we also note that the performance gain

[32]Searching only with fuzzy search both impacts performance and is not a good design choice, so we did not compare against LinkingPark (FS-only).

[33]In practice, we found using different versions of Wikidata influenced the results on SemTab 2021 a lot, especially on the automatically generated data set. For example, the CEA result on HardTablesR2 is 0.940 F1 on our previous dump used in SemTab 2020.

[34]This dataset is created based on the tables from the following biodiversity research papers [49, 50, 51, 52, 53, 54, 55, 56, 57, 58]. According to the license of BioTable dataset, we need to cite the above papers.

[35]Our previous system does not rely on column headers.

[36]To the best of our knowledge, the CPA task of the remaining datasets (linking to Wikidata KG) in SemTab 2020 and SemTab 2021 only evaluate the single subject column scenario.

Table F.8: Average running times (seconds) per table between different systems over the SemTab 2020 datasets. − denotes that the system does not report their latency/speed in the corresponding dataset. †: JenTab only reports latency on the merged Round4 and 2T. For reference, the speed of LinkingPark (Memory-based) is 2.346 s/table, while the speed of LinkingPark (Disk-backed) is 2.931 s/table on the merged split. ‡: DAGOBAH only reports latency on the Round 1-3 datasets. Note that this table is for reference only and not strictly comparable, since the speed of different systems are measured on different hardware.

System	Round1	Round2	Round3	Round4	2T	Avg.
MTab [46]	−	−	−	−	−	1.520
JenTab [22]	1.260	17.744	2.070	$15.409^{\dagger}$		4.270
DAGOBAH [33]	4.619	18.927	15.409	−	−	$12.430^{\ddagger}$
LinkingPark (Memory-based)	0.439	0.595	0.537	1.682	84.339	0.825
LinkingPark (Disk-backed)	0.461	0.607	0.569	2.131	101.579	0.947

Table G.9: Comparison with other top systems in SemTab 2021. Bold scores denote the best performance, while the underlined scores denote secondary performance. ∗ denotes results from SemTab 2020.

System	Tough Tables		HardTablesR2			BioTable			BiodivTab		HardTablesR3		
	CEA	CTA	CEA	CTA	CPA	CEA	CTA	CPA	CEA	CTA	CEA	CTA	CPA
MTab [39]	0.907^{*}	0.728^{*}	**0.985**	**0.977**	0.997	0.964	**0.956**	**0.947**	0.522	0.123	0.968	0.984	0.993
DAGOBAH [59]	**0.923**	**0.832**	0.975	0.976	0.996	**0.970**	0.916	0.899	0.496	**0.381**	**0.974**	**0.990**	0.991
JenTab [60]	0.457	0.697	0.966	0.914	0.996	0.857	0.835	0.899	**0.602**	0.107	0.940	0.942	0.992
LinkingPark	0.908	0.784	0.982	0.959	**0.998**	0.953	0.892	0.899	0.576	0.210	0.959	0.967	**0.999**

Table G.10: Performance of the multi-subject extended algorithm on BioTable dataset. ∗ denotes the result of LinkingPark under strict single subject column assumption, without making the second change specified in Appendix G.

System	CEA	CTA	CPA
LinkingPark (single-subject)	0.953	0.892	0.671^{*}
LinkingPark (multi-subject)	0.964	0.897	0.899

of the extended version is at the cost of the increased computational cost since we currently need to enumerate all column pairs in the table. We treat improving the efficiency of the multi-subject extension version as future work.

Algorithm 3 Extended property linking algorithm

Input: Table T with candidate lists $\{E_{ij}\}$, matching confidence weights $\{\omega_1, \omega_2\}$
Output: Entity property distribution $\{P_{\text{entity},i,j}(p)\}$ and literal property distribution $\{P_{\text{literal},i,j}(p)\}$ for between subject i-index column and non-subject j-index column

```
 1: for $i = 1$ to n; $j = 1$ to n do
 2:     if $i \neq j$ then
 3:         for $k = 1$ to m do
 4:             $\text{prop}_{\text{entity}}(i,j,k) = f_{\text{entity}}(E_{ki}, E_{kj})$
 5:             $\text{prop}_{\text{literal}}(i,j,k), w_{i,j,k} = f_{\text{literal}}(E_{ki}, t_{kj})$
 6:         end for
 7:         $P_{\text{entity},i,j}(p) = \frac{|\{k \mid p \in \text{prop}_{\text{entity}}(i,j,k), k \in \{1,2,\ldots,m\}\}|}{m}$
 8:         $P_{\text{literal},i,j}(p) = \frac{\sum_{k=1; p \in \text{prop}_{\text{literal}}(i,j,k)}^{m} \omega_{i,j,k}}{m}$
 9:     end if
10: end for
11: return $\{P_{\text{entity},i,j}(p)\}$, $\{P_{\text{literal},i,j}(p)\}$
```

Sub-procedure $f_{\text{literal}}(E_{ki}, t_{kj})$

```
1: if $f_{\text{literal\_exact}}(E_{ki}, t_{kj}) \neq \{\}$ then    ▷ Try exact match
2:     return $f_{\text{literal\_exact}}(E_{ki}, t_{kj})$, $\omega_1$
3: else
4:     return $f_{\text{literal\_fuzzy}}(E_{ki}, t_{kj})$, $\omega_2$    ▷ Fuzzy match
5: end if
```

Algorithm 4 Extended iterative entity disambiguation algorithm

Input: Table T with candidate lists $\{E_{ij}\}$, property distributions $\{P_{\text{entity},i,j}(p)\}$, $\{P_{\text{literal},i,j}(p)\}$, parameters $\{\alpha, \beta\}$, maximum number of iterations N, minimum property confidence threshold δ

Output: Entity assignments $\{\hat{e}_{ij}\}$ for each cell t_{ij}

```
1: $e_{ij}^0 = \text{argmax}_{e \in E_{ij}} \alpha \cdot s_{ij}^{\text{row}}(e) + \beta \cdot \text{lexical\_sim}(e, t_{ij})$    ▷ Initialization
2: $\text{M} = \text{gen\_mask}(\{P_{\text{entity},i,j}(p)\}, \{P_{\text{literal},i,j}(p)\})$    ▷ Generate property mask
3: while $t < N$ and any entity assignment has changed do    ▷ Refinement
4:     $s_{ij}^{\text{col}}(e) = \frac{1}{m-1} \sum_{k=1;k \neq i}^{m} \text{ent\_sim}(e, e_{kj}^{t-1})$    ▷ Column score calculation
5:     $e_{kj}^t = \text{argmax}_{e \in E_{ij}} \alpha \cdot s_{ij}^{\text{row}}(e, \text{M}) + \beta \cdot \text{lexical\_sim}(e, t_{ij}) + (1 - \alpha - \beta) \cdot s_{ij}^{\text{col}}(e)$
6: end while
7: return $\{e_{ij}^t\}$
```

Sub-procedure $s_{ij}^{\text{row}}(e, \text{M})$ ▷ Row score calculation

```
1: $r = 0$    ▷ Number of columns having relationship with the column $j$
2: for $k = 1$ to n do    ▷ Enumerate each column except the column $j$
3:     if $j \neq k$ then
4:         if $\text{M}_{kj}$ then    ▷ Treat column $k$ as subject column, column $j$ as non-subject column
5:             $o_{ik}^{\text{entity}}(e) = \max\limits_{p \in f_{\text{entity}}(E_{ik}, \{e\})} (P_{\text{entity},k,j}(p))$    ▷ The entity property support score from column $k$
6:         else
7:             $o_{ik}^{\text{entity}}(e) = 0$
8:         end if
9:         if $\text{M}_{jk}$ then    ▷ Treat column $j$ as subject column, column $k$ as non-subject column
10:            $s_{ik}^{\text{entity}}(e) = \max\limits_{p \in f_{\text{entity}}(\{e\}, E_{ik})} (P_{\text{entity},j,k}(p))$    ▷ The entity property support score from column $k$
11:            $s_{ik}^{\text{literal}}(e) = \max\limits_{(p,w) \in f_{\text{literal}}(\{e\}, t_{ik})} (P_{\text{literal},j,k}(p) \cdot w)$    ▷ The literal property support score from column $k$
12:        else
13:            $s_{ik}^{\text{entity}}(e) = 0$
14:            $s_{ik}^{\text{literal}}(e) = 0$
15:        end if
16:        if $\text{M}_{kj}$ or $\text{M}_{jk}$ then
17:            $r = r + 1$
18:        end if
19:        $s_{ik}(e) = \max(o_{ik}^{\text{entity}}(e), \max(s_{ik}^{\text{entity}}(e), s_{ik}^{\text{literal}}(e)))$    ▷ Select the better support
20:    end if
21: end for
22: return $\frac{1}{r} \sum_{k=1;k \neq j}^{n} s_{ik}(e)$    ▷ Average property support score from each relational column
```

Sub-procedure $\text{gen\_mask}(\{P_{\text{entity},i,j}(p)\}, \{P_{\text{literal},i,j}(p)\})$ ▷ Generate property mask

```
1: $c = \max\limits_{p \in \mathcal{P}} (\max(P_{\text{entity},i,j}(p), P_{\text{literal},i,j}(p)))$
2: if $c \geq \delta$ then
3:     $\text{M}_{ij} = 1$
4: else
5:     $\text{M}_{ij} = 0$
6: end if
7: return M
```