Annotating and Searching Web Tables Using Entities, Types and Relationships

Girija Limaye
IIT Bombay, India
girija@cse.iitb.ac.in

Sunita Sarawagi
IIT Bombay, India
sunita@iitb.ac.in

Soumen Chakrabarti
IIT Bombay, India
soumen@iitb.ac.in

ABSTRACT

Tables are a universal idiom to present relational data. Billions of tables on Web pages express entity references, attributes and relationships. This representation of relational world knowledge is usually considerably better than completely unstructured, free-format text. At the same time, unlike manually-created knowledge bases, relational information mined from "organic" Web tables need not be constrained by availability of precious editorial time. Unfortunately, in the absence of any formal, uniform schema imposed on Web tables, Web search cannot take advantage of these high-quality sources of relational information. In this paper we propose new machine learning techniques to annotate table cells with entities that they likely mention, table columns with types from which entities are drawn for cells in the column, and relations that pairs of table columns seek to express. We propose a new graphical model for making all these labeling decisions for each table simultaneously, rather than make separate local decisions for entities, types and relations. Experiments using the YAGO catalog, DBPedia, tables from Wikipedia, and over 25 million HTML tables from a 500 million page Web crawl uniformly show the superiority of our approach. We also evaluate the impact of better annotations on a prototype relational Web search tool. We demonstrate clear benefits of our annotations beyond indexing tables in a purely textual manner.

1. INTRODUCTION

Relational data is almost universally presented as tables for human consumption. In a recent 500 million page Web crawl, we conservatively estimated [12] that over 25 million tables express relational information, as against implementing visual layout. A given entity may be mentioned in dozens to thousands of such tables, in syntactically different forms (e.g., "Albert Einstein" vs. "Einstein"). Each table contributes valuable facts about entities, their types, and relationships between them, and does so in a manner that is considerably less diverse and less noisy, compared to how facts are expressed in free-format text.

Proceedings of the VLDB Endowment, Vol. 3, No. 1
Copyright 2010 VLDB Endowment 2150-8097/10/09... $ 10.00.

Unfortunately, these "organic Web tables" do not adhere to any uniform schema. Captions, contexts, row and column headers, if present, do not use controlled vocabulary. Text in table cells often mention entities, but, being free-form text, the mentions are potentially ambiguous. In particular, there is no direct way to know that two cells in two different tables refer to the same entity, or to associate an attribute with an entity. These limitations lead to under-exploitation of a rich source of structured information on the Web. At best, keywords in queries can match text inside tables, indexed as an undifferentiated blobs of text.

In recent years, information retrieval (IR) tasks have been enhanced to exploit named entities and types, e.g., in entity ranking tasks. Such queries seek entities of a specified type that are mentioned significantly often near specified words [7]. This has been further extended [8] to ad-hoc retrieval of multiple entities that appear to form a logical record, e.g., name, affiliation and email of database researchers. However, the data source remained completely unstructured text.

Suppose we want to compile a table of footballers (soccer players) and clubs they play for. To extract and reconcile this information from many Web tables, we need to determine that specific columns of these tables mention (subsets of) footballers and clubs. Determining that specific cells mention specific footballers or clubs is also generally useful. Finally, it usually helps to determine that two columns in a table are related in the desired manner. Cell and column annotations of these forms can also boost precision in select queries, e.g., list movies directed by (as against featuring as actor) George Clooney.

1.1 Goal

We are given a catalog comprising a type hierarchy with subtype relations, and entities that are instances of types. We are also given a corpus of tables that are not used for formatting and presentation purposes. (Effective heuristics exist [6] for screening out formatting tables.) Our goal is to annotate each table in the following ways:

- Associate one or more types with each column of the table. If a column is deemed not to have any type in our catalog, determine that as well.
- Annotate pairs of columns with a binary relation in the catalog. If two columns are not involved in any binary relation in our catalog, determine that as well.
- Annotate table cells with entity IDs in the catalog, when there is reason to believe that the cell makes a

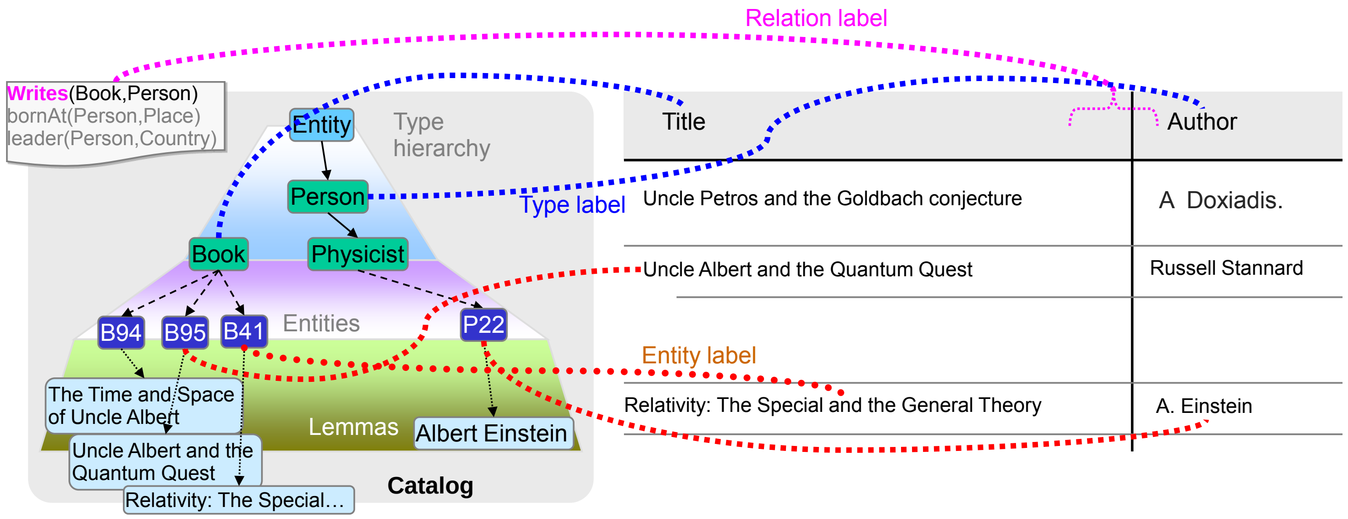


Figure 1: Illustration of cell entity, column type, and relationship labeling.

mention or reference to said entity.

These annotation tasks are challenging. When annotating entity mentions in free-form text [14], the textual context provides clues for disambiguation. In contrast, table cells referring to entities have negligible amounts of additional text. Given each table cell can map to several entities (or none), it is nontrivial to propose one or few types[1] for a column that "explain" (most of) the cells in the column. Finally, there may be no intrinsic clue in a table as to how entities therein are related.

Figure 1 shows a typical scenario. Note that the column header 'Title' can refer to books, movies, or music albums, and "written by" has no word overlap with 'author'. Note also that text similarity is a noisy signal (book title containing 'Albert'). Despite these potential pitfalls, it is possible to explain cells in the first column as mentions of book titles and cells in the second column as elements of 'Person' type based on collective signals.

1.2 Contributions

In this paper we propose machine learning techniques to annotate table cells with entity, type and relation information. We propose a new probabilistic graphical model for simultaneously choosing entities for cells, types for columns and relations for column pairs. We draw the standard types, relations and entities from the YAGO [21] catalog, which has about 250,000 types, two million entities and 99 relations. We train our system and evaluate the accuracy of our annotations using ground truth synthesized from Wikipedia, DBPedia and YAGO. Experiments show that attacking the three subproblems collectively and in a unified graphical inference framework give clear accuracy benefits compared to making local decisions. We also use the trained system to annotate tables in a 500 million page Web crawl. The seed tuples we start with in our catalog are only a small fraction of all the tuples we find and annotate in over 25 million Web tables. We then evaluate the impact of table annotations on a prototype relational Web search tool designed to complete one or more missing fields in a binary relationship.

[1]More than one types are allowed, e.g., German Physicist and Nobel Laureate.

We demonstrate clear benefits of our annotations beyond modeling tables in a purely textual manner.

Outline. In Section 2 we survey related work. In Section 3 we give formal models for the catalog and the source table corpus. Section 4 is the central section that proposes the new model, its associated optimization problem, and its solution. In Section 5 we present a search application to motivate the role of table annotations for improved search results. In Section 6 we describe our experimental testbed and results, and conclude in Section 7.

2. RELATED WORK

Recent years have witnessed active research on bridging the gap between unstructured, semistructured and structured data on the Web. Here we review recent ideas that lead up to our work.

2.1 Web tables and lists

WEBTABLES [6, 5] pioneered the study of tables on the Web as a source of high-quality relational data. A key contribution of WEBTABLES is the collection of attribute co-occurrence statistics, which is used to implement a column thesaurus and propose column auto-completion in queries. The unit of answer in WEBTABLES is a single source table, and the focus is on the ranking of whole source tables. WEBTABLES has no mechanism for annotating cells with entities and columns with types from a catalog. Column names are derived from source tables alone, in the form of text, which is partly why a column name suggestion engine is valuable.

Our system differs in a few fundamental ways. First, our primary goal during preprocessing is to annotate columns with standard type identifiers, and not depend on free-text descriptions of columns.

Second, our goal is to allow more structure in queries, such as the relational expressions $R_1(e_1 \in T_1, E_2 \in T_2)$ (i.e., select-project) and $R_1(e_1 \in T_1, e_2 \in T_2) \wedge R_2(e_2 \in T_2, E_3 \in T_3)$ (i.e., join) where R_1, R_2 are standard relation names, T_1, T_2, T_3 are type literals, E_2, E_3 are entity literals, and e_1, e_2 are entity variables that can be instantiated to literals.

Third, unlike WebTables, our goal is to return a single *synthesized* tables with rows ranked by confidence. Beyond the above queries, note that tagging tables with entities and types lets us express precise join queries without depending on fuzzy text matches. This is left for future work.

A related idea is to curate other Web artifacts like HTML lists into tabular data [12]. No annotation to a standard type or entity catalog was involved in that work. However, our work can further benefit from such additional sources of information.

2.2 Collective entity disambiguation

Another thread of research leading to our work involves annotation of free-form text with references to entities in a catalog. Among the earliest such effort was SemTag [10], which tagged Web documents with references to entities in the TAP catalog [11]. Early work [10, 15, 3] disambiguated each potential entity mention independent of others. Cucerzan [9] and Milne *et al.* [17] were among the first to recognize and exploit the observation that entities mentioned from a single document are likely to be semantically related. Kulkarni *et al.* [14] proposed a precise entity labeling optimization problem that captured local compatibility between mention and entity as well as pairwise similarity between entity labels. Tables with heterogeneous columns violate the basic assumption in all these approaches [9, 17, 14], that entities mentioned on a page are topically homogeneous. This assumption clearly does not apply to tabular data with diverse column types. Our work addresses this important genre of source data.

2.3 Relation extraction

Another common task is to extract instances of relations between named entities from free-form text, e.g., "company A acquired company B for amount M" (see [19] for a survey). Early systems focused on extracting fixed relation types from limited source formats such as news articles. Recently, these have been extended to identify more open-ended relations over the entire Web, based on patterns learned from seed examples or specified manually. A prototypical example is the application of path kernels to dependency parses of sentences [4].

Our data source is very different. On one hand, we seem to have a cleaner source where relations are more explicit, whereas on the other hand, we have little contextual clue.

3. SOURCE MODELS

We are given two input artifacts: a catalog of entities, types and relations, and a corpus of tables. Here we formally define their representation as used in the rest of the work.

3.1 Catalog

The catalog comprises types, entities and relations.

The set of types is $\mathcal{T}$. $T \in \mathcal{T}$ is a type, e.g., *English-language_films*. We will often use canonical strings from Wikipedia or synset names from WordNet [16] to denote types. But a type can be described by more than one such string, which we will call *lemmas* describing the type, $L(T)$. In general, a lemma can have multiple tokens. Internally, each type has a distinct integer ID.

Types are related by a subtype relation $T_1 \subseteq T_2$. This induces a directed acyclic graph where nodes are types and directed edge $T_2 \rightarrow T_1$ denotes $T_1 \subseteq T_2$. We write $T_1 \subseteq^* T_2$ (respectively, $T_1 \subseteq^+ T_2$) if there is a directed path with zero (respectively, one) or more edges from T_2 down to T_1. If not already present, we can create a root type that reaches all other types.

The set of entities is $\mathcal{E}$. Entity $E \in \mathcal{E}$ may be an instance of one or more types T, written as $E \in T$, if there is no other $E \in T' \subseteq T$. This can also be shown as an edge from (the node representing) T to (a node representing) E. If T is transitively reachable from E we write $E \in^+ T$. It is adequate if the raw catalog attaches entities to the most specific types needed; we can always include all ancestor types before starting out.

Let $\mathcal{E}(T) = \{E \in \mathcal{E} : E \in^+ T\}$ be the subset of entities reachable from type T. Likewise, let $\mathcal{T}(E)$ be the subset of types that are ancestors of entity E.

Each entity E has a set of associated *lemmas* $L(E)$. E.g., the city of New York has lemmas *New York*, *New York City* and *Big Apple*. A lemma is a (typically short) sequence of tokens. Lemmas of different entities may be the same (e.g. the state of New York is also called *New York*) or overlapping (e.g. the maker of iPods is called *Apple Computers*).

The set of **b**inary relation names is $\mathcal{B}$. $B \in \mathcal{B}$ is one relation name. We will want to label pairs of source table columns with these canonical relation names. The schema of B is written as $B(T_1, T_2)$. A row or tuple of B is written as $B(E_1, E_2)$, where $E_1 \in T_1, E_2 \in T_2$. Extending to larger arity is left for future work.

The specific catalog we use is YAGO [21], but many others can be modeled in the same way.

3.2 Tables

We preprocess and discard formatting tables as in recent work [6, 5, 12]. We discard tables that use merged rows, columns or cells. I.e., we consider very regular tables where the number of cells is exactly the product of the number of rows and columns. Each row (and each column) may or may not have a header; this is usually clear from HTML formatting. We also capture some amount of textual context around tables. At the end of this process, a table is abstractly represented as:

- The table context, modeled as a short text segment.
- Header cells, if any, denoted by row/column number and cell contents as text segments.
- Number of non-header rows and columns.
- Data cells, each with row+column coordinates, and cell contents represented as a short text segment.

The set of source tables is $\mathcal{S}$ and $S \in \mathcal{S}$ is one source table, with m rows and n columns. By convention we will assume that rows are relation instances and columns are relation attributes, which is the case for all but the smallest tables. We will use $1 \le r \le m$ to index rows and $1 \le c \le n$ to index columns. The text in cell (r, c) will be called D_{rc}. The header text in column c will be called H_c.

4. ALGORITHMS

We model the table annotation problem using a number of interrelated random variables following a suitable joint distribution, represented by a probabilistic graphical model [13]; for a quick primer see Appendix B. The task of annotation then amounts to searching for an assignment of values to the variables that maximizes the joint probability.

4.1 Variables

For a given table $S \in \mathcal{S}$, we associate random variables t_c to denote the type of a column c, e_{rc} to denote the entity label for a cell in row r and column c, and $b_{cc'}$ to denote the relation between column pairs c and c'. Each t_c can be bound to a specific type $T \in \mathcal{T}$ or NA ("no annotation"). Similary, each e_{rc} can be either tied to a specific value $E \in \mathcal{E}$ or NA, and each $b_{cc'}$ can be tied to a specific relation $B \in \mathcal{B}$ or NA denoting no discernible relation between columns c, c'.

4.2 Features and potentials

Intuitively, while assigning values to the variables e_{rc}, t_c and $b_{cc'}$, we need to take into consideration several signals. Following the framework of graphical models, we represent these signals as *features*, and train models that learn how to combine these signals with suitable weights. These features and weights are used to define *potential* functions over subsets of variables, and the product of these potentials gives us the joint distribution over all variables. Thus, our main design problem is choosing a useful set of features.

In the following subsections we will describe different families of features, each measuring some notion of association between source tables and catalogs. Note that no feature is fired if label NA is involved.

4.2.1 Cell text and entity label

Suppose cell (r, c) with cell text D_{rc} is labeled with entity E. How good is this label assignment? Recall that E is known by lemmas in $L(E)$. If any of the lemmas is very similar to D_{rc} then the match is good. Accordingly, we can define a feature $\max_{\ell \in L(E)} \text{sim}(D_{rc}, \ell)$, where sim is the standard TFIDF cosine similarity [18]. We can also use a number of other similarity measures, such as Jaccard or a soft cosine measure [2]. These similarities can be made elements in a vector $\mathbf{f}_1(r, c, E)$.

To balance between the elements of the feature vector, we will use a weight or model vector $\mathbf{w}_1$, and compute $\mathbf{w}_1^\top \mathbf{f}_1(r, c, e_{rc})$, which is a scalar score. The score can, in principle, be negative. These are then combined to declare a *potential* over an entity variable e_{rc} attached to cell (r, c):

$$\phi_1(r, c, e_{rc}) = \exp\left(\mathbf{w}_1^\top \mathbf{f}_1(r, c, e_{rc})\right).$$

4.2.2 Column header and type label

Similar to the previous section, assigning type T to column c is favored if the column header text H_c is similar to one of the lemmas describing T. Again, we can use the standard TFIDF cosine similarity, $\max_{\ell \in L(T)} \text{sim}(H_c, \ell)$, and other similarity measures. Let $\mathbf{f}_2(c, t_c)$ be the feature vector, $\mathbf{w}_2$ be the corresponding weights, and

$$\phi_2(c, t_c) = \exp\left(\mathbf{w}_2^\top \mathbf{f}_2(c, t_c)\right)$$

be the corresponding potential. ϕ_2 tends to be a weaker signal than ϕ_1, because column headers may be omitted, or not match type lemmas as well.

4.2.3 Column type and cell entity

How compatible is it to label a column c with type T and a cell (r, c) in that column with entity E? We wish to create a potential function $\phi_3(T, E)$ to measure this via features $\mathbf{f}_3(t_c, e_{rc})$ as

$$\phi_3(t_c, e_{rc}) = \exp\left(\mathbf{w}_3^\top \mathbf{f}_3(t_c, e_{rc})\right)$$

To get started, we can insist that, unless $E \in^+ T$, $\mathbf{f}_3(\cdots)$ must be zero. How about types that do reach E? Should they all get the same feature value? Suppose for a moment that all cells in a table column have been disambiguated with perfect accuracy to entities. One might argue that the type of the column should be the *most specific* type ancestor of all the entities. We can choose the least common ancestor (LCA). In reality, however, entity labels are uncertain, so insisting on a brittle choice like LCA may be damaging. We will see evidence of this in Section 6. Instead, we will use features to *encourage* specific column types.

The first feature is inspired by the inverse document frequency (IDF) in IR systems [18]. The specificity of a type T can be modeled as $|\mathcal{E}|/|\mathcal{E}(T)|$. If this is large, T is specific. This feature does not depend on the specific cell or entity involved.

The second feature expresses specificity as the distance between E and T. Let $\text{dist}(E, T)$ be the number of edges ($\in$ followed by $\subseteq^*$) on the shortest path between E and T. We want $\text{dist}(e_{rc}, t_c)$ to be generally small. We can therefore use $1/\text{dist}(e_{rc}, t_c)$ as a feature, so that a larger feature value indicates greater favor for t_c, similar to the previous categories of features. There is nothing special in the form $1/\text{dist}(e_{rc}, t_c)$. In information retrieval, damping functions like log or square-root are often used. So we also tried $1/\sqrt{\text{dist}(e_{rc}, t_c)}$. This style of feature has one limitation, which we address next.

Missing links. The above feature "fires" only if $e_{rc} \in^+ t_c$, otherwise it is 0. We can rationalize this via the reasonable convention that if $e_{rc} \notin^+ t_c$, then $\text{dist}(e_{rc}, t_c) = \infty$. One problem with this policy is that catalogs, especially socially-maintained catalogs like Wikipedia (and by inheritance, YAGO) are rarely complete or perfect. In particular, many $\in$ links are missing. For example, at the time of writing, the $\in$ link from Entity Satyajit_Ray to type Indian_film_directors, and the $\subseteq$ link from Universities_in_Toronto to Universities_in_Ontario, are missing.

Therefore, we need positive potentials in selected cases where "$E \in^+ T$" is not known from the catalog but is likely from indirect evidence. Suppose T' is the (only) immediate type ancestor of E. Mining work involving social catalogs often use a relatedness measure between two types [9, 17, 14]. We will use a suitable definition of relatedness to potentially fire a nonzero feature for (E, T) even if $E \notin^+ T$.

As defined before, $\mathcal{E}(T')$ is the set of entities reachable from T'. Consider the quantity $\frac{|\mathcal{E}(T') \cap \mathcal{E}(T)|}{|\mathcal{E}(T')|}$. A large value suggests that the link $E \in^+ T$ may have been missed, because most elements in $\mathcal{E}(T')$ are also in $\mathcal{E}(T)$. Note that we are not modeling this as any kind of probability, but just a hint to the collective annotator. When E has multiple immediate parents types T', we modify the above quantity to $\min_{E \in T'} \frac{|\mathcal{E}(T') \cap \mathcal{E}(T)|}{|\mathcal{E}(T')|}$. Finally, we use this quantity to modify the reciprocal distance feature for entities E not reachable by T to

$$\min_{E \in T'} \frac{|\mathcal{E}(T') \cap \mathcal{E}(T)|}{|\mathcal{E}(T')|} \frac{1}{\min_{E' \in \mathcal{E}(T)} \text{dist}(E', T)}$$

4.2.4 Relation and pair of column types

These features model compatibility between pairs of types T, T' and binary relations B in the catalog. Thus, between every pair of columns c, c' which are likely to be related we get a feature vector $\mathbf{f}_4(b_{cc'}, t_c, t_{c'})$ and the corresponding potential $\phi_4(b_{cc'}, t_c, t_{c'})$ The first feature element in $\mathbf{f}_4$ is set to 1 if there is a schema $b_{cc'}(t_c, t_{c'})$ in the catalog, and 0 otherwise. The second feature measures the fraction of entities under t_c (or $t_{c'}$) that appear in relationship $b_{cc'}$ with an entity in $t_{c'}$ (or t_c).

4.2.5 Relation and entity pairs

If we annotate a relationship $b_{cc'}$ between columns c and c', the entity annotations e_{rc} and $e_{rc'}$ corresponding to different rows can vote for or against it in various ways. This gives rise to another potential $\phi_5(b_{cc'}, e_{rc}, e_{rc'})$ defined through features $\mathbf{f}_5$. The first feature in $\mathbf{f}_5(b_{cc'}, e_{rc}, e_{rc'})$ is 1 if the catalog contains a tuple $b_{cc'}(e_{rc}, e_{rc'})$. The second feature is 1 if relation $b_{cc'}$ is one-to-one or many-to-one and the catalog contains $b_{cc'}(e_{rc}, E')$ for $E' \neq e_{rc'}$, and symmetrically for a one-to-many relation.

For example, in Figure 1 the first feature is 1 if the label of the cell "A. Einstein" is P22, the label of the cell "Relativity: The Special ..." is B96, and `author(B96,P22)` exists in the catalog, and the column pair is labeled with relation name `author`.

Example. We present an example of the resultant graphical model created using the above five kind of potentials in Figure 10.

4.3 Collective objectives

Summarizing, we have variable sets $\mathbf{t} = \{t_c\}, \mathbf{e} = \{e_{rc}\}, \mathbf{b} = \{b_{cc'}\}$, and potentials $\phi_1, \phi_2, \phi_3, \phi_4, \phi_5$ defined on suitable subsets of these variables. For the moment, we assume that all system parameters $\mathbf{w}_1, \mathbf{w}_2, \mathbf{w}_3, \mathbf{w}_4, \mathbf{w}_5$ have been tuned or trained in advance, and we have to assign values to the above variables so as to maximize our objective:

$$\max_{\mathbf{e},\mathbf{t},\mathbf{b}} \underbrace{\prod_{c,c'} \phi_4(b_{cc'}, t_c, t_{c'}) \prod_r \phi_5(b_{cc'}, e_{rc}, e_{rc'})}_{\text{relation}}$$
$$\underbrace{\prod_c \phi_2(c, t_c)}_{\text{columns}} \underbrace{\prod_r \phi_1(r, c, e_{rc})\, \phi_3(t_c, e_{rc})}_{\text{cells}}. \quad (1)$$

The above optimization is called "inference".

The space of values over which the variables range is determined as follows: First, for each cell (r, c) we use a text index to collect candidate entities $\mathcal{E}_{rc}$ based on overlap between cell and lemma tokens. Let $\mathcal{T}(E)$ is the set of all type ancestors of entity E. The space of column labels $\mathcal{T}_c$ is $\bigcup_{E \in \mathcal{E}_{rc}} \mathcal{T}(E)$. The space of relation labels $\mathcal{B}_{cc'}$ is $\bigcup_r \{B : B(E, E')$ exists $, E \in \mathcal{E}_{rc}, E' \in \mathcal{E}_{rc'}\}$. In all cases, we add an additional value "NA" denoting the no annotation option.

We train the various parameters $\mathbf{w}_1, \dots, \mathbf{w}_5$ using the structured learning framework of [22], that generalize Support Vector Machine classifiers to the case when we need to predict multiple variables collectively. The details are skipped because we follow standard machine learning procedures for this training.

4.4 Inference

4.4.1 Special case: no $b_{cc'}, \phi_4, \phi_5$

We first discuss a simplication of optimization (1) where we exclude variables $\{b_{cc'}\}$ and potentials ϕ_4, ϕ_5. Then our objective is

$$\max_{\mathbf{e},\mathbf{t}} \prod_c \phi_2(c, t_c) \prod_{r,c} \phi_1(r, c, e_{rc})\, \phi_3(t_c, e_{rc}). \quad (2)$$

This objective can be solved in polynomial time because the best label for each column can be settled completely independently of other columns. Moreover, once a column label has been fixed, each cell label can be set independent of other cell labels. The pseudocode is shown in Figure 2. Primary key or unique constraints on a column can be handled using a min cost flow formulation [1]. We omit the details, because our real focus is the general case.

```
1: for each column $c$ do
2:     for each type $T \in \mathcal{T}_c$ do
3:         $A_T \leftarrow \phi_2(c, T)$
4:         for each cell $r, c$ in column $c$ do
5:             use a text index to collect candidate entities $\mathcal{E}_{rc}$
               based on overlap between cell and lemma tokens
6:             choose $e^*_{rc} = \arg\max_{E \in \mathcal{E}_{rc}} \phi_1(r, c, E)\phi_3(T, E)$
7:             $A_T \leftarrow A_T \cdot \phi_1(r, c, e^*_{rc}) \cdot \phi_3(T, e^*_{rc})$
8:     finalize $t^*_c = \arg\max_T A_T$
9:     recall and finalize cell assignments $e^*_{rc}$
10: return $\mathbf{t}^*, \mathbf{e}^*$
```

Figure 2: Simplified inference without binary relation variables $b_{cc'}$.

4.4.2 The general case

Inference in the general case (1) is NP-hard, even for a single table; see Appendix C. We resort to an approximate algorithm by drawing on well-known techniques from probabilistic graphical models, specifically, message-passing or belief propagation in factor graphs [13]. A factor graph has two kinds of nodes: (1) *variable* nodes which in our case correspond to the union of types t_c, entities e_{rc}, and relations $b_{cc'}$ variables, and (2) *factor* nodes which correspond to potentials coupling multiple variables. In our case, these are $\phi_3(t_c, e_{rc})$, $\phi_4(b_{cc'}, t_c, t_{c'})$, and $\phi_5(b_{cc'}, e_{rc}, e_{rc'})$.

Inference proceeds by sending messages back and forth between factor nodes and variable nodes according to a given *schedule.* A message $M(i \to f)$ from a variable i to a factor f is calculated by multiplying its own potential with all incoming messages from factors other than f. For example, we compute message from an entity variable e_{rc} to ϕ_3 as

$$M(e_{rc} \to \phi_3) = \phi_1(r, c, e_{rc}) \prod_{c'} M(\phi_5(b_{cc'}, e_{rc}, e_{rc'}) \to e_{rc})$$

A message from a factor f to a variable i is obtained by multiplying f's potential with incoming messages from variables other than i and marginalizing the result on i. For example, a message from ϕ_3 to e_{rc} is computed as:

$$M(\phi_3 \to e_{rc}) = \max_{t_c} \phi_3(t_c, e_{rc}) M(t_c \to \phi_3).$$

Intuitively, this message conveys the belief that factor ϕ_3 has about the label that variable e_{rc} should be assigned. We schedule these messages from entities to ϕ_3 to types and back first. Next, we schedule messages from entities to ϕ_5 to relations and back. Finally, from types to ϕ_4 to relations and back. We repeat this schedule until message values converge from one iteration to the next. In practice we found that convergence was achieved within three iterations. (In the special case of no $b_{cc'}$ variables, this schedule reduces to the direct optimal algorithm shown in Figure 2.) The full set of messages and the overall algorithm appear in Appendix D.

4.5 Baseline annotation algorithms

We will compare our algorithms against two reasonable baseline approaches.

4.5.1 Least common ancestor (LCA)

Let $\mathcal{E}_{rc}$ be the candidate entities to which the cell at (r, c) may be assigned. Recall that $\mathcal{T}(E)$ is the set of all type ancestors of entity E. Consider $\bigcup_{E\in\mathcal{E}_{rc}} \mathcal{T}(E)$. In words, this is the set of all types that may possibly be ancestors of the entity mentioned in cell (r, c), whatever that might be. Therefore, any type that is an ancestor of all cells in the given column must be in $\bigcap_r \bigcup_{E\in\mathcal{E}_{rc}} \mathcal{T}(E)$. Any type in this set that that does not have a descendant also in this set is a candidate for labeling the given column. We report all these types, and evaluate using the F_1 score (harmonic mean of recall and precision). Once a type is assigned to a column, a locally optimal cell entity assignment can be completed using the idea in Figure 2.

4.5.2 Majority

Suppose a cell (r, c) can be assigned entities from set $\mathcal{E}_{rc}$. As before, the cell can potentially belong to any type in $\bigcup_{E\in\mathcal{E}_{rc}} \mathcal{T}(E)$. Let the *vote* for type T be

$$\left|\left\{E : T \in \textstyle\bigcup_{E\in\mathcal{E}_{rc}} \mathcal{T}(E)\right\}\right|$$

We pick types that have more than a threshold F% vote. When F is 50% we get the Majority method and when F = 100% we get LCA. We also report numbers with F in between these two values. In the Majority method we perform entity assignment independently for each cell.

5. A SEARCH APPLICATION

A key motivation behind annotating tables with entities, types, and relations is to be able to ask structured, metadata-cognizant, relational queries over a less structured source. In this section we discuss how our annotation system can assist this process and enhance the quality of responses.

A common form of query one would like to ask of Web tables is:

> Given inputs $R, T_1, T_2, E_2 \in^+ T_2$ where $R(T_1, T_2)$ is in the catalog, return all $E_1 \in^+ T_1$ such that $R(E_1, E_2)$ holds.

This query form is a natural extension of entity search [7, 8] to Web table sources. A common special case of the above query form is to look for entities that have a given value of an attribute.

Figure 3 shows informally how such a query would be processed by a system that does not perform any entity, type or relation annotations. Unlike WebTables, this baseline returns cell contents, not ranked whole tables.

In contrast, in our system, columns have been associated with standard types during preprocessing and indexing, and

```
1: inputs: $R, T_1, T_2, E_2 \in^+ T_2$
2: interpret all inputs as strings
3: look for tables with column headers matching $T_1, T_2$ and
   table context matching $R$
4: for each qualifying table do
5:    look for $E_2$ in the column of $T_2$
6:    for each qualifying row do
7:       collect the cell contents in the $T_1$ column of the row
8: cluster, dedup, rank and present collected cell contents
```

Figure 3: Responding to select-project queries without type annotations.

```
 1: inputs: $R, T_1, T_2, E_2 \in^+ T_2$; $R, T_1, T_2$ are interpreted
    using catalog IDs, $E_2$ if present in catalog
 2: locate all tables that have at least one column $c_1$ labeled
    $T_1$ and a column $c_2$ labeled $T_2$, related by $R$
 3: for each qualifying table do
 4:    if $E_2$ is in the catalog then
 5:       look for cell in column $c_2$ annotated with $E_2$
 6:    else
 7:       look for cell in column $c_2$ with high text similarity
          to the string form of $E_2$
 8:    collect cell in column $c_1$ in this row
 9: aggregate evidence in favor of known entities
10: cluster, dedup, rank and present unannotated cells
```

Figure 4: Responding to select-project queries after type and entity annotations.

this information can be used (typically, interactively [20]) to "harden" the query to a more precise form. Specifically, R, T_1 and T_2 can now refer to precise IDs rather than strings. Figure 4 shows informal pseudocode for how this type information can be exploited.

We will compare these schemes in Section 6.2.

6. EXPERIMENTS

We report on two kinds of experiments: annotation accuracy and the impact of annotation on the search application of Section 5.

YAGO [21] provided the catalog of types (a kind of merger between WordNet synsets and Wikipedia categories), entities, and relations. We used version 2008-w40-2 of YAGO, having 1,941,426 entities, 248,992 types, and 99 relations. We regarded the entity set and their type and relation involvements as sound but potentially incomplete ground truth. Other resources included Wikipedia article text, and a 500-million-page Web crawl. We first extracted over 25 million tables from the crawled corpus, and indexed these tables (including nearby text) using Lucene. The typical number of tables we found per page, and the fraction of those that are relational in nature, are in broad agreement with the experience reported in WebTables [6]. We will call these "Web tables".

6.1 Annotation performance

We collected four table sets with ground truth annotations. A summary of our table sources is shown in Figure 5.

Wiki_Manual: We chose 36 (non-Infobox) tables from Wikipedia article text, based on large content overlap with many Web tables, and manually annotated them with entities, types, and inter-column relations. Description of a few representative tables appears in Appendix E.

Web_Manual: Using the tables from Wiki_Manual as queries, we fetched 371 Web tables similar to them [12]. These were then manually annotated. The main difference between Wiki_Manual and Web_Manual is that the cell, header, and context texts in the latter are more noisy.

Web_Relations: The above two datasets provided only 54 relations. We collected 36 more relations, by using Wiki_Manual to fetch more Web tables, and manually annotated only the relations between column pairs. Cell entities or column types were not labeled.

Wiki_Link: To specifically test the cell entity annotation accuracy at large scale without laborious human judgment, we selected (non-Infobox) tables from Wikipedia text that had more than 90% of their cells linked internally to entities in Wikipedia. This yielded 131 thousand cells with entity annotations spanning 6 thousand tables. No column type or relation annotations were made.

	#Tables	Average	Total annotations		
		#rows	Entity	Type	Rel
Wiki_Manual	36	37	1691	73	10
Web_Manual	371	35	9239	674	44
Web_Relations	30	51	-	-	36
Wiki_Link	6085	20	131807	-	-

Figure 5: Summary of data sets.

6.1.1 *Annotation Quality*

We measure 0/1 loss, i.e., we lose a point if we get a cell wrong, including choosing NA when ground truth was not NA. For column type and relation annotations, we report F_1 score (harmonic mean of recall and precision). If ground truth is missing for a entity, type, or relation, we drop it from the labeling task.

Figure 6 shows the three annotation tasks: entity, type and relation annotation. For each task and each dataset, we show the accuracy (as a percent) of three algorithms: the baseline approaches LCA and MAJORITY, and our algorithm performing collective inference in the full model (1).

Entity annotation accuracy			
Dataset	LCA	MAJORITY	COLLECTIVE
Wiki_Manual	59.75	74.24	**83.92**
Web_Manual	59.68	75.87	**81.37**
Wiki_Link	67.92	77.63	**84.28**
Type annotation accuracy			
Dataset	LCA	MAJORITY	COLLECTIVE
Wiki_Manual	8.63	44.60	**56.12**
Web_Manual	15.16	31.45	**43.23**
Relation annotation accuracy			
Dataset	LCA	MAJORITY	COLLECTIVE
Wiki_Manual	-	62.50	**68.97**
Web_Relations	-	60.87	**63.64**
Web_Manual	-	50.30	**51.50**

Figure 6: Accuracy of entity, type, and relation annotations.

It is immediately evident that COLLECTIVE gives substantially better accuracy at all tasks. In the entity disambiguation task, the typical number of entities between which the algorithms had to choose for each cell was around 7-8. In the column type assignment task, the typical number of types between which the algorithms had to choose for each column was in the hundreds. The accuracies shown in Figure 6 thus represent substantial lift beyond random choice, especially in the challenging open Web domain.

Also note that the accuracy of COLLECTIVE on column types annotation is better for Wiki_Manual compared to Web_Manual, reflecting the more noisy nature of text in Web tables compared to Wikipedia. Further failure analysis (see Appendix F for some anecdotes) revealed that LCA performs particularly poorly at the job because it *overgeneralizes* and MAJORITY suffers because of ambiguities in entities. We hunted for thresholds in-between LCA's 100% and MAJORITY's 50% and obtained the best type accuracy of 46% with a 60% threshold. However, even these numbers are worse than 56% accuracy that COLLECTIVE offers.

6.1.2 *Annotation time*

Figure 7 shows time spent in annotating a snapshot of 250,000 tables from our corpus. The average time per table annotation is 0.7 seconds but there is considerable variation depending on the number of rows, the number of non-numerical columns, and the amount of text in a cell. Further drill-down showed that roughly 80% of the time is spent in probing the lemma index and computing various textual similarity measures between candidate lemmas and cell strings. The inference algorithm accounts for less than 1% of the total time.

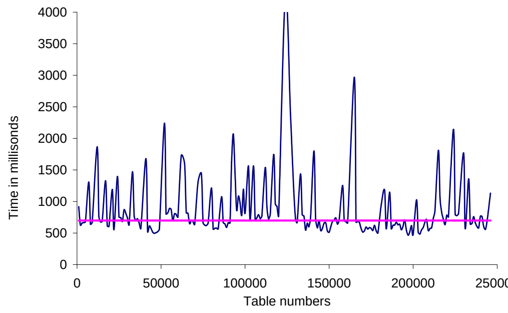


Figure 7: Time spent in annotating tables.

6.1.3 *Training*

For training model parameters $\mathbf{w}_1$ through $\mathbf{w}_5$, we used the Wiki_Manual data using our implementation of a Java-based structured learner [22]. Testing was done on the Wiki_Manual dataset and the Web_Manual datasets. Although our training and test data are not disjoint, we did not observe overfitting issues because the number of features was not too large. We trained our model parameters using the manually collected ground truth and for three different settings of measuring type entity compatibility: $1/\sqrt{\text{dist}}$, 1/dist, and IDF, as discussed in Section 4.2.3. The results appear in Figure 8. $1/\sqrt{\text{dist}}$ appears robust, and IDF on its own performs poorly for type labeling.

6.2 Search

We evaluated entity search queries of the kind described in Section 5 under three settings: a baseline that does no annotations, with only type annotations, and with type and relation annotations.

Entity annotation accuracy			
Dataset	$1/\sqrt{\text{dist}}$	1/dist	IDF
Wiki_Manual	83.92	84.30	85.44
Web_Manual	81.37	80.52	80.06
Type annotation accuracy			
Dataset	$1/\sqrt{\text{dist}}$	1/dist	IDF
Wiki_Manual	56.12	50.36	40.29
Web_Manual	43.23	42.10	25.97

Figure 8: Type-entity compatibility features.

We generated a workload from five relations listed in Appendix G and for each relation randomly selected forty E_2 values in YAGO that participate in the relation. The query was posed on the Web table corpus described above. The answer is a ranked list of entities.

To assess relevance, we used the RDF triples in DBPedia (`http://dbpedia.org/About`) as ground truth. DBpedia's RDF triples are extracted from Wikipedia's Infoboxes but *not* 'organic' tables in Wikipedia page text, so this source is different from Wiki_Manual and Wiki_Link. We score a response using mean average precision (MAP), which is standard in information retrieval (`http://en.wikipedia.org/wiki/Information_retrieval#Mean_Average_precision`).

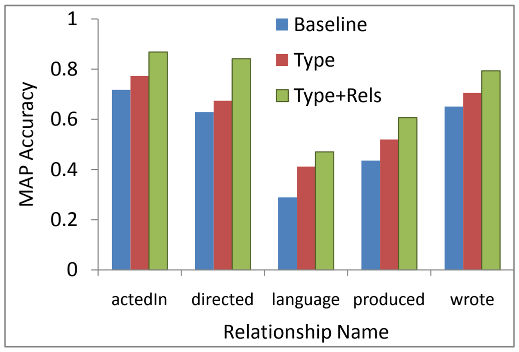


Figure 9: Mean Average Precision (MAP) for attribute-value queries without any annotation (Baseline), with only column type annotations (Type), and with both column type and relation annotations (Type+Rel).

The results are shown in Figure 9: adding type labels is better than baseline, and adding both type and relation labels is best. It might be argued that this comparison is not fair on the baseline, because it does not have the benefit of a structured query language. There are two counter-arguments. First, our goal is not to offer better responses to keyword queries, but to raise the level of interaction and bring it closer to querying structured data, but without the luxury of a clean data source. Second, the gains from our system are also attributable to corpus annotations, not just query structuring. Separating the two effects would be interesting future work.

7. CONCLUSION

We have presented a new system that annotates open-domain tables on the Web with entity, type and relation information. Thereby, it can harness the power of 'organic' Web tables to answer simple relational queries, even though the source tables do not have any uniform or identifiable schema. We gave a precise model for the annotation task, balancing local compatibility between entities and their potential mentions in table cells, and global constraints between relations, types and columns. The Web will never have a complete 'schema'. Socially maintained catalogs will always be incomplete. Our work paves the way to augment catalogs with dynamic relational information. We demonstrate that this approach can lead to better responses to relational queries on the unstructured Web.

8. REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, 1993.

[2] M. Bilenko, R. Mooney, W. Cohen, P. Ravikumar, and S. Fienberg. Adaptive name-matching in information integration. *IEEE Intelligent Systems*, 2003.

[3] R. Bunescu and M. Pasca. Using encyclopedic knowledge for named entity disambiguation. In *EACL*, pages 9–16, 2006.

[4] R. C. Bunescu and R. J. Mooney. A shortest path dependency kernel for relation extraction. In *EMNLP Conference*, pages 724–731. ACL, 2005.

[5] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. WebTables: exploring the power of tables on the Web. *PVLDB*, 1(1):538–549, 2008.

[6] M. J. Cafarella, A. Halevy, Y. Zhang, D. Z. Wang, and E. Wu. Uncovering the relational Web. In *WebDB*, volume 11, Vancouver, June 2008.

[7] S. Chakrabarti, K. Puniyani, and S. Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW Conference*, Edinburgh, May 2006.

[8] T. Cheng, X. Yan, and K. C. Chang. EntityRank: Searching entities directly and holistically. In *VLDB Conference*, pages 387–398, Sept. 2007.

[9] S. Cucerzan. Large-scale named entity disambiguation based on Wikipedia data. In *EMNLP Conference*, pages 708–716, 2007.

[10] S. Dill et al. SemTag and Seeker: Bootstrapping the semantic Web via automated semantic annotation. In *WWW Conference*, 2003.

[11] R. V. Guha and R. McCool. TAP: A semantic web test-bed. *Journal of Web Semantics*, 1(1):81–87, 2003.

[12] R. Gupta and S. Sarawagi. Answering table augmentation queries from unstructured lists on the Web. *PVLDB*, 2(1):289–300, 2009.

[13] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques.* MIT Press, 2009.

[14] S. Kulkarni, A. Singh, G. Ramakrishnan, and S. Chakrabarti. Collective annotation of Wikipedia entities in Web text. In *SIGKDD Conference*, 2009.

[15] R. Mihalcea and A. Csomai. Wikify!: linking documents to encyclopedic knowledge. In *CIKM*, pages 233–242, 2007.

[16] G. Miller, R. Beckwith, C. FellBaum, D. Gross, K. Miller, and R. Tengi. Five papers on WordNet. Princeton University, Aug. 1993.

[17] D. Milne and I. H. Witten. Learning to link with Wikipedia. In *CIKM*, 2008.

[18] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[19] S. Sarawagi. Information extraction. *FnT Databases*, 1(3), 2008.

[20] A. Singh, S. Kulkarni, S. Banerjee, G. Ramakrishnan, and S. Chakrabarti. Curating and searching the annotated web. In *SIGKDD Conference*, 2009. System demonstration.

[21] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: A core of semantic knowledge unifying WordNet and Wikipedia. In *WWW Conference.* ACM Press, 2007.

[22] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6(Sep):1453–1484, 2005.

Acknowledgments. The work reported here was supported in part by research grants from Yahoo! Research, HP Labs, and IBM Faculty awards.

APPENDIX

A. NOTATION

$\mathcal{T}$	Set of all type labels
$T \in \mathcal{T}$	One type label
$T_1 \subseteq T_2$	Subtype relation
$\subseteq^*, \subseteq^+$	Transitive closure of $\subseteq$
$\mathcal{E}$	Set of all entities known in catalog
$E \in \mathcal{E}$	One entity label
$E \in T$	E is a direct instance of T
$E \in^+ T$	E is transitively an instance of T
$\mathcal{E}(T)$	Entities that are transitive instances of T
$\mathcal{T}(E)$	All type ancestors of E
$\mathcal{B}$	All relation names in catalog
$B \in \mathcal{B}$	One relation name (label)
r, c	Row, column of a table
D_{rc}	Text in cell r, c
e_{rc}	Variable representing entity label of cell r, c
H_c	Header text in column c of table
t_c	Variable representing type label of column c
$b_{cc'}$	Variable representing relation label of cols c, c'
NA	special label, "no annotation"

B. GRAPHICAL MODEL FRAMEWORK

A probabilistic graphical model provides a convenient and efficient framework for expressing the joint distribution $\Pr(\mathbf{x})$ over N variables $\mathbf{x} = (x_1, \ldots, x_N)$. Variables are indexed as x_i, x_j, etc. For simplicity, assume each x_i takes *label* values in $[1, M]$, whereas in general each x_i can take values from its own space. Label values will be denoted k, ℓ, etc. A probabilistic graphical model [13] captures the dependencies between elements of $\mathbf{X}$, each a node in the graph, with a sparse set of edges as follows:

We first identify small subsets of variables, called *cliques*, that are highly dependent on each other. We will be choosing single nodes x_i and node pairs x_i, x_j as cliques.

Next, we design *node potentials* $\phi_i : [1, M] \to \mathbb{R}_+$. This is a kind of un-normalized measure of compatibility between the variable and each value. We also design *edge potentials* $\phi_{i,j} : [1, M] \times [1, M] \to \mathbb{R}_+$. This is a kind of un-normalized measure of compatibility between the labels of variables believed to be correlated. $\Pr(\mathbf{x})$ is modeled as

$$\Pr(x_1, \ldots, x_n) = \frac{1}{Z} \prod_i \phi_i(x_i) \prod_{i,j} \phi_{i,j}(x_i, x_j)$$

$$\text{where} \quad Z = \sum_{\mathbf{x} \in [1,M]^N} \prod_i \phi_i(x_i) \prod_{i,j} \phi_{i,j}(x_i, x_j),$$

called the *partition function*, serves to normalize the product of clique potentials to a proper probability.

A common method to define potentials is as a dot product between a *model* vector and a *feature* vector. In case of node potential, we might write $\phi_i(x_i) = \exp\left(\mathbf{w}_1^\top \mathbf{f}(x_i)\right)$, where $\mathbf{f} : [1, M] \to \mathbb{R}^k$ is the feature vector and $\mathbf{w}_1 \in \mathbb{R}^k$ is the model vector where k is the number of features. Similarly, an edge potential would be defined as $\phi_{i,j}(x_i, x_j) = \exp\left(\mathbf{w}_2^\top \mathbf{f}(x_i, x_j)\right)$. The feature vector are designed by the user whereas the model vectors $\mathbf{w}_1, \mathbf{w}_2$ are *trained* from labeled data.

Once $\Pr(\mathbf{x})$ is defined and the model vectors are trained, the *inference problem* is to find $\arg\max_{\mathbf{x}} \Pr(\mathbf{x})$.

In our case, the variables $\mathbf{x}$ are a union of $t_c, e_{rc}, b_{cc'}$ variables. The cliques are defined over pairs of variables (t_c, e_{rc}) and triples $(b_{cc'}, t_c, t_{c'})$ $(b_{cc'}, e_{rc}, e_{rc'})$. The feature vectors used to define the node potentials, edge potentials and triple potentials are explained in Section 4.2.

In Figure 10 we show an example graphical model that arises in annotating a table with three rows and three columns. The variables are represented as circles and potentials are shown as ϕ node. ϕ_1 and ϕ_2 are node potentials respectively on the t_c and e_{rc} variables. $\phi_3(t_c, e_{rc})$ represents an edge potential between t_c and e_{rc} variables and ϕ_4 and ϕ_5 are clique potentials among three variables at a time.

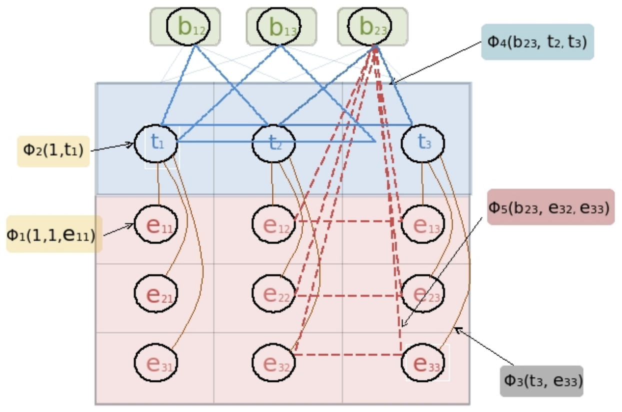


Figure 10: A Graphical model representing type, entity, and relationship variables on a table with three columns and three rows. Accordingly, there are three type variables t_1, t_2, t_3, nine entity variables e_{rc}, and three relation variables $b_{cc'}$.

C. HARDNESS OF INFERENCE

Inference in the general case (1) is NP-hard, even in case of a single table, via a reduction from graph coloring. We give a rough sketch. The graph coloring instance consists of an undirected graph $G = (V, A)$ and a number K, and asks if G is colorable with at most K colors. We build a single table with $|V|$ columns, each column c corresponding to one node, also called c. We create $|V|K$ types in the catalog: K types T_{uk} for each node u. Column u can be assigned only one of the k types $T_{u\cdot}$. For each arc $(u, v) \in A$, we introduce $\binom{k}{2}$ relation schema with a suitably large potential π into the catalog: $B_{uv}(T_{uk}, T_{vk'})$, for $k \neq k'$. All other potentials are zero or one as needed. If there is a K-coloring of G, then the objective for the corresponding graph is $\pi^{|A|}$, and the converse also holds.

D. MESSAGE PASSING ALGORITHM

Please see Figure 11. Training follows a very similar message passing scheme that is standard [13] and is omitted from this version.

E. SAMPLE TABLES FROM Wiki_Manual

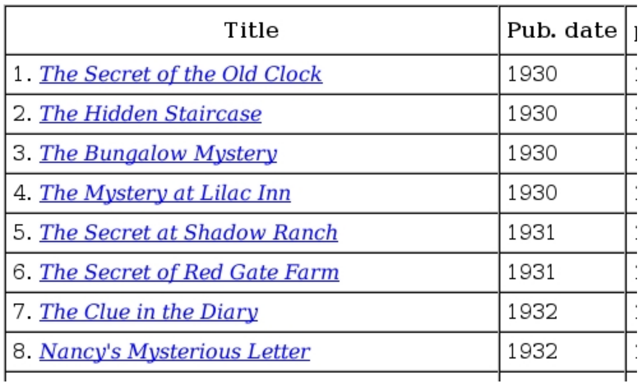
```
1: Get candidate entities, types, and relations as described in Section 4.3
2: Initialize all messages to 1.
3: for Iterations=1 to Max-iterations do
4:   for each column $c$ do
5:     for each row $r$ do
6:       $M(e_{rc} \rightarrow \phi_3(t_c, e_{rc})) = \phi_1(r, c, e_{rc}) \prod_{c'} M(\phi_5(b_{cc'}, e_{rc}, e_{rc'}) \rightarrow e_{rc})$
7:       $M(\phi_3(t_c, e_{rc}) \rightarrow t_c) = \max_{e_{rc}} \phi_3(t_c, e_{rc}) M(e_{rc} \rightarrow \phi_3)$
8:     for each row $r$ do
9:       $M(t_c \rightarrow \phi_3(t_c, e_{rc})) = \phi_2(c, t_c) \prod_{c'} M(\phi_4(b_{cc'}, t_c, t_{c'}) \rightarrow e_{rc}) \prod_{r' \neq r} M(\phi_3(t_c, e_{r'c}) \rightarrow t_c)$
10:      $M(\phi_3(t_c, e_{rc}) \rightarrow e_{rc}) = \max_{t_c} \phi_3(t_c, e_{rc}) M(t_c \rightarrow \phi_3)$.
11:  for each column pair $c, c'$ with candidate relations do
12:    for each row $r$ do
13:      $M(e_{rc} \rightarrow \phi_5(b_{cc'}, e_{rc}, e_{rc'})) = \phi_1(r, c, e_{rc}) \prod_{c'' \neq c'} M(\phi_5(b_{cc''}, e_{rc}, t_{c''}) \rightarrow e_{rc}) \prod_{r} M(\phi_3(t_c, e_{r'c}) \rightarrow e_{rc})$
14:      $M(e_{rc'} \rightarrow \phi_5(b_{cc'}, e_{rc}, e_{rc'}))$ = similar to above.
15:      $M(\phi_5(b_{cc'}, e_{rc}, e_{rc'}) \rightarrow b_{cc'}) = M(e_{rc} \rightarrow \phi_5(b_{cc'}, e_{rc}, e_{rc'})) M(e_{rc'} \rightarrow \phi_5(b_{cc'}, e_{rc}, e_{rc'}))$
16:    for each row $r$ do
17:      $M(b_{cc'} \rightarrow \phi_5(b_{cc'}, e_{rc}, e_{rc'})) = \prod_{r \neq r'} M(\phi_5(b_{cc'}, e_{r'c}, e_{r'c'}) \rightarrow b_{cc'}) M(\phi_4(b_{cc'}, t_c, t_{c'}) \rightarrow b_{cc'})$
18:      $M(\phi_5(b_{cc'}, e_{rc}, e_{rc'}) \rightarrow e_{rc}) = \max_{e_{rc'}, b_{cc'}} \phi_5(b_{cc'}, e_{rc}, e_{rc'}) M(e_{rc'} \rightarrow \phi_5(b_{cc'}, e_{rc}, e_{rc'})) M(b_{cc'} \rightarrow \phi_5(b_{cc'}, e_{rc}, e_{rc'}))$
19:      $M(\phi_5(b_{cc'}, e_{rc}, e_{rc'} \rightarrow e_{rc'})$ = similar to above.
20:    $M(t_c \rightarrow \phi_4(b_{cc'}, t_c, t_{c'})) = \phi_2(c, t_c) \prod_{c'' \neq c'} M(\phi_4(b_{cc''}, t_c, t_{c''}) \rightarrow e_{rc}) \prod_{r} M(\phi_3(t_c, e_{r'c}) \rightarrow t_c)$
21:    $M(t_{c'} \rightarrow \phi_4(b_{cc'}, t_c, t_{c'}))$ = similar to above.
22:    $M(\phi_4(b_{cc'}, t_c, t_{c'}) \rightarrow b_{cc'}) = M(t_c \rightarrow \phi_4(b_{cc'}, t_c, t_{c'})) M(t_{c'} \rightarrow \phi_4(b_{cc'}, t_c, t_{c'}))$
23:    $M(b_{cc'} \rightarrow \phi_4(b_{cc'}, t_c, t_{c'})) = \prod_{r} M(\phi_5(b_{cc'}, e_{rc}, e_{rc'}) \rightarrow b_{cc'})$
24:    $M(\phi_4(b_{cc'}, t_c, t_{c'}) \rightarrow t_c) = \max_{t_{c'}, b_{cc'}} \phi_4(b_{cc'}, t_c, t_{c'}) M(t_{c'} \rightarrow \phi_4(b_{cc'}, t_c, t_{c'})) M(b_{cc'} \rightarrow \phi_4(b_{cc'}, t_c, t_{c'}))$
25:    $M(\phi_4(b_{cc'}, t_c, t_{c'} \rightarrow t_{c'})$ = similar to above.
26:  Exit if converged
27: Recall and finalize labels of variables from messages.
```

Figure 11: Message passing algorithm to get collective entity, type, and relation assignments.

- List of Simpsons episodes with fields episode, director and writer
- List of Presidential libraries with fields president, library and location
- List of nuclear research reactors with fields operator, location and reactor
- List of Newbery medal winners and their respective novels
- List of actors in West Wing and names of their fictional characters

F. LCA OVER-GENERALIZES

LCA over-generalizes because of noise in cell-to-entity annotations and missing links in the catalog. We illustrate this with a real example. Figure 12 shows a Web table with the first column containing titles of Nancy Drew novels. All of these novels appear in YAGO and are correctly labeled with their entity labels. However, the $\in$ link from one of the entities, `The_Clue_of_the_Black_Keys` to the type `Nancy_Drew_books` is missing. The two immediate parents of this entity are `1951_novels` and `Children's_novels`. Furthermore, the $\subseteq$ link from `Nancy_Drew_books` to `Children's_novels` is missing. This makes the LCA choose the root type, `Entity` for the first column.



Title	Pub. date
1. The Secret of the Old Clock	1930
2. The Hidden Staircase	1930
3. The Bungalow Mystery	1930
4. The Mystery at Lilac Inn	1930
5. The Secret at Shadow Ranch	1931
6. The Secret of Red Gate Farm	1931
7. The Clue in the Diary	1932
8. Nancy's Mysterious Letter	1932

Figure 12: Example table where LCA over-generalizes.

G. RELATIONS USED FOR SEARCH QUERY EXPERIMENTS

Figure 13 lists the five relations on which we reported our Search experiments in Section 6.2.

Relation name	Type 1	Type 2
acted in	movie	actor
directed	movie	director
wrote	novel	novelist
official language	country	language
produced	movie	producer

Figure 13: List of relations and their types used in experiments of Section 6.2