Vol.2013-DBS-158 No.25
2013/11/26

Inferring Entity Types from Enumerative Descriptions

Qian Chen[†]　岩井原瑞穂[‡]

† 早稲田大学大学院情報生産システム研究科
〒808-0135 福岡県北九州市若松区ひびきの 1-15

E-mail: † chenqian@asagi.waseda.jp, ‡iwaihara@waseda.jp

Abstract Entity class matching has many real world applications, especially in entity clustering, de-duplication and efficient query processing. Current methods to extract entities from text usually disregard horizontal relationships, concentrating on either a prototype-based entity which lacks relationship between two clusters or a terminological entity where only hierarchical relationships are considered. We focus on enumerative descriptions that enlist entity names, together with parent types, often occurring in web documents as listings and tables. We consider discovering entities and relationships from two strongly related enumerative descriptions.

We propose a RDF schema-based algorithm to capture a probabilistic RDF graph from enumerative descriptions by assigning candidate labels to each related token. Our algorithm is iterative: We infer candidate labels from matching sequential patterns and infer patterns that match well with current instances by updating confidence score on labeling of tokens.

Keyword RDF graph, pattern, Hidden Markov Model

1. Introduction

Enumerative descriptions refer to structured descriptions which contain listings of entities, where the order of listings follows a certain unknown order on the class hierarchy, characterized by a regular expression. There are many forms of web resources that contain such enumerative descriptions, such as product descriptions in e-commerce, Infoboxes of Wikipedia, tables coded by Media Wiki and listings in the Web pages which starting with entity names. Infoboxes, tables and listings contain parent types in areas like section titles and adjoining table cells.

As an example of enumerative descriptions, e-commercial real product descriptions are shown in Table 1. In the first line, the Lumsing Battery is compatible with a list of smart devices of different class. Multiple views need to be considered when we make a choice on whether the type hierarchy is a type tree or a type graph (multiple parents). For example, views are different from categories, prices, OS. In this case, type graphs, not trees, are appropriate. We intend to capture a type graph from product descriptions.

RDF[1] (Resource Description Framework) is a general standard for describing such structured descriptions on Web resources and capturing their relationships. In RDF, entities are represented as sets of <subject, predicate, object> or <subject, property, object> triples. In a triple of predicate, both subject and object are entities, while in a triple of property, its subject is an entity, the property is an attribute name, and the object is a value of the attribute. Because of the dynamic property of Web, the flexibility of the RDF model can make it easy for its schema and data to be modified, integrated and linked to other datasets. In the graph model of RDF, subjects and objects are represented as nodes, and predicates and properties are represented as a directed edge from a subject to an object.

After the release of RDF, a large amount of structured data have been converted to RDF and published on the Web and has been widely used in real applications, especially for datasets whose providers update the data frequently, such as DBpedia [2] (the structured data counterpart of Wikipedia) and Freebase[3]. Also, for a specific domain, it is not enough to simply reuse a general-purpose ontology which has limited coverage. Furthermore, polysemous words need to be resolved into appropriate types, reflecting their contexts. For example, the word “Black” can be a color of phones or cases in the title of ID3 in Table 1.

In this paper, we are interested in the problem of

1 http://www.w3.org/RDF/

2 http://dbpedia.org
3 http://www.freebase.com

Table 1 Product descriptions

ID	Website	Title
1	Amazon	Lumsing 11000mAh 5 x USB External Battery Pack Charger Power Bank For Apple:iPad Mini, iPad 4 3 2, Android Tablets:Samsung Galaxy Tab 2, Note 10.1; Google Nexus 7,10; Acer B1; iPhone 5 4S 4 3GS, iPod; Android Smartphone: Samsung Galaxy S4, S3, S2, Ace,Note 2; HTC One, One X, Desire X; LG Optimus 4X HD, I7, I9;Nokia Lumia 920, Google Nexus 4, Blackberry Z10, Sony Xperia Z; MP3, MP4, GPS, Camera, Game Player
2	Amazon	Anker® Astro3E 10000mAh High Capacity Power Bank Pack Portable External Battery Charger for iPhone 5, 4S, 4, iPad 4, 3, 2, Mini, iPods; Samsung Galaxy S4, S3, S2, Note 2; HTC One, EVO, Thunderbolt, Incredible, Droid DNA; Motorola ATRIX, Droid; Google Nexus 4, Nexus 7, Nexus 10; LG Optimus; PS Vita, GoPro
3	Amazon	GTMax Black Touch Screen Styli Stylus For Samsung, Blackberry, HTC, LG, Pantech, Huawei, iPhone, Motorola, Nokia, Smartphone, iPad, Asus, Acer, Toshiba, Archos Tablet
4	Amazon	DandyCase White/Grey Waterproof Case for Apple iPhone 4, 4S, iPod Touch 3, 4, iPhone 3G, 3GS, & Other Smartphones

discovering and annotating entities in description which contains a list of related entities, while taking a different stance from existing approaches. If a list of related entities contains a mix of known and unknown entities, the type of the latter can be inferred from that of the former and vice versa. Previous works can only annotate entities that are listed in the catalogue (the known entities), but are unable to discover new (or unknown) entities. Besides, we take advantage of the strong relationships between two datasets to build a more complete RDF instance graph based on the same RDF schema, which can link two class hierarchies and extract rich relationships among them. For example, smart devices and accessories form two distinct class hierarchies. However, we find that the compatible relationship between them can help us capture more smart devices information from accessory descriptions which contain a list of compatible device models.

Our research question is stated as to find an efficient approach to infer entity types from enumerative descriptions which contain listings of entities. For example, what we want to extract form Table 1 are both product related entities and product compatibility relationships. Our application scenario is to collect partially typed, structured information from web resources, such as e-commerce sites and Wikipedia, and acquire new typing information by our algorithm.

Our goal motivates us to find a new method to facilitate selection of entity types and automatically expanding RDF graphs. Firstly, since not all type names are captured in the initial given graph, so the method should capture existing types and identify new types (classes). Secondly, since not all type names are listed in the product descriptions, the syntactic structure of the description patterns should be captured. We employ a conventional assumption that an enumeration of types is according to a regular order, such as a navigation path in a RDF schema graph. But we need to reconstruct the most likely navigation path from the enumerative description. At last, we need to improve the accuracy and efficiency of inferring such types on entities.

Exacting hierarchical and diverse entities along with implied relationships between them with high accurately is the main goal of this work. With regards to this, we developed an iterative pattern-based algorithm called HoverTyp (Horizontal-Vertical Type Extractor). The central paradigm used by HoverTyp is an iterative framework of starting with an initial labeled RDF graph and given RDF schema which is used to capture description patterns. They are in turn used to classify more instances and patterns from the dataset. At every stage, captured patterns and candidate instances are scored, reflecting their degree of likelihood. When the score of a match sequence S that is a sequence of pattern matches reaches above a threshold, the process terminates and updates confidence scores on entity types.

The rest of the paper is organized as follows: Section 2 discusses related work. Section 3 describes our data representation framework. Section 4 presents each of the proposed techniques in details. The experimental evaluation is presented in Sections 5. Section 6 concludes the paper with future work.

2. Related Works

RDF is a W3C standard to represent metadata, which has several equivalent formats, for example, triple store, column store, property tables, or graphs. Most of existing work exploit RDF to improve the performance of query systems or to extract ontologies.

In existing graph database researches, Zhao and Han [9] designed a neighborhood signature to directly store labels within *k* tops from each vertex. In contrast, our work involves uncertain candidate labels. Several previous works in the semantic web area considered uncertainty in the RDF data. Furthermore, during data integration [1], the integrated RDF data from different sources may often contain conflicting or duplicate information. Therefore, a new probabilistic RDF data graph model has been proposed. Huang and Liu [2] modeled uncertain RDF data by a probabilistic database. However, this model assumes that RDF triples have independent existence probabilities to appear in reality.

Lian and Chen [3] propose a different format of probabilistic RDF graphs. The node labels are not deterministic (multiple labels with probabilities). Probabilistic RDF graphs are related to our research, because RDF graphs extracted from product descriptions are not deterministic, in the sense that there can be different possible labels to tokens. However, our problem involves a different model of probabilistic RDF graphs which have uncertain vertex labels (but certain edge labels). We need to assign possible labels to tokens and calculate likelihoods by simultaneously aligning with given type hierarchies.

After the release of RDF, large amounts of structured data have been converted to RDF and published on the Web, while most of structured descriptions do not contain type triples. TYPifier [5] is a method proposed to infer the type semantics of structured data and build type hierarchy tree. They use complex pseudo-schema features to indicate missing type information and learn type hierarchies by proposed systems. RDF schema features are also used in our approach with different intention. Moreover, type hierarchies in our RDF graph are more complex and detailed.

Ontologies have played a central role in the development of semantic web. RDF Sentence Graph[8] has been used to automatically summarize ontologies which are widely used in understanding unstructured documents. An ontology typically includes concepts and hierarchical relationships. One of the approaches to learning an ontology from unstructured text is using lexico-syntactic patterns. LASER[4] is an iterative process starting with ISA and HASA seed patterns to effectively discover new patterns. However, this method only concentrate on the hierarchical relationships and ignore horizontal compatible relationships. On the other hand, seed patterns in our method are not pre-defined, but partially given in the initial instance graph. Our algorithm tries to infer types to unknown entities based on discovered patterns in enumerative descriptions.

In most of real-world extraction applications, a pattern-based algorithm is used in an iterative process: Starting with a relatively small set of seed tuples, these extractors iteratively learn patterns that can be instantiated to identify new tuples. I4E[6] is proposed as a graph-based framework that integrates tuples, patterns, and various trace information at each iteration. I4E assigns a confidence score to each pattern based on individual tuples that generate the pattern as well as the collective set of tuples produced by the pattern. In our method we also introduce confidence score on every iteration to update the patterns and prune candidate type labels. However, our problem involves a different model of unknown relationships, and enumerative descriptions contain a list of entities with heterogeneous entity types.

3. Architecture and Background

In this paper, we tackle the problem of inferring entities types from enumerative descriptions. This section outlines the architecture of our system and defines assumptions and principles used in our algorithms.

3.1. System Architecture

HoverTyp is an iterative process. The circle in Figure1 is the main components. It takes preprocessed descriptions with each word tokenized as input. In addition, it receives a given RDF schema and initial RDF instance graph. The next is the main stages:

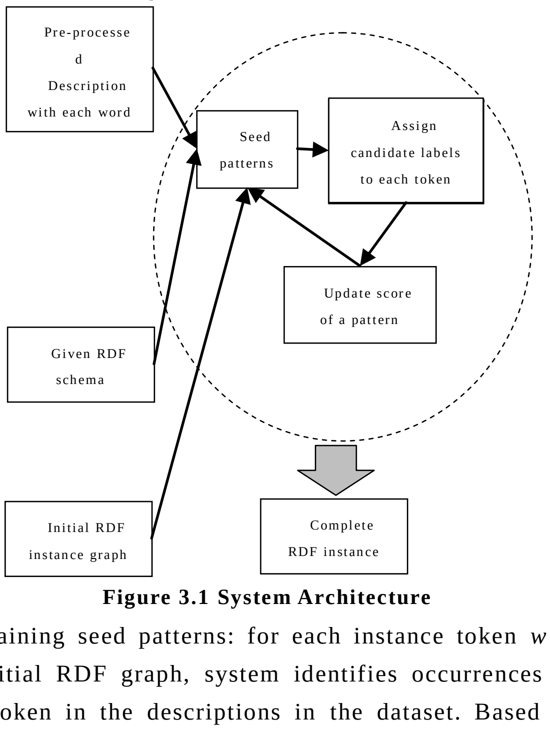

Figure 3.1 System Architecture

Obtaining seed patterns: for each instance token *w* in the initial RDF graph, system identifies occurrences of each token in the descriptions in the dataset. Based on one description in which most of *w* occurs, candidate extraction patterns are generated.

Assigning candidate labels: apply the current set of patterns on each description in the dataset and assign candidate labels on each token in the description based on a Hidden Markov Model (HMM). The candidate type labeled on the token should not conflict with the initial token type. Otherwise, the corresponding pattern cannot be applied on the description.

3.2. Definitions

We utilize the following assumptions and principles in HoverTyp.

Assumptions:

- Basic attributes are appearing as columns in product tables.
- Top-level types, such as model, brand, price, are given in a table.
- An RDF schema for product descriptions is given.
- Relationships connect two classes, but no schema for relationships is given. That is, any two classes can have a relationship.
- A portion of the RDF graph is given as an initial graph. We try to grow this initial graph.

Principles:

- Pattern isolation*: Patterns are not overlapping with each other. Each token in the description must be covered by one and only one pattern.*
- Pattern simplicity: *Patterns needs to be simple for efficient matching.*
- Pattern repetition*: An identical pattern is likely to be repeated within a description.*
- Pattern normalization: *Patterns must satisfy the RDF schema.*

4. Solution and Algorithm

We formalize our problem as augmenting RDF instance graphs through inferring entity types. Intuitively, an entity is represented by a triple in the RDF graph, so what we need to do is to find new vertexes and new edges.

Finding new class types: A title in the product description could enlist multiple types, as shown in Table 1. Since some of these types might be missing in the initial graph, we need to capture such missing types based on the RDF schema. On the other hand, a title may not list up all the class types. So we need to select appropriate class types.

Finding new relationships: Analogous to finding new class types, we need to identify corresponding relationships which link from/to the missing types based on the RDF schema. In this paper, we use description patterns to represent the relationships between entity types.

4.1. Skeleton

We introduce *skeletons* that represent syntactical structures of enumerative descriptions of entities. Skeletons simply capture hierarchal structures of enumerative descriptions. In the next step, we consider mapping between skeletons and the RDF graph. A pattern for capturing entities should be generated from a skeleton and the RDF graph.

For example, below is an external battery pack description with compatible information on smart devices.

PowerGen PGMPP12000 12000mAh External Battery Pack High Capacity Power Bank Charger Triple USB 3Amps output for Apple iPhone 5 4s 4 3Gs 3G, iPod Touch, iPad 1 2 3 4, The New iPad 3/ HTC sensation, XE, XL, One X S V, Thunderbolt, Inspire 4G , EVO 3D, EVO 4G, Desire S Z HD / Samsung Galaxy S3 S2 S 2 II ACE Mini, S Advance, Galaxy Nexus, Nexus 7, Tab / Motorola Atrix 2, Droid 3 X X2 Razr Maxx, Bionic, Triumph

In this description, the delimiters are " " (space), "," (comma) and "/" (slash). We observe that delimiters are hierarchically organized in the ascending order of 1: white space, 2: comma, and 3: slush. From the delimiters, we can construct a tree structure to extract a regular expression.

- We first replace each token with symbol "s". We record the original position of each "s" and the symbol s is not assigned any type. We should note that the proceeding part "… output for" is not considered. The first part "s s s s s s s, s s, s s s s, s s s s /" corresponds to "*Apple iPhone 5 4s 4 3Gs 3G, iPod Touch, iPad 1 2 3 4, The New iPad 3/*".
- We group the lowest level tokens which is separated by spaces and assign symbol "X". The first part "X, X, X, X / X, X, X," corresponds to "*Apple iPhone…XE, XL,*". Here, the first "X" corresponds to the first seven tokens "s s s s s s s".
- Likewise, we group the sequences of X's, delimited by commas, into a symbol "Y":Y / Y
- Finally, we group the sequences of Y's, delimited by slashes, into a symbol "Z".

We obtain the skeleton Z—Y—X—s. The skeleton is not yet mapped to any types.

4.2. Description Patterns

One of the applications of HoverTyp is that for a certain product description, discovering all compatible models which link to this product. So we need to identify which token is class type "Model".

We use description patterns to infer the candidate entity types.

Description patterns: Occurrences of types can be categorized into a collection of patterns which is derived from the RDF schema.

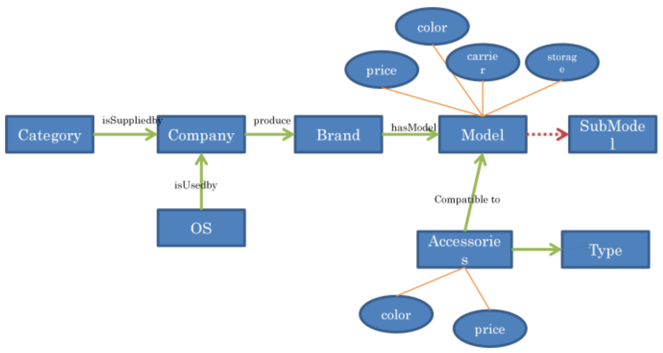

Figure 4.2 Smartdevices RDF schema

Given a seed instance in which each token is marked with correct types in the RDF graph, we fold repetitions and subexpression to obtain a pattern.

- The correct types for the tokens are given as assignment of a type to each symbol 's'.

Skeleton: s s s s s s s, s s, s s s s s, s s s s / s s, s, s, s s s s,

Pattern:c b m m m m m,b m,b m m m m,# # b m/c b,m,m,b m m m,

c: Company b: Brand m: Model, #: non-type tokens ("The New")

- Fold repetitions: If a pattern contains a list of continuous same entity types, we fold these repetitions.

  Pattern: $c\ b\ m^{+}$, $b\ m$, bm^{+}, $b\ m/c\ b$, m, m, bm^{+}

(Note: '+' is one or more repetition. '*' is zero or more.)

- Further fold common subexpressions

  Pattern: $[c[bm^*|m^{+}]^{+}]^{+}$

  (Note: '|' is alternation.)

- We consider assigning types to the skeleton. The skeleton can be translated into a regulation expression by the following mapping:

$$X \rightarrow bm^*|m^{+}$$
$$Y \rightarrow cX^{+}$$
$$Z \rightarrow Y^{+}$$

In the next section, we discuss how to map most plausible regular expressions, based on a probability distribution extracted from the seed instance.

4.3. Constructing a HMM from the seed instance and the patterns

4.3.1. Assigning candidate labels (class names) to each token.

- Values included in the initial RDF graph (seed instances) can be marked up by P(type|value) = 1
- A certain value sometimes has ambiguous meanings. Multiple candidate labels need to be assigned. Eg. "White" can be a color of phones or cases.
- There exist tokens not related to the RDF graph (eg. stop words, adjectives). We need to preprocess descriptions with each word tokenized and decide whether a token is an instance value.
- Tokens like "External Battery Pack" are not used in building the instance graph. But these tokens are characterizing the product itself.

4.3.2. Assigning transition probabilities

Here we construct a Hidden Markov Model (HMM) having states on types. Each edge in the state machine of HMM is labeled with a transition probability on types. From the seed instance, we obtain transition probabilities as follows: We count how many times each subexpression matches, and determine the probabilities on edges originating from the same vertex.

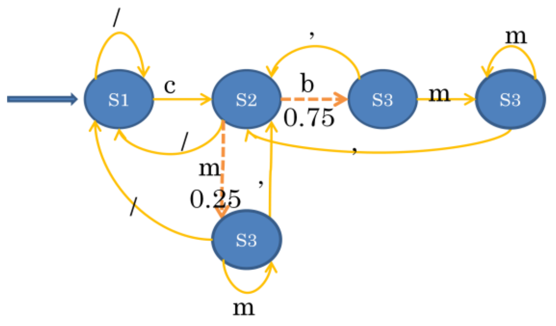

Figure 4.3.1 a pattern automaton with transition probability

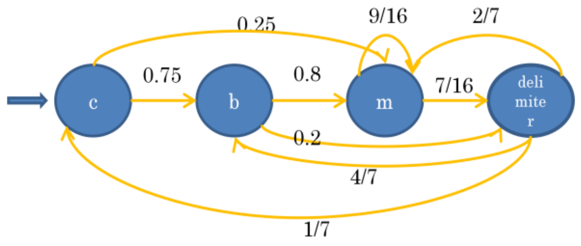

Figure 4.3.2 a pattern HMM with transition probability

For example in Figure 4.3.1, regarding the pattern '[c[bm*|m+]+]+', the subexpression 'bm*' appears six times in the seed instance, and m+ appears twice. Except the alternation '[bm*|m+]', the state machine has only one outgoing edge, so we can assign probability 1 to these edges. But for the alternation '[bm*|m+]', the ratio between the first and second terms is 3:1. From this ratio we assign probabilities 0.75 and 0.25, respectively, to the two outgoing edges corresponding to the alteration. The smaller probability 0.25 represents the irregular case of 'm,m' without 'b'.

We use this pattern automaton to construct a pattern HMM (Figure 4.3.2) with transition probability by count the proportion in the seed instances.

4.4. Assigning types to unknowns

The target instance follows the hierarchical syntactic structure represented by the seed pattern. But delimiters can be different among different seed instances, because descriptions may use different notations depending on their origins. For example, only in the title of ID 1 in Table 1, colon ":" is used.

Now discuss assigning types to unknown tokens. For example: "*Samsung:c Galaxy:b 3:x,4:x*". Here, 'Samsung:c' means that the type of 'Samsung' is 'c' and the symbol 'x' means an unknown type. We need to find a type for each occurrence of 'x'.

We use Viterbi algorithm [7] to find the most likely state sequence for a given description sequence and a HMM. It enumerates all possible state sequences and chooses one that maximizes the score. We use the optimal state sequence to infer the entity types.

The input of the algorithm is:

1) A HMM with state space S = {x_1,…,x_k}, the initial probabilities π_i of being start at state *i* and transitional probabilities a(*i, j*) of transitioning from state *i* to state *j*.

The HMM is constructed from the seed instance and pattern. The initial probabilities π_i can be extracted from the HMM as follows: If the seed instance always starts from "Company", then π_i = 1 for Company and *π_j* = 0 for (*j* != *i*). If there are multiple initial possible types, their frequencies shall be reflected on to their probabilities. Transitional probabilities are also from the HMM through checking how many times each edges appear in the seed instance.

2) Observations {y_1, …,y_T}. In our model, it corresponds to a product description, like "*Apple iPhone 3/3s/4/4s, iPad 2/mini*". Some tokens are assigned types by the seed instance, and the reminders are of unknown types.

3) Emission probabilities P(y_t|*k*). The probability of y_t is observed in state *k*. We can calculate conditional probabilities P(k|y_t) of the opposite direction. Namely, the possibility of in state k when y_t is observed. We apply Bayes' Theorem: P(y_t|k) = P(k|y_t)P(y_t)/P(k) for emission probabilities.

For example, we need to calculate P(Apple|Company) for emission probability. Since "Apple" is in the seed instance, we know that "Apple" type is "Company". This means that P(Company|Apple) = 1.

We are approximating the distribution of the target instance by the distribution of the seed instance, since we do not know the exact distribution of the target. P(Apple) is calculated from how much percentage Apple occupies in the seed instance. Likewise, P(Company) is calculated from how much percentage Company occurs in the seed instance. Then we can obtain the emission probability of P(Apple|Company).

There are tokens of unknown types in the target instance. We need to estimate emission probabilities of unassigned types. We also approximate the distribution of the target by the distribution of the seed instance. We assume that the state probabilities P(k) are identical between the target instance and the seed instance.

For example, the description "*HTC sensation, XE, XL, One X S V*" has eight tokens. Each of the token probability P(y_t) is calculated as 1/8 based on the target instance. We may merge target and seed instances to create a larger frequency distribution. Suppose that in the seed instance, c(Company), b(Brand) and m(Model) occur by the ratio: (0.2, 0,3, 0.5). We employ this as the state probabilities P(k) for the target instance. Now we compute the emission probabilities P(y_t|k) by Bayes' theorem. Here, if we have no knowledge about the description, P(k|y_t) = P(k) and P(y_t|k) = P(y_t). Such as P(One|Company)=P(One)= 1/8. But if we have partial knowledge, such as "HTC" is a company, then we can construct a better belief on P(k|yt). We know that P(c|HTC) = 1, P(c) = 0.2 and P(HTC) = 1/8, then we can calculate the P(HTC|c) = 0.625. Furthermore, we can know that P(b|HTC)=P(m|HTC)=0 and P(HTC|b)= P(HTC|m)=0.

The Viterbi algorithm solves the problem of finding the most optimal state sequence for a given sequence. It uses dynamic programming to solve the following recurrence equations:

$$\begin{aligned} V_{1,k} &= \mathrm{P}\big(y_1 \mid k\big) \cdot \pi_k \\ V_{t,k} &= \mathrm{P}\big(y_t \mid k\big) \cdot \max_{x \in S}\left(a_{x,k} \cdot V_{t-1,x}\right) \end{aligned}$$

Here, $V_{t,k}$ is the probability of the most probable state sequence responsible for the first *t* observations that has *k* as its final state.

4.5. EM algorithm to estimate parameters

We should repeat 4.3 and 4.4 above to improve the overall score, until convergence. We can employ the following approach:

- Expectation-maximization algorithm (EM)
- From the result of 4, update scores (confidence) on labeling of tokens.

5. Conclusion

In this paper, we proposed an iterative process for the entity type resolution on enumerative descriptions. We use a given RDF seed instance to generate a skeleton and a HMM on types. Then, we introduce confidence score on pattern to return possible (entity, class type) pairs as a feedback. Augmented RDF instance graphs have various real world applications. Especially they can be used to improve the result in both the clustering and de-duplication problems, and improving quality of query results.

As future work, we aim to conduct performance evaluation of the proposed method, and study application to various types of partially-typed web resources.

References

[1] X. L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. VLDBJ, 18(2), 2009

[2] H. Huang, C. Liu. Query evaluation on probabilistic RDF databases. In WISE, 2009.

[3] X. Lian and L. Chen. Efficient query answering in probabilistic RDF graphs. SIGMOD'11, 2011.

[4] T.Y. Li, P. Chubak, L.V.S. Lakshmanan. Efficient extraction of ontologies from domain specific text corpora. CIKM'12, 2012.

[5] Y. Ma, T. Tra, V. Bicer. TYPifier: inferring the type semantics of structured data. ICDE '13, 2013.

[6] A.D. Sarma, A.Jain, D. Srivastava. I4E: interactive investigation of iterative information extraction. SIGMOD'10, 2010.

[7] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," IEEE Trans. Inform. Theory,IT-13:260-269, Apr 1967.

[8] X. Zhang, G. Cheng, Y.Z.Qu. Ontology summarization based on RDF sentence graph. In IW3C2, 2007.

[9] P. Zhao and J. Han. On graph query optimization in large networks. PVLDB, 3(1), 2010.