Semi-structured Document Annotation using Entity and Relation Types

Arpita Kundu[1](✉), Subhasish Ghosh[1], and Indrajit Bhattacharya[1]

TCS Research, India
{arpita.kundu1,g.subhasish,b.indrajit}@tcs.com

Abstract. Semi-structured documents such as web-pages and reports contain text units with complex structure connecting these. We motivate and address the problem of annotating such semi-structured documents using a knowledge graph schema with entity and relation types. This poses significant challenges not addressed by the existing literature. The latent document structure needs to be recovered, and paths in the latent structure need to be jointly annotated with entities and relationships. We present a two stage solution. First, the most likely document structure is recovered by structure search using a probabilistic graphical model. Next, nodes and edges in the recovered document structure are jointly annotated using a probabilistic logic program, considering logical constraints as well as uncertainty. We additionally discover new entity and relation types beyond those in the specified schema. We perform experiments on real webpage and complex table data to show that our model outperforms existing table and webpage annotation models for entity and relation annotation.

Keywords: Semi-structured documents · Document annotation · Knowledge discovery · Graphical model · Structure search · Probabilistic logic program.

1 Introduction

Semi-structured or richly structured documents, such as web-pages, reports and presentations, are among the most abundant information sources created by humans for human consumption. These consist of text units connected by spatial structure, and the spatial structures bear evidence of the semantic relations between the text units. There has been a lot of recent interest in extracting information from such semi-structured documents by annotating these with entities and relations from a knowledge graph [18, 10, 9].

However, document structures in the real world are more complex and interesting than those addressed in the literature. One view of such documents is that of complex tables, with cells that span multiple rows and columns. But the table annotation literature so far has only considered simple tables [8, 3, 20, 17]. The second view is that of html structures as in web-pages [10, 9]. Again, this area of research has ignored much of the complexity of such structures.

First, a typical document is not about a single entity and its properties, but about multiple related entities. Secondly, the observed visual structure may not be the true semantic structure. Text units are not always related to the closest neighbors, and the observed structure often leaves out header text units. Thirdly, groups of documents often share a document structure template (or DST), such as distinct structures for movie pages, actor pages, studio pages, etc. in the movies domain. However, the DSTs are not directly observed, and the structure of individual documents may contain minor variations within the theme.

We consider the problem of annotating such semi-structured documents using a knowledge graph schema that specifies entity types and binary relation types between these entity types. The structure of real schemas can be complex. In general, these are directed multi-graphs. In contrast, existing literature [10, 9] only considers schemas that are collections of disconnected star-shapes, each about an entity and its properties. Our observation is that, for complex document structures and complex schemas, the nature of the annotation is also complex. Not all nodes in the document structures correspond to entity types. As a result, individual edges in the document structures cannot be annotated with relation types in the schema. Instead, relation types correspond to document structure paths. This leads to structural constraints on the annotations, which calls for joint annotation of the nodes and paths in the document structure with entity and relation types. Finally, we highlight the task of entity type and relation type discovery. Available knowledge graph schemas for a domain are precise, but far from complete. A typical document corpus mentions many entity and relation types not specified in the schema. Such new entity and relation types need to be discovered from the corpus and incorporated into the schema.

To address these challenges, we propose a two-stage approach. The first stage uses a probabilistic graphical model with structural parameters to capture the complexity of DSTs and variations within a DST. We use a novel structure search algorithm using this model to identify the latent structure and DST for each document. The second stage annotates recovered document structures with entity and relation types. Since this stage requires logical rules to capture background knowledge about constraints while accommodating uncertainty of annotation, we use probabilistic logic programming [14, 13], which naturally supports both. We also propose an iterative algorithm for discovering new entity and relation types by analyzing patterns in the unannotated parts of the document structure.

Using experiments on three real datasets of different types, we demonstrate the flexibility of our approach and improvements in annotation accuracy over state-of-the-art webpage and table annotation models.

2 Related Work

Some of the work on extracting information from semi-structured documents [18, 10, 9] focus specifically on web-pages [10, 9]. These assume documents to be about a single entity so that the document structure is a star. No latent structure different from the observed structure is considered. Though these use a notion of structural templates, documents within a template are assumed to have exactly the same structure. The knowledge graph schema is also assumed to be a star with the topic entity at the center, so that only properties of the topic entity are annotated as relations. Independent annotation of individual document edges is sufficient in this setting.

Annotation of web table using a knowledge graph has received a lot of attention in recent years [8, 3, 20, 17]. These only address simple table structures, whereas the document structures for us are DAGs. These either use graphical models [8, 17] to reason jointly using the table structure, or reduce tables to documents and use deep models for sequence data for annotation [3, 20]. These consider the table structure to be directly observed. These also do not consider shared templates for tables in a corpus.

While we have used ProbLog for probabilistic logic programming, other existing frameworks could have been used [1, 15, 16]. Markov logic network (MLN) [15] combine (undirected) graphical models with logic, which is also our end-to-end goal. However, the existing Alchemy implementation of MLNs cannot perform the sophisticated structure search required by our task.

3 Problem Statement

We begin by defining semi-structured documents and the problem of annotating such documents with a knowledge graph schema.

Document, Structure and Template: We have a set of semi-structured documents D: webpages, reports, slide decks, etc. Each document d_i has a set of text units. In Fig.1, the first column shows two example documents and their text units. Each text unit has a sequence of tokens w_{ij}.

The documents have visible structure connecting the text units based on their relative locations within the document. The second column in Fig.1 shows the corresponding observed structures for the two documents in the first column. Each text unit is a node in this structure. Node 1 is immediately above nodes 2 and 3, while node 2 is on the immediate left of node 3, which explains the edges between them. We consider the observed document structure to be a directed acyclic graph (DAG), where $\pi_{ijj'}$ is a boolean variable that indicates whether node j is a parent of node j' in document i. The parents may be obtained directly from html constructs of webpages, or indirectly from the spatial layout of the text units. To accommodate this, a node can additionally have two dimensional location coordinates $l_{ij} = (x_{ij}, y_{ij})$. The examples illustrate the alternative view of such documents as tables, with integer location coordinates. But observe that the tables are complex - text units can span multiple rows or columns.

The observed structure may not reveal the true semantic relationships. In both example documents, the 'Details' and 'Movie Details' have 'Stars' as parents. Semantically, both should have 'Movie' nodes as their parents. The third column shows the latent structures for the two documents. The latent structure is also a DAG. Let $\pi^*_{ijj'}$ denote the latent parents. Each latent node has its corresponding text w^*_{ij} and location l^*_{ij}.

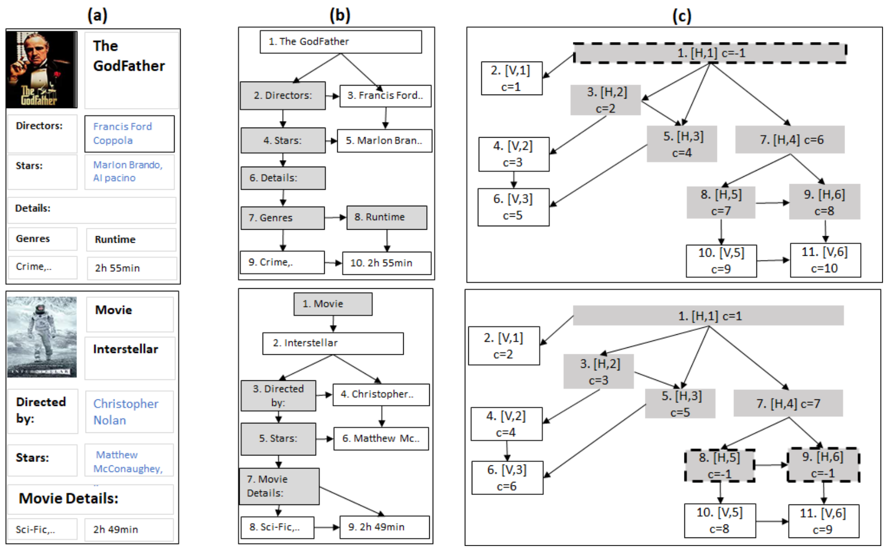

Fig. 1. Example showing (a) two semi-structured documents, (b) their observed structures, and (c) latent structures. Header nodes are shaded gray and node types are mentioned as $[t^p_{ij}, t^s_{ij}]$ inside latent nodes. Corresponding observed nodes (c_{ij}) are also mentioned inside latent nodes. Latent nodes missing in the observed structure ($c = -1$) are marked with dashed borders. Both documents follow the 'Movie' template.

Additionally, the nodes are of different types. At the highest level, these are headers (H) or values (V). The variable $t^p_{ij} \in \{H, V\}$ denotes the primary type of the node. The nodes also have secondary types $t^s_{ij} \in \mathbb{N}$ for both headers and values. These represent different types of values nodes - long textual descriptions, short texts, numbers, etc, and accordingly different types of header nodes associated with these different types of value nodes.

Not all nodes in the latent structure have corresponding nodes in the observed structure. Specifically, header nodes are often meant to be understood from context. Let c_{ij} denotes the corresponding observed node for the j^{th} latent node. Latent nodes that do not have corresponding observed nodes have $c_{ij} = -1$.

Finally, latent structures are often shared between groups of documents in a corpus. Both our example documents are about 'Movies'. As a result, even though they have different observed structures, their latent structures are the same. Other documents about 'Actors' or 'Production Houses' may again have similar structure, different from those of 'Movie' documents. We use $T_i \in \mathbb{N}$ to denote the document structure template (DST in short) of the i^{th} document. Note that two documents with the same DST do not necessarily have identical latent structures. For instance, not all 'Movie' documents may mention run-times. However, they have the same type for the root node of their latent structures, and very similar latent structures overall.

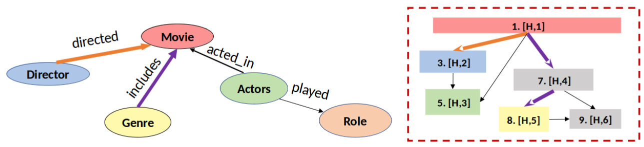

Fig. 2. Example showing a knowledge graph schema with entity and relation types, a latent document structure (only header nodes) and its entity and relation annotation. Latent nodes' colors show their entity annotation. Gray nodes are not annotated with any entity. The orange edge is annotated with the relation 'directed'. The purple path is annotated with the relation 'includes'.

Knowledge Graph Schema: We also have a Knowledge Graph Schema G, specifying the entity types and the relation types for the domain. A toy knowledge graph is shown on the left in Fig.2. Let E be the set of entity types. Let R be the set of binary relation types between entity types in E. The schema graph is a multi-graph in general. In the rest of this paper, we use entities as shorthand for entity types, and relations for relation types.

Annotation: An annotation is a mapping between the latent document structure and the entities and relations. Only header latent nodes ($t^p_{ij} = H$) can correspond to entities. For a latent node, $e_{ij} \in E \cup \{-1\}$ denotes its *entity annotation*. The right side of Fig.2 shows the annotation of the common latent structure of our two example documents in Fig.1(c), with the knowledge graph schema on the left. Some header nodes may not correspond to entities in the given schema G. Such nodes have $e_{ij} = -1$.

Next is *relation annotation*. In the simplest case, an edge in the latent structure corresponds to a relation. However, relations cannot always be mapped to individual edges. For example, node 1 corresponding to 'Movie' is connected to node 8 corresponding to 'Genre' by the *path* $(1, 7, 8)$. Therefore, the entire path corresponds to relation 'includes' between 'Genre' and 'Movie'. This is allowed only because the intermediate node 7 does not correspond to any entity. A latent path p_{il} in the i^{th} document is a sequence of latent nodes of header type ($t^p_{ij} = H$). Let $r_{il} \in R \cup \{-1\}$ denote the *relation annotation* of path p_{il}. Path annotations have to be consistent with entity annotations. If $p_{il} = r$ and the relation r is between entities e_1 and e_2, then the first node j in the path must have $e_{ij} = e_1$ and the last node j' must have $e_{ij'} = e_2$. As a consequence, in a path annotated with any relation ($p_{il} \neq -1$), an internal node j'' cannot be annotated with an entity (i.e. must have $e_{ij''} = -1$). Additionally, path annotations in a document have to be consistent with each other. The same edge cannot appear in two different paths with different relation annotations.

There may be many consistent entity and relation annotations for a document latent structure. Note that annotating every node and edge with -1 is a trivially consistent annotation but one that should ideally have a very low probability. On the other hand, the annotation shown in the example in Fig.2 looks very likely and should have a high probability. Thus, annotation involves logical constraints as well as uncertainty.

Structured Document Annotation Problem: We now define the annotation problem for a semi-structured corpus. We are given a knowledge graph schema G with entities E and relationships R. We are also given a training corpus D of semi-structured documents. For these, the observed document structures are given. In addition, we assume the latent document structure π^*, and the annotations of the latent nodes and paths with entities and relations in the KG are also available for training. Based on the entity annotations, the primary types of some nodes are known to be header, but secondary node types t^s, or the document templates T are not given. The number of secondary node types and the number of templates may also not be known.

Now, given only the observed document structure of a test document d, the task is two fold. (A) **Structure recovery:** Identify the latent document structure π^*_d, along with the primary and secondary node types t^p_d and t^s_d and the document template T_d. (B) **Annotation:** Annotate the recovered latent structure with entities and relations. Note that the relation annotations for different paths needs to be performed jointly along with the entity annotations, because of the constraints on them.

Additionally, we consider the problem of **entity and relation discovery**. Here, we assume that only a subset G^o of the KG schema containing a subset of the entities E^o and relations R^o are observed during training. As a result, the training annotations now only contain the observed entities and relations. No annotations are available for the remaining entities $E - E^o$ and relations $R - R^o$. In our example, the complete schema may contain entity 'Duration' and relation 'has_runtime' between 'Movie' and 'Duration', but this is not in the training schema G^o. Now, an additional task is to recover such latent entities and relations from the training corpus, and annotate the test documents not only with the observed entities and relations, but the latent ones as well.

4 Proposed Approach

Our proposed approach has three stages. As shown in Fig.1, the first stage takes the observed structure of a document and uses a probabilistic graphical model to recover the latent structure, node types and the DST by structure search. As shown in Fig.2, the second stage takes the latent structure and node types, and uses a probabilistic logic program to annotate the nodes and the paths satisfying logical constraints. These two stages are supervised. The final stage analyzes patterns in un-annotated parts of the latent structures to discover new entities and relations.

4.1 Document Structure Recovery using Generative PGM

We define a joint probability distribution over the observed and latent structure variables, and then present a structure recovery algorithm using the model.

Generative Model: The model is iid over documents. We factorize the joint distribution over the variables for each document into 5 major components and provide these with generative semantics as follows:

$$T \sim Cat(\alpha)$$

$$t, \pi^* \sim P(t, \pi^* \mid T; \beta, \mu) \qquad w^*, l^* \sim P(w^*, l^* \mid T, t, \pi^*; h(t), \lambda, \tau, \gamma)$$

$$c, \pi \sim P(c, \pi \mid T, t, \pi^*; \phi) \qquad w, l \sim P(w, l \mid c_i, w^*, l^*)$$

The first step samples the DST T. The second and third steps sample the node types t and the latent structure π^*, content w^* and location l^* conditioned on the DST T. The fourth and fifth steps sample the observed structure π and correspondence c conditioned on the latent structure and the DST, and finally the content and location of the observed nodes conditioned on those for the latent nodes and the correspondence. Next, we define each of these steps in further detail and the parameters involved in each.

When the number of unique DSTs is known ahead of time, the DST T follows a categorical distribution $Cat(\alpha)$ shared by all documents. When the number of DSTs is a variable, following Bayesian non-parametric methods, we instead use a Chinese Restaurant Process (CRP) prior $T \sim CRP(\alpha')$, so that the number of DSTs grows slowly with increasing number of documents [12].

Sampling of the latent structure starts by creating a latent root node, setting its parent $\pi^*_0 = -1$ and sampling its node type t_0 from a DST-specific categorical distribution $Cat(\beta_T)$. The rest of the latent structure is created recursively. For each remaining node i with type t, a child is created (or not) for each unique node type t' by first sampling from a Bernoulli $Ber(\mu_{t,t'}\beta_{T,t'})$ to decide if a child of type t' is to be created, and then, if true, creating a new node j with $\pi^*_j = i$ and $t_j = t'$. The pattern for parent type t and child type t' is captured by $\mu_{t,t'}$, and that for a DST T and a node type t' by $\beta_{T,t'}$. When the number of types is not known in advance, we use the Indian Buffet Process (IBP) $t' \sim IBP(\mu'\beta')$ as the appropriate Bayesian non-parametric distribution [6], so that the number of node types also grows slowly with the number of documents.

For sampling of the content w^* of latent nodes, we make use of the multinomial mixture model with one mixture component for each node type. To make use of precomputed word embeddings, we combine generative mixture models and embeddings [5, 11]. However, instead of using word embeddings, we modify the LFF-DMM model [11] to make use of sequence embeddings, since many text units, such as for 'movie plots', have long sequential content. We first create sequence embeddings of all text units, fine-tuned using BERT [4]. Then the (unnormalized) probability of a text unit content w with embedding $h(w)$ for a node type t with embedding $h(t)$ is defined as the dot product $h(w)h(t)^T$ of the two. The specific content w^*_j of the latent node j is sampled from a categorical distribution over possible text unit contents in the data for the node type t, obtained by applying soft-max on the unnormalized probabilities.

When the data has node locations $l_j = (x_j, y_j)$, we model the location of the j^{th} node as a relative shift (Δ^x_j, Δ^y_j) from the location of its parent node. Viewing the documents as complex tables, the relative shifts are integers. We model the shifts of a node of type t' with parent type t using Poisson distributions: $\Delta^x_j \sim Poi(\tau^x_{tt'})$, $\Delta^y_j \sim Poi(\tau^y_{tt'})$. Spans of nodes are similarly modeled using Poisson distributions $Poi(\gamma^x_{t'})$, $Poi(\gamma^y_{t'})$.

The final step is the sampling of the observed structure conditioned on the latent structure, the DST and node types. The latent nodes are traversed in the topological order of their generation. For the j^{th} latent node of type t, whether it is observed is decided by sampling from a Bernoulli $Ber(\phi_t)$. If it is not observed, then it is skipped and $c_j = -1$. Else, an observed node j' is created with $c_j = j'$. The parent is recorded by setting $\pi_{j'c(\pi^*_j)} = 1$ if π^*_j is observed, or setting $\pi_{j'c(a^*_j)} = 1$, where a^*_j is the nearest ancestor of latent node j which is observed. An observed node j' gets its content w_j and location l_j from its corresponding latent node.

Inference: Now that we have defined the generative model, or equivalently the joint distribution over the observed and the latent variables in a document, we now address the task of inferring the latent structure based on the observed structure, using estimates of the parameters, β, μ, τ, γ and ϕ and the embeddings $h(t)$ for the different node types. We will address the task of parameter estimation after describing our inference algorithm. Specifically, the inference task is to identify, as in the third column of Fig.1, the most likely DST T, latent nodes N^* and parents π^*, node types $t = (t^p, t^s)$, their content w^* and correspondences c between latent and observed nodes, given, as in the second column, the observed nodes N, their parents π, location l and content w. For the inference setting, we assume the number of DSTs and node types to be known. The main challenge in recovering the latent structure is that the number of latent nodes is unknown. Additional latent nodes, which do not have corresponding observed nodes, may need to be introduced into the latent structure.

Algorithm 1: Document Structure Inference

```
   Input: Observed document $N, \pi, w, l$ ; parameters $\beta$, $\mu$, $\tau$, $\phi$, $\{h(t)\}$
1  $N^* = N$, $c_j = j\ \forall j$, $(\pi^*, w^*, l^*) = (\pi, w, l)$, Initialize $t, T$
2  repeat
       // Update node types, parents and DST
3      for $j \in N^*$ do
4          $t_j = \arg\max_t P(t_j = t \mid T, t, \pi^*, w^*_j, l^*_j)$
5          $\pi^*_j = \arg\max_{j'} P(\pi^*_j = j' \mid T, t, l^*)$
6      $T = \arg\max_k P(T = k \mid t, \pi^*)$
7      for each missing header sub-type $t'$ in document: do
8          $L = P(N, \pi, w, l, N^*, T, \pi^*, t, w^*, c)$ // Record likelihood before insertion
9          $N^* = N^* \cup \{n'\}$, $t_{n'} = t', \pi^*_{n'} = 0, c_{n'} = -1$
10         For $j \in N^*$: $\pi^*_j = \arg\max_{j'} P(\pi^*_j = j' \mid T, t, l^*)$
11         $L' = P(N, \pi, w, l, N^*, T, \pi^*, t, w^*, c)$ // Check likelihood after insertion
12         If $L' < L$, revert to earlier $N^*, \pi^*$
13         Else $w^*, l^* \sim P(w^*, l^* \mid T, t, \pi^*; h(t'), \lambda, \tau, \gamma)$
14 until convergence
```

The high-level inference algorithm is shown in Algorithm.1. The inference algorithm first initializes all the latent variables - the latent structure is set to the initial structure, node types are set based on their 'local' content (w) and location (l) features, and the DST is assigned randomly. Then it iteratively updates these variables until convergence. It is essential for the inference algorithm to be iterative since the assignment to the latent variables depend on each other even when the parameters are known. Our algorithm follows the spirit of belief propagation [19], which is known to converge in two passes for polytrees. However, our setting is complicated by the search over new nodes.

The first part of the update, reassigns the node type t and parent π^* for each of the existing current nodes and the document template, based on the *current assignment of all the other variables*. Updates are made using the posterior distributions for these variables given the other assignments. All the posterior distributions are categorical. The node type depends on the content and location, as well as the node types of the parent and all child nodes, and the DST. The parent depends on the DST, the types of all nodes as well as their locations. The DST depends on the types and parents of all nodes. The variables are set to the mode of the posterior distribution. In each iteration, this takes $O(N(N+n_t)+n_T)$ time, where N is the current number of nodes in the document, n_t is the number of node types, and n_T is the number of DSTs.

The second part in each update iteration is the search over new nodes. The algorithm identifies a missing node type in the document, iteratively finds the best position to introduce it into the document by reassigning parents of all nodes, and finally decides to add it if this leads to an improvement in the overall likelihood. In general, this takes $O(n_t N^2)$ time. The structural constraints in our specific setting simplify the search to some extent. Specifically, (a) only nodes with primary type header can be missing and (b) header nodes can only be children of header nodes. When only a few document nodes are missing, the complexity reduces to $O(N^2)$. Some domains may have some additional constraints, such as (c) a header node has at most K children of a specific header node type, and (d) there are at most K' nodes corresponding to a header node type in a document. In our experimental domains, both K and K' were 1. We also found that the algorithm converges in a few iterations in such settings.

Learning: The parameter learning task is to estimate parameters (α, μ, β, τ, γ, ϕ) and the embeddings $h(t)$ for each node type from the training documents. We assume that each training document has the entity and relation annotations over the true structure of the document. For the purpose of training the graphical model for structure recovery, this means that the true nodes N^* and the true parents π^* for these are provided. Additionally, since only header nodes can correspond to entities, the primary nodes types for a subset of nodes are implicitly known to be header. However, the primary types for the remaining nodes are not known. Further, the secondary node type for any of the nodes, as well as the DST for the documents are not known. Since the number of distinct node types and distinct DSTs are also typically difficult to provide as inputs, these are also assumed to be unknown.

We use a hard version of the expectation maximization (EM) algorithm for parameter learning. In each iteration, the E-step performs inference for the latent variables, which are the node types t and the DST T. These are done following the first part of Algo. 1. The only difference is that new types and new DSTs need to be considered, beyond those already encountered in the training data. These are done using the CRP for the DST and the IBP for the node types, both of which reserve a constant probability for unseen types in their prior. The M-step takes maximum a posteriori (MAP) estimates of all the parameters using current assignments to the latent variables. For the embedding $h(t)$ of each node type t, instead of the true MAP, for simplicity we take the average of the embeddings of the text units currently assigned to the specific type t. The iterations continue until there are no changes to the assignments.

4.2 Document Structure Annotation using PLP

At this stage, for each document, we have the output of structure recovery, which includes N^*, π_j^*, t_j and T. We also have the knowledge graph schema with the entities E and binary relations R defined over these. The task is to find the entity annotation $e_j \in E \cup \{-1\}$ for each header node n_j and the relation annotation $r_l \in R \cup \{-1\}$ for each header path p_l, as in Fig.2. The goal is to find the most likely consistent annotation.

The probability of a consistent annotation is determined by the parameters θ_{et}, which is the probability of entity label e for a node of type t, and $\psi_{re_1e_2}$, which is the probability of a relation label r between entity pair (e_1, e_2). The joint probability of a document annotation is the product of the individual entity and relation annotations. All inconsistent annotations have zero probability.

To address this problem, we define a valid joint probability distribution over entity and relation annotations that follow logical rules encoding the constraints. A natural framework for this is probabilistic (first order) logic programming (PLP) [14, 13], which combines first order logic programming such as Prolog, with probabilistic semantics. Specifically, we use ProbLog [2, 13], but other frameworks [1, 15, 16] can be substituted without much difficulty.

A probabilistic logic program is composed of a set of ground facts (or predicates), a set of uncertain facts, and a set of first order logic rules encoding background knowledge over the predicates. These define a distribution over possible worlds of assignments to the remaining (query) predicates. The probabilities can be learnt from evidence facts. We first define our reduction of the entity and relation annotation task to a probabilistic logic program, and then describe the inference and learning algorithms.

Probabilistic Logic Program for Annotation: For document structure annotation, we set the input document header structure as the ground facts: $\text{edge}(n_j, n_{j'}) = 1$ if $\pi_j^* = j'$ or $\pi_{j'}^* = j$, and $\text{type}(t, n_j) = 1$ if $t_j = t$. The uncertain facts are $eLabel(e, t)$ for node with type t having entity label e, and $rLabel(r, e_1, e_2)$ for entity pair e_1, e_2 getting relation label r. These are specified using the parameters that determine the probability of an annotation, using the notion of annotated disjunctions, as follows: θ_{et} :: $eLabel(e, t); \ldots$ and

$\psi_{re_1e_2}$ :: $rLabel(r, e_1, e_2)$; . . . .. The entity and relation annotations form the query predicates eAnnot(e, n_j) and rAnnot(r, p_l).

The rules capture background knowledge about constraints to define the consequences of assignments:

$$\text{eAnnot}(e, n) := \text{type}(t, n), \text{eLabel}(e, t).$$
$$\text{rAnnot}(r, p) := \text{path}(p), \text{eAnnot}(e_1, \mathit{head}(p)), \text{eAnnot}(e_2, \mathit{last}(p)), \text{rLabel}(r, e_1, e_2).$$
$$\text{path}(p) := \mathit{len}(p, 2), \text{edge}(\text{head}(p), \text{last}(p)).$$
$$\text{path}(p) := \text{path}(\mathit{tail}(p)), \text{edge}(\text{head}(p), \text{head}(\text{tail}(p))), \text{eAnnot}(\text{head}(\text{tail}(p)), -1).$$

The first rule defines the probability of an entity annotation of a node n with entity e, in terms of the type t of the node, which is a ground fact, and the entity label e for a node of type t, which is an uncertain fact. Similarly, the second rule defines the probability of a relation annotation of a path p with relation r, in terms of p being a logical path, the probability of entity label e_1 for the head node of p, the probability of entity label e_2 for the last node of p, and the probability of relation label r between entities e_1 and e_2. The final two rules define logical paths. We represent paths as lists, and for notational convenience assume functions head(l), tail(l) and last(l) for lists. The simplest path has length 2 and consists of an edge between the head and the last node. Alternatively, p is a logical path if its tail is a logical path, there is an edge between the head of p and the head of the tail, and the head node of the tail has entity annotation -1.

Inference: The inference task uses known estimates of the probabilities θ (for eLabel (e, t)) and ψ (for rLabel(r, e_1, e_2)) and the ground predicates edge(n, n') and type(t, n) to infer the query predicates eAnnot(e, n) and rAnnot(r, p). To do this in ProbLog, we execute the query rAnnot$(_, _)$ to get the probability of different relation annotations for each path. We annotate each path with the relation which has the highest probability. We also annotate the first and last node of each path with the first and last entity of the selected relation type.

Parameter Learning: The learning task is to estimate the values of parameters θ_{et} and $\psi_{re_1e_2}$ from training data containing entity annotations for header nodes and relation annotations for header paths. This is done by providing to ProbLog as evidence ground facts eAnnot(e, n) and rAnnot(r, p) prepared from training data.

4.3 Entity and Relation Discovery

We finally describe our approach for entity and relation discovery, given the latent header structure and the entity and relation annotations using the existing (incomplete) knowledge graph. Our discovery algorithm is iterative. It identifies as candidates for a new entity, a sets of nodes C^e_t with the same secondary header type t but not yet assigned to any entity. The algorithm takes the largest such set and assigns all those nodes to a new entity id. Next, it looks for new relations between existing entities including the newly discovered one. It identifies as candidates for a new relation, a sets of header-type paths $C^r_{e,e'}$ with the same entities e and e' assigned to the head and last entities but with the intermediate nodes not assigned to any entity. It takes the largest such set and assigns all those paths to a new relation. This continues while the sizes of best candidate entity set and the best candidate relation set are larger than ϵ_e and ϵ_r respectively, which are the entity and relation discovery thresholds.

5 Experiments

We now report experiments for evaluating our proposed model, which we name **PGM-PLP**. We first describe our datasets and baselines for comparison.
Datasets: We use three real datasets of different types.
Web Complex: This dataset is created from the website of a large multinational company. This is our primary dataset as this contains all discussed complexities of semi-structured documents. This has 640 webpages following 6 DSTs. We convert the html of the pages to a DAG structure containing the locations of the text units. The DAG structure of a webpage contains 6-18 nodes. We have also annotated each webpage with the gold-standard structure. Nodes are annotated using 40 node types (20 for headers, 20 for values). The schema has 11 entity types and 13 relation types. Nodes and paths in the gold-standard structure are annotated with entities and relations. A webpage has 3-6 nodes with entity annotations and 2-6 paths with relation annotations. We use 15% of the pages as training, 15% for validation and 70% as test.
SWDE: This is a benchmark webpage dataset for entity annotation [10]. It contains gold annotations of 4-5 predicates of a single entity for 4 verticals, with 10 websites in each vertical and 200-2000 pages for each site. The schema has a star shape, with a central entity and 4-5 secondary entities. We use the same train-test setting as [7]. We converted the DOM tree structure of each webpage to DAG structure with only the text units as nodes. This dataset does not differentiate between observed and true structures, and also does not identify DSTs.
Tab Complex : This is a proprietary dataset containing real tables from the pharmaceuticals industry. The tables are complex with 2-level hierarchy in value cells under some columns. We use this dataset to demonstrate that our solution is not restricted to webpages, but can handle document structure with only spatial locations of text units and no associated html. This has 76 tables of 9 types. The number of tables for each type ranges from 4 to 14. The total number of cells per table ranges from 13 to 8292, corresponding to 36 different cell types. The gold-standard structure is the same as the observed structure. The associated knowledge graph schema has 18 entity types and 26 relation types. Each table contains gold-standard entity and relation annotations. A table has 2-5 entity annotations and 1-6 relation annotations. We use 15% of the data as training, 15% for validation, and remaining for test.

Models: We next describe the models used for comparison.
PGM-PLP: This is our proposed model that recovers latent structure as well as the DST, followed by entity and relation annotation. The model hyper-parameters

Table 1. Entity (E) and relation (R) annotation performance on Web Complex, Tab Complex and SWDE for PGM-PLP, CERES and Limaye. Limaye cannot handle html structure in SWDE.

	Web Complex		Tab Complex		SWDE							
					Movie		NBA Player		University		Book	
	E	R	E	R	E	R	E	R	E	R	E	R
Limaye	0.81	0.61	0.93	0.86	-	-	-	-	-	-	-	-
CERES	0.65	0.28	0.64	0.31	1.00	1.00	1.00	1.00	1.00	1.00	0.95	0.92
PGM-PLP	**0.97**	**0.96**	**0.95**	**0.92**	1.00	1.00	1.00	1.00	0.99	0.99	**1.00**	**1.00**

- the CRP parameters α', the IBP parameters μ', β' and the entity and relation discovery thresholds ϵ_e and ϵ_r - are tuned using the validation sets.

Ceres: CERES[10] is the state-of-the-art on entity information extraction from web-pages. Though it can be trained using distant supervision, for fair comparison we provide it with the same training data as the other models. It does not recover latent structure and also does not consider DSTs for webpages. As no implementation is publicly available, we have used our own implementation. Since CERES can only handle star schemas, we provide it with the best possible star approximation of the schemas. It uses html and text features for annotation. However, since html features are not available for Web Complex and Tab Complex, its entity annotation classifier only uses text-based features.

Limaye: This is a common baseline for annotation of simple tables [8] that uses joint inference using a probabilistic graphical model. It performs comparably with more recent table annotation models [17]. Since semi-structured documents can alternatively be represented as (complex) tables, we use this as the representative of table annotation approaches. Note that this approach does not recognize DSTs or latent structures. Since Limaye, like other table annotation approaches, can only handle simple tables, we approximate our datasets as simple tables as faithfully as possible to create the input for Limaye, as follows. We define a complex table as a tree where each leaf node is a simple table, and each internal node has the structure of a simple table but each cell points to another internal or leaf node. Sibling tables (internal or leaf) do not necessarily have the same dimensions. We simplify such tables in a bottom-up fashion. An internal node is simplified only when all its children are simple tables. We order the cells of an internal (table) node and merge its children sequentially and pairwise. The merge operation for two tables of different dimensions takes all combination of the rows of the two tables, and columns from both.

Annotation Experiments: We first present results for entity and relation annotation in Table 1. We use micro-averaged F1, which is the weighted average of the F1 scores for the different classes (entities or relations). Limaye is defined only for tables and cannot annotate html structures as in SWDE.

We observe that for Web Complex, which has the most complex document structure and knowledge graph schema, PGM-PLP significantly outperforms both Limaye and CERES. This reflects the usefulness of structure and DST recovery, and joint annotation. Limaye performs much better than CERES since it is able to handle arbitrary schemas, while CERES takes in a star-approximation. Unlike Web Complex, Tab Complex does not have any latent structure different from the observed structure. As a result, performance improves for both Limaye and CERES, with CERES still lagging Limaye. However, PGM-PLP is still significantly better than CERES and also outperforms Limaye. The smaller gap between PGP-PLP and Limaye is also because of lower structural complexity in Tab Complex compared to Web Complex. For SWDE, both PGM-PLP and CERES are able to annotate almost perfectly when provided with the same training setting as in [7].

Table 2. Performance for identification of parent (F1), node type (NMI) and DST (NMI) for PGM and its three ablations.

	Parent	Node type	DST
PGM(-T)	0.82	0.73	0.64
PGM(-L)	0.97	0.85	0.88
PGM(-S)	0.78	0.82	0.79
PGM	**0.98**	**0.95**	**0.99**

Table 3. Entity (E) and relation (R) discovery performance (NMI) for PGM-PLP with varying percentage of hidden entities during training.

	Web Complex		Tab Complex	
% Hidden	E	R	E	R
30	0.96	0.95	0.92	0.91
60	0.95	0.95	0.91	0.91
100	0.95	0.94	0.88	0.90

Document Structure Recovery: Next, we report performance for document structure recovery. Specifically, we evaluate the goodness of parent identification of nodes, node type identification (at the secondary level), and DST identification. Note that the first is a binary classification task, so that we evaluate using F1. The second and the third are clustering tasks, with no supervision provided during training. Accordingly, we use normalized mutual information (NMI) over pair-wise clustering decisions for evaluation. The other models do not address this task. Therefore, we compare against ablations of our own model. This task does not involve the PLP stage. So we use ablations of PGM. Ablation **-T** does not use the textual content, **-L** does not use the locations of the text units, and **-S** does not use structural patterns between parent and child types as captured by the parameter β. We report performance only for Web Complex, which has gold standards for all of these aspects of the document structure.

From Table 2, we can see importance of each of these aspects for structure recovery. PGM is able to recover the structure almost perfectly. Not surprisingly, text content has the biggest individual impact, most of all on the DST. Structural pattern has the next biggest impact. Location has the smallest individual impact, but still results in a significant difference in performance.

Entity and Relation Discovery: We next report performance for entity and relation discovery. In this experiment, we hide some entities and all their relations from the training schema and the training annotations. We evaluate how accurately occurrences of these entities and relations are recognized in the test documents. This is again a clustering task, since these entity and relation labels are not seen during training. Accordingly, we evaluate performance using NMI.

Table 4. Ablation study for entity (E) and relation (R) detection on 3 datasets. PGM(-S)-PLP keeps original document structure. PGM-LP uses Prolog instead of ProbLog.

	Web Complex		Tab Complex		SWDE							
					Movie		NBA Player		University		Book	
	E	R	E	R	E	R	E	R	E	R	E	R
PGM(-S)-PLP	0.41	0.11	0.88	0.80	1.00	1.00	1.00	1.00	0.99	0.99	0.99	0.98
PGM-LP	0.21	0.10	0.35	0.20	0.00	0.00	0.31	0.20	0.00	0.00	0.24	0.10
PGM-PLP	0.97	0.96	0.95	0.92	1.00	1.00	1.00	1.00	0.99	0.99	1.00	1.00

Note that the other approaches cannot address this task. So we evaluate only PGM-PLP. Since SWDE has a simple star schema to begin with, we evaluate only for Web Complex and Tab Complex. Table 3 records performance for varying percentages of hidden entities during training. Performance is extremely robust for Web Complex. Even when no initial schema is provided, entities and relations are detected very accurately based on the recovered document structure properties. This is based on the accurate recovery of the secondary header types and clear path patterns between these in the document templates.

Ablation Study for Annotation: Finally, after presenting the usefulness of our overall model, we report performance of different ablations of PGP-PLP for the end task of entity and relation annotation. **PGM(-S)-PLP** performs only partial structure recovery. It identifies node types and DSTs but keeps the original observed structure. Its second stage is the same as in PGM-PLP. **PGM-LP** has the same first stage as PGM-PLP. However, instead of a probabilistic logic program in the second stage, it uses a deterministic logic program, specifically ProLog, with no uncertain facts. In Table 4, we record performance of these two ablated versions along with that of the full model for all three datasets. PGM(-S)-PLP performs the worst for Web Complex because of its structural complexity, where structure recovery has the biggest impact on both entity and relation annotation. In contrast, it almost matches the performance of the full model for SWDE, where the structural complexity is the least. On the other hand, PGM-LP takes a very significant hit across all three datasets. This shows the importance of accounting for uncertainty when annotating the recovered structures, and that of learning the parameters of this model.

6 Conclusions

We have formalized the general problem of annotation of complex semi-structured documents with a knowledge graph schema containing entity and relationship types. We have addressed the problem using a two-stage solution that performs structure recovery using a probabilistic graphical model, followed by joint annotation of nodes and edges of the structure using a probabilistic logic program. We also address the discovery of new entity and relation types not in the schema. We show the usefulness of our model by experiments on real data for both web-pages and complex tables, and we show that we outperform both state-of-the-art table and web-page annotation approaches.

References

1. Bach, S.H., Broecheler, M., Huang, B., Getoor, L.: Hinge-loss markov random fields and probabilistic soft logic. Journal of Machine Learning Research (JMLR) **18**, 1–67 (2017)
2. De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S.E.: Probabilistic Inductive Logic Programming. Springer-Verlag, Berlin Heidelberg (2008)
3. Deng, L., Zhang, S., Balog, K.: Table2vec: Neuralword and entity embeddings for table population and retrieval. In: SIGIR (2019)
4. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: Pre-training of deep bidirectional transformers for language understanding. In: NAACL HLT (2019)
5. Dieng, A.B., Ruiz, F.J.R., Blei, D.M.: Topic modeling in embedding spaces. Transactions of the Association for Computational Linguistics **8**, 439–453 (2020)
6. Griffiths, T.L., Ghahramani, Z.: The indian buffet process: An introduction and review. Journal of Machine Learning Research **12**(32), 11851224 (2011)
7. Gulhane, P., Madaan, A., Mehta, R., Ramamirtham, J., Rastogi, R., Satpal, S., Sengamedu, S., Tengli, A., Tiwari, C.: Web-scale information extraction with vertex. In: ICDE (2011)
8. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. Proc. VLDB Endow. **3**(1-2) (Sep 2010)
9. Lockard, C., Shiralkar, P., Dong, X.L.: OpenCeres: When open information extraction meets the semi-structured web. In: NAACL-HLT (2019)
10. Lockard, C., Dong, X.L., Einolghozati, A., Shiralkar, P.: Ceres: Distantly supervised relation extraction from the semi-structured web. Proc. VLDB Endow. **11**(10), 10841096 (2018)
11. Nguyen, D.Q., Billingsley, R., Du, L., Johnson, M.: Improving topic models with latent feature word representations. Transactions of ACL **3**, 299–313 (2015)
12. Orbanz, P., Teh, Y.: Modern bayesian nonparametrics. In: NIPS Tutorial (2011)
13. Raedt, L.D., Kimmig, A.: Probabilistic programming. In: Tutorial at IJCAI (2015)
14. Raedt, L.D., Poole, D., Kersting, K., Natarajan, S.: Statistical relational artificial intelligence: Logic, probability and computation. In: Tutorial at Neurips (2017)
15. Richardson, M., Domingos, P.: Markov logic networks. Machine Learning **62**(12), 107136 (Feb 2006)
16. Sato, T., Kameya, Y.: Parameter learning of logic programs for symbolic-statistical modeling. Journal of Artificial Intelligence Research **15**, 391454 (2001)
17. Takeoka, K., Oyamada, M., Nakadai, S., Okadome, T.: Meimei: An efficient probabilistic approach for semantically annotating tables. In: AAAI (2019)
18. Wu, S., Hsiao, L., Cheng, X., Hancock, B., Rekatsinas, T., Levis, P., Ré, C.: Fonduer: Knowledge base construction from richly formatted data. In: SIGMOD (2018)
19. Yedidia, J.S., Freeman, W.T., Weiss, Y.: Understanding Belief Propagation and Its Generalizations, p. 239269. Morgan Kaufmann Publishers Inc. (2003)
20. Zhang, S., Balog, K.: Ad hoc table retrieval using semantic similarity. In: WWW (2018)