A Hybrid Machine-Crowdsourcing System for Matching Web Tables

Ju Fan†, Meiyu Lu†, Beng Chin Ooi†, Wang-Chiew Tan§, Meihui Zhang†

† *National University of Singapore*
{fanj, lumeiyu, ooibc, mhzhang}@comp.nus.edu.sg

§ *University of California*
tan@cs.ucsc.edu

***Abstract*—The Web is teeming with rich structured information in the form of HTML tables, which provides us with the opportunity to build a knowledge repository by integrating these tables. An essential problem of web data integration is to discover *semantic correspondences* between web table columns, and schema matching is a popular means to determine the semantic correspondences. However, conventional schema matching techniques are not always effective for web table matching due to the *incompleteness* in web tables.**

In this paper, we propose a two-pronged approach for web table matching that effectively addresses the above difficulties. First, we propose a concept-based approach that maps each column of a web table to the best concept, in a well-developed knowledge base, that represents it. This approach overcomes the problem that sometimes values of two web table columns may be disjoint, even though the columns are related, due to incompleteness in the column values. Second, we develop a hybrid machine-crowdsourcing framework that leverages human intelligence to discern the concepts for "difficult" columns. Our overall framework assigns the most "beneficial" column-to-concept matching tasks to the crowd under a given budget and utilizes the crowdsourcing result to help our algorithm infer the best matches for the rest of the columns. We validate the effectiveness of our framework through an extensive experimental study over two real-world web table data sets. The results show that our two-pronged approach outperforms existing schema matching techniques at only a low cost for crowdsourcing.

I. Introduction

The Web contains a vast amount of structured and unstructured information, such as HTML tables and text documents. Structured information, in the form of web tables, can be extracted from the Web to improve search results and to enable knowledge discovery. Indeed, the topic of exploring and discovering information in web tables has attracted much interest in research community in recent years [1], [2], [3], [4], [5], [6], [7].

Very often, information from different web tables need to be consolidated together to build comprehensive knowledge about various entities or concepts. An essential step towards consolidating or integrating knowledge from different web tables is to discover the *semantic correspondences* between the columns of different web tables. The problem of discovering the semantic correspondences between two tables is known as *schema matching*, which is a topic that has been extensively studied in the past decade or so (e.g., see surveys [8], [9]). Even though numerous solutions have been proposed in the past for solving the schema matching problem, web tables are inherently *incomplete*, making existing schema matching solutions inadequate for matching columns of web tables.

Table T_1: *Top Rated Movies*

Title	Directed By	Language
Les Misérables	T. Hooper	EN
Life of PI	A. Lee	EN
Inception	C. Nolan	EN

Table T_2: *Highest Weekend Box Office*

Movie	Director	Written By
Up	P. Docter	P. Docter
Avatar	J. Cameron	J. Cameron
The Pianist	R. Polanski	W. Szpilman

Table T_3: *Top Rated Storybooks*

Title	Written By	Language
Les Misérables	V. Hugo	French
Life of PI	Y. Martel	English
Harry Potter	J. K. Rowling	English

Table T_4: *Best Sellers*

Book	Author
Harry Potter	J. K. Rowling
Twilight	S. Meyer
Lincoln	D. H. Donald

Fig. 1. Web table examples: table T_1 and T_2 are about movies, while T_3 and T_4 are about books.

The first type of incompleteness in web tables arises from the fact that web tables typically contain only a limited amount of information, since a web table is usually extracted from a single web page. Hence, given two web tables, even if two columns from these web tables model the same real-world concept (i.e., there is a semantic correspondence between them), it can happen quite often that they may contain only a few values in common or they may be completely disjoint. For example, consider the web tables of school names extracted from individual web pages of U.S. school districts. These web tables will typically contain no more than 20 schools in any school district and they are unlikely to share values (i.e., names of schools). A significant number of instance-based conventional schema matching techniques (e.g., see surveys [8], [9]) rely primarily on the similarity between the values of two columns to determine whether a semantic correspondence exists between the two columns. Thus, these conventional techniques may conclude that there is no semantic correspondence between the web tables mentioned above.

The second type of incompleteness in web tables arises from the fact that information about the schema of a web table may not always be available. Traditional schema matching techniques over relational databases rely heavily on the availability of metadata, such as column names and information on data types (again, see surveys [8], [9]). However, column names (i.e., table headers) in web tables are often undefined (i.e., marked up with `<th>` tag) or missing. Furthermore, data types are not explicitly specified in web tables. Even if the header names are available, they can sometimes be meaningless or unreliable [6]. As such, the traditional schema matching approaches, which depend on schema-level information to find the matches, do not tend to work well over web tables.

To address the limitations of conventional schema matching techniques for matching columns of web tables, we propose a concept-based approach that exploits well-developed knowledge bases with fairly wide coverage and high accuracy, such as FREEBASE [10], to facilitate the matching process over web tables. The FREEBASE knowledge base contains about 23 million instances over 6,852 different concepts, such as people, location, film, etc. These concepts are rather comprehensive and cover a significant amount of real-world instances in various domains. One fundamental idea in our approach is to first map values of each web table column to one or more concepts in FREEBASE by searching the values over an instance-to-concept index that is built over FREEBASE. Columns that are mapped to the same concept are then matched with each other (i.e., there is a semantic correspondence between them). Obviously, our technique does not rely on the availability of metadata, such as column names, (but we do exploit them if such information is available) since we use only the values of columns to determine the best concepts representing the values. Furthermore, by first mapping values of different web table columns to concepts and then determining whether or not a pair of columns is semantically related based on whether they match the same concept, we can now detect semantic correspondences even between columns that may not share any values in common. For example, with this approach, two columns listing the names of schools from two web tables of two different school districts will be matched to the same concept "*Education/US_Schools*", even though the column values are likely to be disjoint, since the instances of this concept overlap with the values of each of the columns.

It should be mentioned that prior work, such as [3], [6], use similar ideas: labels are annotated on columns of tables with missing headers, and binary relationships are annotated on pairs of columns, based on information that is extracted from the Web or ontologies such as YAGO [11]. However, these are all pure *machine-based* approaches and they do not always work well on some inherently difficult matching issues. For example, in Figure 1, the values of T_1.`Title` and, respectively, T_3.`Title`, can refer to both movie titles and book titles. However, in the context of the values in the other columns in T_1 and T_3, it is clear that T_1.`Title` refers to movie titles while T_3.`Title` refers to book titles. Even though it is possible for prior work [3] to also take into account values of other columns to collectively decide an appropriate concept for a set of values, we observe that such classification tasks are actually quite effortless for human who can produce more reliable classification results. With this observation in mind, we investigate a hybrid machine-crowdsourcing based approach where, intuitively, the machine will do most of the "easy" work on matching up pairs of columns through concepts and defer to the crowd to discern the concepts only for the columns that machines consider "difficult" (but is still quite an effortless task for the crowd).

Contributions. To the best of our knowledge, we are the first to exploit crowdsourcing for web table schema matching and also the first to adopt a hybrid machine-crowdsourcing based approach for a general solution to the matching problem. In this paper, we describe our design of the hybrid machine-crowdsourcing based web table matching framework. Our framework automatically assigns the most "beneficial" column-to-concept matching tasks to the crowd under a given budget k (i.e., the number of microtasks) for crowdsourcing and utilizes the crowdsourcing result to help algorithms infer the matches of the rest of tasks.

One of the fundamental challenges in the design of our framework is to determine what constitutes a "beneficial" column and should therefore be crowdsourced to determine the right concept for that column. To this end, we propose a utility function that takes both matching difficulty and influence of a column into consideration. Intuitively, the *matching difficulty of a column* refers to the degree of ambiguity of the set of column values. The *influence of a column* refers to the extent to which the knowledge of the right concept of that column will help infer the concepts of other columns. Naturally, we prefer crowdsourcing columns that are more difficult and have greater influence on other columns. We prove that the problem of selecting the best k columns to maximize the expected utility is NP-hard in general, and subsequently, we design a greedy algorithm that derives the best k columns with an approximation ratio of $1-\frac{1}{e}$, where e is the base of the natural logarithm.

Summary. We summarize our contributions below. 1) We propose a hybrid machine-crowdsourcing framework as a general solution for the web table schema matching problem. 2) We present a concept-based schema matching approach that first maps each column to a concept in a catalog, and then matches pairs of columns based on whether they belong to the same concept. 3)We propose a utility function and effective algorithms to select the best k columns for crowdsourcing. 4) We conduct extensive experiments and show that, with a fairly small amount of crowdsourcing cost, our hybrid framework outperforms existing web table annotation approaches [3], [6] and conventional schema matching techniques.

II. HYBRID MACHINE-CROWDSOURCING FRAMEWORK

In this section, we first formally define the fundamental problems of our work, and then introduce the architecture of our machine-crowdsourcing hybrid framework.

A. Definitions

A *web table corpus* is a set of *web tables*, denoted by $\mathcal{T} = \{T_1, T_2, \ldots, T_{|\mathcal{T}|}\}$. We will often write T to denote the name of the web table and the relation that interprets it. A web table T consists of a set of columns, denoted as $T.\mathcal{A} = \{A_1, A_2, \ldots, A_{|T.\mathcal{A}|}\}$. For web tables, not all of the column names are present in general but we abuse the notation A_i to denote the name of the column (if available) or to symbolically refer to that column. Naturally, in any web table column $T.A_i$, there is a set of values associated with $T.A_i$, which is given by the projection of T on attribute A_i. Figure 1 shows four examples of web tables, where each cell with a gray background contains a column name, and each cell with a white background contains a value.

Table Match. Like prior work on schema matching [8], [9], we say that there is a *semantic correspondence* (or simply, correspondence) between two columns A and A' across two tables T and, respectively, T', if $T.A$ and $T'.A'$ are *semantically related*. We call the set of such column correspondences for web tables a *table match*. In other words, a *table match* between two web tables T_1 and T_2, denoted as M_{T_1,T_2}, is a set of all pairs of columns between T_1 and

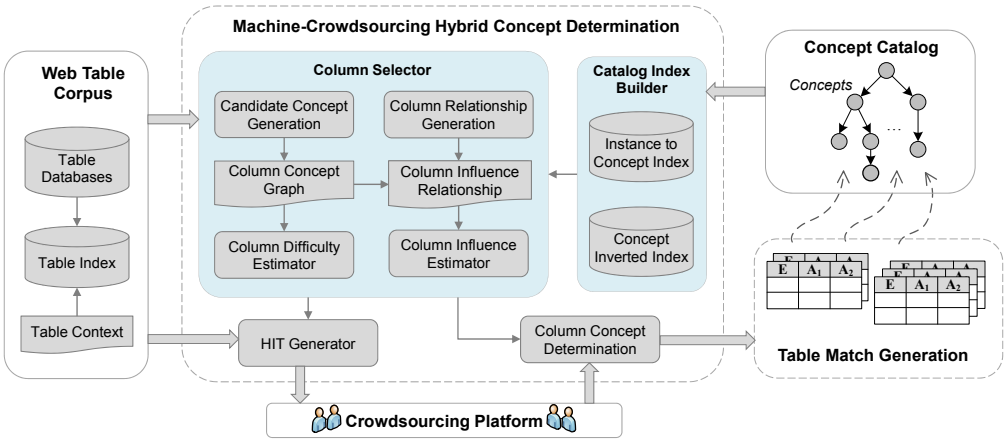


Fig. 2. Hybrid machine-crowdsourcing system architecture.

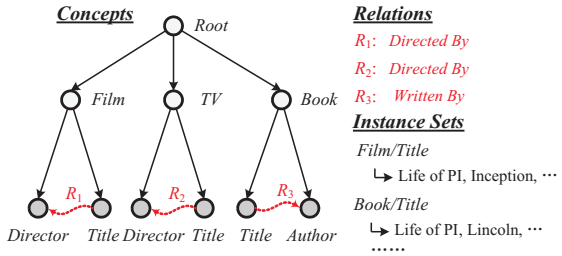


Fig. 3. An example of concept catalog.

T_2 such that for every pair $(A_i, A_j) \in M_{T_1,T_2}$ ($A_i \in T_1$ and $A_j \in T_2$), there is a correspondence between A_i and A_j. In Figure 1, there are two correspondences in M_{T_1,T_2}: (T_1.`Title`, T_2.`Movie`) and (T_1.`DirectedBy`, T_2.`Director`). The other table match M_{T_3,T_4} consists of the correspondences { (T_3.`Title`, T_4.`Book`), (T_3.`WrittenBy`, T_4.`Author`)}.

Concept Catalog. As mentioned in the Introduction, we exploit the availability of a *concept catalog* which is a triple $\langle \mathcal{C}, \mathcal{I}, \mathcal{R} \rangle$. The set $\mathcal{C}$ is a set of concepts, $\mathcal{I}$ denotes a set of instances, where each instance may belong to one or more concepts, and $\mathcal{R}$ is a set of binary relations that captures the set of relationships between concepts. We use the notation $\mathcal{I}(C)$ to denote the set of all instances from $\mathcal{I}$ that belong to the concept C. The set of concepts form a directed acyclic graph, where concepts are the nodes of the graph, and there exists an edge from a concept C_i to C_j if $\mathcal{I}(C_j) \subseteq \mathcal{I}(C_i)$. Pairs of instances of concepts can also be related through relationships that are captured by the relations in $\mathcal{R}$.

In this paper, even though we use FREEBASE as the concept catalog, other catalogs could also be used in place of FREEBASE. Figure 3 shows an example of a concept catalog where the set $\mathcal{C}$ is the nodes in the graph, and each node has an associated set of instances. Note that a solid arrow connecting two concepts in FREEBASE represents the concepts have a "whole-part" relationship. For example, the arrows from concept `Film` to its child concepts represent that `Film` consists of several other concepts, e.g., `Film/Title` and `Film/Director`. In the figure, we only attach the instances to leaf concept nodes (on the right). The instances of an intermediate concept node are obtained by including all the instances of its leaf concepts. The red dotted lines indicate the relations in $\mathcal{R}$. For example, R_1 is the "Directed By" relation that contains pairs of instances from two concepts, `Film/Title` and `Film/Director`.

For ease of presentation, we summarize the notations (some only introduced later) we use in this paper in Table I.

TABLE I. TABLE OF NOTATIONS.

e_{ij}	a match between column A_i and concept C_j
$\mathcal{A}^q$	a column set selected for crowdsourcing
E^q	crowdsourcing result for $\mathcal{A}^q$
$D(A_i)$	the difficulty of column A_i
$\text{Inf}(A_i \mid E^q)$	the influence of E^q on column A_i
$U(\mathcal{A}^q \mid E^q)$	the utility of $\mathcal{A}^q$ given E^q

B. System Architecture

Our hybrid machine-crowdsourcing framework is illustrated in Figure 2. We develop our system within the CDAS (Crowdsourcing Data Analytics System) project, along the lines of [12], [13].

Our framework differs from traditional schema matching techniques in two key aspects: we exploit a concept catalog to determine column correspondences and we leverage the wisdom of the crowd to improve the matching quality. Our framework takes as input a web table corpus (the left side of Figure 2) and a concept catalog (the right side of Figure 2). A number of auxiliary indices are constructed over the web tables and the concept catalog to improve efficiency, and table matches are determined through the following two phases.

Phase I: Concept Determination. In this phase, we leverage both the machine and the crowd to determine the concept that best represents the values of each column of every web table. The central component in this phase is the *Column Selector* module, which selects the most beneficial columns for crowdsourcing given a budget limit (further details in Sections III and IV). For the selected columns, the *HIT Generator* will create a series of questions accordingly for the crowd. An example of the type of questions asked is shown in Figure 4, where a concept is represented as a path from the root to the concept itself in the concept graph. After this, given the answers returned by the crowd, *Column Concept Determination* component will determine the best concept for each column by aggregating information from both the machine and the crowd (Section V).

In the *Column Selector* component, the web tables are first fed into the *Candidate Concept Generation* module to generate candidate concepts for each column, and a *Column Concept Graph* is derived to maintain such associations (Section III-A). After this, a utility function, which quantifies the usefulness of

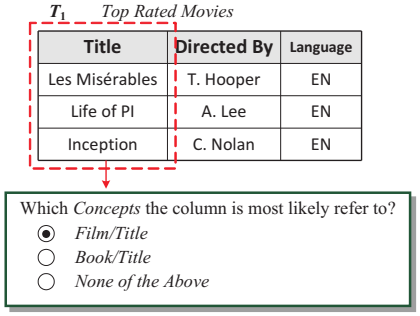


Fig. 4. A crowdsourcing microtask interface.

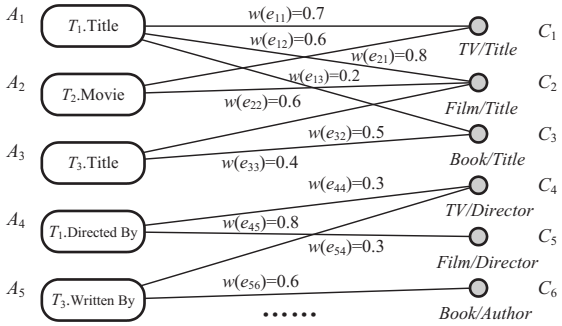


Fig. 5. An example of column concept graph.

columns, is used to determine the columns that will be most beneficial for crowdsourcing. Specifically, given a column, the utility function takes into consideration two factors. The first factor is the difficulty for machine to determine the concept for the column, which is computed by *Column Difficulty Estimator* (Section III-B). The second factor is the degree of influence of the column, if verified by the crowd, on inferring the concepts of other columns. To compute this factor, the *Column Relationship Generation* module analyzes the influence relationship between columns. After this, based on the derived relationship, the *Column Influence Estimator* estimates the extent to which the knowledge of the right concept of the column will help infer the concepts of other columns (Section III-C).

Phase II: Table Match Generation. In this phase, we consider all pairs of columns from distinct tables. For every such pair of columns that are assigned to the same concept, we create a semantic correspondence between the two columns. This phase is straightforward, and thus we shall focus on Phase I hereafter.

III. Column Utility

In our framework, a machine based similarity function is first applied to find the candidate concepts for each column. Then we would like to select the most useful columns for crowdsourcing under a given budget. On one hand, we prefer columns which are difficult for machines to determine their concepts. On the other hand, we favor the columns, if verified by the crowd, whose results would have greater influence on inferring the concepts of other columns. In this section, we will first present the machine technique to generate the candidate concepts in Section III-A, then study the column difficulty and influence in Section III-B and Section III-C respectively, and finally present a utility function which captures the usefulness of columns by considering both difficulty and influence.

A. Candidate Concept Generation

Machine-based techniques for matching web table columns to concepts in a catalog typically employ similarity functions. A similarity function takes the values of column A and a concept C as inputs, and outputs the likelihood that A and C are related. If the likelihood is positive, then we say that C is a *candidate concept* for column A.

Matching Likelihood. We employ a straightforward similarity function to measure the likelihood of matching a table column to a concept in the catalog. More specifically, let $e_{ij} = \langle A_i, C_j \rangle$ be a possible match between column A_i and concept C_j, the likelihood of this match is

$$w(e_{ij}) = \frac{|A_i.\mathcal{V} \cap \mathcal{I}(C_j)|}{|A_i.\mathcal{V}|} \quad (1)$$

where $A_i.\mathcal{V}$ denote the set of values in column A_i, and $\mathcal{I}(C_j)$ the set of instances attached to concept C_j in the catalog. Note that the above matching likelihood can also be measured by any reasonable similarity function, such as the sophisticated techniques proposed in [3], [6] and functions that support fuzzy matching [14]. The focus of this work is on effectively exploiting the power of crowdsourcing to improve the matching results, so we do not look deep into these sophisticated techniques in this paper. We demonstrate in our experiments (see Section VI-C) that, in cooperation with crowdsourcing, even this simple method can significantly outperform the sophisticated techniques [3], [6].

Column Concept Graph. We represent the matches between table columns and their candidate concepts as a bipartite graph $G(\mathcal{A} \cup \mathcal{C}, \mathcal{E})$, where $\mathcal{A}$ is the set of columns in $\mathcal{T}$, and $\mathcal{C}$ the set of concepts in the catalog. A match $e_{ij} = \langle A_i, C_j \rangle$ is represented as an edge between A_i and C_j with the matching likelihood $w(e_{ij})$ computed via Equation (1) as weight. The set of all edges of A is denoted by $\mathcal{E}(A)$. With a slight abuse of notation, we will use $e_{ij} = \langle A_i, C_j \rangle$ to refer to either a match between column A_i and concept C_j, or an edge in column concept graph, depending on the context. Figure 5 shows an example of column concept graph for the web tables in Figure 1. For simplicity, we only present five columns and six candidate concepts, where every edge is labeled with its corresponding matching likelihood. For example, column A_1 is associated with candidate concepts C_1, C_2, and respectively, C_3 represented as edges e_{11}, e_{12} and, respectively, e_{13}.

B. Modeling the Difficulty of a Column

We would like to identify table columns for which it will be difficult for machine-based methods to identify the correct concepts. Intuitively, given a column A, it is difficult for a machine to determine the best concept to represent A if the weights of edges in $\mathcal{E}(A)$ are almost identical. On the other hand, if there is a clear "winner" (i.e., an edge whose weight is clearly higher than the rest) among all the edges in $\mathcal{E}(A)$, then it is easy for an algorithm to determine which is the best concept to match A.

Based on the observation, we model *difficulty of a column* as the amount of entropy in the distribution of edge weights associated with that column. The reason is that entropy has the following properties: if the distribution is skewed (e.g., one edge has a much higher likelihood than the others), then its entropy is low; if it is close to a uniform distribution, its entropy is high. Formally, the *difficulty* of column A, denoted as $D(A)$, is:

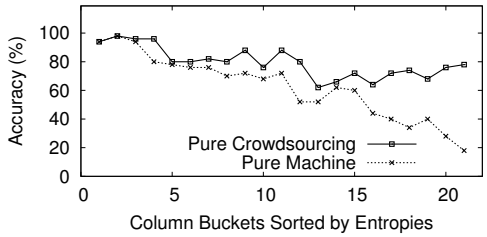


Fig. 6. The effect of entropy on performance.

$$D(A) = -\sum_{e \in \mathcal{E}(A)} \frac{w(e)}{Z} \cdot \log \frac{w(e)}{Z} \quad (2)$$

where the matching likelihood $w(e)$ is computed via Equation (1), and $Z = \sum_{e \in \mathcal{E}(A)} w(e)$ is used for normalization. Note that Equation (2) can still be used to compute difficulty when other models are applied to compute $w(e)$.

We use WWT dataset (See Section VI-A) to examine the effectiveness of our column difficulty model. We sort columns in the ascending order of their entropies, and divide them into equal-size buckets. Each bucket contains 50 columns, and so the i-th bucket consists of columns ranging from $50i$ to $50(i+1)$. We then apply machine algorithm (Equation 1) and pure crowdsouring scheme to the columns in each bucket to find the best concepts. For machine algorithm, we choose the concept with the highest matching likelihood as its answer. For crowdsourcing, we publish each matching task to 3 workers and get the answer via majority-voting. Once the answers are obtained, accuracy is computed for each bucket by a comparison with ground truth. Figure 6 shows the results, where x-axis represents the bucket number. We can see that the accuracy of pure crowdsourcing scheme remains relatively stable while the accuracy of pure machine method decreases significantly as entropy increases. This clearly validates that our proposed model can effectively capture the difficulty of a column, and crowdsourcing is much more accurate than machine on difficult columns.

Example 1: Consider column A_3 and A_4 in Figure 5 (refer to Figure 1 for table values). As the values in column A_3 can refer to both film and book (with similar matching likelihood 0.5 and 0.4), it is quite difficult for the machine to choose the correct concept ($D(A_3) = 0.99$). For column A_4, since "A. Lee" and "C. Nolan" are film directors only, it is easier for the machine to make the right decision ($D(A_4) = 0.85$).

C. Modeling the Column Influence

The knowledge of the column-to-concept associations for some columns can help us infer the concepts for the other columns. To model the column influence, we consider two types of influence relationships: *intra-table influence* and *inter-table influence*. In what follows, we discuss each influence relationship separately. For ease of presentation, we shall first describe the influence model for one single column, and then introduce the aggregated influence for a column set.

Intra-Table Influence. Intuitively, the columns in a table are usually coherent in semantics and tend to be highly correlated. For instance, if we know that the column T_1.`Title` in Figure 7(a) matches with concept `Film/Title`, we can conjecture that the likelihood that the intra-table column T_1.`Directed By` matches with `Film/Director` is higher than that with other candidate concepts, such as `TV/Director`.

We formalize our intuition as follows. Let $e^q = \langle A^q, C^q \rangle$ denote the crowdsourcing result where column A^q is known to match with concept C^q. Consider another edge $e_{ij} = \langle A_i, C_j \rangle$. If A_i and A^q are in the same table, we estimate the influence of e^q on e_{ij}, denoted by $P(e_{ij} \mid e^q)$, as the *relatedness* between concepts C^q and C_j. Consider the example in Figure 7(a). We can estimate the influence of crowdsourcing result e_{12} on e_{45}, $P(e_{45} \mid e_{12})$ as the relatedness between C_2 and C_5.

A straightforward method to compute concept relatedness is to set $P(e_{ij} \mid e^q) = 1$ if there is a relation from C^q to C_j in the catalog; otherwise $P(e_{ij} \mid e^q) = 0$. A drawback of this method is that it does not consider the relative "strength" of different relations between concepts. For example, consider concepts `Film/Director` and `Language/HumanLanguage` in the concept catalog. Suppose the former concept only has a connecting concept `Film/Title` while the latter has more relations that respectively connect to concepts `Film/Title`, `Movie/Title` and `TV/Title`. Intuitively, `Film/Director` should have larger influence on `Film/Title`, compared with `Language/HumanLanguage`. In other words, a title column is more likely to be `Film/Title` given that the same table contains `Film/Director` while having `Language/HumanLanguage` column is not sufficient to infer the domain of the title column. However, the simple method fails to differentiate `Film/Director` and `Language/HumanLanguage` in this case.

To address this problem, we further consider pairs of instances that participate in the relations in the catalog. Let $\langle C^q, C_j \rangle$ be the set of instance pairs that participate in the relation between C^q and C_j in the concept catalog, and $\langle A^q, A_i \rangle$ be the set of value pairs in column A^q and A_i. We denote all the candidate concepts of column A_i as $\mathcal{C}(A_i)$. We estimate intra-table influence $P(e_{ij} \mid e^q) = \frac{|\langle C^q, C_j \rangle \cap \langle A^q, A_i \rangle|}{\sum_{C_m \in \mathcal{C}(A_i)} |\langle C^q, C_m \rangle \cap \langle A^q, A_i \rangle|}$. Using this method, in the above example, `Film/Director` would have larger influence on `Film/Title` than `Language/HumanLanguage`.

Inter-Table Influence. The basic idea of inter-table influence is that similar columns are more likely to match to the same concept. For example, if we know T_2.`Movie` is similar to T_1.`Title`, then whenever one of them is determined to match to concept `Film/Title`, the other would have more likelihood to be inferred to match to the same concept.

Formally, consider a known match $e^q = \langle A^q, C^q \rangle$ and a candidate match e_{iq} between column A_i and the same concept C^q. We measure the influence of e^q on e_{iq}, denoted as $P(e_{iq} \mid e^q)$, by considering the *relatedness* between A_i and A^q. In Figure 7(b), given crowdsourcing result e_{12}, we can compute $P(e_{22} \mid e_{12})$ as the relatedness between column A_1 and A_2.

We adopt a concept-based method to measure the column relatedness. More specifically, for each column A_i, we generate a *concept vector*, denoted as $\overrightarrow{C}(A_i)$. Each dimension in $\overrightarrow{C}(A_i)$ represents a concept in the catalog (e.g., C_j), and the weight of this dimension is the matching likelihood computed by Equation (1) (e.g., $w(e_{ij})$). The relatedness between A_i and A^q is computed as the cosine similarity of their concept vectors, i.e., $P(e_{iq} \mid e^q) = \texttt{cosine}\big(\overrightarrow{C}(A_i), \overrightarrow{C}(A^q)\big)$.

Figure 7(b) shows an example. The concept vectors for column A_1, A_2, and A_3 are depicted in the left part of the figure. Clearly, the concept vector of A_1 is more similar to that of A_2, compared with A_3. Therefore, the influence of e_{12} on e_{22} is larger than e_{32}.

One may argue that we can also use concept-based relatedness to directly generate column correspondences. However, as shown in our technical report [15], this method produces low F-Measure. As a result, we do not treat high concept-based relatedness between two columns as "hard evidences" that the two columns are semantically correlated. Instead, we use such relatedness as an influencing factor on inferring the concept of a given column.

Aggregated Influence. We are now ready to discuss the influence of multiple crowdsourcing columns. Let $\mathcal{A}^q$ denote the set of columns that are selected for crowdsourcing, and E^q denote the matches returned by the crowd for $\mathcal{A}^q$. For each $A \in \mathcal{A}^q$, there is exactly one match $\langle A, C\rangle$ in E^q which states that column A is matched with concept C. We assume that the influences of matches in E^q on a candidate match are independent with each other. We use $P(e_{ij} \mid E^q)$ to represent the influence of E^q on e_{ij}, which can be interpreted as the probability that the candidate match e_{ij} is influenced by at least one match in E^q. We can first compute the probability of the complementary event, i.e., e_{ij} cannot be influenced by any match in E^q. Then, the influence of a column set, $P(e_{ij} \mid E^q)$ can be computed as:

$$P(e_{ij} \mid E^q) = 1 - \prod_{e^q \in E^q} \big(1 - P(e_{ij} \mid e^q)\big) \tag{3}$$

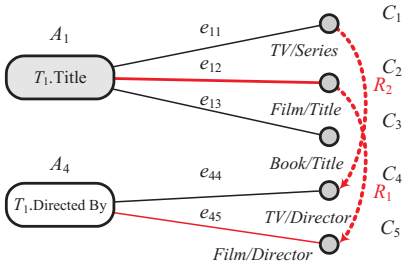
where $P(e_{ij} \mid e^q)$ is either intra- or inter-table influence. In this way, all the possible influences on a particular candidate match are aggregated.

Example 2: Consider the column concept graph in Figure 5. Suppose column A_2 and A_4 are selected for crowdsourcing and the returned result is $E^q = \{e_{22}, e_{45}\}$. We next examine the influences of e_{22} and e_{45} on a candidate match, say e_{12}. 1) As Figure 7(a) shows, there exists a relation R_1 between concept C_2 and C_5, so the intra-table influence of e_{45} on e_{12} is $P(e_{12} \mid e_{45})$. 2) As shown in Figure 7(b), the concept vectors of column A_1 and A_2 are similar. Hence, the inter-table influence of e_{22} on e_{12} is $P(e_{12} \mid e_{22})$. Finally, according to Equation (3), the aggregated influence is $P(e_{12} \mid E^q) = 1 - \big(1 - P(e_{12} \mid e_{45})\big) \cdot \big(1 - P(e_{12} \mid e_{22})\big)$.

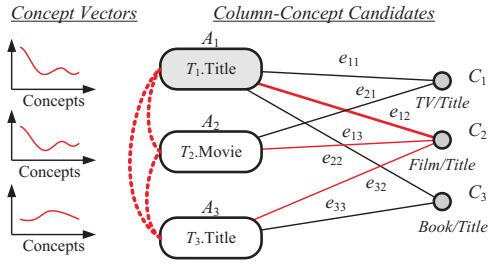
We next describe the influence of the crowdsourcing result E^q on a column $A_i \in \mathcal{A}$, denoted as $\text{Inf}(A_i \mid E^q)$. Similar with the above, we say a column is influenced by E^q if at least one of its candidate matches is influenced by E^q. With the independence assumption, $\text{Inf}(A_i \mid E^q)$ is computed as

$$\text{Inf}(A_i \mid E^q) = 1 - \prod_{e_{ij} \in \mathcal{E}(A_i)} \big(1 - P(e_{ij} \mid E^q)\big) \tag{4}$$

Utility of crowdsourcing columns $\mathcal{A}^q$ with result E^q. Next we would like to propose a utility function to measure the benefit of columns. As mentioned earlier, our utility function considers both column difficulty and the influence on other columns. We use $U(\mathcal{A}^q|E^q)$ to denote the utility of $\mathcal{A}^q$, given that the crowd returns E^q as the result. For each column $A_i \in \mathcal{A}$, we first examine the influence $\text{Inf}(A_i \mid E^q)$ of E^q on A_i.



(a) Intra-table influence.



(b) Inter-table influence.

Fig. 7. Influence of the crowdsourcing result.

Note that $\text{Inf}(A_i \mid E^q) = 1$ if $A_i \in \mathcal{A}^q$. Otherwise, $\text{Inf}(A_i \mid E^q)$ can be computed using Equation (4). Also, we consider the difficulty $D(A_i)$ of A_i. Combining the above two factors, we measure the utility $U(\mathcal{A}^q|E^q)$ as:

$$U(\mathcal{A}^q|E^q) = \sum_{A_i \in \mathcal{A}} D(A_i) \cdot \text{Inf}(A_i \mid E^q) \tag{5}$$

IV. Utility-Based Column Selection

This section presents a utility-based method for column selection. We first introduce the expectation of utility, named expected utility, for column selection in Section IV-A, and then develop an efficient algorithm for column selection given a budget limit in Section IV-B.

A. Expected Utility of Columns

The utility introduced in Section III assumes that the crowdsourcing result of the selected columns is given. However, in the phase of column selection, the result of the columns is unknown. Thus, we introduce the *expected utility* to model the expectation of utility for the selected columns when considering all the possible crowdsourcing results.

Formally, let $\mathcal{E}(\mathcal{A}^q)$ be the Cartesian product of the candidate edge sets associated with columns $\mathcal{A}^q$, i.e., $\mathcal{E}(\mathcal{A}^q) = \mathcal{E}(A_1) \times \mathcal{E}(A_2) \times \ldots \times \mathcal{E}(A_k)$ where $A_i \in \mathcal{A}^q$. An element $E^q \in \mathcal{E}(\mathcal{A}^q)$ represents a possible result of $\mathcal{A}^q$. Based on all the results in $\mathcal{E}(\mathcal{A}^q)$, we define the expected utility of $\mathcal{A}^q$ as:

$$\begin{aligned} E[U(\mathcal{A}^q)] &= \sum_{E^q \in \mathcal{E}(\mathcal{A}^q)} P(E^q) \cdot U(\mathcal{A}^q|E^q) \\ &= \sum_{A_i \in \mathcal{A}} D(A_i) \sum_{E^q \in \mathcal{E}(\mathcal{A}^q)} \text{Inf}(A_i \mid E^q) \cdot P(E^q) \end{aligned} \tag{6}$$

where $U(\mathcal{A}^q|E^q)$ is the utility in Equation (5), and $P(E^q)$ is the probability that $\mathcal{A}^q$ obtains the result E^q.

Next, we explain how to compute $P(E^q)$. Let e^q denote an edge in E^q corresponding to column A^q. By assuming the

crowdsourcing results of different columns are independent of each other, we can estimate $P(E^q) = \prod P(e^q \mid A^q)$, where $P(e^q \mid A^q)$ is the probability that the crowdsourcing result of A^q is e^q. Since the crowdsourcing result is unknown at this stage, we use either a uniform distribution or the matching likelihood $w(e^q)$ (Section III-B) to estimate it.

Example 3: Take Figure 5 as an example. Suppose there are two crowdsourcing columns, i.e., $\mathcal{A}^q = \{A_1, A_5\}$. By considering all possible crowdsourcing results, we have six possible correct edge sets, i.e., $E_1^q = \{e_{11}, e_{54}\}$, $E_2^q = \{e_{12}, e_{54}\}$, $E_3^q = \{e_{13}, e_{54}\}$, $E_4^q = \{e_{11}, e_{56}\}$, $E_5^q = \{e_{12}, e_{56}\}$ and $E_6^q = \{e_{13}, e_{56}\}$. Suppose that we use the uniform assumption, we have $P(E_i^q) = 1/6$ where $i = 1 \dots 6$. Finally, we can compute the expectation of utility based on Equation (6).

For ease of presentation, we use $\text{Inf}(A_i \mid \mathcal{A}^q)$ to denote the sum $\sum_{E^q \in \mathcal{E}(\mathcal{A}^q)} \text{Inf}(A_i \mid E^q) \cdot P(E^q)$ in Equation (6). Obviously, the intuition of $\text{Inf}(A_i \mid \mathcal{A}^q)$ is the expected influence of the crowdsourcing columns $\mathcal{A}^q$ on column A_i. Then, the expected utility can be simply represented as the weighted summation of the expected influence on columns, where the weight is column difficulty, i.e.,

$$E[U(\mathcal{A}^q)] = \sum_{A_i \in \mathcal{A}} D(A_i) \cdot \text{Inf}(A_i \mid \mathcal{A}^q) \tag{7}$$

B. Algorithm for Column Selection

Our problem of *crowdsourcing column selection*, which we define next, is defined based on the expected utility.

Definition 1 (k Column Selection Problem): Given a column concept graph G, a concept catalog $\langle \mathcal{C}, \mathcal{I}, \mathcal{R} \rangle$ and a budget k, the column selection problem finds a subset $\mathcal{A}^q \subseteq \mathcal{A}$ of columns so that $|\mathcal{A}^q| \leq k$ and the expected utility in Equation (7) is maximized.

Theorem 1: The k column selection problem is NP-hard.

Proof: (*sketch*) We prove that the k column selection problem is NP-hard by a reduction from the k *Maximum Coverage* (KMC) problem, which is known to be NP-hard.

Recall that an instance of the KMC problem $(E, \mathcal{S}, k)$ consists of a universe of items $E = \{s_1, s_2, \dots, s_n\}$, a collection of subsets of the universe E, i.e., $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ where any $S_i \in \mathcal{S}$ satisfies $S_i \subseteq E$, and a number k. The optimization objective is to select k subsets from $\mathcal{S}$, denoted by $\mathcal{S}'$, so that the number of covered items $|\bigcup_{S \in \mathcal{S}'} S|$ is maximized.

Our reduction creates the set $\{A_1, A_2, \dots, A_n\}$ of columns based on E in our column concept graph G, where each column A_i is associated to exactly two candidate concepts C_i and C_i'. Furthermore, $\mathcal{R}$ is constructed as follows: given an element s_j in E, if $s_j \in S_i$, we add two relations: one is from concept C_i to C_j, and the other is from concept C_i' to C_j'. We can show that using this construction of G and $\mathcal{R}$, there is zero inter-table influence. We can also show that the expected intra-table influence $\text{Inf}(A_j \mid A_i)$ calculates to 1 if $s_j \in S_i$ and 0 otherwise. Each set S_i in the KMC problem corresponds to the column A_i and the elements covered by S_i correspond to the set of columns A_j influenced by A_i, where $\text{Inf}(A_j \mid A_i) > 0$. We can show that selecting the k best columns $\mathcal{A}^q$ so that the expected utility is maximized is equivalent to finding the k best sets so that the number of elements covered is maximized. A complete proof can be found in our technical report [15]. ■

Despite the above result shows computing the "best" crowdsourcing columns is intractable in general, we show that the expected utility possesses two good properties, namely monotonicity and submodularity. These properties enable us to develop an algorithm which greedily determines the columns that maximize the expected utility and has an approximation ratio of $1 - 1/e$, where e is the base of the natural logarithm.

In order to prove the aforementioned two properties, we first provide a formula to compute the *delta expected influence* of a column A^q, i.e., the increase in expected influence when adding A^q to the crowdsourcing column set $\mathcal{A}^q$, denoted by $\triangle\text{Inf} = \text{Inf}(A_i \mid \mathcal{A}^q \cup \{A^q\}) - \text{Inf}(A_i \mid \mathcal{A}^q)$.

Lemma 1: The delta expected influence $\triangle\text{Inf} = \text{Inf}(A_i \mid \mathcal{A}^q \cup \{A^q\}) - \text{Inf}(A_i \mid \mathcal{A}^q)$ can be computed as

$$\triangle\text{Inf} = \text{Inf}(A_i \mid A^q) \cdot \big(1 - \text{Inf}(A_i \mid \mathcal{A}^q)\big) \tag{8}$$

Proof: Based on the definition of $\text{Inf}(A_i|\mathcal{A}^q)$ in Section IV-A, we must have $\text{Inf}(A_i \mid \mathcal{A}^q \cup \{A^q\}) = \sum_{E^q \in \mathcal{E}(A^q)} \sum_{e^q \in \mathcal{E}(A^q)} \text{Inf}(A_i \mid E^q \cup \{e^q\}) \cdot P(E^q \cup \{e^q\})$. Then, from Equation (3) and (4), we know that $\text{Inf}(A_i \mid E^q \cup \{e^q\}) = 1 - \big(1 - \text{Inf}(A_i \mid E^q)\big)\big(1 - \text{Inf}(A_i \mid e^q)\big)$. Since we also know that $P(E^q \cup \{e^q\}) = P(E^q)P(e^q)$, we can prove that $\text{Inf}(A_i|\mathcal{A}^q \cup \{A^q\}) = \text{Inf}(A_i|\mathcal{A}^q) + \text{Inf}(A_i \mid A^q) \cdot \big(1 - \text{Inf}(A_i \mid \mathcal{A}^q)\big)$. Then, we can easily obtain that $\triangle\text{Inf} = \text{Inf}(A_i|A^q) \cdot \big(1 - \text{Inf}(A_i|\mathcal{A}^q)\big)$. Thus, we prove the lemma. ■

Now, we are ready to prove the monotonicity and submodularity of the expected utility, which is shown as follows.

Lemma 2: The expected utility given by Equation (7) is monotone and submodular.

Proof: Since linear combination of monotone and submodular functions is also monotone and submodular, we only need to show that the term $D(A_i) \cdot \text{Inf}(A_i \mid \mathcal{A}^q)$ is monotone and submodular.

We first prove the monotonicity. That is, given two sets of crowdsourcing columns $\mathcal{A}_1^q \subseteq \mathcal{A}_2^q$, we must have $\text{Inf}(A_i \mid \mathcal{A}_1^q) \leq \text{Inf}(A_i \mid \mathcal{A}_2^q)$. Consider each possible crowdsourcing result $E_1^q \in \mathcal{E}(\mathcal{A}_1^q)$. We can find a collection of edge sets, each of which, E_2^q, is a superset of E_1^q, i.e., $\mathcal{E} = \{E_2^q \mid E_2^q \supseteq E_1^q, E_2^q \in \mathcal{E}(\mathcal{A}_2^q)\}$. From Equation (3) and (4), we know the influence satisfies $\text{Inf}(A_i \mid E_2^q) \geq \text{Inf}(A_i \mid E_1^q)$ given $E_2^q \supseteq E_1^q$. Since we also know $P(E_1^q) = \sum_{E_2^q \in \mathcal{E}} P(E_2^q)$, we can prove the monotonicity $\text{Inf}(A_i \mid \mathcal{A}_1^q) \leq \text{Inf}(A_i \mid \mathcal{A}_2^q)$. Hence, $D(A_i) \cdot \text{Inf}(A_i \mid \mathcal{A}^q)$ is also monotone.

Next, we prove the submodularity property. That is, given two sets of columns $\mathcal{A}_1^q \subseteq \mathcal{A}_2^q$ and a column A^q, we must have $\text{Inf}(A_i \mid \mathcal{A}_1^q \cup \{A^q\}) - \text{Inf}(A_i \mid \mathcal{A}_1^q) \geq \text{Inf}(A_i \mid \mathcal{A}_2^q \cup \{A^q\}) - \text{Inf}(A_i \mid \mathcal{A}_2^q)$. According to Lemma 1, we have the following equation $\text{Inf}(A_i \mid \mathcal{A}^q \cup \{A^q\}) - \text{Inf}(A_i \mid \mathcal{A}^q) = \text{Inf}(A_i \mid A^q) \cdot \big(1 - \text{Inf}(A_i \mid \mathcal{A}^q)\big)$. Then, due to the proved monotonicity of $\text{Inf}(A_i \mid \mathcal{A}^q)$, we have $\text{Inf}(A_i \mid \mathcal{A}_1^q) \leq \text{Inf}(A_i \mid \mathcal{A}_2^q)$. Thus, we prove the submodularity. ■

Based on Lemma 2, we develop a greedy-based approximation algorithm to find the k best crowdsourcing columns.

Algorithm 1: *SelectColumns* $(G, \langle \mathcal{C}, \mathcal{I}, \mathcal{R} \rangle, k)$

```
begin
    for each column A in 𝒜 do
        Compute difficulty D(A) using G;
        Compute influence Inf(A_j | A) for each A_j ∈ 𝒜 ;
    𝒜^q ← ∅ ;
    for i = 1 to k do
        for each column A in 𝒜 − 𝒜^q do
            △Ũ(A) ← ComputeDeltaUtility(A, 𝒜, 𝒜^q) ;
        A* = arg max_{A∈𝒜−𝒜^q} {△Ũ(A)} ;
        𝒜^q ← 𝒜^q ∪ {A*} ;
        Update utility for ∀A ∈ 𝒜 using the new 𝒜^q ;
    return 𝒜^q ;
end
```

Algorithm 2: *ComputeDeltaUtility* $(A, \mathcal{A}, \mathcal{A}^q)$

```
begin
    △Ũ(A) ← 0 ;
    for each column A' in 𝒜 − 𝒜^q do
        △Inf = Inf(A' | 𝒜^q ∪ {A}) − Inf(A' | 𝒜^q) ;
        △Ũ(A) ← △Ũ(A) + D(A') · △Inf ;
    return △Ũ(A) ;
end
```

Greedy-based Approximation Algorithm. The pseudo-code of our greedy-based algorithm is shown in Algorithm 1. For each column $A \in \mathcal{A}$, the algorithm computes difficulty $D(A)$, and its influence on each column $A_j \in \mathcal{A}$, i.e., $\text{Inf}(A_j \mid A)$.

Next, the algorithm initializes an empty set of crowdsourcing columns $\mathcal{A}^q \leftarrow \emptyset$. Then, it iteratively selects the next best column by computing the *delta expected utility*, i.e., the increase in expected utility, of each column. In each iteration, given the current $\mathcal{A}^q$, the algorithm examines every column $A \in \mathcal{A} - \mathcal{A}^q$, and computes the delta expected utility of adding A to $\mathcal{A}^q$, which is denoted by $\triangle\widetilde{U}(A)$. Then, the algorithm selects the column A^* with the maximum $\triangle\widetilde{U}(A)$ and inserts it to the set of crowdsourcing columns. Finally, the algorithm updates the expected utility based on the new $\mathcal{A}^q$, and continues to the next iteration.

The important task in Algorithm 1 is to compute the delta expected utility $\triangle\widetilde{U}(A)$ and the computation is described in Algorithm *ComputeDeltaUtility*. According to Equation (7), the function examines each column $A' \in \mathcal{A} - \mathcal{A}^q$ and computes the delta expected influence $\triangle\text{Inf} = \text{Inf}(A' \mid \mathcal{A}^q \cup \{A\}) - \text{Inf}(A' \mid \mathcal{A}^q)$. Based on Equation (8), the value $\triangle\text{Inf}$ is computed incrementally as $\triangle\text{Inf} = \text{Inf}(A' \mid A) \cdot \big(1 - \text{Inf}(A' \mid \mathcal{A}^q)\big)$. Thus, we can materialize the expected influence $\text{Inf}(A' \mid \mathcal{A}^q)$ for each column $A' \in \mathcal{A} - \mathcal{A}^q$, which is computed in last iteration. When examining column A, we only need to on-the-fly compute $\text{Inf}(A' \mid A)$, and then compute the delta expected influence incrementally.

Proposition 1: The greedy-based algorithm described in Algorithm 1 achieves an approximation ratio of $1 - 1/e$, where e is the base of the natural logarithm.

Proof: Given the monotone and submodular properties of the expectation of utility in Lemma 2, the approximation ratio of the greedy algorithm is $1 - 1/e$ as shown in [16]. ■

Algorithm 3: *DetermineConcepts* $(\mathcal{A}, E^q)$

```
begin
    for each column A in 𝒜 do
        if A ∈ 𝒜^q then Insert ⟨A, C⟩ into r* ;
        else
            for each candidate edge e ∈ ℰ(A) do
                Compute w(e) via Equation (1) ;
                Compute P(e | E^q) via Equation (3) ;
                P(e) ← α · w(e) + (1 − α) · P(e | E^q) ;
            Insert column A and the concept of the edge
            with maximum P(e) into r* ;
    return r*          /* Column-concept pairs */;
end
```

V. Concept Determination

After the crowdsourcing result is obtained, we can then proceed to determine which candidate edge is the best for each column. Formally, given the crowdsourcing result E^q of the set $\mathcal{A}^q$ of columns and a column A_i, we wish to find the edge $e_{ij} \in \mathcal{E}(A_i)$ with the maximum probability $P(e_{ij})$ as the edge for column A_i, where $P(e_{ij})$ denotes the probability that an edge $e_{ij} = \langle A_i, C_j \rangle$ is correct.

To compute the probability $P(e_{ij})$, we consider two sources of evidences. One evidence, denoted as $\theta_{\mathtt{M}}$, is based on the machine. We utilize the matching likelihood $w(e_{ij})$ in Section III-B to capture if the machine considers edge e_{ij} to be correct. The other evidence, denoted as $\theta_{\mathtt{q}}$ is based on the influence of the crowdsourcing result E^q. When considering this evidence, we employ $P(e_{ij} \mid E^q)$ in Section III-C to capture the confidence that e_{ij} is inferred to be correct given E^q. Then, we combine the two evidences as follows.

$$\begin{aligned} P(e_{ij}) &= P(e_{ij} \mid \theta_{\mathtt{M}}) \cdot P(\theta_{\mathtt{M}}) + P(e_{ij} \mid \theta_{\mathtt{q}}) \cdot P(\theta_{\mathtt{q}}) \\ &= w(e_{ij}) \cdot P(\theta_{\mathtt{M}}) + P(e_{ij} \mid E^q) \cdot P(\theta_{\mathtt{q}}) \end{aligned} \tag{9}$$

where $P(\theta_{\mathtt{M}})$ and $P(\theta_{\mathtt{q}})$ are prior probabilities representing our belief on the machine and the crowdsourcing influence respectively. Since we only have two evidences, we simply denote $P(\theta_{\mathtt{M}})$ as α and $P(\theta_{\mathtt{q}})$ as $(1 - \alpha)$. The value of α is determined by experiments.

Algorithm 3 presents the pseudo-code which determines the best concept for each column. The algorithm takes the set $\mathcal{A}$ of columns and the result E^q of the crowdsourcing columns $\mathcal{A}^q$ as input. It iteratively examines each column $A \in \mathcal{A}$ and determines the best concept of A. The algorithm considers two cases. If $A \in \mathcal{A}^q$, the algorithm outputs the concept determined by crowdsourcing. Otherwise, the algorithm computes $P(e) = \alpha w(e) + (1 - \alpha) P(e \mid E^q)$ for each candidate edge $e \in \mathcal{E}(A)$. Finally, the algorithm finds the edge with maximum $P(e)$ (ties are broken arbitrarily) and takes the corresponding concept as the matched concept of column A.

VI. Experiments

This section evaluates the performance of our approach to web table matching. First, we examine the accuracy of column-to-concept matches produced by our hybrid machine-crowdsourcing method. Then, we compare the hybrid method with existing web table annotation techniques. Finally, we evaluate our concept-based approach on web table matching, and compare it with conventional schema matching techniques.

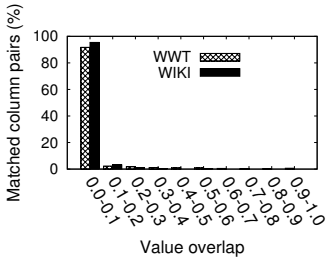


Fig. 8. Value overlap of matched column pairs.

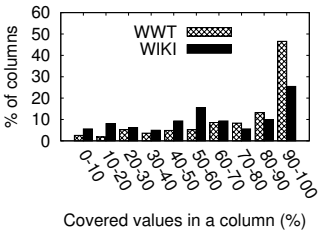


Fig. 9. The coverage of FREEBASE over column values.

A. Experimental Setup

Web Table Datasets. We conduct extensive experiments on two real-world web table datasets, WWT and WIKI, for evaluation. Table II shows the statistics of these two datasets.

TABLE II. STATISTICS OF WEB TABLE DATASETS.

Dataset	# tables	# columns	# labeled columns
WWT	607	1,166	1,166
WIKI	92,618	189,072	161

1) WWT [3]: We randomly select 607 web tables from WWT and manually label string columns with FREEBASE concepts as the ground truth. The number of the considered columns is 1,166.

2) WIKI [17]: The web table corpus extracted from Wikipedia data dump contains $92,618$ web tables with $189,072$ string columns. Due to its size, it is not feasible to manually label FREEBASE concepts for all the columns, and we therefore randomly select 161 columns and label them with concepts as ground truth.

On both datasets, we derive correct column matches based on the ground truth. Then, we measure the *value overlap* between each pair of matched columns, i.e., the Jaccard similarity between the value sets of the two columns. Figure 8 shows the result. The value overlap of more than 90% of the column pairs is smaller than 0.1 in each dataset. This result illustrates the incompleteness of web tables presented in Introduction: it is quite often that the matched columns have a few values in common or they may be completely disjoint.

Concept Catalog. We use FREEBASE [18] as our concept catalog. FREEBASE has 22,985,650 instances over 6,852 different concepts and 6,826 relations. For example, the concept "Film/Title" has 146,147 instances, including "Titanic", "Inception", etc. Next, we examine the coverage of FREEBASE over the values in a web table column. Given a column A_i and its ground-truth concept C_j in FREEBASE, we compute the percentage of A_i's values that are covered by the instance set of C_j. As shown in Figure 9, 82% (66%) of the columns on the WWT (WIKI) contain more than 50% values which are covered by the correct concepts. This illustrates that FREEBASE has a large coverage on the column values, and thus it can be employed for web table matching.

Crowdsourcing on AMT. We use Amazon Mechanical Turk (AMT) as the platform for crowdsourcing. We generate the human-intelligence tasks (HITs) and publish the HITs on AMT. We take 20 microtasks as one HIT, where each microtask contains a table column and its candidate concepts. To assist workers in understanding the column, we provide the entire web table as the context (see Figure 4 as an example). We pay \$0.2 each time a worker completes an HIT and \$0.005 to AMT for publishing each HIT. We employ the multi-assignment strategy by assigning each HIT to 3 workers and combining their answers via majority voting. In addition, we use qualification test to estimate worker's accuracy and filter out those who are not qualified to perform our HITs.

B. Hybrid Machine-Crowdsourcing Method

In this set of experiments, we examine our hybrid machine crowdsourcing method for generating column-to-concept matches. We use *accuracy* as the evaluation metric. For each column, we compare the concept produced by one method against the ground truth, and compute accuracy as the percentage of the correct column-to-concept matches with respect to all columns. Since we have ground truth for all columns on WWT, we use this dataset to compare our strategies against the alternatives in each component of our hybrid method.

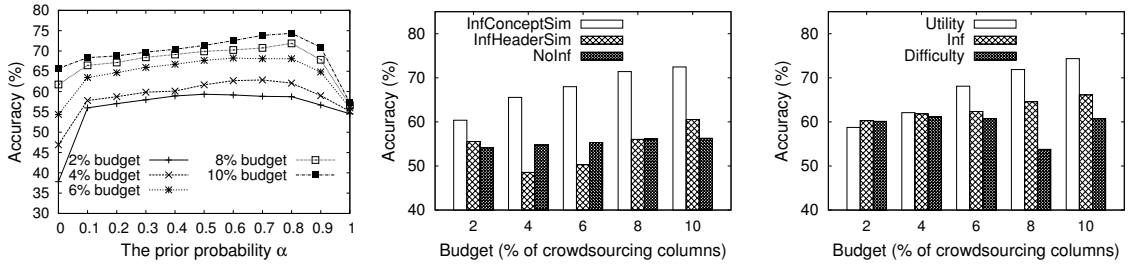
1) Machine-Based vs. Crowdsourcing-Based: We first evaluate the pure machine-based method. For each column, this method computes the matching likelihood to each concept in the catalog using Equation 1 (refer to Section III-A), and selects the concept with the maximum matching likelihood. The experimental result shows that the method achieves only 54% on accuracy. From a careful error analysis, we observe that for 31% columns their ground-truth concepts are not consistent with the concepts with the maximum matching likelihood , and thus the method fails to discover the correct concept matches for these columns. For the remaining columns, 60% of them have multiple concepts (including the ground-truth one) with the maximum matching likelihood. In this case, the machine-based method may produce false positives.

TABLE III. EVALUATING CROWDSOURCING-BASED METHOD (WWT).

Metric	# asg. = 1	# asg. = 2	# asg. = 3
Accuracy	68%	70%	77%
Cost	\$12	\$24	\$36

Next, we evaluate the pure crowdsourcing-based method by generating HITs for all the 1,166 columns. We assign the same microtask to different workers (varying the number of workers from 1 to 3) and choose the result via majority voting. The accuracy of crowdsourcing is shown in Table III. The results show that the crowdsourcing-based method significantly outperforms the machine-based methods. Even if we assign each microtask to only one worker, crowdsourcing still outperforms the machine-based method by 14%. The experimental results confirm our claim that human can produce more reliable column-to-concept matching results. However, in the crowdsourcing-based method, the higher the accuracy we want to achieve, the more money we have to pay. For instance, to improve the accuracy from 68% to 77%, we need to pay additional \$24 to publish more microtasks on AMT. Clearly, we need to combine the machine-based and crowdsourcing-based methods to produce accurate column-to-concept matches while lowering the cost.

2) Evaluation on Hybrid Machine-Crowdsourcing Concept Determination: To evaluate our concept determination method proposed in Section V, we first randomly select $x\%$ of columns for crowdsourcing and obtain the concepts produced by the crowd. Next, we utilize Algorithm 3 to determine concepts for the other columns. Finally, we compute the accuracy as



(a) Effect of α. (b) Effect of influence. (c) Effect of column selection.

Fig. 10. Evaluation of machine-crowdsourcing concept determination (WWT).

follows. We pick the columns with ground truth excluding the crowdsourcing ones, and compute the percentage of columns whose concepts are correctly determined. We use the same accuracy metric in Section VI-C. We run the experiment three times to obtain different sets of columns for crowdsourcing, and compute the average accuracy.

We first empirically learn the prior probabilities of the machine (α in Section V) and crowdsourcing influence ($1-\alpha$). The priors are used to reconcile the machine-generated $w(e)$ and the influence of the crowdsourcing result $P(e \mid E^q)$. We vary α from 0 to 1 and examine the accuracy of our algorithm with different budgets (i.e., the percentages of columns for crowdsourcing). As shown in Figure 10(a), depending only on either crowdsourcing influence ($\alpha = 0$) or the machine-generated likelihood ($\alpha = 1$) cannot achieve satisfactory accuracy. In contrast, reconciling the machine result and the crowdsourcing influence ($0 < \alpha < 1$) produces much better accuracy. We also observe that $\alpha = 0.8$ achieves the best accuracy in different budgets. Thus, we set 0.8 as the default value of α in the rest of the experiments.

Next, we study the effectiveness of the column influence model (Section III-C). Our model `InfConceptSim` not only considers intra-table influence, but also uses concept-based relatedness to compute inter-table influence. We compare it against the following two baselines. 1) `NoInf` does not consider column influence at all. Specifically, it employs the crowdsourcing result for the $x\%$ of columns, and the result produced by the pure machine-based method for the remaining columns. 2) `InfHeaderSim` also considers intra-table influence, but for inter-table influence it utilizes the Jaccard similarity on the token sets of column headers.

As illustrated in Figure 10(b), our model `InfConceptSim` outperforms the two baselines at every budget. Compared to `NoInf`, we can see that using crowdsourcing results to infer the concepts of other columns can improve the accuracy by 12% on average (across different budgets). In addition, `InfConceptSim` achieves higher accuracy than `InfHeaderSim` which depends on header similarity for modeling influence. The low accuracy of `InfHeaderSim` is attributed to the high heterogeneity in column headers: columns with similar headers can refer to different concepts (false positives); columns corresponding to the same concept may have dissimilar headers (false negatives).

We also observe that, given larger crowdsourcing budgets, our influence model can obtain higher accuracy. This meets our expectation, since the more crowdsourcing columns, the more reliable column-to-concept matches that we can use to infer other matches. In contrast, `NoInf` remains nearly unchanged when increasing the number of crowdsourcing columns, because it can not take advantage of crowdsourcing. In addition, we notice that `InfHeaderSim` does not follow a certain trend with increasing crowdsourcing budgets. This is because `InfHeaderSim` influences other columns simply based on the header similarity. However, the influences are not always reliable due to the heterogeneity in web table headers.

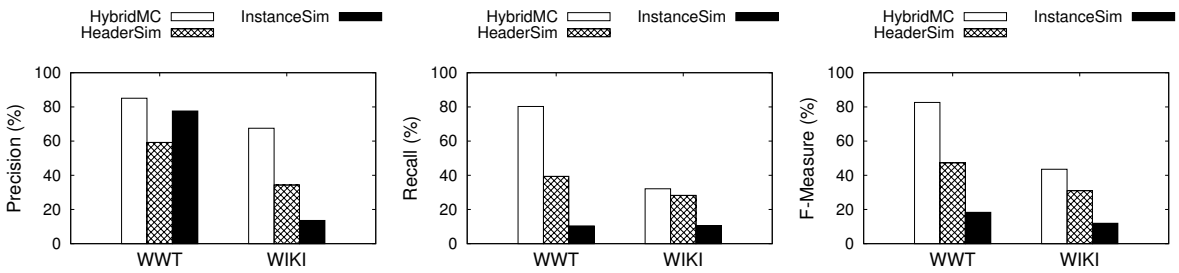
3) Evaluation on Column Selection: We evaluate our utility function on selecting crowdsourcing columns. This function, denoted as `Utility`, quantifies the usefulness of a column by considering both its difficulty and influence on other columns, and we employ Algorithm 1 to select the best columns that maximize the expected utility. We compare `Utility` against the following two baselines. 1) Method `Difficulty` considers only column difficulties. It selects the most difficult columns. 2) Method `Inf` considers only column influence. It selects the columns which have the largest influence.

Figure 10(c) summarizes the results. Our column selection approach `Utility` achieves the best accuracy in all the cases except 2% budget (23 out of 1,166 columns for crowdsourcing). When the number of crowdsourcing columns is small, their influence on inferring the concepts of other columns is limited. Therefore, all three column selection alternatives produce comparable accuracies. With increasing budget, the improvement of `Utility` over the other two alternatives becomes more notable. The strength of `Utility` is mainly attributed to the combination of both column difficulty and influence in our utility function. In contrast, `Difficulty` and `Inf` only consider either difficulty or influence. `Difficulty` selects columns which are difficult for the machine. The selected columns, however, may not be helpful on inferring other column-to-concept matches. Similarly, the columns selected by `Inf` have large influence on other columns, which may be easy for the machine to determine their concepts and hence are not ideal for crowdsourcing.

We have also roughly estimated the possible cost when applying our hybrid machine-crowdsourcing method on a large-scale web table corpus based on the scalability of the WWT dataset (see our technique report [15] for details). We estimate that, on the large WIKI dataset with 189,072 columns, to achieve accuracy of 60%, 65% and 70%, the crowdsourcing costs are about $200, $340 and $500, respectively. Such cost is affordable in real applications.

C. Comparison with Table Annotation

We compare our hybrid concept determination method `HybridMC` with two recently proposed table annotation techniques [3], [6], which also discern concepts for table columns.



(a) Precision. (b) Recall. (c) F-Measure.

Fig. 11. Evaluation of discovering column correspondences on both datasets.

1) `Annotate1` [3] devises a collective graphical model to annotate table columns with concepts in a knowledge base. It considers not only the similarities between column headers and concept names, but also the matching degree between column values and concept instances.

2) `Annotate2` [6] utilizes more sophisticated scoring schemes based on the maximum likelihood hypothesis to annotate columns with concepts.

We have implemented the above two methods and used them to annotate columns with FREEBASE concepts. In particular, on the WIKI dataset, we first run one method to determine concepts for all the 189,072 columns, and then compute accuracy based on the 161 columns with manually labeled ground-truth concepts (see Table II).

TABLE IV. COMPARISON OF CONCEPT DETERMINATION.

WWT data set		WIKI data set	
Method	**Accuracy**	**Method**	**Accuracy**
`HybridMC`	**75.0%**	`HybridMC`	**54.0%**
`Annotate1`	58.7%	`Annotate1`	35.4%
`Annotate2`	52.1%	`Annotate2`	32.3%

Table IV presents the result on the two datasets, which shows that our proposed method achieves better accuracy than the annotation techniques. For the WWT dataset, our method improves the accuracy by 16.3% and 22.9% against `Annotate1` and `Annotate2` respectively. Similarly, we achieve about 20% improvement on the WIKI dataset. This is because the machine-based annotation techniques cannot effectively produce high-quality results on some inherently difficult matching issues. Further, the improvement in accuracy could be achieved without incurring a high cost. Specifically, `HybridMC` only publishes 116 (10%) columns costing $3.69 for the WWT dataset, and 190 (0.1%) columns costing $6.15 for the WIKI dataset respectively. This confirms the feasibility of using our method in the real-world web table matching applications for improving their column-to-concept matches.

D. Evaluation on Table Matching

In this section, we evaluate the performance of our hybrid approach on discovering column correspondences. Given the ground truth, we compute the *precision* as the number of correct correspondences over the number of returned ones. We compute the *recall* as the number of correct correspondences over the number of all correspondences in the ground truth. Finally, we compute the *F-measure* as $\frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$.

Our hybrid approach `HybridMC` generates a correspondence when the two columns refer to the same concept. We shall now compare our approach with the following two conventional schema matching methods. 1) `HeaderSim` employs schema-level information by computing Jaccard similarity between column headers, and it produces a correspondence if the similarity exceeds a threshold. We tune the threshold such that `HeaderSim` achieves the best F-measure. 2) `InstanceSim` makes use of the instance-level information and it produces a correspondence if the two columns share common values.

As shown in Figure 11, `HybridMC` outperforms the conventional techniques by a wide margin on F-Measure. As analyzed in Section VI-B2, the weak performance of `HeaderSim` is due to the heterogeneity in column headers. `InstanceSim` performs poorly on recall due to the fact that the semantically related columns may quite often share very few common values or even be completely disjoint. In summary, the results confirm our claim that conventional schema matching techniques are inadequate for matching columns in web tables.

VII. RELATED WORK

There is a large body of works on processing structured data on the web, including web tables [2], [1], [3], [6], HTML lists [19], [20] and the hidden Web [21], [22]. In this paper, we pay our special attention on web tables.

Web table annotation [6], [3] is closely related to our work. Limaye et al. [3] proposed a probabilistic graphic model to collectively annotate web tables using YAGO [11]. They annotate table cells with entities, columns with types and column pairs with relations. The idea is to use joint inference to enhance the quality of all the annotations. Later Venetis et al. [6] solved the same problem using a Web-scale knowledge base, where all the class labels and relationships are extracted from the Web. Both of them focus on improving the quality of pure machine-based approach, while our work concentrates more on leveraging the power of crowdsourcing and building a hybrid machine-crowdsourcing framework. The main problem we tackle is to wisely choose the columns for crowdsourcing and effectively utilize the crowdsourcing results to achieve overall high quality. To the best of our knowledge, we are the first to exploit crowdsourcing for web table schema matching.

Some work recently has concentrated on soliciting the user feedback to overcome the limitations of machine approaches. One of the key problems is to determine in what order the candidates are presented to the users for their feedback. Jeffery, Franklin, and Halevy [23] defined a holistic utility function to rank the candidate matches. They assume a query workload is available, and their utility function is crafted over the workload. In contrast, we define the column utility based

on the column difficulty and its influence on other columns. Stonebraker et. al [24] propose a data curation system that entails machine learning approaches with human assists. They employ human domain experts with one or more areas of expertise to answer questions, e.g., verifying if two attributes are matched. In contrast, our work does not assume the crowd has specific expertise, and provides a hybrid framework to better leverage the intelligence from the non-expert crowd.

Another direction of crowdsourcing-based integration is entity resolution. In [25], the objective is to ask as few questions as possible to cover all the candidate entity matches. The work in [26] introduced a probabilistic approach for question selection in crowdsourcing entity resolution framework. The key contribution of this work is that they took transitive closure into consideration when selecting questions. Our work is to discover column correspondences and we only choose the most valuable columns for crowdsourcing under a given budget.

There also exists much work on studying the quality of crowdsourcing answers, such as predicting the minimal number of workers to satisfy a required accuracy, and choosing the correct answer in a wise manner rather than simple majority voting [12]. These techniques can be easily adopted by our framework to improve the accuracy of our approach.

VIII. Conclusion and Future Work

We have described a hybrid machine-crowdsourcing framework to effectively discover the schema matches for web tables. To the best of our knowledge, our system is the first hybrid machine-crowdsourcing system for tackling the web table matching problem. Unlike traditional schema matching techniques, the machine part of our framework leverages a concept-based approach. Due to the inherent semantic heterogeneity in web tables, pure machine algorithms cannot always work well. To this end, we harness the power of human intelligence as the crowdsourcing part of our framework to further improve the matching quality. As a first attempt towards the development of a hybrid machine-crowdsourcing system, we have proposed an effective utility function to measure the benefit of crowdsourcing columns. Our experiments on two real-world web table datasets reveal that our hybrid machine-crowdsourcing framework is promising; even with a first-cut model of the utility function, our hybrid system already provides a much higher accuracy, compared to existing methods, with a low crowdsourcing cost. In this paper, we made a simplification that the crowd was assumed to produce perfect answer, which is not always the case. As part of our future work, we plan to take the crowd accuracy into account in the model. Another direction is to look into an online scheme that crowdsources questions in multiple batches instead of crowdsourcing all questions in one single phase. The challenge is then how to optimally batch the questions and how to effectively adjust the utilities according to partial crowdsourcing results. One possible approach is to investigate active learning techniques that proactively adjust the crowdsourcing columns based on a subset of answers provided by the crowd for the utility function.

Acknowledgment

The work in this paper was in part supported by the Singapore Ministry of Education Grant No. R-252-000-454-112. Tan was partially supported by NSF grant IIS-0905276 and a Google Faculty Award. Part of this work was done while Tan was visiting the National University of Singapore.

References

[1] M. J. Cafarella, A. Y. Halevy, and N. Khoussainova, "Data integration for the relational web," *PVLDB*, vol. 2, no. 1, pp. 1090–1101, 2009.

[2] M. J. Cafarella, A. Y. Halevy, D. Z. Wang, E. Wu, and Y. Zhang, "WebTables: exploring the power of tables on the web," *PVLDB*, vol. 1, no. 1, pp. 538–549, 2008.

[3] G. Limaye, S. Sarawagi, and S. Chakrabarti, "Annotating and searching web tables using entities, types and relationships," *PVLDB*, vol. 3, no. 1, pp. 1338–1347, 2010.

[4] R. Pimplikar and S. Sarawagi, "Answering table queries on the web using column keywords," *PVLDB*, vol. 5, no. 10, pp. 908–919, 2012.

[5] A. D. Sarma, L. Fang, N. Gupta, A. Y. Halevy, H. Lee, F. Wu, R. Xin, and C. Yu, "Finding related tables," in *SIGMOD*, 2012, pp. 817–828.

[6] P. Venetis, A. Y. Halevy, J. Madhavan, M. Pasca, W. Shen, F. Wu, G. Miao, and C. Wu, "Recovering semantics of tables on the web," *PVLDB*, vol. 4, no. 9, pp. 528–538, 2011.

[7] M. Yakout, K. Ganjam, K. Chakrabarti, and S. Chaudhuri, "InfoGather: entity augmentation and attribute discovery by holistic matching with web tables," in *SIGMOD*, 2012, pp. 97–108.

[8] P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later," *PVLDB*, vol. 4, no. 11, pp. 695–701, 2011.

[9] E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *VLDB J.*, vol. 10, no. 4, pp. 334–350, 2001.

[10] K. D. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *SIGMOD*, 2008, pp. 1247–1250.

[11] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *WWW*, 2007, pp. 697–706.

[12] X. Liu, M. Lu, B. C. Ooi, Y. Shen, S. Wu, and M. Zhang, "CDAS: A crowdsourcing data analytics system," *PVLDB*, vol. 5, no. 10, pp. 1040–1051, 2012.

[13] J. Gao, X. Liu, B. C. Ooi, H. Wang, and G. Chen, "An online cost sensitive decision-making method in crowdsourcing systems," in *SIGMOD Conference*, 2013, pp. 217–228.

[14] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 1, pp. 1–16, 2007.

[15] J. Fan, M. Lu, B. C. Ooi, W.-C. Tan, and M. Zhang, "A hybrid machine-crowdsourcing system for matching web tables," in *Technical Report*, 2013, http://www.comp.nus.edu.sg/~cdas/pdfs/crowdweb.pdf.

[16] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher, "An analysis of approximations for maximizing submodular set functionsłi," *Mathematical Programming*, vol. 14, no. 1, pp. 265–294, 1978. [Online]. Available: http://www.springerlink.com/index/H474774824K48231.pdf

[17] Metaweb Technologies, "Freebase wikipedia extraction (wex)," http://download.freebase.com/wex.

[18] Google, "Freebase Data Dumps," https://developers.google.com/freebase/data.

[19] R. Gupta and S. Sarawagi, "Answering table augmentation queries from unstructured lists on the web," *PVLDB*, vol. 2, no. 1, pp. 289–300, 2009.

[20] H. Elmeleegy, J. Madhavan, and A. Y. Halevy, "Harvesting relational tables from lists on the web," *VLDBJ*, vol. 20, no. 2, pp. 209–226, 2011.

[21] S. Raghavan and H. Garcia-Molina, "Crawling the hidden web," in *VLDB*, 2001, pp. 129–138.

[22] W. Wu, A. Doan, and C. T. Yu, "WebIQ: Learning from the web to match deep-web query interfaces," in *ICDE*, 2006, p. 44.

[23] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy, "Pay-as-you-go user feedback for dataspace systems," in *SIGMOD*, 2008, pp. 847–860.

[24] M. Stonebraker, D. Bruckner, I. F. Ilyas, G. Beskales, M. Cherniack, S. B. Zdonik, A. Pagan, and S. Xu, "Data curation at scale: The data tamer system," in *CIDR*, 2013.

[25] J. Wang, T. Kraska, M. J. Franklin, and J. Feng, "CrowdER: Crowdsourcing entity resolution," *PVLDB*, vol. 5, no. 11, pp. 1483–1494, 2012.

[26] S. E. Whang, P. Lofgren, and H. Garcia-Molina, "Question selection for crowd entity resolution," in *PVLDB*. Stanford InfoLab, August 2013.