Towards the Completion of a Domain-Specific Knowledge Base with Emerging Query Terms

Sihang Jiang[†], Jiaqing Liang[†¶], Yanghua Xiao[†§*], Haihong Tang[‡], Haikuan Huang[‡], Jun Tan[‡]
[†]*Shanghai Key Laboratory of Data Science, School of Computer Science, Fudan University, China*
[‡]*Alibaba Group, Zhejiang, China*
[§]*Shanghai Institute of Intelligent Electronics & Systems, Shanghai, China*
[¶]*Shuyan Technology, Shanghai, China*
{*tedsihangjiang, l.j.q.light*}*@gmail.com, shawyh@fudan.edu.cn,*
piaoxue@taobao.com, {*haikuan.hhk, tanjun.tj*}*@alibaba-inc.com*

***Abstract*—Domain-specific knowledge bases play an increasingly important role in a variety of real applications. In this paper, we use the product knowledge base in the largest Chinese e-commerce platform, Taobao, as an example to investigate a completion procedure of a domain-specific knowledge base. We argue that the domain-specific knowledge bases tend to be incomplete, and are oblivious to their incompleteness, without a continuous completion procedure in place. The key component of this completion procedure is the classification of emerging query terms into corresponding properties of categories in existing taxonomy. Our proposal is that we use query logs to complete the product knowledge base of Taobao. However, the query driven completion usually faces many challenges including distinguishing the fine-grained semantic of unrecognized terms, handling the sparse data and so on. We propose a graph based solution to overcome these challenges. We first construct a lot of positive evidence to establish the semantical similarity between terms, and then run a shortest path or alternatively a random walk on the similarity graph under a set of constraints derived from a set of negative evidence to find the best candidate property for emerging query terms. We finally conduct extensive experiments on real data of Taobao and a subset of CN-DBpedia. The results show that our solution classifies emerging query terms with a good performance. Our solution is already deployed in Taobao, helping it find nearly 7 million new values for properties. The complete product knowledge base significantly improves the ratio of recognized queries and recognized terms by more than 25% and 32%, respectively.**

***Keywords*-Product Knowledge Base; E-commerce; Knowledge Base Completion;**

I. Introduction

Knowledge bases (KBs) play an increasingly important role in many real applications. Great efforts have been devoted to the construction of KB, such as DBpedia [1], CN-DBpedia [2], Yago [3], Probase [4] and ProbasePlus [5], due to KBs containing a great amount of machine-readable knowledge. These KBs have been extensively used in query understanding, recommendation and question answering. Although most of the large scale KBs are open domain, there has been a recent trend to build *domain-specific knowledge bases* (DKBs). A DKB usually has a higher coverage over domain-specific facts, making them a more feasible option for domain-specific applications. Many DKBs thus have been built to enable smart domain applications, such as MusicbrainZ [6] which is a music KB used for music recommendation, PubMed [7] which is a medical KB for biomedical literature searching and GeoNames[1] which is a geography KB providing positioning service.

In this paper, we focus on DKBs serving for entity search, such as Taobao's product KB or movie domain KB, which are used for user's intention understanding. More specifically, we mainly use the product KB of Taobao (the largest Chinese C2C platform) as an example to show the usage and incompleteness of a DKB and to study *how to complete a DKB*.

A. Problem Statement

The product KB in Taobao is in the form of CPV (Category-Property-Value) triples. For convenience, we mainly use CPV triples to represent the triples in a DKB, while it is exactly the same as a SPO (Subject-Property-Object) triple. A *"category"* is a collection of items or products. Every category is then associated with several *"properties"* (or predicates), and every property is further associated with several optional *"values"*. For example, Figure 1 shows an example of CPV KB, where "Dress" is a category, "length of sleeves" and "shape of sleeves" are the two properties of the category, and "lotus leaf sleeves" and "lantern sleeves" are two values of the "shape of sleeves" property. Every item in Taobao belongs to one of the leaf categories and is described by specifying several property-value pairs.

The CPV KB has been widely used in a lot of tasks in Taobao. For example, in query understanding, as a user searches for "red dress with lantern sleeves", the CPV KB helps recognize the category "Dress" as well as corresponding property-value pairs such as (color, red) and (shape of sleeves, lantern sleeves). In this way, the users' intention is

*Yanghua Xiao is the corresponding author. This paper was supported by National Key R&D Program of China (No. 2017YFC0803700), by National NSFC (No.61732004) and by Shanghai Municipal Science and Technology project (No.16JC1420400).

[1]http://www.geonames.org/

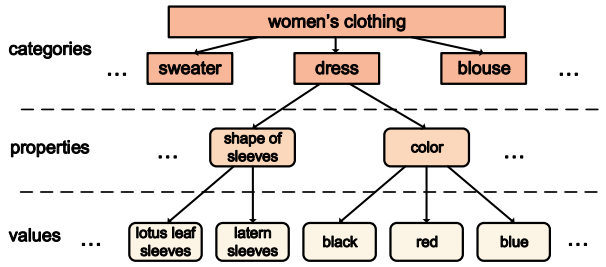

Figure 1. A snapshot of product KB in Taobao. Categories-properties-values are the building blocks of the product KB.

understood, which enables Taobao to recommend the most relevant items in the category by matching such property-value requirements.

However, the DKBs are always *incomplete*, which significantly hurts their applications. There are three major reasons. *Firstly, values relevant to user experience are usually missing.* The first version of DKB was constructed for the convenience of item management. It usually had a quite limited coverage over the values reflecting user experience and thus can hardly support query understanding. For example, in order to distinguish different kinds of paintings on clothes, there exist coarse-grained values under the "pattern" property, such as "human face", "animals" and "geometry paintings". However, instead of searching for a coarse-grained value like "animals", the users are more likely to search for clothes with a more specific pattern, such as "dogs". Without understanding "dogs" is a fine-grained value under the "pattern" property, the recommendation system returns the clothes with patterns "hot dogs" as a result according to string similarity. *Secondly, a manually built DKB usually has a limited coverage over non-typical values.* A DKB usually is manually built by domain experts. The manual construction ensures the precision of KB, but human experts tend to focus on the *typical values* used by shoppers. For example, there are a huge number of brands for dresses in Taobao product KB. The domain experts can only include the most typical brands into the properties section. *Thirdly, there are many newly emerging values.* For example, Donald Trump's supporters may search for T-shirts with the keywords "Donald Trump portrait" during the presidential campaign. Since "Donald Trump portrait" is a new value and is absent in CPV KB, it is difficult to find relevant product suggestions for the queries.

Our statistical study further verifies the incompleteness of DKBs. The current product KB of Taobao contains more than thirty thousand categories. However Taobao only recognizes 30.9% queries, and the remaining 69.1% queries contain unrecognized terms. Here we consider the query terms not in CPV KB as unrecognized terms. We denote the percentage of queries containing unrecognized terms as $percentage_{qu}$ and the percentage of terms unrecognized as $percentage_{ut}$. We show these percentages in three different typical categories of Taobao in Table I. The incompleteness

Table I
PERCENTAGE OF QUERIES CONTAINING UNRECOGNIZED TERMS ($percentage_{qu} = \frac{\#\text{QUERIES CONTAINING UNRECOGNIZED TERMS}}{\#\text{ALL QUERIES}}$) AND PERCENTAGE OF TERMS UNRECOGNIZED ($percentage_{ut} = \frac{\#\text{UNRECOGNIZED TERMS}}{\#\text{ALL QUERY TERMS}}$)

Category	$percentage_{qu}$	$percentage_{ut}$
Dress	51.4%	62.5%
T-shirts	51.5%	72.1%
Shoes	48.6%	66.0%

of product KB causes great difficulty in recognizing the missing values that are used in real queries. As a result, the platform has a limited capability in understanding queries and user's intention.

The direct sources that provide new values are query logs. There are also two reasons for using query logs. *Firstly, the terms mentioned in query logs are fresh*. The emerging terms in query logs allow us to construct a fresh KB. *Secondly, using query logs enforces the KB to reflect the user experience.* There is a significant gap between the product KB that is constructed by the administrator and the one required by consumers. Closing the gap and completing the product KB with more dimensions of user experience are critical for building a KB that is powerful for understanding search intent of consumers.

In this paper, we propose to complete a DKB with emerging query terms. The completion is critical for ensuring the *completeness* and *freshness* of a DKB. For product KB in Taobao, the completion procedure aims at increasing the coverage of product KB by supplementing new values for existing properties. The completion problem is reduced to classification of emerging query terms into existing properties for each category. The categories and properties in DKB in general have relatively low update frequency compared to values, because the schema of a DKB seldom changes. Hence in this paper, we focus on the completion of DKB with new *values*, rather than categories and properties.

B. Previous Solution

A lot of solutions have been proposed for KB completion. In general, these approaches can be roughly divided into two categories, *supervised approaches* and *unsupervised approaches*. The supervised approaches use labeled data to train a classifier and predict the properties for each new value; and the unsupervised approaches use great amount of textual information to mine patterns or to build a k-NN classifier with semantic representations. However, all of the previous approaches in general are not suitable for our problem.

Weakness of supervised approaches: Our problem can be viewed as a relation extraction problem, where we take a new value and a category as inputs and predict the property (or relation) between them. Great efforts have been devoted to relation extraction and many supervised approaches have been proposed. However, the supervised approaches require a large amount of labeled data, especially deep learning

based approaches [9], [10], [11], while it is hard for a specific domain to collect such large amount of labeled data. The product KB contains millions of properties where 66% properties contain less than 20 values, which makes it difficult to train an effective classifier for so many labels using supervised models. Therefore, an effective unsupervised approach is expected for our problem.

Weakness of existing unsupervised approaches: Existing unsupervised approaches use great amount of textual information to mine patterns or to build a k-NN classifier with semantic representations [12], [13]. However, these approaches are not suitable for our problem. It is hard to collect large amount of external information in a specific domain. Also, most queries consist of disordered terms, which makes it hard to mine good patterns or learn good representations. For example, the distributions of topics for words in LDA [14], [15] can be used as semantic representations to build a k-NN classifier. Due to the frequency-based probabilistic modeling, LDA based methods cannot capture deep semantic information from text. Two values with the same LDA topic do not necessarily belong to the same property. For example, "lace" and "hollow-carved" are two terms under the same topic "sexy", since people tend to use "lace" and "hollow-carved" as keywords when searching for sexy clothes. However, "lace" is a value of "fabric" property, whereas "hollow-carved" is a value of the "design details" property.

C. Our Idea and Contribution

We use query logs to complete the DKB. We propose a graph based solution. Our solution accepts emerging query terms and the existing KB as inputs, and finds the top candidate properties for all the emerging query terms. The key is evaluating the relevance between an emerging query term and properties in KB, which is further reduced to the evaluation of the relevance between an emerging query term and existing values of a property. The basic idea of our solution is first collecting all positive and negative evidence about whether two terms belong to the same property or not (see Section IV). Then, we use positive evidence to build a term similarity graph, and use negative evidence as constraints (see Section V-A). Finally, we propose two alternative graph exploration mechanisms: *shortest path exploration* and *random walk*, under the constraints derived from negative evidence, to evaluate the relevance between an emerging query term and existing values of properties (see Section V-B). It is worth mentioning that, in order to ensure the quality of the KB, all the results still require manual verification. We therefore select the top-k most likely properties for each emerging query term as the output. This will not increase the labor cost and will help improve the effectiveness of our model, allowing more recommended properties to be checked manually.

Our contribution is summarized as follows:

- We propose the problem to complete a DKB with emerging query terms and elaborate its challenges. As far as we know, this is the first effort in completing a large-scale real-world DKB with emerging query terms.
- We propose a graph based solution to solve the problem. The solution is unsupervised, requiring no labeled data. Our solution is interpretable by providing the paths for tracing, which is critical for its real application.
- We conduct extensive experiments to show the effectiveness of the proposed solution on real data. The solution is deployed in Taobao, finding nearly 7 million new triples for the Taobao product KB. The new KB improves the ratio of recognized queries and recognized terms by more than 25% and 32%, respectively.

The rest of this paper is organized as follows. In Section II, we review the related work. In Section III, we formalize the problem and give an overview of our proposed solution. In Section IV, we elaborate the external evidence we use, and in Section V, we introduce our main methods and analyze the time complexity. In Section VI, we describe our experimental settings and results. In Section VII, we show the deployment of our method in Taobao, and finally we conclude this work in Section VIII.

II. Related Work

Our research is related to KB completion and knowledge guided algorithms.

KB completion: KB completion methods can be roughly divided into two categories: internal methods and external methods [16]. Internal methods only use knowledge in KB. These methods focus on finding missing relations or types for the existing values or subjects in KB[17], [18], [19]. However finding new values for a DKB is more useful in real applications and more challenging. So we pay more attention on new values rather than existing values in a DKB, and we thus have to rely on external information. External methods use external information to achieve a good performance. The external information consists of two parts, structured information and unstructured information. Structured information such as HTML tags and tables can be viewed as a kind of patterns, by which the missing relations or entity types can be found [20], [12], [13]. However, structured information is not necessarily available in specific domains. For example, in our problem setting, there are only short text without any extra tags. Plain texts (i.e., unstructured information) instead are widely available in most domains, which motivates many works to use textual information to learn extraction patterns [21], [22], [23]. Some other works directly find a latent representation for entities or values from texts and build a corresponding classifier for prediction [9], [10], [11]. However, these works require large amount of textual information and it is hard for specific domains to collect a large amount of textual information. Biperpedia [24] uses both KB and query logs as input for KB completion. However, Biperpedia focuses on finding attributes, while our method aims at finding new values. It is worth mentioning that, entity resolution

Table II
NOTATIONS

Notations	Meanings	Notations	Meanings
C	the set of all categories in CPV KB	c	one category that $c \in C$
P	the set of all properties belonging to c	p_i	the i-th property that $p_i \in P$
p	one property	V_i	the set of values belonging to p_i
V	the set all existing values in c, $V = \bigcup V_i$	U	the set of emerging query terms
T	the set of all terms, $T = V \cup U$	t	an emerging query term
v	a existing value	x, y	a term

Notations	Meanings
pe_{ed}	positive evidence measured by edit distance
pe_{es}	positive evidence measured by embedding similarity
pe_{ia}	positive evidence measured by using indicative affixes
ne_{co}	negative evidence measured by using frequency of co-occurrence
ne_{cate}	negative evidence measured by using distribution of categories
ne_{pos}	negative evidence measured by using distribution of Part-of-Speech tags
ne_{st}	negative evidence measured by using distribution of semantic tags

(ER) [25] is a related task of KB completion. ER focuses on identifying same entities in different descriptions, but our method aims at finding new values.

Knowledge guided algorithms: A few works use knowledge in form of positive and negative evidence to guide training. Lee et al. [26] tried to identify the same entities from two taxonomies by building a similarity graph. Then, the authors used negative evidence to split the graph into several components. However, this work focused on mapping one taxonomy to another, and we focus on DKB completion. LDA [14] is a probabilistic topic model, which aims at inferring the probability of belonging to each topic for documents and words. By imposing Dirichlet prior, it outperforms other traditional topic models on extracting semantic information, e.g. PLSI [27]. Andrzejewski et al. [15] encoded both positive evidence and negative evidence, which they refer to as Must-Links and Cannot-Links respectively, into a LDA model by extending the *Dirichlet prior* to *Dirichlet Forest prior*. In addition to this, the Must-Links and Cannot-Links must be completely correct, but external evidence always contains errors without manual labeling, which requires the model to tolerate the noisy evidence.

III. SOLUTION FRAMEWORK

In this section, we first formulate our problem and then present the framework of our solution.

Problem formulation: Our goal is to classify each emerging query term into a corresponding property. Considering that (1) most of emerging query terms are missing values belonging to a certain property in CPV KB and (2) classification results require manual verification, we directly use emerging query terms as candidates and try to classify them into corresponding properties. As we consider all existing properties as candidate properties, we give each emerging query term in KB a score to quantify how likely it belongs to a given property. Given a category c and an emerging query term t, let P be the set of properties of c. Our goal is to find the property $p \in P$ with the highest score:

$$\arg\max_{p \in P} score(p, t) \tag{1}$$

, where $score(p, t)$ measures the likelihood that t belongs to p. Since the results will go through manual verification, in the real implementation we will return the top-k properties $p \in P$ with the highest score $score(p, t)$. We give the notations used in this paper in Table II.

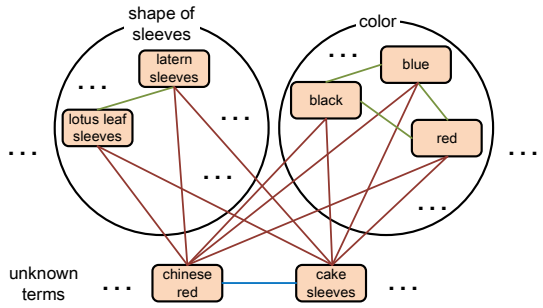

Figure 2. An example of a term similarity graph. Each circle represents a property while the terms in the circle are the values under this property.

Solution Framework: Next we outline the framework of our method, which is illustrated in Algorithm 1. There are three major steps. In the first step, for all terms (those existing values in the KB as well as any emerging query terms), we collect positive evidence and negative evidence to measure whether two terms belong to the same property. In the second step, we use positive evidence to build a term similarity graph, where each edge represents that the connected terms belong to the same property (a sample of this term similarity graph is shown in Figure 2). Then we use the negative evidence to identify term pairs that are not likely to belong to the same property. In the third step, we employ two alternative graph exploration mechanisms (the shortest path and random walk) under the constraint that the edges rejected by negative evidence are forbidden to go. The exploration helps calculate the likelihood score that an emerging query term belongs to a given property. Finally, we recommend the proper properties for each emerging query term according to Equation 1.

Algorithm 1 Framework of our solution.

Input:
A category c;
All the properties P belonging to c;
All the values V belonging to c;
All the emerging query terms U;
All the query logs Q;
Output:
Step 1: Collect positive evidence PE and negative evidence NE for all the terms $T = U \cup V$;
Step 2: Generate a term similarity graph G using positive evidence PE;
Step 3: Run a shortest path (or random walk) exploration on graph G under constraints specified in NE;
For each emerging query term $t \in U$, find $\hat{p}(t) = \arg\max_p score(p, t)$;
return $\hat{p}(t)$;

Alternatively, we can build the term similarity graph for multiple categories even all categories. However, this solu-

tion has two weakness. First, terms have different meanings in different categories and some meaningless edges are introduced to our method. For example, term "large" in air conditioner category is obviously different from term "large" in dress category. Second, a graph built for a single category is smaller, alleviating the difficulty to handle big graph. Hence we apply our method on each category.

In Section IV-A and Section IV-B, we elaborate the positive and negative evidence respectively. In Section V-A, we then introduce how we build the term similarity graph G, in Section V-B we describe the scoring mechanism based on a shortest path exploration and random walk exploration, and in Section V-C we introduce the time complexity of our methods.

IV. External Evidence

In general, to determine whether an emerging query term should be classified into an appropriate property, we need the positive evidence (or negative evidence) that supports (or rejects) the same membership between the emerging query term and the existing values of the property. Given two terms x and y, any evidence that supports x and y belonging to the same property is called *positive evidence*, and any evidence that rejects x and y belonging to a same property is called *negative evidence*. We rely on positive evidence to determine whether an emerging query term belongs to an existing property. However, positive evidence usually has errors. We still need negative evidence to reconcile the side effect of unreliable positive evidence. It is worth mentioning that, most of the following evidence are language independent and thus are useful in different languages.

A. Positive Evidence

In general, the more similar two terms are, the more probable the two terms belong to the same property. Hence, we use similarity as a kind of positive evidence. Next we will first propose three individual similarity metrics and then aggregate them to drive a stronger positive evidence. We focus on the similarity between an emerging query term $x \in U$ and any other term $y \in U \cup V$.

Edit Distance: First, we use Levenshtein distance to measure the string similarity between two terms. Levenshtein distance is a kind of edit distance, which allows removal, insertion or substitution of a character in string. It quantifies the *dissimilarity* of two strings. In order to get a normalized similarity score for two terms, we define pe_{ed} as follows:

$$pe_{ed}(x,y) = 1 - \frac{EdtDis(x,y)}{\max(len(x), len(y))} \tag{2}$$

, where $EdtDis$ is the Levenshtein distance, and $len(x)$ measures the string length of term x.

Embedding based Similarity: Word embedding is a continuous word representation, which captures the semantics of words in a low dimensional vector space. We use all the titles of clicked items as documents, and use word2vec [28], [29], [30] model to train the word vector for each term in the titles. We only use the title to train the word vector because titles in general are much cleaner than other free-text such as product description or comments. We use cosine similarity to measure embedding similarity and then normalize the cosine similarity into the range of $[0, 1]$ by the following equation:

$$pe_{es}(x,y) = \frac{1 + cosine(v_x, v_y)}{2} \tag{3}$$

, where v_x, v_y are the word vectors of x and y, respectively, and $cosine(v_x, v_y)$ is the cosine similarity between v_x and v_y.

Affix based Similarity: We observe that many terms have *indicative affixes*. Note that a term with an indicative affix is usually related to a particular property. For example, the affix "sleeves" of term "lotus leaf sleeves" strongly suggests that the term belongs to the property "shape of sleeves". For an arbitrary term pair (x, y), there are two possible cases to address.

1) When x and y are both emerging query terms, that is $x, y \in U$, the same affix of x and y implies that they are more likely to belong to the same property. We define sim_1 as follows:

$$sim_1(x,y) = \frac{len(afx(x,y))}{\max(len(x), len(y))} \tag{4}$$

, where afx finds the same indicative affix between x and y, afx returns an empty string when x and y don't share the same indicative affix.

2) When x is an emerging query term, and y is an existing value, that is $x \in U$ and $y \in V$, we find that some x shares affix with property p that y belong to. For example, a "style" affix has a strong probability of being related to the "style" property in the clothes related categories. Thus x shares similarity with all values $v \in p$, when x shares the same affix with p. For example, "Japanese style" deserves to have a high similarity score with all values in property "style", even when the value has no affix "style" (such as "kawaii"). We define sim_2 to reflect this observation:

$$sim_2(x,y) = \frac{len(afx(x))}{len(x)} \delta(p_a(afx(x)), y) \tag{5}$$

, where afx returns the affix in x, $p_a(a)$ returns the property of highest association with indicative affix a, and $\delta(p, y)$ is an indicator function which indicates whether $y \in p$.

Put together, the affix based similarity score pe_{ia} is defined as follows:

$$pe_{ia}(x,y) = \begin{cases} sim_1(x,y) & x \in U \text{ and } y \in U \\ sim_2(x,y) & x \in U \text{ and } y \in V \end{cases} \tag{6}$$

We still need an algorithmic solution to find an affix in a term (i.e., to compute $afx(x)$), to find a common affix for a pair of terms (i.e., to compute $afx(x,y)$) and to find the most associative property for an affix (i.e., to compute

$p_a(a)$). Specifically, we first enumerate all the 1-grams and 2-grams in all queries for the category, and collect the high frequency ones as affixes and look for the longest affix in a term x as $afx(x)$. For any two terms x, y, their longest common affix is the result of $afx(x, y)$. Then, for every affix a and a property p, we consider the frequency of values in P that contains an affix a as a TF (term frequency) and consider the number of properties that have a value with a as affix to be IDF (inverse document frequency). Finally, we get a tf-idf score for (a, p). The property with the highest tf-idf score is $p_a(a)$.

Aggregated Positive Evidence: Now, we have positive evidence from three different principles, and each positive evidence is in the range of $[0, 1]$. Considering the sparsity of the data, we use noisy-or model to strengthen all the evidence, and map all scores into one score in the range of $[0, 1]$:

$$PE(x, y) = 1-(1-pe_{ed}(x, y))(1-pe_{es}(x, y))(1-pe_{ia}(x, y))$$

Intuitively, $PE(x, y)$ has a high score as long as at least one source of evidence gives a high score, and PE increases when more evidence is available. For example, "Japanese style" and "kawaii" share low string similarity, but affix based evidence sufficiently supports the same membership between them. This leads to a high PE score between "Japanese style" and "kawaii".

B. Negative Evidence

The positive evidence might introduce false positives. String similarity is only a syntactic similarity and the word embedding based similarity only considers the context of each term. Negative evidence is also necessary to prevent the noise produced by the positive evidence. We collect negative evidence from query logs. Similarly, each negative evidence is also in the range of $[0, 1]$, which quantifies the likelihood that two terms belong to two different properties. The larger the score is, the more likely the two terms belong to two different properties. Furthermore, we take negative evidence from all sources available into consideration to avoid the wrong rejection of positive evidence.

Principle 1 (Co-occurrence): A pair of terms that frequently co-occur in the same queries are quite likely to belong to two different properties.

Co-occurrence in queries: As claimed in Principle 1, the terms that belong to the same property are less likely to co-occur in the same query. The ultimate objective of the user is to find the target product with minimal cost. This objective implies that when a user drafts a query he or she will use the least amount of terms. Another implication is that the user will not use terms that are semantically conflicting with each other in a query. For example, a user will not likely specify a query like "women dress red white", because "red" and "white" in this context are conflicting with each other and therefore confuses the search engine. As a result, a real user tends to use only one word to describe each aspect of the target item. In other words, the terms co-occurring in the same query are less likely to belong to the same property. We define ne_{co} as follows:

$$ne_{co}(x, y) = 1 - e^{-co(x,y)} \tag{7}$$

where co is the frequency that x and y co-occur in the same query.

Principle 2 (Distribution of Categories): A pair of terms that have different distributions of categories are more likely to belong to two different properties.

Distribution of categories: Each query has its corresponding intended search results, which is usually embodied in the categories of the clicked items. It is obvious that the terms that belong to the same property should occur in queries searching for similar categories. For example, "lotus leaf sleeves" and "lantern sleeves" belong to the same property "shape of sleeves", and they should therefore occur in the queries searching for similar categories of items that have sleeves. Thus, if two terms have different distributions of categories, it is more likely that they belong to different properties. Let $P_{cate}(x)$ be the distribution of categories for term x. We use a normalized symmetric KL divergence to model the dissimilarity of two distributions:

$$KL_{ns}(D_1, D_2) = 1 - e^{-\frac{KL(D_1,D_2)+KL(D_2,D_1)}{2}} \tag{8}$$

, where $KL(D_1, D_2)$ is the KL divergence between distribution D_1 and D_2. Then we have the following negative evidence:

$$ne_{cate}(x, y) = KL_{ns}(P_{cate}(x), P_{cate}(y)) \tag{9}$$

Principle 3 (Distribution of POS Tags): A pair of terms that have different distributions of POS tags are more likely to belong to two different properties.

Distribution of POS tags: Also, the terms belonging to the same property should have similar Part-of-Speech (POS) tags. For example, the values belonging to property "shape of sleeves" are all noun, like "lotus leaf sleeves" and "lantern sleeves". So, if two terms have different POS tags, it is more likely that they belong to different properties. Note that, the terms may have several POS tags. For example, "red" and "black" in the "color" property may be tagged as nouns or adjectives. Then we should therefore consider the distribution of POS tags of one term. Let $P_{pos}(x)$ be the distribution of POS tags for term x. We therefore have the following negative evidence:

$$ne_{pos}(x, y) = KL_{ns}(P_{pos}(x), P_{pos}(y)) \tag{10}$$

Principle 4 (Distribution of Semantic Tags): A pair of terms that have different distributions of semantic tags are quite likely to belong to two different properties.

Distribution of semantic tags: Similarly, the terms belonging to the same property should have similar semantic tags. For example, the value belonging to property "brand" should be tagged as brand, and the values belonging to property "color" should be tagged as color. We use a semantic tagger, which is a sequence-to-sequence model [31] trained from 8 million labeled sentences, to collect semantic tags. Let $P_{st}(x)$ be the distribution of semantic tags for term

x. We have the following negative evidence:

$$ne_{st}(x,y) = KL_{ns}(P_{st}(x), P_{st}(y)) \tag{11}$$

Aggregated negative evidence: Now, we have negative evidence derived from four different principles, and each negative evidence is in the range of $[0,1]$. Since a single negative evidence is weak, we cannot simply reject a claim that x and y belong to the same property with only weak negative evidence. In other words, we reject the claim when all the evidence sources have a high score rejecting the claim. This motivates us to integrate the scores in the following way:

$$NE(x,y) = ne_{co}(x,y) \cdot ne_{cate}(x,y) \cdot ne_{pos}(x,y) \cdot ne_{st}(x,y)$$

Let trh_{ne} be a threshold, we consider $NE > trh_{ne}$ to be negative evidence, which means the negative evidence is strongly rejected by all sources. We denote all edges $< x, y >$ with $NE(x,y) > trh_{ne}$ by $\mathcal{NE}$, which will be used as constraints in our algorithmic solution.

V. Method

In this section, we first introduce how we generate the term similarity graph in Section V-A. Then we elaborate two graph exploration based algorithms to evaluate relevance between a term and property in Section V-B. Finally, we analyze the time complexity of our methods in Section V-C.

A. Term Similarity Graph

For each category, we generate the term similarity graph $G = (T, E)$, where T is the set of terms in queries for the category[2] and each vertex in T represents a term. The edges connecting two terms represent that two terms belong to the same property, weighted by the accumulated positive evidence score between these terms.

We have two kinds of terms *existing values* (V) and *emerging query terms* (U). For each pair of terms (x, y), we have two kinds of edges connecting x and y. (1) $x, y \in V$, we set the weight as 1 (maximal edge weight) when x and y belong to the same property. (2) $x \in U, y \in U \cup V$, the weight of edge $< x, y >$ is $PE(x,y)$ which quantifies the strength of the positive evidence that supports the same membership of x and y. We show an example term similarity graph in Figure 2. There are two different properties: "shape of sleeves" and "color"; and two emerging query terms: "cake sleeves" and "Chinese red". We establish edges between emerging query terms and existing values.

Definition 1 (Term Similarity Graph): A term similarity graph for a category c, denoted by $G = (T, E)$, is an edge weighted graph, with vertex set $T = U \cup V$, where U is the set of all emerging query terms and V is the set of values in KB. For each $< x, y > \in E$, the weight $w(x,y)$ is defined as:

$$w(x,y) = \begin{cases} 1 & x, y \in V \text{ and } \phi(x,y) = 1 \\ PE(x,y) & x \in U, y \in U \cup V \end{cases} \tag{12}$$

[2]Each item has corresponding categories. Thus, we use clicked items when searching with a query to identify the terms for a specific category.

, where $\phi(x,y) = 1$ represents that x and y belong to the same property.

B. Algorithms

Basic Idea: Recall that our ultimate objective (Equation 1) is finding the most relevant property for each emerging query term according to $score(p,t)$. Since each property p has already a lot of values in KB, we reduce the evaluation of $score(p,t)$ to the evaluation of relevance between t and existing values of p:

$$score(p,t) = \max_{v_i \in V_p} score'(v_i, t) \tag{13}$$

, where V_p is the set of values under p, and $score'(v_i, t)$ is the relevance score between a value v_i of p and the emerging query term t. Here, $\max$ is just one of the alternative implementations, while average or other aggregation mechanisms could also be used. However, $\max$ function is proved to be the best choice in our experiments. Next, we elaborate two graph exploration mechanisms to estimate the similarity between two terms. With such graph exploration mechanisms, our methods are interpretable because we are able to know the reason why an emerging query term is related to a property by tracing the paths.

Shortest Path under Constraints: We evaluate $score'(v_i, t)$ with the term similarity graph G. One direct idea is that *if a path connecting two terms exhibits high plausibility, the two terms are quite likely to be similar with each other*. Thus, $score'(x,y)$ is reduced to the evaluation of the plausibility of the path connecting them:

$$score'(x,y) = \max_{r \in PATH(x,y)} s(r) \tag{14}$$

, where $PATH(x,y)$ is the set of all simple paths between x and y, and $s(r)$ calculates the *plausibility* score of simple path r. $s(r)$ is defined as the product of weights of all edges in path r: $s(r) = \prod_j w(e_j)$, where e_j is the j-th edge in path r, and $w(e_j)$ is the weight of edge e_j.

Thus, our goal is equivalent to find the path $\hat{r}$ that maximizes $score'(x,y)$ for each term pair (x,y):

$$\begin{aligned} \hat{r} &= \arg\max_r s(r) \\ &= \arg\max_r \ln s(r) \\ &= \arg\min_r \sum -\ln w(e_j) \end{aligned} \tag{15}$$

The last equation holds because $w(e_j)$ is in the range of $[0,1]$. Then, the goal is equivalent to find a shortest path between each term pair (x,y) under the new edge weight. Hence, we change the term similarity graph G to a new graph by replacing the weight $w(e)$ with $-\ln w(e)$ for each edge e.

Another concern is that we need to forbid the shortest path exploration going through edges with high negative evidence. We reflect this by multiplying the edge weights with a parameter $\lambda \ll 1$ to punish the negative edges. Specifically, the graph G will be changed to a new graph G',

and the weight of edges will be set to $-I(x,y)\ln w(x,y)$, where $I(x,y)$ is an indicator function:

$$I(x,y) = \begin{cases} 1 & <x,y> \notin \mathcal{NE} \\ \lambda & <x,y> \in \mathcal{NE} \end{cases} \tag{16}$$

Then, we run the shortest path algorithm on the new graph G' and get one of shortest paths $\hat{r}(x,y)$ for each term pair (x,y). The shortest path $\hat{r}(x,y)$ enables us to compute the relevance between two terms, as defined in Equation 14.

Random Walk under Constraints *:* The shortest path algorithm considers only one path between emerging query term t and value v. However, the neighbors in other paths clearly have also contribution to the similarity of term pairs. For example, if t is similar to another term t_1 and t_1 is similar to term t_2, it is quite likely that t is also similar to t_2. In other words, a single shortest path is not enough for the accurate estimation of term similarity. It motivates us to employ a random walk procedure to fully use the information in the term similarity graph. The underlying principle is that *if a random walk procedure starting from term x has a large probability to reach term y on term similarity graph G, x and y are similar to each other.*

The key to employ a random walk based similarity estimation is to define the transition probabilities $P_1(y|x)$ from x to y. The direct implementation in our setting is normalizing the positive evidence of term x by a softmax function:

$$P_1(y|x) = \frac{\exp(w(x,y))}{\sum_{i \in Nei(x)} \exp(w(x,i))} \tag{17}$$

where $Nei(x)$ is the neighbors of x in G. However, this generic definition needs to be improved to accommodate our settings: *First, we award the self-transition probability for existing values in KB*. Note that for an emerging query term x we hope the random walk procedure helps find the most relevant existing value in KB. Thus, when a random walk meets an existing value in V, we would like the procedure to be *"trapped"* (that is the random walk has a significant probability to transit to itself) in the self-transition, so that only existing values in KB will stand out with a significant probability to be reached. For this purpose, we enforce a probability $\theta \geq 0.5$ for self-transition when $x \in V$. *Second, we punish the walk through the edges rejected by negative evidence.* Specifically, we punish the transition probability of term pairs rejected by negative evidence by multiplying it with a parameter $\lambda \ll 1$. We use Equation 16 to modify $w(x,y)$ and derive a new exponent $I(x,y)w(x,y)$ for any term pairs.

Thus, the transition probability is defined as follows:

$$\hat{P}_1(y|x) = \begin{cases} P_1(y|x) & x \in U \\ \dfrac{(1-\theta)\exp(I(x,y)w(x,y))}{\sum_{i \in Nei(x)} \exp(I(x,i)w(x,i))} & x \in V \text{ and } x \neq y \\ \theta & x \in V \text{ and } x = y \end{cases} \tag{18}$$

Now, the transition probability is a matrix $\mathbf{A}$ where $\mathbf{A}_{x,y} = \hat{P}_1(y|x)$. The matrix $\mathbf{A}$ is a stochastic matrix with each rows summing to 1. Finally, we perform a random walk procedure: calculate the probability that one term x reaches to term y within k steps, denoted by $\hat{P}_k(y|x)$. Here, $\hat{P}_k(y|x) = [\mathbf{A}^k]_{x,y}$ and we use it as $score'(x,y)$.

C. Complexity Analysis

Considering that the shortest path under constraints method can be viewed as a single-source shortest paths problems [32] for each emerging query term, the time complexity of it is $O(|U||E| + |U||T|\log|T|)$, while $|E|$ is the number of all edges in term similarity graph, $|T|$ is the number of all vertexes in term similarity graph, and $|U|$ is the number of emerging query terms.

The time complexity of the Constrained Random Walk method is $O(\Delta|T|^3)$, which equals to time cost of a matrix multiplication problem, where Δ is the random walk step.

The time cost of our methods is determined by the size of term similarity graph. The time cost is acceptable when we run the algorithm in specific domains, in which the size of term similarity graph is small. The experimental results on the efficiency of our method can be found in Section VI-D.

VI. Experiments

In this section, we evaluate our shortest path under constraints algorithm (SP) and random walk under constraints approach (RW) for CPV KB completion and a movie domain KB completion. We compare our method with 8 baselines. And our method beats all of them. This section is organized as follows. First, we report the datasets we used and experimental settings in Section VI-A and Section VI-B. Then, we present 8 different baselines in Section VI-C. In Section VI-D, we show the effectiveness and efficiency of our methods, the effectiveness of all the external evidence, and then apply our methods in other domains and show some real examples.

A. Datasets

Table III
Dataset statistics

Category	#All Terms	#Existing Values	#Emerging Query Terms
Dress	706	502	204
Air Conditioner	127	91	36
Perfume	236	205	31
T-shirt	433	305	128
Movie	1577	1409	168

Table III presents the statistics of 5 datasets from different specific domains. The category "Dress", "Air Conditioner", "Perfume", and "T-shirt" are categories from CPV KB, and category "Movie" is generated from CN-DBpedia [2].

For CPV KB data, we use an old CPV KB to collect the values and properties from a given category, then we collect all terms from query logs within a single week. For each term t in the term set, as long as the term is not one of the values where $t \notin V$, then it is considered to be an emerging query term. Our goal is then to recommend the top-k most likely properties for each emerging query term.

For "Movie" domain, we randomly select a subset triples of CN-DBpedia which are in category "Movie". We randomly select about 10% triples as emerging query terms, and group several values as queries.

B. Experimental Settings

We conduct our experiment on a 24-Core 2.3GHz CPU, GTX-1080 GPU, 64GB RAM machine. We use the GPU for deep learning based methods to speed up training and predicting. We vary the value of parameters λ (negative edge punishment parameter), θ (self-transition ratio), and trh_{ne} (negative evidence selection threshold) of our method, and test them on category "Dress". We get the best performance when $\lambda = 0.1$, $\theta = 0.9$, $trh_{ne} = 0.94$.

C. Baseline Algorithms

Here we briefly introduce eight different baseline methods.

- **Embedding Similarity Based Method (Max-ES):** As Equation 3 shows, we use cosine similarity between term embedding to represent the similarity of two terms. The higher the similarity is, the more possible the two terms belong to the same property. We then define $score(p,t) = \max_{v_j \in p}(pe_{es}(t, v_j))$.
- **Embedding Similarity Based Method (Avg-ES):** This method is similar to the **Max-ES** one, but instead uses the arithmetic mean of embedding similarities: $score(p,t) = avg(pe_{es}(t, v_j))$, where $v_j \in p$.
- **Positive Evidence Only (PE):** We only conduct the random walk algorithm on the graph G that is generated by positive evidence, which is a special case of RW when the constraint is empty.
- **LDA:** We use LDA [14] as our basic topic model baseline. We consider each query as a document, every document as a mixture of several topics. Each topic consists of many words. We then use the distribution of topics to calculate $score'(t,v) = KL_{ns}(P_{topic}(t), P_{topic}(v))$. Specifically, we set 30 topics, $\alpha = 2$, $\beta = 0.01$ in LDA, here α and β are hyperparameters for the document-topic and topic-word Dirichlet distribution, respectively.
- **DF-LDA:** We use DF-LDA [15] as our baseline. Similar to LDA, we consider each query to be a document, every document to be a mixture of several topics and each topic to be a collection of many words. We then use negative evidence as Cannot-Links in DF-LDA. Specifically, we set 30 topics, $\alpha = 2$, $\beta = 0.01$, $\eta = 100$, here α and β are hyperparameters and η is a strength parameter. We set the number of Cannot-Links to 200, and the following experiment will show that the time cost is unacceptable when the number of Cannot-Links is greater than 200.
- **METIC:** We use METIC [33] as our baseline. Our problem can be viewed as an entity typing problem. We consider each emerging query term as an entity and each property as a type, then the goal is to find the real type for each entity. METIC uses description as inputs to train a complex bi-LSTM network for type prediction, and we use queries as text inputs to predict the corresponding property for each emerging query term.
- **Neural Relation Extraction (NRE):** As we have mentioned above, our problem can be viewed as a relation extraction problem. Given a category and an emerging query term, our goal is to find the relation between the category and the emerging query term. We thus use a sentence-level attention-based relation extraction model [11] as our baseline. This work considers that sentences contain the information about relations and uses an attention-based neural network to encode the sentences as the representation of the relation.
- **transE+CNN:** KG embeddings can be used for knowledge graph completion, and KG embeddings using external textual information [34] can complete KG with unseen values. This method learns a KG embedding using textual information, they explore two encoders including continuous bag-of-words and deep convolutional neural network to extract semantics of entity descriptions. We take CPV KG and queries as input for training.

D. Experimental Results

Next, we conduct experiments to evaluate the effectiveness and efficiency of our approaches. We also evaluate the effectiveness of all the external evidence. Finally, we apply our methods in other domain.

1: Effectiveness

Table IV
HITS@K FOR BASELINES AND OUR METHODS.

Hits@k	Dress			Air conditioner			Perfume			T-shirt		
	k=1	k=3	k=5	k=1	k=3	k=5	k=1	k=3	k=5	k=1	k=3	k=5
Max-ES	0.47	0.61	0.72	0.34	0.63	0.81	0.47	0.75	0.90	0.36	0.56	0.69
Avg-ES	0.32	0.51	0.61	0.17	0.35	0.76	0.31	0.64	0.85	0.28	0.50	0.66
PE	0.51	**0.88**	**0.93**	0.39	0.77	0.91	0.50	0.78	0.93	0.40	0.79	**0.89**
LDA	0.25	0.43	0.65	0.24	0.56	0.66	0.32	0.79	0.90	0.14	0.42	0.63
DF-LDA	0.24	0.39	0.49	0.20	0.50	0.61	0.31	0.73	0.88	0.10	0.33	0.58
METIC	0.07	0.32	0.37	0.56	0.72	0.72	0.55	0.72	0.76	0.19	0.28	0.37
NRE	0.33	0.39	0.46	0.31	0.33	0.42	**0.60**	0.75	0.80	0.15	0.23	0.40
transE+CNN	0.11	0.27	0.42	0.15	0.40	0.63	0.12	0.40	0.72	0.11	0.27	0.43
SP	**0.59**	0.79	0.85	**0.58**	0.77	0.93	0.48	**0.82**	**0.94**	**0.44**	0.74	0.86
RW	0.51	**0.88**	**0.93**	0.40	**0.79**	**0.96**	0.50	0.81	**0.94**	0.40	**0.80**	**0.89**

For each emerging query term, all the methods recommend the k most relevant properties. Here, we use Hits@k as our evaluation metric. In the experiment, k is set to 1, 3 and 5. The results of all methods are shown in Table IV. *As we can see, our method beats the other baselines.*

As we can see from the table, baseline1 beats baseline2, which proves that *the max function is better than average function.* Intuitively, the difference between existing values in the same property may be very large, and an existing value may have both similar values and dissimilar values in the same property. So, an average function $avg(score(t,v))$ is affected by dissimilar values.

SP is better when we only need the most relevant property, while RW works better when more relevant properties are preferred. SP finds only a simple path with the maximum weight, and the connecting terms most likely belong to the same property. However, RW finds all possible simple paths with their corresponding weights. Each weight represents the probability that two terms belong to the same property according to this path. So, the values with only one shortest path connecting to the term have a lower rank under RW. RW considers more similar existing values than SP, and this avoids noise. However, this method may give the wrong properties a higher score.

DF-LDA gets a worse result than LDA itself because there exist errors in the negative evidence, while DF-LDA requires that all Cannot-Links must be correct. However, both SP and RW work better when taking negative evidence into consideration. This proves that *both SP and RW are good at noise tolerance*.

As we have mentioned above, all the deep learning based models perform worse than our methods due to the limited number of training data. Compared to other baselines, the deep learning based models still achieve a worse result. This result reveals that the lack of data limits the performance of deep learning models in specific domains.

2: Efficiency

The efficiency is also quite important in real applications. So, we compare the efficiency of different methods in the category "Dress". The time consumed is shown in Table V.

Table V
TIME COST IN CATEGORY "DRESS"

Method	Time Cost (s)
LDA	13500
DF-LDA	67740
METIC	351
NRE	32
transE+CNN	45
SP	4260
RW	3

The result shows that our method is much faster than LDA based methods. And the RW is even faster than deep learning based methods. The deep learning based models takes little time to train and predict, because the scale of training data is small and these models use a GTX-1080 to speed up. However, these models still take more time than RW because the matrix multiplication in RW is quite simple. Also, both LDA and DF-LDA require complex sampling, which takes lots of time. In contrast, SP and RW have low time complexity. *We highlight that our methods are less time-consuming than LDA or DF-LDA.*

It is worth mentioning that, the amount of time used in DF-LDA increases when taking more negative evidence into consideration. Hence we conduct a further experiment to show the correlation between time cost and the number of Cannot-Links in DF-LDA. The result is shown in Figure 3.

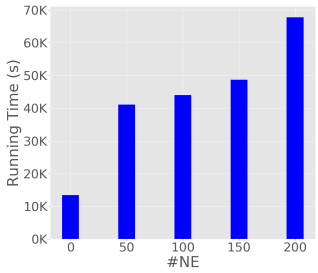

Figure 3. Time cost of DF-LDA. Here $\#NE$ represents the number of negative evidence we use.

We can see that the time grows in a super-linear way when taking more negative evidence as input. Also, the memory need increases with the number of negative evidence and it begins to run out of all the $64GB$ of memory when only 250 sets of negative evidence had to be considered. However, we have at least thousands of negative evidence, the time cost and memory need are unacceptable.

3: Effectiveness of External Evidence

In order to see the effectiveness of different positive evidence, we use embedding based similarity alone, embedding similarity with edit distance, all the positive evidence to aggregate three kinds of positive evidence respectively. Then, we use 4 different sets of negative evidence to prove the effectiveness of different negative evidence.

Table VI
EFFECTIVENESS OF DIFFERENT EXTERNAL EVIDENCE IN CATEGORY "DRESS"

No.	Experiment Setting	Hits@1	Hits@3	Hits@5
1	pe_{es}	45.75	81.44	88.39
2	$pe_{es} + pe_{ed}$	50.42	87.25	92.92
3	$pe_{es} + pe_{ed} + pe_{ia}$	50.85	87.68	93.06
4	$PE + ne_{co}$	50.99	87.68	93.06
5	$PE + ne_{cate}$	50.85	87.68	93.20
6	$PE + ne_{st}$	50.99	87.68	93.06
7	$PE + ne_{pos}$	50.99	87.68	93.20

As we can see in Table VI, the accuracy increases when we use more positive evidence, so *all the positive evidence has a significant impact on the overall results*. Also, we see that results of experiments that take one kind of negative evidence into consideration are better than the result derived when considering only positive evidence. This suggests that *all the negative evidence is effective for our task.*

Table VII
ACCURACY OF EXTERNAL EVIDENCE

Domain	Dress	Air	Perfume	T-shirts
#NE	64824	7608	30145	48228
Error Rate	9.75%	15.83%	32.96%	17.28%

As mentioned before, there exist errors in negative evidence, since that negative evidence is generated automati-

cally. The number of the negative evidence for each category and their error rates are reported in Table VII. This shows that methods using negative evidence should have the ability of noise tolerance. A smaller trh_{ne} means more negative evidence and a larger trh_{ne} means the lower error rate of negative evidence. Thus, finding a best trh_{ne} is a trade-off between the number of negative evidence and error rate.

4: Adapt to Other Domains

We apply our methods in other domains, and here we use our method to complete a movie domain KB. We use all the positive evidence, which is edit distance, embedding based similarity, and the affix based similarity. For negative evidence, we use distribution of POS tags and semantic tags, since actors may appear in the same queries and we only have query logs for category "Movie".

Table VIII
EFFECTIVENESS OF OUR METHOD IN CATEGORY "MOVIE"

Method	Hits@1	Hits@3	Hits@5
Max-ES	0.19	0.47	0.64
Avg-ES	0.00	0.00	0.01
PE	0.24	0.55	0.72
LDA	0.01	0.07	0.28
DF-LDA	0.06	0.07	0.28
METIC	0.08	0.50	0.64
NRE	0.12	0.52	0.67
transE+CNN	0.02	0.05	0.08
SP	**0.25**	0.55	0.72
RW	0.24	**0.59**	**0.76**

Our methods achieve a good result on the movie DKB completion task. The result is shown in Table VIII. As we can see from the table, though we use less external evidence, our methods still perform well and beat all the other baselines. The result shows that the small scale of training data limits the performance of deep models.

5: Case Study

We present several cases to show the effectiveness of our method in Table IX. We show emerging query terms together with their frequency and the existing value with highest $score'(t, v)$. The emerging query terms shown in the first part of the table are high frequency terms, those in the second part are long-tailed emerging query terms, and those in the third part are error cases. Our method can find a good result for both long-tailed emerging query terms and high frequency emerging query terms, and do not rely on semantic similarity. Our methods can also handle spelling mistakes, as shown by how "hanging bag skirt", which is a spelling mistake in Chinese and means "strap skirt", was still recognized, and its most similar value was "strap skirt". We also show some error cases. There are mainly two kind of errors. First, some emerging query terms belong to some properties that are not in CPV KB. For example, "spring" belongs to property "season", and there is no such property. Such error needs human experts to label new properties. Second, some terms need more information for understanding. Our method doesn't know "small daisy" is a kind of flower, and thus "small daisy" shares low similarity with other flowers in CPV KB. Such error needs more external knowledge, e.g. KB rules, for understanding query terms.

Table IX
NEW VALUES AND THEIR MOST SIMILAR VALUES

Case Type	Emerging Query Terms (#num)	Existing Value
Good Cases - High Frequency	vest(81646)	sleeveless
	large size(69205)	average size
	thin waist(11063)	elastic waist
Good Cases - Low Frequency	loose sleeves(104)	bat sleeves
	collarless(109)	double collar
	hanging bag skirt(209)	strap skirt
Error Cases	custom made	milky
	spring	British style
	small daisy	purple

VII. DEPLOYMENT

We have already deployed RW for CPV KB completion in Taobao. We provide top-5 relevant properties for each emerging query terms for manual verification. We collect every week's query logs as the input, and only a few people do the manual verification. Up to now, we have helped add 6968260 new CPV triples from a duration of only a year, and we are able to add more than 20000 new CPV triples per day. After the deployment of our method, the percentage of unrecognized queries dropped from 69.10% to 43.57%, and the percentage of unrecognized terms dropped from 69.19% to 36.31%. The more complete CPV KB improves the percentage of recognized queries and recognized terms by 25.53% and 32.88%, respectively. In addition to this, the precision of our method increases when taking a more complete CPV KB as input, because a more complete CPV KB will help to calculate the $score(p, t)$ more accurately.

It should be pointed out that, although our algorithm is fast, data acquisition and preprocessing is not fast and will slow the process down. Fortunately, completing a DKB doesn't require a quick feedback, so the preprocessing is completely acceptable. The main bottleneck is the manual verification due to the high human cost. Thus, one of our future works is to further reduce human efforts.

VIII. CONCLUSION

In this paper, we propose to complete domain-specific knowledge bases using queries as external information. In detail, we propose a framework to find the corresponding property for each emerging query term. First, we collect positive evidence and negative evidence to measure whether two terms belong to the same property. Then, we generate a term similarity graph using positive evidence and calculate a score on the graph under the constraints derived from negative evidence to predict the membership of each emerging query term. Extensive quantitative experiments have been conducted to justify the efficiency and effectiveness of our method.

Discussion: We identify some future work for our domain-specific completion method. First, though deep learning based methods suffer from the lack of training data in specific domains, the neural networks are able to find some useful latent features. Second, there might be some more effective methods using external evidence to find the reasonable score for each (emerging query term, property) pair. For example, KB rules [35], [36] help to build a better term similarity graph, by adding weights on paths. Third, though we provide top-k relevant properties for each emerging query terms for manual verification, there still need lots of human labeling. It is important to get a smaller k, so that human efforts can be further reduced. Fourth, an interesting direction is to find some more effective way to aggregate external evidence. Finally, this research focuses on the classification of existing properties. How to automatically cluster the emerging query terms belonging to new property is another interesting direction.

REFERENCES

[1] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," *The semantic web*, pp. 722–735, 2007.

[2] B. Xu, Y. Xu, J. Liang, C. Xie, B. Liang, W. Cui, and Y. Xiao, "Cn-dbpedia: A never-ending chinese knowledge extraction system," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2017, pp. 428–438.

[3] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 697–706.

[4] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: A probabilistic taxonomy for text understanding," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 481–492.

[5] J. Liang, Y. Xiao, H. Wang, Y. Zhang, and W. Wang, "Probase+: Inferring missing links in conceptual taxonomies," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 6, pp. 1281–1295, 2017.

[6] A. Swartz, "Musicbrainz: A semantic web service," *IEEE Intelligent Systems*, vol. 17, no. 1, pp. 76–77, 2002.

[7] K.-I. Goh, M. E. Cusick, D. Valle, B. Childs, M. Vidal, and A.-L. Barabási, "The human disease network," *Proceedings of the National Academy of Sciences*, vol. 104, no. 21, pp. 8685–8690, 2007.

[8] M. Wick and B. Vatant, "The geonames geographical database," *Available from World Wide Web: http://geonames. org*, 2012.

[9] G. Ji, K. Liu, S. He, and J. Zhao, "Distant supervision for relation extraction with sentence-level attention and entity descriptions," in *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[10] Y. Wu, D. Bamman, and S. Russell, "Adversarial training for relation extraction," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 1778–1783.

[11] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, "Neural relation extraction with selective attention over instances," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2016, pp. 2124–2133.

[12] D. Ritze, O. Lehmberg, and C. Bizer, "Matching html tables to dbpedia," in *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*. ACM, 2015, p. 10.

[13] C. Ran, W. Shen, J. Wang, and X. Zhu, "Domain-specific knowledge base enrichment using wikipedia tables," in *Data Mining (ICDM), 2015 IEEE International Conference on*. IEEE, 2015, pp. 349–358.

[14] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[15] D. Andrzejewski, X. Zhu, and M. Craven, "Incorporating domain knowledge into topic modeling via dirichlet forest priors," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 25–32.

[16] H. Paulheim, "Knowledge graph refinement: A survey of approaches and evaluation methods," *Semantic web*, vol. 8, no. 3, pp. 489–508, 2017.

[17] J. Sleeman, T. Finin, and A. Joshi, "Topic modeling for rdf graphs." in *LD4IE@ ISWC*, 2015, pp. 48–62.

[18] M. Nickel, V. Tresp, and H.-P. Kriegel, "Factorizing yago: scalable machine learning for linked data," in *Proceedings of the 21st international conference on World Wide Web*. ACM, 2012, pp. 271–280.

[19] J. Sleeman and T. Finin, "Type prediction for efficient coreference resolution in heterogeneous semantic graphs," in *Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on*. IEEE, 2013, pp. 78–85.

[20] E. Munoz, A. Hogan, and A. Mileo, "Triplifying wikipedia's tables." *LD4IE@ ISWC*, vol. 1057, 2013.

[21] A. P. Aprosio, C. Giuliano, and A. Lavelli, "Extending the coverage of dbpedia properties using distant supervision over wikipedia." in *NLP-DBPEDIA@ ISWC*, 2013.

[22] D. Gerber, S. Hellmann, L. Bühmann, T. Soru, R. Usbeck, and A.-C. N. Ngomo, "Real-time rdf extraction from unstructured data streams," in *International Semantic Web Conference*. Springer, 2013, pp. 135–150.

[23] D. Gerber and A.-C. N. Ngomo, "Bootstrapping the linked data web," in *1st Workshop on Web Scale Knowledge Extraction@ ISWC*, vol. 2011, 2011.

[24] R. Gupta, A. Halevy, X. Wang, S. E. Whang, and F. Wu, "Biperpedia: An ontology for search applications," *Proceedings of the VLDB Endowment*, vol. 7, no. 7, pp. 505–516, 2014.

[25] V. Christophides, V. Efthymiou, and K. Stefanidis, "Entity resolution in the web of data," *Synthesis Lectures on the Semantic Web*, vol. 5, no. 3, pp. 1–122, 2015.

[26] T. Lee, Z. Wang, H. Wang, and S.-w. Hwang, "Web scale taxonomy cleansing," *Proceedings of the VLDB Endowment*, vol. 4, no. 12, pp. 1295–1306, 2011.

[27] T. Hofmann, "Probabilistic latent semantic analysis," in *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1999, pp. 289–296.

[28] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *ICLR workshop*, 2013.

[29] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[30] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *hlt-Naacl*, vol. 13, 2013, pp. 746–751.

[31] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, 2014, pp. 3104–3112.

[32] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *Journal of the ACM (JACM)*, vol. 34, no. 3, pp. 596–615, 1987.

[33] B. Xu, Z. Luo, L. Huang, B. Liang, Y. Xiao, D. Yang, and W. Wang, "Metic: Multi-instance entity typing from corpus," in *Proceedings of the 2018 ACM on Conference on Information and Knowledge Management*. ACM, 2018.

[34] R. Xie, Z. Liu, J. Jia, H. Luan, and M. Sun, "Representation learning of knowledge graphs with entity descriptions." in *AAAI*, 2016, pp. 2659–2665.

[35] S. Ortona, V. V. Meduri, and P. Papotti, "Robust discovery of positive and negative rules in knowledge bases," in *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018, pp. 1168–1179.

[36] L. Galárraga, C. Teflioudi, K. Hose, and F. M. Suchanek, "Fast rule mining in ontological knowledge bases with amie+," *The VLDB Journal The International Journal on Very Large Data Bases*, vol. 24, no. 6, pp. 707–730, 2015.