Dependency-Aware Core Column Discovery for Table Understanding

Jingyi Qiu[1], Aibo Song[1], Jiahui Jin[1](✉), Tianbo Zhang[1], Jingyi Ding[1], Xiaolin Fang[1], and Jianguo Qian[2]

[1] School of Computer Science and Engineering, Southeast University, Nanjing, China
{jingyi_qiu,absong,jjin,tianbozhang,jingyi_ding, xiaolin}@seu.edu.cn

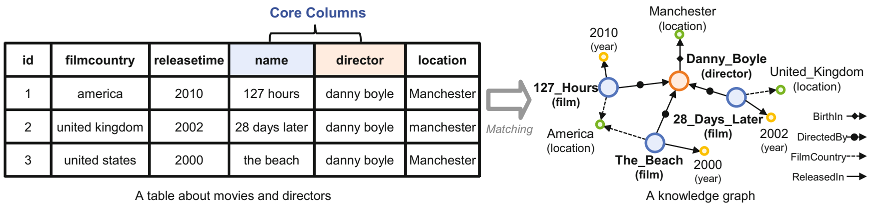
[2] State Grid Zhejiang Electric Power Company, LTD, Hangzhou, China
qian_jianguo@zj.sgcc.com.cn

Abstract. In a relational table, core columns represent the primary subject entities that other columns in the table depend on. While discovering core columns is crucial for understanding a table's semantic column types, column relations, and entities, it is often overlooked. Previous methods typically rely on heuristic rules or contextual information, which can fail to accurately capture the dependencies between columns and make it difficult to preserve their relationships. To address these challenges, we introduce Dependency-aware Core Column Discovery (DaCo), an iterative method that uses a novel *rough matching* strategy to identify both inter-column dependencies and core columns. Unlike other methods, DaCo does not require labeled data or contextual information, making it suitable for practical scenarios. Additionally, it can identify multiple core columns within a table, which is common in real-world tables. Our experimental results demonstrate that DaCo outperforms existing core column discovery methods, substantially improving the efficiency of table understanding tasks.

Keywords: table understanding · core column · semantic dependency

1 Introduction

Tabular data, which includes web tables, CSV files, and data lake tables, is a valuable resource for developing knowledge graphs [11,27] and question answering systems [41]. However, as tabular data often lacks well-defined schemas, the process of table understanding (also known as table interpretation or table annotation) is critical in assigning semantic tags from knowledge graphs to mentions within the table [21,32]. To do this effectively, it is necessary to identify the core columns [53], also known as subject columns, which represent the primary subject entities on which other columns in the table depend [50,51,54]. Once these core columns have been identified, understanding the remaining non-core columns becomes significantly simplified [13,38]. Existing methods for discovering core columns mainly rely on heuristic rules, such as the leftmost rule [52], or contextual information [6,12]. However, these methods may not be practical or applicable in scenarios like data lakes or enterprise tables [19,20], where structures are often incomplete, data is noisy, and there may be multiple core columns. Therefore, identifying inter-column dependencies and discovering core columns can be challenging.



id	filmcountry	releasetime	name	director	location
1	america	2010	127 hours	danny boyle	Manchester
2	united kingdom	2002	28 days later	danny boyle	manchester
3	united states	2000	the beach	danny boyle	Manchester

Fig. 1. A table about movies and directors. Prior works usually choose the leftmost column *i.e.,*"id" as the core column, but the values within the "id" column are general and do not refer to entities in the knowledge graph [18]. In contrast, our method selects the "name" and "movie director" columns as core columns because they correspond to the entities on which the entities listed in the "filmcountry", "releasetime", and "location" columns depend, according to the knowledge graph.

This paper aims to tackle the problem of core column discovery for table understanding by identifying the minimum set of columns that other columns depend on[1]. While discovering core columns is crucial for table understanding, it is often overlooked [6]. Existing methods rely on heuristic rules [16,52,54] or machine learning techniques [45], but these may not capture true dependencies [24,47] or require large amounts of labeled data. Pre-trained models show promise [10,12,44,48], but their effectiveness depends on the corpus they were trained on and may not adapt well to specific domains. Unlike keys discovered based on uniqueness in data profiling, core column discovery focuses more on semantic inter-column dependency, *i.e.,*whether there are entity-attribute relationships [24,53,54]. Furthermore, research on discovering multiple core columns often suffers from poor performance due to their reliance on exact matching [18,46].

Identifying core columns while preserving their dependencies is a challenging task. Using a knowledge graph (KG) as a reference may appear to be a solution, but it is intricate due to *(1) Data incompleteness and inconsistency.* Real-life KGs are inherently incomplete, which means that a significant portion of mentions in a table may not have corresponding entries in the KG, named unlinkable mention [36]. Additionally, inconsistencies in representation between tables and KGs can make it challenging to map these mentions accurately. *(2) Noisy table metadata.* Noisy column names, captions, or lack of metadata [17] in tables make it difficult to annotate column types accurately, which is crucial for determining dependent relationships among columns. *(3) Multiple core columns.* Detecting multiple core columns [18,46] in tabular data is challenging due to their unpredictable number and locations. Although prior

[1] By dependency, we are implying that column y is an attribute of column x if y depends on x [28].

research [8,26,27,36,49,55] has attempted to tackle these issues, they are computationally expensive and rely on the success of core column discovery, leading to a dilemma.

We propose a solution called Dependency-Aware Core Column Discovery (DaCo) to address the above challenges. DaCo is a two-level iterative process that identifies inter-column dependencies and core columns. Outer iteration performs sampling and discovery iteratively until the termination test is satisfied, which considerably reduces computational complexity when matching tables to knowledge graphs. Inner iteration identifies the core columns based on the sampled rows. It establishes a rough matching between each mention (column type) and a set of candidate KG entities (KG types), providing a coarse understanding of the table, which is necessary for DaCo to handle situations where the table has only small portions of overlaps with the KG[2]. Based on this rough matching, DaCo calculates the dependency score for each pair of columns. If the score exceeds a certain threshold, we consider the existence of an dependent relationship between the columns and identify the core column set according to the inter-column dependencies. Once a core column is found, we update the threshold and repeat the inner iteration until the core column set size converges. *To the best of our knowledge,* DaCo *is the first approach to discover the core column for table understanding while preserving inter-column dependencies without requiring table metadata, contextual information and exact table-to-KG matching.*

Overall, our contributions are as follows. *(1) Rough matching and dependency scoring.* We propose a novel approach that combines rough matching and dependency scoring to explore the semantic dependencies between columns, which enables the extraction of a core column based on inter-column dependencies. *(2) Two-level iterative algorithm.* To improve the effectiveness and efficiency of the dependency and core column discovery, we introduce a two-level iterative algorithm, named DaCo, that performs sampling in outer iteration and discovery in inner iteration until convergence. *(3) Extensive evaluations.* Extensive experiments were conducted on core column discovery and table understanding, revealing that DaCo outperforms other approaches in terms of effectiveness with precision of 85%-90% on core column set discovery and nearly 40% improvement on fine grained column type prediction and relation extraction tasks. Source code is provided on github[3].

The rest of our paper is organized as follows. Sect. 2 provides a definition of the core column discovery problem. Rough matching strategy and the dependency score are presented in Sect. 3. The two-level iterative algorithm is presented in Sect. 4. Our experimental results and related works are showcased in Sect. 5 and 6, respectively. Finally, we conclude our paper in Sect. 7.

2 Preliminary

A review of basic notations is listed in Table 1. Assume three infinite alphabets, $\varUpsilon$, $\varTheta$, and $\varPhi$ for table mentions, graph node labels, and graph edge labels.

[2] Our solution can be applied to table understanding tasks since research on table understanding assumes an overlap between the table and the KG.

[3] https://github.com/barrel-0314/daco.

Definition 1 (Table and Column Set). *A table T is a collection of data organized into m rows and n columns, each cell contains a mention, which refers to the textual content for representing an entity in KG and often shares relationships with other mentions in the same row. The mention in i^{th} row and the x^{th} column of table T is denoted by $t_{i,x}$, where $t_{i,x} \in \Upsilon$. $S \subseteq X$ is a column set of T, where set X is defined as the set of all possible column indices in the table, ranging from 1 to n. The projection of T on column x is represented as $T[x]$.*

Table 1. Notations and their descriptions.

Notations	Descriptions
T, $T[x]$, $t_{i,x}$	a table, the x^{th} column of table T, the mention located in the i^{th} row and the x^{th} column in T
C, G, v, τ	core column set, KG, matching function for mention and type
$\mathsf{dep}(x, y)$, ζ	semantic dependency score between column x and y, threshold of semantic dependency
$\hat{v}^k_{i,x}$, θ_x	the k^{th} entity in rough matching of $t_{i,x}$, a type in $\tau(x)$
q, η, γ	parameter for calculating $\mathsf{dep}_\tau(\cdot, \cdot)$, the threshold and decay parameter of iterations
a,$\hat{a}$,ε	the ground truth, the sample and a threshold of a
T_s, m_s	sampled table and the number of sampling rows

To avoid the influence of noisy metadata, we exclude headers and captions from our definition, which distinguishes our approach from previous works [12,54]. Without contextual information, annotating mentions, column types, and inter-column relationships by matching tables to a KG is crucial for accurately identifying inter-column dependencies.

Remark: We assume that each cell in the table refers to a specific entity. However, it's common for tables to include cells that do not correspond to any particular entity. For example, in Fig. 1, the values in the "id" column may not have corresponding entities in a KG. Even if numerical strings can be treated as text for candidate selection, they often matches literals in the KG rather than entities, which means it lacks type attributes or does not share consistent type in the same column. Besides, the "id" column lacks connecting edges with mentions in the KG within the same row. In this case, the content of the "id" column cannot be considered as mentions and does not provide entity information. Therefore, when computing the semantic dependency relationships and identifying the core columns, we exclude columns where every cell cannot be matched to the KG.

Definition 2 (Knowledge Graph). *A knowledge graph, denoted by $G = (V, E, L)$, is a directed graph that contains nodes representing entities and edges connecting the nodes. The set of nodes and edges are denoted by V and E respectively. A labeling function represented by L is also present in the graph. Each node $v \in V$ and edge $e \in E$ has a label denoted by $L(v) \in \Theta$ and $L(e) \in \Phi$ respectively. We can represent the type of a node v as the value of its"type" attribute. We use θ to denote a type. Because an entity may be related to multiple types, we use set $\tau(v)$ to denote the type set of v.*

Assuming that the KG is complete and that the table can completely overlap the KG, we can match the table and KG exactly by using table annotation. To achieve this, we use the entity annotation function $v(\cdot)$ to map a table mention to a corresponding entity in the KG. In other words, for any given table mention $t_{i,x}$, we have $v(t_{i,x}) \in V$. Additionally, we use the type annotation function $\tau(\cdot)$ to assign a set of semantic types to each column in the table. Specifically, for any given column x, we have $\tau(x) \subseteq \cup_{v\in V}(\tau(v))$. Furthermore, we use the relationship annotation function $\rho(\cdot,\cdot)$ to assign a property from the KG to the relationship between two columns. In other words, for any two columns x and y, we have $\rho(x,y) \in \cup_{e\in E}(\{L(e)\})$. If this is no relationship between x and y, we let $\rho(x,y) = null$. With the exact table annotation, we define the semantic dependency between two columns as follows:

Definition 3 (Semantic Dependency). *Given two columns x and y in the same table, we can determine if column y is semantically dependent on column x by examining each pair of mentions $t_{i,x}$ and $t_{i,y}$. If there exists a KG edge e_i starting from the entity represented by $t_{i,x}$ and pointing to the entity represented by $t_{i,y}$ for every row i, we establish that column y is semantically dependent on column x. Formally, if each edge $e_i = (v(t_{i,x}), v(t_{i,y})) \in E$ satisfies the relation label $L(e_i) = \rho(x,y)$, then we denote this semantic dependency as $x \rightarrow y$.*

We expand the idea of column-wise semantic dependency to include column-set-wise semantic dependency. We denote the semantic dependency between two sets of columns, S_1 and S_2, as $S_1 \rightarrow S_2$. Here, S_1 and S_2 are subsets of X, and for each y in S_2, there exists an x in S_1 that satisfies $x \rightarrow y$. Also we let column x depend on itself, *i.e.,* $x \rightarrow x$. In the subsequent sections, we will refer to semantic dependency as "dependency" if there is no confusion.

Example 1. Consider the table in Fig. 1. There is a semantic dependency between the 5^{th} and 6^{th} columns, specifically, *i.e.,*$5 \rightarrow 6$. This relationship stems from the presence of an edge labeled as "BirthIn" between $v(\text{“}dannyboyle\text{”})$ and $v(\text{“}manchester\text{”})$, meaning that $\rho(5,6) = \text{“}BirthIn\text{”}$. Additionally, we can also observe a semantic dependency between the "name" column and several other columns (*i.e.,*"filmcountry", and "releasetime"), resulting in a column-set-wise dependency of $\{4,5\} \rightarrow \{2,3,6\}$.

Because KG is far from complete and overlapping for real-life table to be matched, accurately identifying inter-column dependencies becomes infeasible. To account for real-world situations, we propose a dependency score $\mathsf{dep}(x,y)$ that gauges the possibility of column y being dependent on column x. If the value of $\mathsf{dep}(x,y)$ surpasses a predetermined threshold ζ, then we establish a relation between x and y, such that $(\mathsf{dep}(x,y) > \zeta) \Rightarrow (x \rightarrow y)$. Sect. 3 explains how to calculate $\mathsf{dep}(x,y)$. By utilizing these dependency scores, we can define the core column discovery problem as follows.

Definition 4 (Core Column Discovery Problem). *Given a table with the set of columns (denoted by X), the problem is to discover a set of core columns C that is the smallest subset of X on which all other columns in the table depend, i.e.,$C \rightarrow X$ and for all C' that satisfies $C' \rightarrow X$, $|C| \leq |C'|$.*

Determining the core column remains an NP-hard problem even when the dependency scores $\mathsf{dep}(\cdot,\cdot)$ and the threshold ζ are provided. This can be demonstrated through a reduction from the directed dominating set problem [39]. In the next sections, we show how to measure $\mathsf{dep}(\cdot,\cdot)$ (Sect. 3) and how to iteratively compute $\mathsf{dep}(\cdot,\cdot)$ to discover the core column (Sect. 4).

3 Rough Matching and Dependency Scores

This section aims at measuring the dependency score $\mathsf{dep}(\cdot,\cdot)$. The semantic dependency definition implies that if column y depends on column x, each pair of corresponding entities $v(t_{i,x})$ and $v(t_{i,y})$ should share the same relationship as the columns themselves. In other words, $\rho(x,y) = L((v(t_{i,x}), v(t_{i,y})))$, where $t_{i,x}$ and $t_{i,y}$ are mentions in the i^{th} row of columns x and y, respectively. However, this entity-centric matching approach may be too strict for real-life tables and KGs, since the mentions in the table may not always be linkable to entities in the KG. Additionally, there might not exist a shared relationship capable of connecting all pairs of matched entities due to the incompleteness of KG.

To overcome these limitations, we propose a column-centric strategy called *rough matching* to replace the fine-grained entity-centric matching. By taking the column-centric approach, we can relax the constraints on linkability and account for the incompleteness of KGs which provides a more flexible and robust method for measuring the dependency score of real-world tables.

3.1 Rough Matching

The rough matching approach generate candidate entities for each mention and assigns potential types $\tau(x)$ to each column x, rather than providing precise annotations. Let $t_{i,x}$ be the i^{th} mention in column x. In this approach, we produce the top-K candidate entities for $t_{i,x}$, denoted as $\hat{\mathrm{v}}(t_{i,x}) = \{\hat{v}^k(t_{i,x})\}_{k=1}^{K}$, where $\hat{v}^k(t_{i,x})$ represents the k^{th} candidate entity. Note that any label similarity function can be employed to select candidates by comparing the mention and the entity name, and we use edit distance as the label similarity measure here. For simplicity, we represent $\hat{v}^k(t_{i,x})$ as $\hat{v}^k_{i,x}$. To determine a potential type set for column x, we extract the types of mentions' candidate entities from the knowledge graph by using the edges labeled "type" and then merge the type sets. We accomplish this by combining the sets $\tau(\hat{v}^k_{i,x})$ for all $i \in [1,m]$ and $k \in [1,K]$, where m is the number of rows and $\tau(\cdot)$ is a function that assigns type sets to entities and columns. The resulting union is the type set of column x, *i.e.*,$\tau(x)$.

In our approach, the type sets $\tau(x)$ and $\tau(y)$ are essential for calculating $\mathsf{dep}(x,y)$. Nevertheless, $\tau(\cdot)$ may include many irrelevant types as the top-k candidates chosen for each mention might not pertain to the table's topic. To filter those irrelevant types, we define a score function $s_\tau(\theta,x)$ to determine the relevance of a particular type θ to column x as follows:

$$s_\tau(\theta,x) = \frac{1}{mK}\sum_{i=1}^{m}\sum_{k=1}^{K} w^k_{i,x} s^v_\tau(\theta, \hat{v}^k_{i,x}), \tag{1}$$

where $s_{\tau}^{v}(\theta, \hat{v}_{i,x}^{k}) = \mathbb{I}[\theta \in \tau(\hat{v}_{i,x}^{k})]$ and $\mathbb{I}[\cdot]$ is the indicator function that takes the value 1 if its argument is true and 0 otherwise determines whether $\theta \in \tau(\hat{v}_{i,x}^{k})$ and takes the value 1 if it does, otherwise it is 0. $s_{\tau}(\theta, x)$ can be seen as the weighted sum of the $s_{\tau}^{v}(\theta, \hat{v}_{i,x}^{k})$ for each candidate.

We employ the weight $w_{i,x}^{k}$ to distinguish whether $\hat{v}_{i,x}^{k}$ is a correct match for $t_{i,x}$. Ideally, all correct entities corresponding to the same column should have the same set of types. Therefore, we use the Jaccard function to measure the similarity between the type set of $\hat{v}_{i,x}^{k}$ and those of other candidates. If the types of $\hat{v}_{i,x}^{k}$ are more consistent with those of other candidates, it is more likely to be a correct match. To compute $w_{i,x}^{k}$, we use the following equation:

$$w_{i,x}^{k} = \frac{\sigma_{i,x}^{k}}{mK} \sum_{j=1}^{m} \sum_{l=1}^{K} Jaccard(\tau(\hat{v}_{i,x}^{k}), \tau(\hat{v}_{j,x}^{l})), \tag{2}$$

where the weight $\sigma_{i,x}^{k}$ evaluates whether $\hat{v}_{i,x}^{k}$ correctly matches $t_{i,x}$ based on the inter-column relationship. Since the inter-column relationship is not predetermined, our algorithm adjusts $\sigma_{i,x}^{k}$'s value dynamically with Eq.(7).

After computing $s_{\tau}(\theta, x)$ for each type θ, we cluster the types into two groups using k-means ($k = 2$) and select the types with the highest scores as $\tau(x)$. We also select candidate entities whose types belong to the updated $\tau(x)$. Here, k-means is used rather than selecting top-k types or setting a threshold because k-means eliminates the need to set parameter values.

3.2 Dependency Score

After performing a rough matching, we proceed to calculate the dependency score $\mathsf{dep}(x, y)$ for columns x and y. To consider that the semantic dependencies between corresponding entities have similar relationships as the columns themselves, we divide the score into two parts. The first part, denoted as $\mathsf{dep}_{\hat{v}}(x, y)$, considers the dependencies between candidate entities in $\hat{\mathbf{v}}_{i,x}$ and $\hat{\mathbf{v}}_{i,y}$. The second part, denoted as $\mathsf{dep}_{\tau}(x, y)$, evaluates the dependency between two columns based on $\tau(x)$ and $\tau(y)$. Then $\mathsf{dep}(x, y)$ is a linear combination of the two part of scores, which is as follows.

$$\mathsf{dep}(x, y) = \alpha \mathsf{dep}_{\hat{v}}(x, y) + (1 - \alpha)\mathsf{dep}_{\tau}(x, y) \tag{3}$$

where $\alpha \in [0, 1]$ is a parameter for balancing $\mathsf{dep}_{\hat{v}}(\cdot, \cdot)$ and $\mathsf{dep}_{\tau}(\cdot, \cdot)$ and defaults to 0.5.

We show how to compute $\mathsf{dep}_{\hat{v}}(x, y)$. Our main idea is to count the number of edges in the KG that link the candidate entities of column x with those of column y. When there are more edges present, it suggests a stronger likelihood of interdependence between these columns. Formally, if there exists a relationship $\rho(x, y) \in \Phi$ connecting x and y, the corresponding entities should be connected by edges in E between two columns, *i.e.*,$(v(t_{i,x}), v(t_{i,y})) \in E$. However, it is difficult to calculate an exact value for $\rho(x, y)$ using $\hat{v}_{i,x}$ and $\hat{v}_{i,y}$. Nonetheless, since the number of edges $|E|$ is considerably smaller $|V|^2$ in a KG, there are barely any edges existed between two randomly selected entities. Hence, if y does not depend on x, it is difficult for there to exist edges from $\hat{\mathbf{v}}_{i,x}$ to $\hat{\mathbf{v}}_{i,y}$. Conversely, x has a higher dependency with y when there are several edges from $\hat{\mathbf{v}}_{i,x}$ to $\hat{\mathbf{v}}_{i,y}$, which can be used as an indicator for the existence of $\rho(x, y)$. Based on this observation, we compute $\mathsf{dep}_{\hat{v}}(x, y)$ as follows:

$$\mathsf{dep}_{\hat{v}}(x,y) = \frac{n_e(x,y) - \min\{n_e\}}{\max\{n_e\} - \min\{n_e\}} \tag{4}$$

where $n_e(x,y) = \sum_{i=1}^{m} \sum_{\hat{v}_{i,x}^k \in \hat{\mathbf{v}}_{i,x}} \sum_{\hat{v}_{i,y}^l \in \hat{\mathbf{v}}_{i,y}} \mathbb{I}[(\hat{v}_{i,x}^k, \hat{v}_{i,y}^l) \in E]$. $n_e(x, y)$ represents the total number of edges between candidates $\hat{\mathbf{v}}_{i,x}$ and $\hat{\mathbf{v}}_{i,y}$ in each row of columns x and y. To normalize $\mathsf{dep}_{\hat{v}}(x, y)$ to the range of $[0, 1]$, we apply min-max normalization to $n_e(x, y)$. Here, $\max\{n_e\}$ and $\min\{n_e\}$ represent the max and min values of n_e over all pairs of columns, respectively.

Next, we demonstrate the computation of $\mathsf{dep}_{\mathcal{T}}(x, y)$, which tackles scenarios where a relationship exists between two columns, but there are no direct connections among the candidate entities. Our main idea is to assess the inter-column dependency by examining the correlation between the candidate types of column x, *i.e.,*$\tau(x)$, and the candidate types of column y, *i.e.,*$\tau(y)$ based on the KG ontology. This involves computing the correlation of θ_x and θ_y for each type pair (θ_x, θ_y) where $\theta_x \in \tau(x)$ and $\theta_y \in \tau(y)$. The correlation, denoted as $Corr(\theta_x, \theta_y)$, takes into account the ontology and can be computed using the method described in [58], which considers the specificity of types and the distance between types in the ontology graph. However, the method presented in [58] can only determine the existence of a relationship between columns x and y, without indicating its direction, *i.e.,*whether it is $x \to y$ or $y \to x$. To address this issue, we calculate the proportion of head nodes with type θ_x and θ_y, denoted as $h(\theta_x)$ and $h(\theta_y)$, respectively. The ratio of $h(\theta_x)/(h(\theta_x) + h(\theta_y))$ serves as a useful reference for determining the direction of dependency between x and y. If this value is greater, it signifies that θ_x has more head entities and θ_y has fewer head entities, implying that the direction of dependency is more likely to be $x \to y$ rather than $y \to x$. Overall, we have

$$\mathsf{dep}_{\mathcal{T}}(x,y) = \frac{\mathcal{T}(x,y) - \min_{x',y' \in X, x' \neq y'} \mathcal{T}(x',y')}{\max_{x',y' \in X, x' \neq y'} \mathcal{T}(x',y') - \min_{x',y' \in X, x' \neq y'} \mathcal{T}(x',y')} \tag{5}$$

where $\mathcal{T}(x,y) = \sum_{\theta_x \in \tau(x)} \sum_{\theta_y \in \tau(y)} \frac{h(\theta_x)}{h(\theta_x)+h(\theta_y)} \sqrt[q]{Corr(\theta_x, \theta_y)}$. To balance the scores, we introduce the parameter q due to the exponential nature of $Corr(\theta_x, \theta_y)$ as calculated by [58], where $q > 0$. This equation considers potential type matches, computes scores for each pair, and normalizes them to 0 to 1. The score is determined by a combination of the relationship between the two column types in the ontology graph and the proportion in which the two column types appear as head entities in the knowledge graph. Moreover, since the values of $\mathsf{dep}_{\mathcal{T}}(\cdot, \cdot)$ for different (θ_x, θ_y) pairs are always greater than zero, it is possible to identify the core column using $\mathsf{dep}(x, y)$ even if $\mathsf{dep}_{\hat{v}}(x, y) = 0$.

4 DaCo: A Two-Level Iterative Algorithm

This section presents DaCo, a two-level iterative algorithm for identifying core columns based on dependency scores. The algorithm includes an outer iteration and an inner iteration (Alg. 1). The outer iteration handles large tables using a sampling method with a termination check to avoid biased samples. The inner iteration refines rough matching results to discover core columns. An example of applying DaCo is shown in Fig. 2, which demonstrates the process of discovering core columns from the table of Fig. 1.

4.1 Outer Iteration

In the process, the outer iteration randomly selects rows from a table and proceeds with the inner iteration until the termination check is met. To be specific, every cycle of the outer iteration involves the following steps.

- **Initialization** (lines 2–5). We randomly select m_s rows for T to form a sub-table T_s and generate the candidate entities for each mention in T_s.

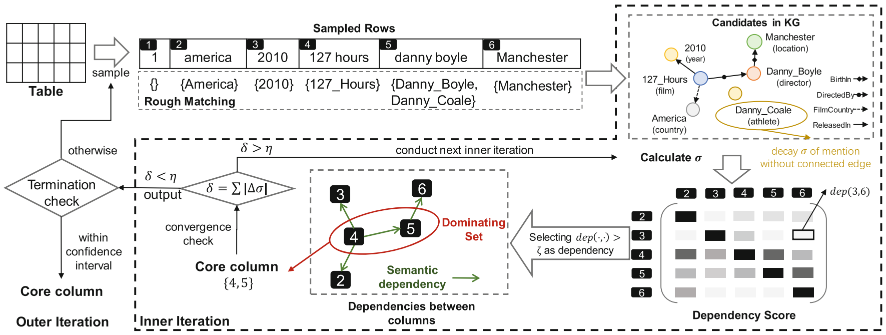


Fig. 2. An example of DaCo algorithm : (i) In the outer iteration, it keeps sampling rows and discovering the core columns from the table until the results are unbiased. Before inner iteration, it conducts rough matching for each mention of the sampled rows. If one column (*e.g.,* column 1) cannot generate entity candidates in KG or its candidates do not connect with candidates in other columns, this column will be excluded for the following operations. (ii) In the inner iteration, the weights of candidate entities are adjusted based on their connecting edges in the KG, and the dependency scores are computed. Based on the calculated dependency scores, core column discovery can be transformed into a problem of finding the dominating set in a directed graph: Set a threshold and consider each column as a vertex, connecting two columns if their dependency score exceeds the threshold. The inner iteration terminates and outputs the set of core columns when erroneous candidates such as $Danny\_Coale$ are filtered out, and the candidates' weights are stabilize, ensuring the convergence of the process.

- **Perform Inter Iteration** (lines 6–13). We iteratively determine the core column set C based on the sampled rows with detailed information in Sect. 4.2.
- **Termination Check** (line 14). To confidently terminate the algorithm, it is necessary to verify that the core column set C discovered from T_s is equivalent to that obtained from T. We represent a core column set as an n-bit feature vector $\mathcal{C}$, where if x is a core column, then $\mathcal{C}[x] = 1$, otherwise $\mathcal{C}[x] = 0$. The feature vectors of the core column sets discovered from T_s, T, and row i, are denoted by $\mathcal{C}$, $\mathcal{C}_T$, and $\mathcal{C}_i$, respectively. Since $\mathcal{C}_T$ is not known in advance, it must be discovered from each row. For each column x, we have defined a function $a(x)$ to measure the degree of consistency between $\mathcal{C}_T[x]$ and $\mathcal{C}_i[x]$ for all rows i, such that $a(x) = \frac{1}{m}\sum_{1\le i\le m}[\mathcal{C}_T[x] \equiv \mathcal{C}_i[x]]$.

By introducing $a(x)$ and Rademacher random variables [33], we can estimate the confidence interval of the samples, *i.e.,* whether the core column of the sample is consistent with the one of the entire table. If a sample falls within the confidence interval, it indicates that the sample is unbiased and the result can be returned. Otherwise, it suggests that the biased sample causes its core column inconsistent with the one of the entire table, requiring a new round of sampling. Additionally, we compute $\hat{a}(x) = \frac{1}{m}\sum_{1 \le i \le m_s}[\mathcal{C}[x] \equiv \mathcal{C}_i[x]]$ for T_s. We have the following theorem regarding the confidential interval of $\hat{a}(x)$.

Theorem 1. *Given a confidence level* $1-\varepsilon > 0$*, the confidence interval of* $\hat{a}(x)$ *satisfies the following bound:*

$$Pr\left(\sup_{x \in X} |\hat{a}(x) - a(x)| \le 2 \max_{x \in X} \sqrt{\frac{2\hat{a}(x) \cdot \ln(n)}{m_s}} + \sqrt{\frac{2\ln(2/\varepsilon)}{m_s}}\right) \ge 1 - \varepsilon. \quad (6)$$

Algorithm 1: DaCo Algorithm

```
Input: Table $T$, iteration threshold $\eta$
Output: Core column set $C$
while $flag = False$ ;                                        /* Outer iteration */
do
    sample $T_s$ from $T$;
    generate $\hat{\mathbf{v}}$ for each mention in $T_s$;    /* Generate candidate entities */
    $r \leftarrow 0, \quad \delta \leftarrow \infty$;
    while $\delta > \eta$ ;                                   /* Inner iteration */
    do
        calculate $\sigma$ with Eq. (7);                      /* Compute weights of candidates */
        calculate dep$(\cdot,\cdot)$ with Eq. (3);            /* Compute dependency scores */
        $\zeta \leftarrow$ k-means(dep$(\cdot,\cdot)$);       /* Update $\zeta$ */
        $C \leftarrow$ CoreColumnSet($\zeta$, dep$(\cdot,\cdot)$) ;   /* Discover core column set */
        $r \leftarrow r + 1$;
        $\delta \leftarrow$ ConvergenceCheck($\zeta$);
    $flag \leftarrow$ TerminationCheck($C, T_s, \varepsilon$);
return $C$
```

Sketch of Proof. Theorem 1 can be proved by introducing Rademacher random variable and Massart's lemma [33].

The termination check of the algorithm involves computing a bound based on Theorem 1. This bound is evidently determined by ε, C, and T_s. If the resulting bound is less than a predetermined threshold, the algorithm terminates.

4.2 Inner Iteration

During the inner iteration, we use sampled sub-table to generate core column sets. The following steps are repeated in each iteration.

- **Update Dependency Scores** (lines 8–9). In each iteration, we update the dependency scores dep$(\cdot,\cdot)$ of all pairs of columns by adjusting the weights of candidate entities through Eq. (2). This involves changing the value of $\sigma^k_{i,x}$ which evaluates candidate entities based on the relationship between the two columns. To achieve

this, we initialize $\sigma_{i,x}^{k,0}$ as 1 and compute $\sigma_{i,x}^{k,r}$ in the r^{th} iteration by Eq. (7) using the core column set discovered in the previous iteration.
The main idea of Eq. (7) is to determine whether a candidate is a correct match entity by examining the existence of connecting edges between candidates, and to lower the weight of candidates that are more likely to be incorrect matches. Assuming $t_{i,x}$ is a mention in a core column, we determine the correctness of $\hat{v}_{i,x}^{k}$ by examining whether there exists an edge between it and a candidate of any non-core column. If such an edge exists, we set $\sigma_{i,x}^{k,r}$ to $\sigma_{i,x}^{k,r-1}$. Otherwise, we reduce the importance of $\hat{v}_{i,y}^{k}$ by multiplying it with factor γ, where $0 < \gamma < 1$. Consequently, we compute $\sigma_{i,x}^{k,r}$ as either $\sigma_{i,x}^{k,r-1}$ or $\gamma\sigma_{i,x}^{k,r-1}$:

$$\sigma_{i,x}^{k,r} = \begin{cases} 1, & \text{if } r = 0 \\ \sigma_{i,x}^{k,r-1}, & \text{if } r > 0 \wedge \exists \hat{v}_{i,y'}^{l} \text{ such that } (\hat{v}_{i,x}^{k}, \hat{v}_{i,y'}^{l}) \in E \\ \gamma\sigma_{i,x}^{k,r-1}, & \text{otherwise,} \end{cases} \tag{7}$$

where y' is any non-core columns. In a similar way, we compute the weight $\sigma_{i,y}^{k,r}$ for candidate entities in the non-core columns. Once all the weights have been computed, we update the dependency scores with Eq. (3).

- **Discover Core Column Set** (lines 10–11). After $\mathsf{dep}(\cdot,\cdot)$ is updated, we utilize a k-means ($k = 2$) cluster algorithm to divide $\mathsf{dep}(\cdot,\cdot)$ into two groups, one with higher $\mathsf{dep}(\cdot,\cdot)$ and the other with lower $\mathsf{dep}(\cdot,\cdot)$. ζ is set as the smallest value in the higher-score group. Then, C can discovered by finding the minimize size of core column set that satisfies $\mathsf{dep}(x, y) > \zeta$ for $\forall x \in C, y \in X \backslash C$. Since the problem is NP-hard, we apply a heuristic method where we repeatedly identify and mark a column as the core column if it results in the determination of the maximum number of non-core columns.
- **Convergence Check** (line 13). The inner iteration stops when the core column set remains unchanged. To track the progress of convergence, we use the change of $\sigma_{i,x}^{k,r}$ as a metric. When $\sigma_{i,x}^{k,r}$ doesn't change, the output of $s_\mathcal{T}(\theta, x)$, $\tau(x)$, $\mathsf{dep}(x, y)$ also remain unchanged, which leads to a converged core column C. Therefore, to check for convergence, we need to identify any differences in σ, specifically $|\sigma_{i,x}^{k,r} - \sigma_{i,x}^{k,r-1}|$. We define the check variable δ as $\sum_{x\in X}\sum_{i=1}^{m}\sum_{k=1}^{K}(|\sigma_{i,x}^{k,r} - \sigma_{i,x}^{k,r-1}|)/(mnK)$. Convergence of $\sigma_{i,x}^{k,r}$ can be proved by applying Cauchy's convergence criterion [30]. It follows that both $\tau(x)$ for every column x, $\mathsf{dep}(x, y)$ and C are also convergent, indicating the guaranteed convergence of the inner iteration. To terminate the inner iteration, we set the threshold η to stop at $\delta < \eta$. In Sect. 5.6, we discuss the impact of η.

5 Experiment

We conduct experiments to evaluate the performance of DaCo. There are four research questions to seek in this section:

- **RQ1:** How does DaCo perform compared with other baselines of core column set discovery task?

- **RQ2:** To what extent do multiple core columns, unlinkable mention portion, and the number of sampled rows impact the effectiveness of DaCo?
- **RQ3:** What is the impact of parameter settings on the performance of DaCo?
- **RQ4:** How does DaCo improve the performance of table understanding?

5.1 Experimental Settings

Dataset: We use two main data sources:

- **Table Corpus:** We conducted experiments for core column set discovery on SEM[4] [3], T2D [1], GIT [2], WIKI [4] and TUS [34]. In all datasets, except T2D, we manually identified the single core column in each table. We excluded blank tables and tables without core columns in GIT. For TUS, we labeled it by sampling 300 tables. Additionally, we introduce a new dataset called MULTICC, consisting of 94 tables, which is collected from SEM and WIKI for multiple core column discovery, by adding new columns generated with reference to the KG. The statistics of these datasets are summarized in Table 2. We use the portion of unlinkable mentions (UMP) in each table corpus to indicate the overlap between tables and KGs.
- **Knowledge Graphs:** We use subsets of DBpedia [25], *i.e.,mappingbased_ objects_en*, *mappingbased_literals_en*, *instance_types_en*, *DBpedia Ontology*.

Table 2. Statistics of table datasets.

Dataset	$\|T\|$	$\bar{m}$	$\bar{n}$	$\max_m$	$\max_n$	$\sum m$	$\sum n$	*UMP[%]*
SEM	180	1080.21	4.46	15,478	8	194,438	803	6.74
T2D	233	121.60	4.950	586	14	28,333	1,153	30.53
GIT	460	47.27	18.00	1,015	75	38,478	14,652	53.92
WIKI	428	35.55	3.79	465	6	15,215	1,622	61.10
TUS	5049	1932.12	10.98	4,987	44	9,755,274	55,438	>72.12
MULTICC	94	1132.23	3.47	15,477	7	106,430	326	28.99

Baselines: We compared DaCo with ten baseline methods, including five heuristic-based methods (LEFT, UNI, SUP, CONN, MIX [14]), two database-based methods (HPIValid (HPI) [5], GORDIAN (GOR) [40]), and three machine learning-based methods (SVM [45], NK [6], TURL [12]). SUP, CONN, and MIX are based on disambiguation results and linked edges between columns. HPI and GOR are key discovery approaches in data profiling, where we implement modules such as preprocessing, sampling, tree search and validation of HPI, prefix tree creation and merging, finding non-keys, pruning, and computing keys from non-keys of GOR. SVM and NK are supervised learning methods. TURL is a pretrained table representation model, which

[4] It includes tough tables generated from SemTab for dealing with the tabular data to KG matching problem.

Table 3. Precision, accuracy and recall of core column discovery. Pre, Acc and Rec represent precision, accuracy and recall, respectively. The bolds denote the best results.

	Method	SEM			T2D			GIT			WIKI			TUS		
		Pre	Acc	Rec	Pre	Acc	Rec	Pre	Acc	Rec	Pre	Acc	Rec	Pre	Acc	Rec
Heuristic	LEFT	.811	.898	.811	.906	.941	.906	.594	.865	.594	.624	.778	.624	.380	.862	.380
	UNI	.617	.816	.617	.644	.790	.644	.757	.911	.757	.675	.794	.675	.280	.859	.280
	SUP	.378	.590	.378	.498	.663	.498	.622	.868	.622	.304	.601	.304	.200	.825	.200
	CONN	.439	.628	.439	.451	.642	.451	.639	.878	.639	.344	.631	.344	.180	.816	.180
	MIX	.361	.571	.361	.506	.671	.506	.598	.857	.598	.278	.584	.278	.160	.814	.160
DB	HPI	.633	.810	.633	.691	.812	.691	.735	.904	.735	.703	.811	.703	.400	.882	.400
	GOR	.628	.817	.628	.640	.778	.640	.737	.903	.737	.668	.789	.668	.337	.862	.337
ML	SVM	.400	.498	.400	.652	.474	.652	.587	.854	.587	.175	.482	.175	.060	.661	.060
	NK	.806	.810	.806	.871	.902	.871	.535	.761	.535	.643	.726	.643	.360	.858	.360
	TURL	.683	.689	.683	.652	.776	.652	.613	.834	.613	.591	.695	.591	.080	.760	.080
DaCo		**.850**	**.901**	**.850**	**.949**	**.970**	**.949**	**.798**	**.930**	**.798**	**.918**	**.959**	**.918**	**.800**	**.959**	**.800**

we fine-tuned it by using the cross-entropy loss function and 70 tables in T2D with their core column labels.

Parameter Default Configurations: Parameter q for calculating $d(x, y)$ is 1, convergence threshold η=0.01, decay parameter γ=0.85 [4], sample number m_s=10.

5.2 Overall Performance (RQ1)

In this study, precision, accuracy, and recall results of different approaches are reported on five table corpora (Table 3). Notice that all tables in the corpus have a single core column, resulting in identical precision and recall values.

The evaluation of various methods indicates DaCo's superiority over state-of-the-art baselines in all five benchmark datasets. DaCo attains precision improvements ranging from 3.9%-40% and accuracy improvements from 0.3%-14.8%. Precision ranges between 80% to 95% and accuracy varies from 90%-97% across different table corpora. DaCo performs better than baselines even with incomplete data (e.g., WIKI and TUS), making it effective for various types of tables, particularly those with more unlinkable mentions. These results demonstrate DaCo's robustness for core column detection in table understanding.

5.3 Effects of Unlinkable Mentions (RQ2-1)

We conducted an experiment to study the impact of Unlinkable Mention Portion (UMP) in tables by randomly setting cells to blank with varying rates to simulate UMP. The results of this experiment are presented in Fig. 3.

Our experimental findings demonstrate that DaCo outperforms existing approaches, particularly when dealing with tables containing numerous unlinkable mentions. Despite a decrease in precision with an increase in UMP, DaCo achieves a precision of over 70% and yields improvements of approximately 10% and 20% on GIT and WIKI, respectively, when UMP is 90%. The superior performance of DaCo can be

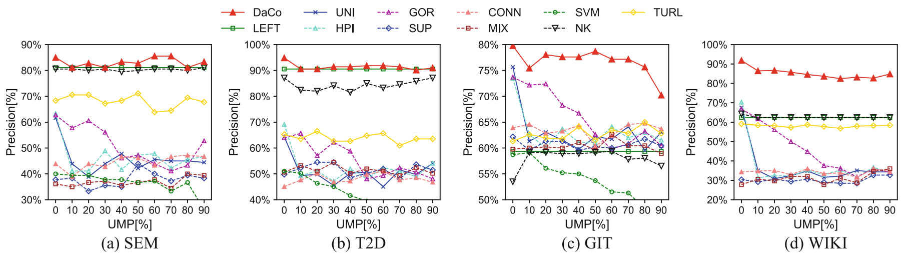


Fig. 3. Effects of unlinkable mention. Precision of four table corpus are depicted with UMP ranging from 10% to 90%.

attributed to its ability to extract valid dependencies using rough matching, even in the presence of a scarcity of linkable mentions. Additionally, our results highlight that DaCo has a distinct advantage in processing real-life tables with incomplete data.

5.4 Effects of Row Sampling (RQ2-2)

In order to further investigate the effectiveness of DaCo, we conduct experiments within the range of sampled row numbers ($m_s = [2, 20]$) in each outer iteration. For comparison, baselines sample m_s rows in this experiment. As shown in Fig. 4, we compare the results with DaCo^{-}, which represents our model without termination check module, *i.e.,*running once in outer iteration.

Based on the results shown in Fig. 4, DaCo outperforms other methods in most cases, with its precision gradually increasing and leveling off as the number of sampled rows increases. Even only with two rows, DaCo reports precision about 75%-90%. On GIT and WIKI, DaCo outperforms other methods on any of m_s with an improvement of at least 3% and 18% respectively. This further demonstrates that DaCo can achieve good results even with a small number of sampled rows on tables with many unlinkable mentions, making it suitable for real-life tables with various size.

Comparing DaCo with DaCo^{-}, DaCo achieves higher precision, especially with a lower number of sampled rows, *e.g.,*it brings an improvement on precision to DaCo^{-} about 10%, 12%, 11% and 8% on different table corpus. This is because a smaller

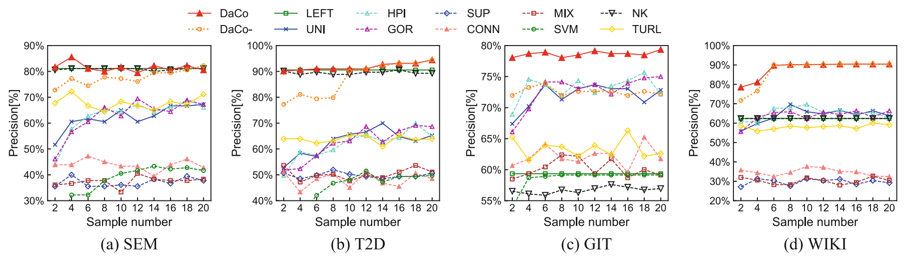


Fig. 4. Sample number effect on core column discovery. Sample number m_s is ranged from 2 to 20. The whole table will be input when it has less number of rows than m_s.

number of sampled rows is more likely to result in biased samples. Termination check can prevent this. Therefore, the existence of the termination check module improves the precision of results, particularly when working with small tables.

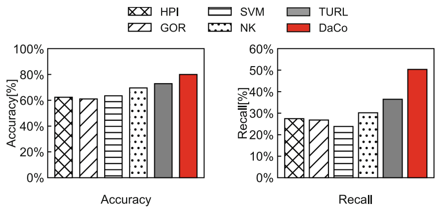


Fig. 5. Results on MULTICC

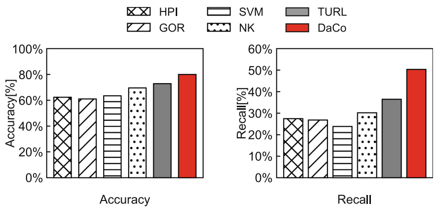


Fig. 6. Effects of varying q and η

5.5 Effects of Multiple Core Columns (RQ2-3)

In order to evaluate the performance of DaCo in identifying multiple core column from tables, we conducted experiments on MULTICC. We extend the DB and ML approaches to support multi core columns discovery for comparison of accuracy and recall, as shown in Fig. 5. DB is designed to implement the identification of unique column combinations, which inherently allows for the discovery of multiple core columns. ML involves classifying each column, and in cases where multiple columns in a table are classified as core columns, it provides results for multiple core column discovery.

The evaluation results show that DaCo outperforms state-of-the-art methods in identifying tables with multiple core columns, achieving an accuracy improvement of 8% and a recall improvement of 13%. Specifically, DaCo achieves an accuracy of 80% and a recall of 50% for multiple core column discovery, which indicates its effectiveness in discovering multiple core columns even without metadata and complete data.

5.6 Effects of Varying Parameter Settings (RQ3)

We investigated the impact of parameter settings, specifically the values of q in $\text{dep}_\tau(\cdot,\cdot)$ and η in the inner iteration. Figure 6 reveals an initial increase, followed by fluctuation within a certain range, and eventually a decrease in precision on both of q and η. We found that the highest precision for core column discovery in both datasets was achieved when q=1.00 and η=0.01.

Table 4. Effects on table understanding tasks' precision.

	Method	Column Type Prediction						Entity Linking						Relation Extraction		
		T2K			Hybrid I			T2K			Hybrid I			T2K		
		SEM	T2D	WIKI	SEM	T2D	WIKI	SEM	T2D	WIKI	SEM	T2D	WIKI	SEM	T2D	WIKI
Heuristic	LEFT	.320	.220	.620	.381	.354	.380	.438	.312	.217	.582	.100	.178	.326	.096	.153
	UNI	.330	.142	.553	.381	.241	.391	.334	.178	.234	.410	.073	.206	.278	.104	.170
	SUP	.300	.152	.448	.356	.288	.201	.330	.183	.165	.218	.047	.070	.120	.054	.082
	CONN	.305	.114	.306	.332	.202	.191	.324	.123	.131	.231	.053	.076	.166	.079	.079
	MIX	.282	.156	.435	.349	.302	.198	.319	.188	.175	.192	.048	.067	.104	.073	.075
DB	HPI	.339	.188	.553	.380	.259	.423	.334	.226	.224	.385	.075	.207	.260	.131	.190
	GOR	.314	.149	.541	.385	.239	.392	.345	.169	.219	.413	.077	.199	.255	.118	.181
DaCo		**.795**	**.646**	**.630**	**.795**	**.646**	**.630**	**.457**	**.347**	**.290**	**.590**	**.114**	**.287**	**.733**	**.533**	**.442**

5.7 Performance on Table Understanding Tasks (RQ4)

We present various experiments to demonstrate the effectiveness of DaCo in three primary tasks in table understanding [53]: Column Type Prediction (CTP), Entity Linking (EL), and Relation Extraction (RE). We integrate DaCo with two core column-based table understanding methods, T2K [38] and Hybrid I [13], and evaluate the results. We do not compare Hybrid I's RE task as it does not include a RE implementation. Both methods are core column based table understanding approaches. The CTP and RE tasks are subsequently completed based on these linked entities or entity candidate sets. Table 4 summarizes the precision of the two table understanding models using core column discovery.

The table shows that DaCo significantly outperforms state-of-the-art baselines in CTP, achieving precision values of 60%-80% compared to the precision values of T2K and Hybrid I. This is because DaCo returns a more accurate or fine-grained column type than baselines. For example, in Fig. 1, DaCo returns "Director" while T2K returns "Person" for the 5^{th} column , where we identify "Director" as the correct result. DaCo's advantage in CTP is due to the use of dependency analysis and two-level iteration to refine the type of each column during core column discovery, which yields more precise column types. T2K and Hybrid I often predict a general type by majority voting of top-1 results. In EL, DaCo also achieves a slight improvement in precision by 0.8%, 3.5%, and 5.6% over state-of-the-art baselines when inputting the core columns identified by DaCo. For RE, DaCo again outperforms the state-of-the-art baselines, achieving precision values of 44%-73% and improving RE by 30%-50%.

6 Related Work

Table Understanding. The comprehension of table semantics for downstream applications is essential, and this task often involves referencing a knowledge graph [53]. Table understanding research can be divided into core column-based and core column-free. Core column-free techniques generate representations from supervised learning or context in table surroundings [9,35,37,42]. In contrast, core column-based methods integrate semantics more efficiently from core column entities to non-core column attributes [16,27,31,56,57]. We consider discovering core columns is a crucial task for successful table understanding. However, there is currently a lack of deep investigation into this area [6].

Core Column Discovery. Core column discovery is a crucial task in web table understanding [53] that can provide input to multiple downstream applications [18,26,27,36,51]. There are three main categories approaches for core column discovery: (1) heuristic-based. Heuristic approaches often select the leftmost column [7,52,54] or the most unique column [16,23,55]. However, these approaches only focus on a single core column and lack explanation; (2) machine learning-based. They treat core column discovery as a binary classification problem for each column [6,45], which suffer from the unavailability of training data due to the limited labeled tables and the lack of metadata; (3) databased-based. These methods aim to identify core columns based on unique column detection and functional dependency [22,29], which often demand numerous tables to explore inter-table information for reducing noise. Our approach is based on the perspective of mining inter-column dependencies semantically, which is ignored by most previous work.

Dependencies in Database. Functional dependencies are fundamental to discover keys, rules, and schema in relational data. They allow for the determination of whether one column set can uniquely determine the value of another column set [24]. In response to the growing demands of real-world datasets, extensions to functional dependencies have been proposed [15,24,43,47]. However, semantic dependencies are defined on tables using KG and discovered by relations in KG but not value uniqueness in functional dependency studies.

7 Conclusion

DaCo is a novel method for discovering core column that can handle challenges such as incomplete and inconsistent data, lack of metadata, and multiple core columns. This is achieved by defining a new semantic dependency that measures the inter-column relationship and a two-level iterative algorithm to obtain the core columns. Extensive experiments demonstrate the effectiveness of our model on various aspects and improvements in table understanding tasks.

Supplemental Material Statement: Code for our DaCo and dataset are available from GitHub at https://github.com/barrel-0314/daco.git.

Acknowledgements. We would like to thank Jiaoyan Chen for his useful comment on this paper. This work is supported by the State Grid Technology Project "research and application of key technologies for automatic graphic construction of power grid control system driven by model and data", the National Natural Science Foundation of China under the grant numbers [62061146001, 62072099, 62232004], the "Zhishan" Scholars Programs of Southeast University, and the Fundamental Research Funds for the Central Universities.

References

1. T2d gold standard for matching web tables to dbpedia (2015). http://webdatacommons.org/webtables/goldstandard.html
2. Gittables benchmark-column type detection (2021). https://zenodo.org/record/5706316#.YxAVU9NBw2x
3. Semtab 2021: Semantic web challenge on tabular data to knowledge graph matching (2021), http://www.cs.ox.ac.uk/isg/challenges/sem-tab/2021/
4. Bhagavatula, C.S., Noraset, T., Downey, D.: TabEL: entity linking in web tables. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9366, pp. 425–441. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25007-6_25
5. Birnick, J., Blasius, T., Friedrich, T., Naumann, F., Papenbrock, T., Schirneck, M.: Hitting set enumeration with partial information for unique column combination discovery. In: Proceedings of the VLDB Endowment, vol. 13, pp. 2070–2083 (2020)
6. Bornemann, L., Bleifuß, T., Kalashnikov, D.V., Naumann, F., Srivastava, D.: Natural key discovery in wikipedia tables. In: Proceedings of The Web Conference 2020, pp. 2789–2795 (2020)
7. Cafarella, M.J., Halevy, A., Wang, D.: WebTables: exploring the power of tables on the web. In: Proceedings of the VLDB Endowment, pp. 538–549 (2008)
8. Cafarella, M.J., Halevy, A., Wang, D., Wu, E., Zhang, Y.: Uncovering the relational web. In: Proceedings of the 11th International Workshop on Web and Databases (2008)
9. Chen, J., Jiménez-Ruiz, E., Horrocks, I., Sutton, C.: ColNet: embedding the semantics of web tables for column type prediction. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 29–36 (2019)
10. Chen, Z., Trabelsi, M., Heflin, J., Xu, Y., Davison, B.D.: Table search using a deep contextualized language model. In: Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval. pp. 589–598 (2020)
11. Chirigati, F., Liu, J., Korn, F., Wu, Y., Yu, C., Zhang, H.: Knowledge exploration using tables on the web. In: Proceedings of the VLDB Endowment, vol. 10, pp. 193–204 (2016)
12. Deng, X., Sun, H., Lees, A., Wu, Y., Yu, C.: TURL: table understanding through representation learning. In: Proceedings of the 2022 ACM SIGMOD International Conference on Management of Data, vol. 14, pp. 33–40 (2022)
13. Efthymiou, V., Hassanzadeh, O., Rodriguez-Muro, M., Christophides, V.: Matching web tables with knowledge base entities: from entity lookups to entity embeddings. In: Proceedings of the International Semantic Web Conference, pp. 260–277 (2017)
14. Ermilov, I., Ngomo, A.-C.N.: TAIPAN: automatic property mapping for tabular data. In: Blomqvist, E., Ciancarini, P., Poggi, F., Vitali, F. (eds.) EKAW 2016. LNCS (LNAI), vol. 10024, pp. 163–179. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49004-5_11
15. Fan, W., Wu, Y., Xu, J.: Functional dependencies for graphs. In: Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data, pp. 1843–1857 (2016)
16. Gentile, A.L., Ristoski, P., Eckel, S., Ritze, D., Paulheim, H.: Entity matching on web tables: a table embeddings approach for blocking. In: Proceedings of the 20th International Conference on Extending Database Technology, pp. 510–513 (2017)
17. Harmouch, H., Papenbrock, T., Naumann, F.: Relational header discovery using similarity search in a table corpus. In: 2021 IEEE 37th International Conference on Data Engineering, pp. 444–455. IEEE (2021)
18. Ho, V.T., Pal, K., Razniewski, S., Berberich, K., Weikum, G.: Extracting contextualized quantity facts from web tables. In: Proceedings of the Web Conference 2021, pp. 4033–4042 (2021)

19. Ibrahim, Y., Riedewald, M., Weikum, G., Zeinalipour-Yazti, D.: Bridging quantities in tables and text. In: Proceedings of IEEE 35th International Conference on Data Engineering, pp. 1010–1021 (2019)
20. Khatiwada, A., et al.: Santos: relationship-based semantic table union search. CoRR abs/2209.13589 (2022)
21. Korini1, K., Peeters, R., Bizer, C.: SOTAB: the WDC schema.org table annotation benchmark. In: Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching co-located with the 21st International Semantic Web Conference, vol. 3320, pp. 14–19 (2022)
22. Kruit, B., Boncz, P., Urbani, J.: Extracting N-ary facts from wikipedia table clusters. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management, pp. 655–664 (2020)
23. Kruit, B., Boncz, P., Urbani, J.: TAKCO: a platform for extracting novel facts from tables. In: Companion Proceedings of the Web Conference, pp. 705–707 (2021)
24. Kruse, S., Naumann, F.: Efficient discovery of approximate dependencies. In: Proceedings of the VLDB Endowment, vol. 11, pp. 759–772 (2018)
25. Lehmann, J., et al.: Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web **6**(2), 167–195 (2014)
26. Lehmberg, O., Bizer, C.: Web table column categorisation and profiling. In: Proceedings of the 19th International Workshop on Web and Databases, pp. 1–7 (2016)
27. Lehmberg, O., Bizer, C.: Stitching web tables for improving matching quality. In: Proceedings of the VLDB Endowment, vol. 10, pp. 1502–1513 (2017)
28. Lehmberg, O., Bizer, C.: Profiling the semantics of N-ary web table data. In: Proceedings of the International Workshop on Semantic Big Data, vol. 5, pp. 1–6 (2019)
29. Lehmberg, O., Bizer, C.: Synthesizing N-ary relations from web tables. In: Proceedings of the 9th International Conference on Web Intelligence, Mining and Semantics, vol. 17, pp. 1–12 (2019)
30. Li, Z.: Cauchy convergence topologies on the space of continuous functions. Topol. Appl. **161**, 321–329 (2014)
31. Luzuriaga, J., Munoz, E., Rosales-Mendez, H., Hogan, A.: Merging web tables for relation extraction with knowledge graphs. IEEE Trans. Knowl. Data Eng. **35**(2), 1803–1816 (2023)
32. Marzocchi, M., Cremaschi, M., Pozzi, R., Avogadro, R., Palmonari, M.: MammoTab: a giant and comprehensive dataset for semantic table interpretation. In: Proceedings of the Semantic Web Challenge on Tabular Data to Knowledge Graph Matching co-located with the 21st International Semantic Web Conference, vol. 3320, pp. 28–33 (2022)
33. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. The MIT Press (2018)
34. Nargesian, F., Zhu, E., Pu, K.Q., Miller, R.J.: Table union search on open data. In: Proceedings of the VLDB Endowment, vol. 11, pp. 813–825 (2018)
35. Neumaier, S., Umbrich, J., Parreira, J.X., Polleres, A.: Multi-level semantic labelling of numerical values. In: Groth, P., et al. (eds.) Proceedings of the 15th International Semantic Web Conference, pp. 428–445 (2016)
36. Nguyen, P., Kertkeidkachorn, N., Ichise, R., Takeda, H.: TabEAno: table to knowledge graph entity annotation. CoRR abs/2010.01829 (2020)
37. Pham, M., Alse, S., Knoblock, C.A., Szekely, P.: Semantic labeling: a domain-independent approach. In: Groth, P., et al., (eds.) Proceedings of the 15th International Semantic Web Conference, pp. 446–462 (2016)
38. Ritze, D., Lehmberg, O., Bizer, C.: Matching html tables to DBpedia. In: Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, pp. 1–6 (2015)
39. Shyu, S.j., Yin, P., Lin, B.M.T.: An ant colony optimization algorithm for the minimum weight vertex cover problem. Ann. Oper. Res. **131**, 283–304 (2004)

40. Sismanis, Y., Brown, P., Haas, P.J., Reinwald, B.: GORDIAN: efficient and scalable discovery of composite keys. In: Proceedings of the VLDB Endowment, pp. 691–702 (2006)
41. Sun, H., Ma, H., Yih, W.t., Yan, X.: Table cell search for question answering. In: Proceedings of the 25th International Conference on World Wide Web, pp. 771–782 (2016)
42. Takeoka, K., Oyamada, M., Nakadai, S., Okadome, T.: Meimei: an efficient probabilistic approach for semantically annotating tables. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 281–288 (2019)
43. Tan, Z., Ran, A., Ma, S., Qin, S.: Fast incremental discovery of pointwise order dependencies. In: Proceedings of the VLDB Endwment, vol. 13, pp. 1669–1681 (2020)
44. Trabelsi, M., Chen, Z., Zhang, S., Davison, B.D., Heflin, J.: StruBERT: structure-aware BERT for table search and matching. In: Proceedings of the Web Conference 2022, pp. 442–451 (2021)
45. Venetis, P., et al.: Recovering semantics of tables on the web. In: Proceedings of the VLDB Endowment, vol. 4, pp. 528–538 (2011)
46. Wang, N., Ren, X.: Identifying multiple entity columns in web tables. Int. J. Softw. Eng. Knowl. Eng. **28**(3), 287–309 (2018)
47. Wei, Z., Hartmann, S., Link, S.: Discovery algorithms for embedded functional dependencies. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp. 833–843 (2020)
48. Yin, P., Neubig, G., Yih, W.T., Riedel, S.: TaBERT: pretraining for joint understanding of textual and tabular data. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, pp. 8413–8426 (2020)
49. Zhang, M., Chakrabarti, K.: InfoGather+ semantic matching and annotation of numeric and time-varying attributes in web tables. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 145–156 (2013)
50. Zhang, S., Balog, K.: Ad hoc table retrieval using semantic similarity. In: Proceedings of the World Wide Web Conference, pp. 1553–1562 (2018)
51. Zhang, S., Balog, K.: On-the-fly table generation. In: Proceedings of the 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pp. 595–604 (2018)
52. Zhang, S., Balog, K.: Auto-completion for data cells in relational tables. In: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, pp. 761–770 (2019)
53. Zhang, S., Balog, K.: Web table extraction, retrieval, and augmentation: a survey. ACM Trans. Intell. Syst. Technol. **11**, 13:1-13:35 (2020)
54. Zhang, S., Meij, E., Balog, K., Rernanda, R.: Novel entity discovery from web tables. In: Proceedings of International World Wide Web Conference, pp. 1298–1308 (2020)
55. Zhang, X., Chen, Y., Chen, J., Du, X., Zou, L.: Mapping entity-attribute web tables to web-scale knowledge bases. In: Meng, W., Feng, L., Bressan, S., Winiwarter, W., Song, W. (eds.) DASFAA 2013. LNCS, vol. 7826, pp. 108–122. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37450-0_8
56. Zhang, Z.: Towards efficient and effective semantic table interpretation. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 487–502. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_31
57. Zhang, Z.: Effective and efficient semantic table interpretation using TableMiner+. Semantic Web **8**(6), 921–957 (2017)
58. Zhu, G., Iglesias, C.A.: Computing semantic similarity of concepts in knowledge graphs. IEEE Trans. Knowl. Data Eng. **29**(1), 72–89 (2017)