Generating Query Facets using Knowledge Bases

Zhengbao Jiang, Zhicheng Dou, *Member, IEEE,* and Ji-Rong Wen, *Senior Member, IEEE*

Abstract—A query facet is a significant list of information nuggets that explains an underlying aspect of a query. Existing algorithms mine facets of a query by extracting frequent lists contained in top search results. The coverage of facets and facet items mined by this kind of methods might be limited, because only a small number of search results are used. In order to solve this problem, we propose mining query facets by using knowledge bases which contain high-quality structured data. Specifically, we first generate facets based on the properties of the entities which are contained in Freebase and correspond to the query. Second, we mine initial query facets from search results, then expanding them by finding similar entities from Freebase. Experimental results show that our proposed method can significantly improve the coverage of facet items over the state-of-the-art algorithms.

Index Terms—Query Facets, Knowledge Bases, Query Dimensions

1 Introduction

With the help of search engines, users can quickly find web pages containing the information they want by issuing queries and receiving search results comprised of "ten blue links". However, previous studies show that many users are not satisfied with this kind of conventional search result pages [1], [2], [3]. Users often have to click and view many documents to summarize the information they are seeking, especially when they want to learn about a topic that covers different aspects. This usually takes a lot of time and troubles the users. An automatic summarization of search results may help users understand the query without browsing many pages, hence could save their time.

Mining query facets (or query dimensions) is an emerging approach to solve the problem above. Table 1 shows the top facets for query "Pentax," "Beijing subway," and "Tom Cruise". Pentax is a Japanese camera brand. Its query facets cover aspects about related camera brands, Pentax's SLR cameras, Pentax's small digital cameras, and different kinds of optical devices. These query facets help users learn about the topic "Pentax," and at the same time, users can further narrow down their information needs based on these facets.

Existing query facet mining algorithms mainly rely on the top search results from search engines [4], [5], [6], [7]. Dou et al. first introduced the concept of query dimensions [4], which is the same concept as query facet discussed in this paper. They proposed QDMiner, a system that can automatically mine query facets by aggregating frequent lists contained in the results. The lists are extracted by HTML tags (like <select> and <table>), text patterns, and repeat content blocks contained in web pages. Kong et al. [6] proposed two supervised methods, namely QF-I and QF-J,

TABLE 1
Example query facets.

query	Pentax
1	k7, k200d, istd, kx, k100d, k10d, ...
2	optioi 10, optio h90, optio rz10, optio rs1000, ...
3	binoculars, spotting scopes, range finders, ...
4	canon, casio, nikon, kodak, sony, olympus, ...
query	Beijing subway
1	line 1, line 2, line 4, line 5, line 10, line 13, batong line, ...
2	xizhimen, jianguomen, dongzhimen, chongwenmen, ...
3	forbidden city, temple of heaven, tiananmen square, ...
query	Tom Cruise
1	knight and day, top gun, valkyrie, mission impossible, ...
2	katie holmes, nicole kidman, mimi rogers, suri cruise, ...
3	actor, producer, director, writer

to mine query facets from the results. In all these existing solutions, facet items are extracted from the top search results from a search engine (e.g., top 100 search results from Bing.com). More specifically, facet items are extracted from the lists contained in the results. The problem is that the coverage of facets mined using this kind of methods might be limited, because some useful words or phrases might not appear in a list within the search results used and they have no opportunity to be mined.

In order to solve this problem, we propose leveraging a knowledge base as a complementary data source to improve the quality of query facets. Knowledge bases contain high-quality structured information such as entities and their properties and are especially useful when the query is related to an entity. We propose using both knowledge bases and search results to mine query facets in this paper. The reason why we don't abandon search results is that search results reflect user intent and provide abundant context for facet generation and expansion. Our target is to improve the recall of facet and facet items by utilizing entities and their properties contained in knowledge bases, and at the same time, make sure that the accuracy of facet items are not harmed too much. Our approach consists of two

- *Zhengbao Jiang and Zhicheng Dou are with the School of Information and DEKE, Renmin University of China, Beijing 100872, P.R. China. E-mail: rucjzb@163.com, dou@ruc.edu.cn*
- *Ji-Rong Wen is with the School of Information and Beijing Key Laboratory of Big Data Management and Analysis Methods, Renmin University of China, Beijing 100872, P.R. China. E-mail: jirong.wen@gmail.com*

*Manuscript received ***, 2016; revised ***, 2016.*

methods which are **facet generation** and **facet expansion**. In facet generation, we directly use properties of entities corresponding to a query as its facet candidates. In facet expansion, we expand initial facets mined by traditional algorithms such as QDMiner to find more similar items contained in a knowledge base such as Freebase[1]. The facets constructed by the two methods are further merged and ranked to generate final query facets. More specifically,

(1) **Facet Generation**: We propose directly mining query facet candidates from Freebase. Given a query, we first retrieve relevant entities from Freebase, then obtain all the properties of these entities. For example, for the query "Beijing subway", we first retrieve entity *Beijing Subway* and its properties. These properties include *Transit Lines* with values such as *Line 1*, *Line 2*, *Airport Express*, and etc. We call these properties as direct properties. Sometimes, such properties are not sufficient. To get more properties of an entity, we further obtain "properties of properties". For example, each subway line in the example above has a property *Stations*, then *All the Stations of Each Line* could be combined as an extended property of entity *Beijing Subway*. We denote this kind of properties as second-hop properties. In the same way, we could obtain multi-hop properties. In this paper, we use both direct and second-hop properties as facet candidates. Note that if query matches no entities, no candidates will be generated in this step.

(2) **Facet Expansion**: We use QDMiner to mine initial query facets, then use Freebase to expand these facets to get similar items. We propose two different ways to expand a facet. **First**, we try to assign each facet to a suitable property of the entities corresponding to the query, and add the target entities of the property to enrich the facet. We denote this method as **property based facet expansion**. For example, for the query "Michael Jackson," an initial facet mined by QDMiner is comprised of his compositions "you rock my world," "butterflies," "man in the mirror," "thriller," "cry," etc. We find that entity *Michael Jackson* has a property *Nominated Works* which covers most items in this facet, hence we could use other target entities of this property to enrich this initial facet. When no relevant entities or properties are retrieved, we use the **second** way – **type based facet expansion**. We find a common type that covers most facet items. For example, "Kristen Stewart," "Robert Pattinson," "Taylor Lautner," etc. is a facet of query "Eclipse." The best type covering most facet items in Freebase is *Celebrities*. However, this type is too broad and it contains many entities which are irrelevant to the facet. Thus we propose mining some constraints that could be used together with the type to retrieve relevant entities from Freebase more specifically. The constraints grasp the relationship between facet and the query more precisely, which could guarantee the accuracy of expansion. In the example above, we find out that all initial facet items are actually actors of the film *Eclipse*, hence we use a constraint (*Eclipse* $\in$ *Films Starring In*) which means that the celebrity must have a property *Films Starring In* and *Eclipse* must appear in the target values of the property to narrow down semantic scope.

In this paper, we use both facet generation and facet expansion to mine query facets rather than just one of them. Facet generation outputs candidates only for queries matching with Freebase entities while type based expansion is more general. Facet generation may introduce some new facets contained in Freebase which are complement to the ones mined by traditional methods.

(3) The facet candidates constructed by facet generation and expansion are further merged, because there might be duplicate items within these candidates. We then re-weight the final facets by checking the occurrence of the facet items within top search results.

We denote the solution above which generates new facets and expand existing facets using Freebase with **QDM$_{\text{KB}}$** in this paper. Please note that we actually leverage the advantages of both knowledge bases and search results to generate high-quality query facets, hence QDM$_{\text{KB}}$ has high potential to outperform the state-of-the-art algorithms which solely use search results for facet mining. The contributions are two-fold:

(1) By leveraging both knowledge bases and search results, QDM$_{\text{KB}}$ breaks the limitation of only using search results to generate query facets, thus could improve the quality of facets, especially recall. Although in practice, it is impossible and unnecessary to show hundreds of facet items to users, recall is still of great importance. First, for some short facets such as "founders" of query "Google" and "family members" of query "Tom Cruise," users want exactly total answers. Second, even for long facets which cover dozens of items, users still may have the potential to explore as many suggested items as possible. Listing all these items aside the traditional "ten blue links" is distracting. Instead, we could use a "more" link to guide the users who want to explore more to another single page.

(2) Knowledge bases act not only as supplemental data sources, but also bring structured information to query facets. Different items among facets mined by traditional methods are isolated and lean, while during the process of our algorithm, we actually link some facet items to knowledge bases, which could yield many benefits such as (a) finding more information related to each facet item through the link structure of knowledge bases; (b) using the types or properties in knowledge bases as a potential explanation of the meaning of each facet.

We use two existing datasets that are used by QDMiner [4], namely UserQ and RandQ, to evaluate the proposed method. Experimental results show that our proposed method QDM$_{\text{KB}}$ significantly outperforms all state-of-the-art methods including QDMiner, QF-I, and QF-J in terms of rp-nDCG. It yields significantly higher recall of facet items.

The remainder of this paper is organized as follows. We introduce related works in Section 2. Our approach is introduced in Section 3. Experimental methodology and results are introduced in Section 4 and Section 5. We conclude our work in Section 6.

2 Related Work

2.1 Query Facet Mining and Faceted Search

Query facets summarize a query in different aspects. They may help users quickly understand important aspects of the query and help them explore information. Dou et al. first introduced this problem and proposed QDMiner algorithm

1. http://www.freebase.com/

[4]. QDMiner first extracts frequent lists in top search results using predefined patterns, then weights each list and groups them into final facets. Similar to QDMiner, Kong and Allan [6] developed supervised approaches, namely QF-I and QF-J, to mine query facets. Facet item candidates are extracted from frequent lists which are obtained in a similar way as QDMiner. Then two Bayesian models are learned to estimate how likely a candidate is a facet item and how likely two candidates belong to the same facet. All the existing works are based on top search results, hence the quality of final facets might be limited. If some words or phrases don't appear in a list within top search results, they have no opportunity to be facet items.

Different from query facet mining that generates facets for each query without any domain assumptions or prior knowledge, some traditional faceted search approaches are mainly built on a specific domain or predefined facet categories. The problem of automatically mining facet metadata and mapping documents onto these categories has been studied for years [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18], [19]. A robust review of faceted search is beyond the scope of this paper. Dakka et al. [12], [13], [14] developed methods to extract facet hierarchies for a text corpus or a text database then assign each document to those facets. One main difference is that we aim to mine several semantically coordinate lists of items to guide users' search, while Dakka's methods focus on building concept hierarchies. E. Stoica et al. [19] proposed Castanet to automatically generate domain-specific faceted metadata from textual descriptions of items based on existing external lexical database WordNet. Li et al. [11] proposed Facetedpedia base on the internal link structure and categories of Wikipedia. Different from the approaches above, query facet mining constructs specific facets for arbitrary query without predefined domains or categories.

2.2 Set Expansion

The problem of facet expansion we discuss in this paper is related to the problem of set expansion, which focuses on expanding a partial set of "seed" objects into a more complete set. The problem of set expansion has been well studied. The well-known examples of web-based set expansion systems are Google Sets [20], SEAL (Set Expander for Any Language) [21], [22], [23], [24], and NeedleSeek [25], [26]. The basic idea of SEAL is that different terms in the same set should appear in similar context. SEAL first searches semi-structure documents that contain all the seeds, then constructs a wrapper for each document to extract more terms with similar context. The final rank of all the candidates is conducted by random walk in a graph.

Similar to set expansion, we also aim to find new items that are similar to seed items to enrich the facet. However, the notable difference between query facet expansion and set expansion is that we have higher requirement on item accuracy for facet expansion. The expanded facet items should not be just relevant to each seed item – they should also be relevant to the query context, including query itself and its top search results. Utilizing the context will help measure the relevance of discovered items and control the quality of the facet. For example, a set including "Sam Smith," "Pharrell Williams," and "Carrie Underwood" could be expanded to a set of "pop stars" under no context, while given the query "57th grammy," it is more accurate to expand this set to a collection of "57th grammy award winners."

2.3 Query Recommendation

The problem of query ambiguity and broadness has been discussed for several years, and query recommendation (or query suggestion) is a popular way to help users specify their information needs and issue the correct query. Existing query recommendation techniques are used to propose alternative queries that are semantically similar to the user's original query [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37]. These suggested queries are provided to users and users can select one of them which better describes their information needs. The problem of mining facets is different from query recommendation, as the main purpose of mining facets is to summarize the information contained in the query, rather than to find some query reformulations. There might be different applications of using query facets. For example, for the query "Beijing Subway," there exists a facet "line1," "line2," "line4," "line5," etc. which includes transit lines that are direct summarization of the query information rather than query reformulations. It would be natural to use query facets as structural query suggestions, and we will investigate the solution to exploit query facets for generating query suggestions.

2.4 Query-based Summarization

Text summarization technique is a fundamental research area. Existing methods could be distinguished following different criterias: summary construction methods (abstractive/extractive), number of sources for the summary (single/multiple documents), summary trigger (generic/query-based). Different approaches of query-based summarization include graph-based, language model-based and machine learning. The detailed illustration could be found in [38] and [39]. Although both summarization and query facets aim to conclude valuable information from retrieved documents, they vary in the way of presentation. Summarization uses a flat list of sentences, while query facet mining digs deeper to generate multiple lists of semantically related terms. Organizing information as lists is a more intuitive way to guide user's browsing.

3 Mining Query Facets

A query facet is comprised of homogeneous or coordinate items which describes or summarizes an important aspect of a query. A facet item is an information nugget which could be represented in the form of words, phrases, or short sentences. A single query may include multiple facets, as shown in Table 1.

Existing approaches, such as QDMiner, QF-I, and QF-J, mainly exploit frequent lists contained in top search results for generating query facets. These lists are extracted by predefined patterns described in Table 2. Using these patterns, we could extract many coordinate items such as watch brands in both Example 1 and Example 2 shown below. In QDMiner, the extracted lists are weighted by their frequencies and are grouped together into query facets. In

TABLE 2
Text patterns for extracting lists.

Pattern type	Examples
free text	item{, item}* (and|or) {other} item
HTML	<select><option>item</option>...</select> <ul><li>item</li>...</ul> <table><tr><td>item</td></tr>...</table>

QF-I and QF-J, the items contained in extracted lists make up the candidate set. Two Bayesian models are trained to predict whether a candidate is a facet item and whether two candidates belong to the same facet.

Example 1 *We shop for gorgeous watches from* *Seiko*, *Bulova*, *Lucien Piccard*, *Citizen*, *Cartier* *or* *Invicta*.

Example 2 <ul><li><a ...>Omega SA</a></li>
<li><a ...>Jaeger-LeCoultre</a></li>
<li><a ...>Rolex</a></li>...</ul>

All existing approaches heavily rely on top search results which is defective. We propose mining query facets based on both knowledge bases and search results, because:

(1) There are some important items in the top retrieved documents, but they are not represented in a list. For example, the following paragraph is in the top retrieved documents of the query "best restaurants in Boston".

Example 3 *I lived in Boston when I was a child. My father always took me to* *Mistral restaurants* *at that time, and I thought it was the best restaurants in Boston. After I grow up, my girlfriend likes to eat at* *No.9 Park*. *I promise that's the best restaurants I have ever seen...*

In this example, there are two restaurant names "Mistral restaurants" and "No.9 Park" which are good facet item candidates for the query. However, it is unable to extract them by the patterns used in existing works, because they are not represented in a list.

(2) Sometimes some items are not even in top search results, so we have no opportunity to extract them.

(3) We suggest that knowledge bases (like Wikipedia or Freebase) are very good data sources for generating query facets. Knowledge bases contain high-quality structured information. By leveraging a knowledge base, we may have the opportunity to extract both "Mistral restaurants" and "No.9 Park" as instances of hotel for Example 3.

(4) The sole use of knowledge bases is not enough and we still need to leverage search results. The quality of information contained in a knowledge base is generally high, whereas using search results has other benefits. First, search results may contain fresh facet items which are not included in a knowledge base, especially the latest information. For example, search results for the query "Apple" may include information about "iPhone 6" earlier than a knowledge base. Second, search results help identify the importance of entities. For instance, in Freebase, both *Apple* and *Samsung* belong to the type *Mobile Brands*. No information is provided about which brand is more popular whereas using search results of the query "mobile brands," we can at least know which brand is mentioned more frequently. Third, it is not easy to directly generate facets based on a knowledge base for queries which are not entities. We need search results to generate initial facets for this kind of queries.

In the remaining part of this section, we will introduce $\mathbf{QDM_{KB}}$ in detail.

3.1 Overall Solution

In this paper, we use Freebase for mining query facets. Freebase is a large knowledge base consisting of data harvested from sources such as Wikipedia, NNDB, FMD, and MusicBrainz, as well as individually contributed data from users. Freebase provides a JSON-based HTTP API[2] to programmers, and we can use Metaweb Query Language (MQL) to customize a query to retrieve data. In this paper, we use Search API and Topic API to retrieve entities, properties, and types. All entities, properties and types in Freebase are presented in *italics* in this paper.

The overall solution, namely $\text{QDM}_{\mathbf{KB}}$, is shown in Figure 1. We use both facet generation and facet expansion to construct facet candidates. These candidates are further merged and ranked to generate final facets. Given a query q issued to QDM_{KB}, the algorithm has four major steps:

(1) Facet Generation

- **Retrieve relevant entities**
  Based on Freebase Search API, we retrieve a list of entities $E(q)$ relevant to query q
- **Retrieve related properties**
  Based on Freebase Topic API, for each entity $e \in E(q)$, we retrieve all its direct properties $P_1(e)$ and further construct its second-hop properties $P_2(e)$. $P(e) = P_1(e) \cup P_2(e)$ and $P(q) = \bigcup_{e\in E(q)} P(e)$. We remove properties in $P(q)$ which have only one target entity and the left ones are facet candidates.

(2) Facet Expansion

We first use existing state-of-art algorithm QDMiner to generate a set of initial facets $F(q)$. For each facet $f \in F(q)$, we use the following steps:

- **Property based facet expansion**
  We try to assign f to the most appropriate property $p \in P(q)$ and use other entities in p to enrich f. The property p should cover most items in f, meanwhile its size should be as small as possible to avoid including unrelated items.
- **Type based facet expansion**
  If $P(q)$ is empty or no suitable p is found in $P(q)$, we try to assign f to the most fine-grained type t which covers most items in f and use other entities in t to enrich f. Because Freebase types are usually too broad, we further construct some constraints to narrow down the semantic scope just like the example in Section 1. The constraints precisely characterize the relationship between facet and query context and thus improve the accuracy of expansion.

(3) Facet Grouping

All the facet candidates constructed by facet generation and expansion might have duplicate entities. We use the Quality Threshold algorithm to cluster them into the final facets by grouping similar candidates together.

(4) Facet Weighting

2. https://developers.google.com/freebase/

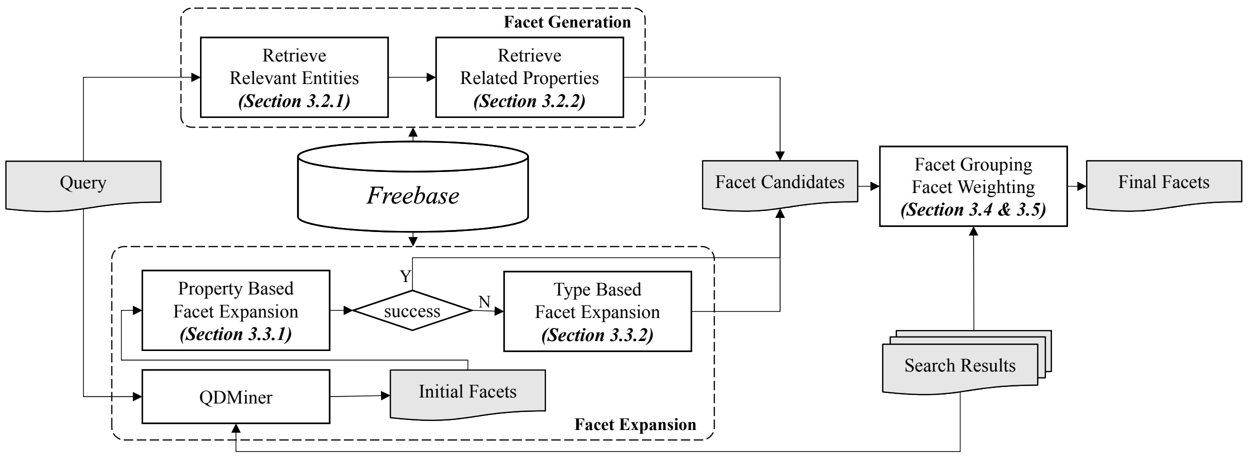

Facet Generation
Retrieve Relevant Entities (Section 3.2.1)
Retrieve Related Properties (Section 3.2.2)
Query
Freebase
Facet Candidates
Facet Grouping Facet Weighting (Section 3.4 & 3.5)
Final Facets
Property Based Facet Expansion (Section 3.3.1)
success
Y
N
Type Based Facet Expansion (Section 3.3.2)
Search Results
QDMiner
Initial Facets
Facet Expansion

Fig. 1. Overall solution of **QDM$_{\text{KB}}$**.

We weight each final facet in the same way as QDMiner. A good facet should be supported by many websites and contain informative items. These final facets are sorted by weights to form the output of QDM$_{\text{KB}}$.

By utilizing the high-quality information of knowledge bases and user intent reflected in search results, QDM$_{\text{KB}}$ has potential to outperform existing algorithms that are solely based on top search results. In the remaining parts of this section, we will introduce each component in detail.

3.2 Facet Generation

Given a query q issued to QDM$_{\text{KB}}$, we try to find relevant entities in Freebase and construct properties of these entities. These properties are inherent summarizations of corresponding entity in different aspects thus can be used directly as facet candidates. If the query matches no entities in Freebase, no candidates will be generated in this step. We will use Freebase Search API to retrieve relevant entities and use Topic API to retrieve properties.

3.2.1 Retrieving Relevant Entities

We use Freebase Search API [3] to retrieve entities for an input query string. Based on the introduction to the Search API, retrieved Freebase entities are ranked by a relevance score that is a function of its inbound and outbound link counts in Freebase and Wikipedia. We assume that the ranking quality is reasonably good, and we can retrieve relevant entities in top search results for most queries. However, Freebase also returns partially relevant entities. For example, for the input query "windows," Freebase returns highly relevant entities like *Microsoft Windows* and *Windows 7*, meanwhile it also returns partially relevant entities like *.NET Framework* and *Visual Basic* at lower positions. In this paper, we just want exactly matched entities in most cases. To select this kind of entity results, we use the following method to further select entities from the results returned by Freebase:

(1) Retrieve top five entities from Freebase. We retrieve only top five entities for each query, because our initial study shows that Freebase can return correct entities within top five results for most queries.

(2) Calculate similarity between each entity and the input query string. Because an entity may be presented in different ways, we use all aliases of the entity together with its name. In Freebase, there might be multiple aliases for an entity. For example, the entity *Microsoft Corporation* has aliases like "microsoft," "ms," and "msft". The similarity between an input query string qs and an entity e is calculated as follows.

$$Sim(e, qs) = \max_{a \in \text{alias}(e)} \frac{|a \cap qs|}{|a \cup qs|} \tag{1}$$

Here alias(e) is all the aliases of an entity e, $|a \cap qs|$ is the number of common terms shared by a and qs, and $|a \cup qs|$ is the number of terms that either appears in a or in qs.

(3) Remove the entities if $Sim(e, qs) < t_{sim}$, where t_{sim} is a threshold for controlling similarity.

(4) The remaining list of entities, represented by $E(qs)$, are assumed to be the retrieved entities for string qs.

We directly use query q issued to QDM$_{\text{KB}}$ as the input string to Search API and follow the steps above to obtain relevant entities set $E(q)$. Because a query may have various meanings, it is reasonable to keep multiple entities instead of just the top one.

3.2.2 Retrieving Related Properties

For each entity $e \in E(q)$, we use Freebase Topic API [4] to retrieve direct properties associated with their target entities $P_1(e)$ and further obtain second-hop properties $P_2(e)$.

Each Freebase entity belongs to different types, and each type contains different properties. For example, entity *Steven Spielberg* belongs to types *Film Producer (/film/producer)*, *Film Director (/film/director)*, *TV Producer (/tv/tv_producer)*, and etc. Type *Film Director* includes properties such as *Films Directed (/film/director/film)* which has targets entities like *Schindler's List*, *Saving Private Ryan*, and *War Horse*. Type *TV Producer* contains properties such as *TV Programs Produced (/tv/tv_producer/programs_produced)* which includes target entities like *Animaniacs*, *Band of Brothers*, and *Halo*. We call these properties and their target entities as **direct properties** which could be retrieved by Topic API.

In addition to direct properties, we further obtain "properties of properties" just like the example of *Beijing Subway*

3. https://developers.google.com/freebase/v1/search-cookbook

4. https://developers.google.com/freebase/v1/topic-overview

showed in Section 1. For *Steven Spielberg*, we can further retrieve properties for each of his directed films. All these films have a property *Actors (/film/film/starring)*. Then *All Actors of Directed Films* is a **second-hop properties** of *Steven Spielberg*. We can iteratively continue this process to generate multi-hop properties. The more hops we use, the more related entities we find, but the less relevant the retrieved entities are. All the retrieved properties and target entities compose a radial graph that centers on the source entity which, in the case above, is *Steven Spielberg*. Considering the trade-off between recall and precision, we only use the direct and second-hop properties in this paper. We leave the exploration of using more hops to future work.

Assume that p is a direct property of entity e, $V(e,p)$ is all the target entities the property contains for entity e, $P_1(e)$ is a set of triples with the form of <e, p, $V(e,p)$>. Let $e_t \in V(e,p)$ is one of the target entities and $p^{'}$ is a direct property of e_t. We use the following method to construct second-hop properties $P_2(e)$ for entity e.

Sometimes the same direct property of peer target entities may include duplicate entities. For example, *Saving Private Ryan* and *War Horse* are two target entities belonging to the property *Films Directed* of entity *Steven Spielberg*. Property *Genres (/film/film/genre)* of *Saving Private Ryan* has target entities including *War Film*, *Action Film*, and *Drama*, whereas the corresponding property of *War Horse* includes *War Film* and *Drama*. In this case, we tend to merge all *Genres* of different films to form one second-hop property, i.e., *All Genres of Directed Films* of *Steven Spielberg*. That is to say, if properties $p^{'}$ of each entity e_t in $V(e,p)$ share many common target entities, we tend to treat all these $p^{'}$ as a whole. From the viewpoint of source entity *Steven Spielberg*, the property *All Genres of Directed Films* makes more sense than many separate properties such as *Genres of His Saving Private Ryan* and *Genres of His War Horse*. In contrary, if properties $p^{'}$ vary from each other, such as *Actors* of different films directed by *Steven Spielberg*, each of them will be treated as a separate second-hop property. We use <e, $p \to p^{'}$, $\cup_{e_t \in V(e,p)} V(e_t, p^{'})$> to denote merged second-hop properties. Separate second-hop property is denoted by <e, $p \overset{e_t}{\to} p^{'}$, $V(e_t, p^{'})$> where $p \overset{e_t}{\to} p^{'}$ indicates that this property is associated with the entity $e_t \in V(e,p)$. We calculate a diversity score for each second-hop property as follows.

$$Diversity(e,p,p^{'}) = \frac{|\bigcup_{e_t \in V(e,p)} V(e_t,p^{'})|}{\sum_{e_t \in V(e,p)} |V(e_t,p^{'})|} \quad (2)$$

We create separate second-hop properties $p \overset{e_t}{\to} p^{'}$ if $Diversity(e,p,p^{'})$ is larger than 0.5 which means that if on average each target entity occurs less than twice among $p^{'}$ of every $e_t \in V(e,p)$, separate second-hop properties are constructed; otherwise, we create a merged one $p \to p^{'}$ which contains all unique target entities from $p^{'}$ of every $e_t \in V(e,p)$. All merged and separate second-hop properties make up set $P_2(e)$.

For each $e \in E(q)$, we obtain both $P_1(e)$ and $P_2(e)$, $P(e) = P_1(e) \cup P_2(e)$, $P(q) = \bigcup_{e \in E(q)} P(e)$. We remove all the triples in $P(q)$ whose $|V(e,p)| <= 1$ which means that the property has no more than single value. Examples of single-value properties for entity *Steven Spielberg* include *Nationality (/people/person/nationality)* and *Gender (/people/person/gender)*. We remove all single-value properties because they don't have enough entities to form a facet. All left properties are facet candidates.

3.3 Facet Expansion

Given a query q and its top search results $D(q)$, assume $F(q)$ are initial query facets extracted by QDMiner. Recall that QDMiner is a system that can automatically mine query facets by extracting and clustering frequent lists from top search results. As introduced in Section 1, QDMiner may suffer from the limitation of the search results. We propose further expanding the facets to find more peer items by leveraging a knowledge base.

Assuming that $f \in F(q)$ is an initial facet generated by QDMiner. We propose two different ways to expand f based on Freebase, namely **property based** and **type based facet expansion**, as introduced in Section 3.1.

3.3.1 Facet Expansion based on Properties

Using the method introduced in Section 3.2.1and 3.2.2, we obtain all properties $P(q)$ for query q. Note that these properties include the direct properties, the merged second-hop properties, and the separate second-hop properties. For each initial facet $f \in F(q)$, we try to find the most suitable property $p \in P(q)$ that can cover most items in f.

For each entity $e \in E(q)$, we use the following equation to calculate the ranking score of each property $p \in P(e)$:

$$score(e,f,p) = idf(p) \cdot sim(e,f,p) \quad (3)$$

Where $idf(p)$ is the inverted entity frequency for the property. We have $idf(p) = \log \frac{|E|}{|V(e,p)|}$ where $|E|$ is the total number of entities contained in Freebase. $|V(e,p)|$ is the number of target entities of property p. We assume that if property p has a large number of target entities, then p tends to be a common property. A common property is less distinguishable and useless to be a facet candidate. $sim(e,f,p)$ is the similarity between the facet and the property.

$$sim(e,f,p) = \frac{\sum_{I_i \in f} I(|E(I_i) \cap V(e,p)| > 0)}{|f|} \quad (4)$$

We use the same method introduced in Section 3.2.1 to retrieve entities for each facet item. $|f|$ is the number of items contained in the initial facet. For each $I_i \in f$, if $E(I_i)$ has overlap with $V(e,p)$, $I(|E(I_i) \cap V(e,p)| > 0)$ is 1. The numerator is the count of facet items which could be found in property p. We remove the properties with $sim(e,f,p) < t_{overlap}$, then rank all left properties base on Equation (3) and select the top 1 as final property assignment result.

Table 3 shows a list of property candidates ranked by $score(e,f,p)$. The first initial facet includes all the operating systems under query "windows". According to $score(e,f,p)$, the most probable underlying property is a second-hop property of entity *Windows* which includes all the operating systems developed by *Microsoft*. It extends raw facet size from 32 to 50. The property in second position is *Softwares* of *Microsoft* which is too broad and thus has a lower $idf(p)$. The third property is a direct property *Versions*

Included of *Windows* which doesn't cover most items in initial facet. The second example shows an initial facet of Beijing tourist attractions and the best matching property we select is also a second-hop property of *Beijing Subway* which is exactly what we want. The property characterizes the relationship between queries and facets, so that it could be used as a potential label of the facet which helps user understand the underlying meaning.

3.3.2 Facet Expansion based on Types

If $P(q)$ is empty or no suitable property is found, the method introduced in Section 3.3.1 fails. In this section, we propose directly finding similar items for a facet without the premise that query matches entities in Freebase. More specifically, we try to find a formalized query that can be used to retrieve similar entities. First, we select the most possible type for the facet; second, we discover some constraints that can narrow the scope; finally, we combine the type and constraints to make up a single MQL query to retrieve more candidates from Freebase. For example, assume that the initial facet is "E.T.," "Saving Private Ryan," etc. for query "Spielberg's movies". By checking the common types shared by these movies, we may select *Film (/film/film)* as the main type. This type almost covers all facet items, but it is too broad and contains many irrelevant films of which the director is not *Spielberg*. If we figure out that all items in the initial facet are directed by *Spielberg*, we can use (*Spielberg* ∈ *Directors*) as a constraint to narrow down the search scope which means that the retrieved entities must have the property *Directors* and *Spielberg* must appear in the target values of this property. This might significantly improve precision of expanded facet items, as well as system efficiency because much less entities are retrieved.

(1) Type Weighting

Recall that $I_i \in f$ is a facet items in facet f, and $E(I_i)$ is the corresponding entities retrieved from Freebase, and $e \in E(I_i)$ is one of these entities. Assume that $T(e)$ is the list of types of entity e. We use the following equation to calculate the weight of a type t for facet f.

$$score(f,t) = idf(t) \cdot weight(f,t) \tag{5}$$

$$weight(f,t) = \frac{1}{|f|} \sum_{I_i \in f} \sum_{e \in E(I_i)} \sum_{t_e \in T(e)} \frac{I(t,t_e)}{\sqrt{R(I_i,e)}\sqrt{R(f,I_i)}} \tag{6}$$

Where $I(t,t_e) = 1$ if t equals to t_e, i.e., e belongs to type t. $R(I_i,e)$ denotes the rank of entity e within the retrieved entities for facet item I_i. $R(f,I_i)$ is the rank of facet item I_i within the initial facet. The higher a facet item and an entity are ranked, the higher the weights of corresponding types of the entity are.

Similar to $idf(p)$, $idf(t)$ is the inverted entity frequency of type t. We have $idf(t) = \log \frac{|E|}{|E_t|}$ where $|E_t|$ is the number of entities belonging to type t.

We remove all the types whose $weight(f,t) < t_{overlap}$, then rank left ones based on Equation (5), and select the top 1. If no types are left, we directly output the initial facet.

Figure 2 shows the types of entities in Freebase. Most types are too broad. The most fine-grained type of *Schindler's List* is *Biographical Film* which fails to capture the relationship between facets and the query "Steven Spielberg."

(2) Query Dependent Constraint

As we mentioned before, if we directly retrieve all the entities of the selected type (for example, all films), there might be a large number of entities irrelevant to the query context, which is also the biggest difference between set expansion and query facet expansion. To utilize query context, we need to generate a more specific query to narrow down the scope of entities. We try to generate a MQL query for retrieving entities which belong to the selected type and have some common properties with the same target entities. For example, for a facet comprised of "Saving Private Ryan," "E.T.," etc., we try to generate a MQL query like:

```
[{

    "/film/film/director": [{
        "mid": "/m/06pj8"  //Steven Spielberg
    }]
    "type": "/film/film",
}]
```

For each facet item $I_i \in f$, we retrieve all its properties $P_1(I_i)$ from Freebase using the method introduced in Section 3.2.2. The reason why we don't use $P_2(I_i)$ is that initial facets usually have many items, and it is costly to further construct second-hop properties for each item. We then transform all the properties and their values into a list of pairs <p, e_t> which are comprised of a property p, and a target entity e_t. For example, the property *Directed By (/film/film/directed_by)* of entity *Saving Private Ryan* is transformed to pair </film/film/directed_by, *Steven Spielberg*>. Similarly, *E.T.* has the same pair </film/film/directed_by, *Steven Spielberg*> as *Saving Private Ryan*. For each facet item $I_i \in f$, we obtain a set of pairs. We then check each pair, and select the ones whose target entity e_t is related to the query q. We assume that entity e_t is related to the query q if e_t has at least one alias that is contained by the query, i.e., $\exists a \in \text{alias}(e_t),\ a \cap q = a$. By confining the target entity of the pair to be related to query, we characterize the relationship between the facet and the query and thus guarantee expansion precision.

We then calculate the weight of each pair using the following equation. If a pair is included by many facet items just like the example above, it has a high weight.

$$weight(f,p,e_t) = \frac{1}{|f|} \sum_{I_i \in f} \sum_{e \in E(I_i)} \frac{HasPair(e,p,e_t)}{\sqrt{R(I_i,e)}\sqrt{R(f,I_i)}} \tag{7}$$

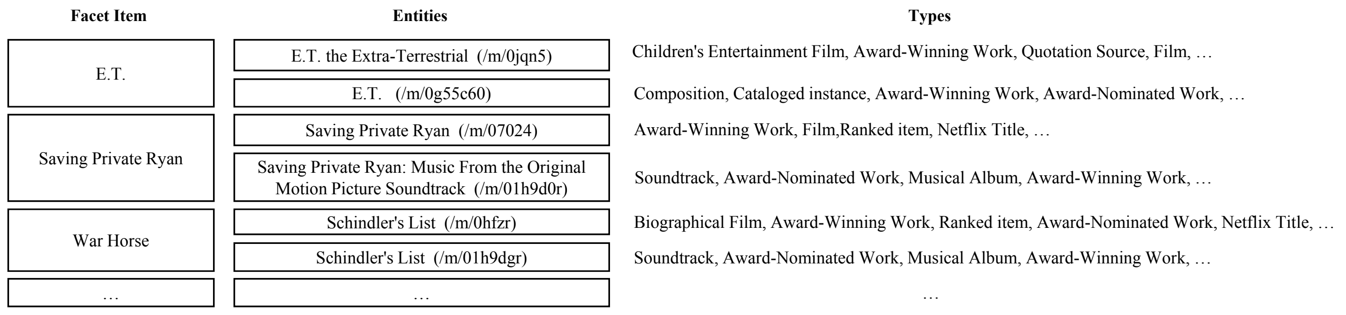
$HasPair(e,p,e_t) = 1$ if entity e has a property p containing a target entity e_t; otherwise, $HasPair(e,p,e_t) = 0$.

We remove pairs whose $weight(f,p,e_t) < t_{overlap}$. For each e_t among left pairs, we select the top 1 (p, e_t) pair to generate query constraints in the final MQL query (the final MQL may contain multiple pair constraints). If no candidate pairs are left, we assume that there is no consistent constraint for this type and in this case, we use all the entities of the type for facet expansion.

An alternative for constructing constraints is to use traditional relation extraction methods. For example, the simplest way is to identify entity *Spielberg* and his movies in the

TABLE 3
Property based facet expansion.
All properties are in the form of triple <source entity, property, target entities> and ranked by their scores.
Note that property could be direct or second-hop. Each second-hop property has an arrow.
The first property of "windows" means that *Windows* has a property *Developer* with a value *Microsoft*. *Microsoft* has a property *Developed OS*.

query: **windows** facet: **windows vista, windows xp, windows 7, windows 2000, windows nt 3.1, windows 95, windows 98, windows server 2008, ...** (32 total)
<*Windows*, /developer $\xrightarrow{Microsoft}$ /operating_systems_developed, [*MS-DOS, Windows 98, Windows XP, Windows 7, Windows Server, ...* (50 total)]>
<*Windows*, /developer $\xrightarrow{Microsoft}$ /software, [*Outlook, Office, PowerPoint, SharePoint, WordPad, Visual C++, IE, Window 7, ...* (329 total)]>
<*Windows*, /includes_os_versions, [*Windows 1.0, Windows 2.0, Windows NT, Windows XP, Windows Vista, Windows 7, ...* (11 total)]>
query: **beijing subway** facet: **forbidden city, temple of heaven, tiananmen square, summer palace, the forbidden city, the great wall, lama temple, ...** (8 total)
<*Beijing Subway*, /area_served $\xrightarrow{Beijing}$ /tourist_attractions, [*Forbidden City, Summer Palace, Ming tombs, Beihai Park, Tiananmen Square, ...* (33 total)]>
<*Beijing Subway*, /area_served $\xrightarrow{Beijing}$ /location, [*Beijing National Stadium, Peking University, Tanzhe Temple, Wangfujing, ...* (309 total)]>
query: **michael jackson** facet: **you rock my world, butterflies, man in the mirror, thriller, cry** (5 total)
<*Michael Jackson*, /nominated_work, [*Beat It, Billie Jean, We Are the World, Bad, Man In the Mirror, Earth Song, Dangerous, ...* (168 total)]>
<*Michael Jackson*, /track_contributions [*Who Is It, I Can't Help It, Off the Wall, Will You Be There, Cry, In the Closet, ...* (151 total)]>
<*Michael Jackson*, /tracks_produced, [*Billie Jean, Get on the Floor, Bad, Heaven Can Wait, Black or White, Muscles, Threatened, ...* (113 total)]>



Facet Item	Entities	Types
E.T.	E.T. the Extra-Terrestrial (/m/0jqn5)	Children's Entertainment Film, Award-Winning Work, Quotation Source, Film, …
	E.T. (/m/0g55c60)	Composition, Cataloged instance, Award-Winning Work, Award-Nominated Work, …
Saving Private Ryan	Saving Private Ryan (/m/07024)	Award-Winning Work, Film,Ranked item, Netflix Title, …
	Saving Private Ryan: Music From the Original Motion Picture Soundtrack (/m/01h9d0r)	Soundtrack, Award-Nominated Work, Musical Album, Award-Winning Work, …
War Horse	Schindler's List (/m/0hfzr)	Biographical Film, Award-Winning Work, Ranked item, Award-Nominated Work, Netflix Title, …
	Schindler's List (/m/01h9dgr)	Soundtrack, Award-Nominated Work, Musical Album, Award-Winning Work, …
…	…	…

Fig. 2. Freebase entities and types corresponding to items of facet "directed films" of query "Spielberg."
"E.T." matches two entities, a film(/m/0jqn5) and a composition(/m/0g55c60). The music's types include *Composition*, *Cataloged instance*, etc.

text and extract their underlying relation "director of" by some predefined patterns. To retrieve more similar entities from Freebase, we need to use structured query language (such as MQL). Although the relationship extracted may be reasonable, how to translate or map the textual form relation to structured query remains a problem. How to employ relation extraction methods is our future work.

Table 4 shows three examples of type based facet expansion. The first one is a facet of characters of TV program *General Hospital*. The best matching type *TV Character* is too broad to be a query facet. By utilizing the property constraint, the characters are required to appear in program *General Hospital*, which is more specific and precise. The second case is a facet consisting of actors in film *Eclipse*. Type *Celebrity* is too crude to characterize the relationship while the property constraint does. Note that our method accommodates the situation where initial facets contain noises such as "new moon" because the foundation of our approach is voting. The last example uses three property constraints to describe the relationship between the query "chinese women tennis player" and the facet, and achieves an exact comprehension of the raw query.

A problem worth consideration is the ambiguity problem. The entities we retrieve for each facet item may not correspond to the genuine meaning. Regarding a facet which is comprised of "Apple," "Samsung," "Nokia," "Blackberry," "LG," etc., for the item "Apple", we may retrieve the entity *Apple (fruit)*. However, by using voting based scores, the ambiguity problem is addressed potentially. Assume that only a small portion of items have ambiguous meanings, the effect of ambiguity could be solved because the voices of ambiguity entities are too weak to make a difference. In the example above, although "Apple" has an ambiguous meaning *Apple (fruit)* whose type is *Fruit (/user/rcheramy/default_domain/fruit)*, all the other items tell that the facet is about *Mobile Phone Brands (/user/robert/mobile_phones/mobile_phone_brand)*.

3.4 Facet Grouping

After facet generation and expansion, we collect a set of facet candidates which consist of (1) new facet candidates generated from Freebase; (2) facet candidates expanded by the property based and type based methods. There might be duplicate items within these facet candidates, hence we need to further group these facets. We use the QT algorithm (Quality Threshold) to cluster facet candidates [40]. QT is a clustering algorithm that groups data into high quality clusters. Compared to other clustering algorithms, QT ensures quality by finding large clusters whose diameters don't exceed a user-defined diameter threshold. This method prevents dissimilar data from being forced under the same cluster and ensures good quality of clusters. In

TABLE 4
Type based facet expansion.
For the first example, retrieved entities must belong to type *TV character* and appear in *General Hospital*.

	query: **general hospital recaps** facet: **sonny corinthos, jason morgan, lucky spencer, luke spencer, dante falconeri, nikolas cassadine, ...**
type constraints	/tv/tv_character </appeared_in_tv_program, *General Hospital*>
	query: **eclipse** facet: **kristen stewart, robert pattinson, twilight, taylor lautner, breaking dawn, new moon, ashley greene, ...**
type constraints	/celebrities/celebrity </performance_in, *Eclipse*>
	query: **chinese women tennis players** facet: **zheng jie, peng shuai, li na**
type constraints	/tennis/tennis_tournament_champion </nationality, *China*>, </gender, *Female*>, </profession, *Tennis player*>

QT, the number of clusters is not required to be specified. We use the complete linkage for calculating the diameter, i.e., diameter is calculated using the longest distance among all the points in the cluster.

$$Dia(C) = \max_{f_1 \in C, f_2 \in C} Dis(f_1, f_2) \tag{8}$$

And the distance $Dis(f_1, f_2)$ between facet f_1 and facet f_2 is calculated as follows.

$$Dis(f_1, f_2) = \frac{|f_1 \cap f_2|}{\max |f_1|, |f_2|} \tag{9}$$

3.5 Facet Weighting

For each final facet f, we calculate a weight to evaluate its importance. Consistent with QDMiner, we assume that a good facet is usually supported by many websites and appear in many documents. A good facet contains items that are informative to the query. Therefore, we evaluate the importance of each facet f by the following components:

(1) S_{DOC}: **result matching weight**. Good facet item should *frequently* occur in *highly ranked* results. We let $S_{\text{DOC}} = \sum_{d \in R} (s_d^m \cdot s_d^r)$, where $s_d^m \cdot s_d^r$ is the supporting score by each result d, and:

- s_d^m **is the percentage of items contained in** d. A facet f is supported by a document d, if d contains some facet items of f. The more items d contains, the stronger it supports f. Suppose $N_{f,d}$ is the number of items which appear both in list f and document d, and $|f|$ is the number of facet items contained in facet f, we let $s_d^m = \frac{N_{f,d}}{|f|}$.
- s_d^r **measures the importance of document** d. The documents ranked higher in the original search results are usually more relevant to the query, hence they are more important. We simply let $s_d^r = 1/\sqrt{rank_d}$, where $rank_d$ is the rank of document d. The higher d is ranked, the larger its score s_d^r is.

(2) S_{IDF}: **average invert document frequency (IDF) of items**. A list comprised of common items in a corpus is not informative to the query. We calculate the average IDF value of all items, i.e., $S_{\text{IDF}} = \frac{1}{|f|} \cdot \sum_{e \in f} idf_e$. Here $idf_e = \log \frac{N - N_e + 0.5}{N_e + 0.5}$, where N_e is the total number of documents that contain item e in the corpus and N is the total number of documents. We use the ClueWeb09 collection[5], which includes about one billion webpages, as our reference corpus in counting N_e and N.

We combine the components above, and evaluate the importance of a facet f by Equation (10).

$$S_f = S_{\text{DOC}} \cdot S_{\text{IDF}} \tag{10}$$

Finally, we sort all final facets by weights to form output of QDM$_{\text{KB}}$. The reason why we use search results instead of knowledge bases to weight and rank each facet is that search results reflect users' attention and intent. We don't have much information about the popularity of entities in the current Freebase. Recall the example used in Section 3. If a user searches "mobile phone" , both *Apple* and *Samsung* are facet item candidates which, in Freebase, are just two distinct nodes with similar properties and types. While by search results, we could confirm which one appears more frequently to justify its importance.

4 Experimental Setup

4.1 Datasets

We use the same datasets as QDMiner to evaluate our approach. The first dataset, namely, **UserQ**, is comprised of 89 queries issued by QDMiner users. The second dataset is **RandQ** which includes 105 randomly sampled English queries from a query log of a commercial search engine.

We construct the ground truth as follows. For each query in both dataset, ground truth facets are manually created by annotators who need to do careful survey on the topics related to the query. Then, each facet is rated by at least five subjects in three levels, namely Good, Fair, and Bad. Based on our investigation, the facets created by human annotations can cover most useful facets returned by different algorithms. In fact, the ground truth used in this paper is built based the on one used in [4]. Because we introduce more baseline methods and propose QDM$_{\text{KB}}$, some unlabeled items emerge. After adding these unlabeled items to ground truth, the newly constructed ground truth can be viewed as a superset of the old one, which causes the difference of experiment result of QDMiner (rp-nDCG decrease from 0.212 to 0.183 as shown in Table 6). Note

5. http://boston.lti.cs.cmu.edu/Data/clueweb09/

TABLE 5
Statistics of datasets.

Item	UserQ	RandQ	Item	UserQ	RandQ
#queries	89	105	#docs per query	99.8	99.5
#direct-entity queries	47	55	#lists per doc	44.1	37.0
#general queries	42	50	#items per list	9.7	10.1

that all the annotations are made according to SERP of each facet item returned by search engine rather than knowledge bases, so there is no association between the construction of ground truth and experimental performance.

On average, each query in UserQ has 4.9 Good, 5.3 Fair, 4.4 Bad facets, while each query in RandQ has 2.9 Good, 2.1 Fair, 2.1 Bad facets. The reason why queries in UserQ have more facets is that queries in RandQ are sampled from logs of a commercial search engine which have many meaningless queries. In experiment section, we split each dataset into two parts, namely direct-entity and general queries. Direct queries are the queries matching entities in Freebase, i.e., $|E(q)| > 0$. General queries are the left ones, i.e., $|E(q)| = 0$. The detailed statistics of the datasets are displayed in Table 5. The datasets were published at http://www.playbigdata.com/qd2/datasets.aspx. More information regarding the datasets can be found in [4].

4.2 Evaluation Metrics

Similar to QDMiner [4], we follow two aspects to evaluate query facets, namely quality of clustering and ranking effectiveness. For clustering quality, we use existing metrics such as NMI. For ranking effectiveness, we use nDCG and some tuned editions of nDCG to take precision and recall into consideration. To evaluate ranking effectiveness, each facet f is assigned a relevance score by aligning this facet to a ground truth facet $f^{'}$ which covers maximum number of items in f. Because a ground truth facet may be assigned to multiple facets, the nDCG (including fp-nDCG and rp-nDCG) scores may exceed 1. So we follow [4] to credit only the highest assigned facet, and skip all later ones. The metrics are listed as follows.

NMI - *Normalized Mutual Information*. We use this metric to evaluate whether facet items are split into correct facets.

nDCG - *Normalized Discounted Cumulative Gain*. nDCG measure is widely used in information retrieval. This measure is appropriate to evaluate the ranking of query facets.

fp-nDCG - *purity aware nDCG based on the first appearance of each class*. Suppose that each output facet f_i is assigned to a labeled class $f_i^{'}$. Then the ranking quality of top p facets is calculated by $nDCG_p = \frac{DCG_p}{IDCG_p}$, where $DCG_p = \sum_{i=1}^{p}(w_i \cdot DG_i)$, $w_i = \frac{f_i^{'} \bigcap f_i}{f_i}$, and $IDCG_p$ is the ideal ordering score. DG_i is the discounted gain of the i^{th} facet, and $DG_i = \frac{2^{r_i}-1}{log_2(1+i)}$, where r_i is $f_i^{'}$'s ranking score.

rp-nDCG - *recall and purity aware nDCG*. rp-nDCG is calculated based on all output facets, and we let $w_i = \frac{f_i^{'} \bigcap f_i}{f_i} \frac{f_i^{'} \bigcap f_i}{f_i^{'}}$ for rp-nDCG.

F1-nDCG - *F1 aware nDCG*. F1-nDCG is similar with rp-nDCG, except that $w_i = \frac{2p_i r_i}{p_i + r_i}$, $p_i = \frac{f_i^{'} \bigcap f_i}{f_i}$, $r_i = \frac{f_i^{'} \bigcap f_i}{f_i^{'}}$.

We further use the evaluation metrics ***PRF*** and ***wPRF*** proposed by Kong and Allan [6]. PRF is a harmonic mean of precision of facet terms, recall of facet terms and facet clustering F1. wPRF further takes into account the different ratings associated with query facets, and it uses weighted facet term precision, recall, and clustering F1. Note that neither PRF nor wPRF accounts for facet ranking effectiveness.

We calculate the evaluation metrics above based on the top 5 facets for each query and use two-tailed paired t-test for statistically significance testing with p-value lower than 0.01. In all the following experiment result tables, significance test results compared to different baselines are marked using various symbols.

4.3 Experimental Settings

We implement several existing approaches to verify whether QDM_{KB} could outperform the state-of-the-art algorithms. We implement **QDMiner** [4], the first query facet mining algorithm, and the supervised method **QF-I** and **QF-J** proposed by Kong and Allan [6]. As the problem of facet expansion is related to the problem of set expansion, we also experiment with expanding facets using **SEAL** which is one of the state-of-the-art set expansion methods.

For QDMiner, we mine query facets based on top 100 results from a commercial search engine, which is consistent with [4]. For QF-I and QF-J, we use the same lists extracted from search results as QDMiner for fair comparison. We use the same features, standardization method, and stratification method, as those in [6]. For QDMiner, QF-I, and our proposed method QDM_{KB}, we use 5-fold cross validations and tune corresponding parameters based on rp-nDCG. We use rp-nDCG as the main metric because it considers every aspect including ranking quality, clustering quality, and item recall of generated facets. For QDM_{KB}, we tune $t_{overlap}$ used in Section 3.3. Based on our observation, we empirically set $t_{sim} = 0.3$ used in Section 3.2.1 which could filter apparently not exactly matched entities. For QDMiner, we tune the clustering parameters - the diameter threshold for a cluster and the weight threshold for a valid cluster. For QF-I, we tune the weight threshold for facet terms and the diameter threshold for clustering. For QF-J, there are no parameters that need to be tuned.

5 Experimental Results

5.1 Overall Results

We show the overall results of our proposed method and baselines on UserQ and RandQ in Table 6 and 7 respectively.

(1) **QDM_{KB} significantly outperforms all baselines in terms of rp-nDCG and F1-nDCG, on both UserQ and RandQ**. For UserQ, rp-nDCG is improved to 0.239 which surpasses strongest baseline QDMiner's 0.183 by about 30%. P value of t-test is 6.249E-5, t value is 4.205, the degree of freedom df is 88. F1-nDCG is improved to 0.438 from 0.402 by about 9%, which is also a significant improvemet. For RandQ, QDM_{KB} raises rp-nDCG to 0.285 compared to QDMiner's 0.234 which is a huge promotion. P value of t-test is 6.981E-7, t value is 5.285, the degree of freedom df is 104. F1-nDCG is improved to 0.502 from 0.463 by about 8%, which is significant. This means that the overall quality of

TABLE 6
Overall results on UserQ. The best result is in bold.
Pre is precision. Rec is recall.

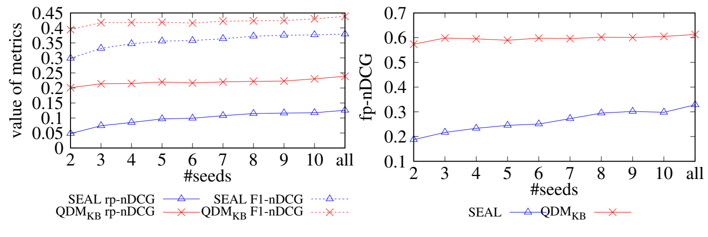
Metric	NMI	F1	Pre	Rec	nD CG	fp-n DCG	**rp-n DCG**	F1-n DCG	PRF	wPRF
QDM$_{\mathrm{KB}}$	$\mathbf{.841}^{\dagger}_{\circ}$	$.339^{\star\ddagger}_{\circ}$	$.711^{\dagger\ddagger}_{\circ}$	$.236^{\star}_{\ddagger}$	$.807^{\dagger}_{\ddagger}$	$.613^{\dagger\ddagger}_{\circ}$	$\mathbf{.239}^{\star\dagger}_{\ddagger\circ}$	$\mathbf{.438}^{\star\dagger}_{\ddagger\circ}$	$\mathbf{.410}^{\star\ddagger}_{\circ}$	$.428^{\star\ddagger}_{\circ}$
QDMiner*	.820	.270	**.796**	.169	**.821**	**.620**	.183	.402	.333	.349
QF-I†	.736	**.350**	.514	.284	.589	.334	.136	.317	.403	**.431**
QF-J‡	.821	.253	.506	.179	.533	.383	.119	.243	.315	.336
SEAL$^{\circ}$	.754	.270	.265	**.334**	.793	.328	.126	.379	.324	.376

TABLE 7
Overall results on RandQ.

Metric	NMI	F1	Pre	Rec	nD CG	fp-n DCG	**rp-n DCG**	F1-n DCG	PRF	wPRF
QDM$_{\mathrm{KB}}$	$\mathbf{.800}^{\star}_{\dagger}$	$\mathbf{.412}^{\star\dagger}_{\ddagger\circ}$	$\mathbf{.608}^{\dagger\ddagger}_{\circ}$	$.380^{\star\dagger}_{\ddagger}$	$.832^{\dagger}_{\ddagger}$	$.604^{\dagger\ddagger}_{\circ}$	$\mathbf{.285}^{\star\dagger}_{\ddagger\circ}$	$\mathbf{.502}^{\star\dagger}_{\ddagger\circ}$	$\mathbf{.473}^{\star\dagger}_{\ddagger\circ}$	$\mathbf{.494}^{\star\dagger}_{\ddagger\circ}$
QDMiner*	.758	.301	.592	.251	**.837**	**.611**	.234	.463	.354	.377
QF-I†	.639	.297	.320	.352	.579	.275	.138	.332	.349	.388
QF-J‡	.771	.171	.282	.161	.511	.318	.108	.225	.223	.251
SEAL$^{\circ}$	.768	.207	.166	**.423**	.822	.320	.168	.425	.261	.319

facets mined by our algorithm is higher than those mined by QDMiner, QF-I and QF-J. This confirms the effectiveness of leveraging knowledge bases for mining query facets. The high-quality information contained in knowledge bases is helpful for generating better query facets. The average size of query facets in QDMiner is 26.4. QDM$_{\mathrm{KB}}$ improves it to 36.1.

(2) QDM$_{\mathrm{KB}}$ gets comparable results with QDMiner in terms of fp-nDCG. There is a slight drop from 0.62 to 0.613 on UserQ and from 0.611 to 0.604 on RandQ. It should be emphasized that there is a trade-off between precision and recall. The more entities we expand, the more risk including unrelated items, which harms fp-nDCG. In the first example of Table 8, QDM$_{\mathrm{KB}}$ correctly uses type-based expansion to add more mobile phone brands to the initial facet, which improves facet recall as well as precision because the initial facet contains a noise "iphone." While in the second example, QDM$_{\mathrm{KB}}$ mistakenly aligns the initial facet which indicates components of mobile phone to a meaningless property. Although slightly inferior to QDMiner, our method significantly outperforms other baselines on fp-nDCG. This means that the ranking and clustering quality are as good as original methods. Recall is improved without significant loss of precision. We further calculate precision, recall, and F1 on facet item level (different from [4], [7] where F1 indicates clustering F1). Specifically, on UserQ, precision drops from 0.796 to 0.711 by about 11%, while recall increases from 0.169 to 0.236 by about 40%; on RandQ, QDM$_{\mathrm{KB}}$ gets comparable precision and significantly higher recall than QDMiner. QDM$_{\mathrm{KB}}$ gets worse precision than QDMiner on UserQ. After investigation, we find this is because that queries in UserQ are more entity-like and more expanded entities are included. The more entities we expand, the more risk including unrelated items, which harms precision. How to more accurately characterize the relationship between facets and queries needs our further effort. However, compared to QF-I and SEAL which have high recall and significantly lower precision, QDM$_{\mathrm{KB}}$



(a) rp/F1-nDCG with various #seed (b) fp-nDCG with various #seed

Fig. 3. Performance comparison of QDM$_{\mathrm{KB}}$ and SEAL

achieves a reasonable balance between these two aspects. To take ranking quality into consideration, we calculate F1-nDCG. QDM$_{\mathrm{KB}}$ gets highest F1-nDCG (0.438 and 0.502 on UserQ and RandQ respectively) and significantly outperforms other baselines.

(3) QDM$_{\mathrm{KB}}$ significantly outperforms QF-I in terms of rp-nDCG, F1-nDCG, and fp-nDCG on both datasets. In terms of PRF and wPRF, QDM$_{\mathrm{KB}}$ shows comparable results on UserQ and superior results on RandQ compared to QF-I. These results indicate that the facets generated by QDM$_{\mathrm{KB}}$ have comparable recall of facet items with QF-I and much better ranking quality than QF-I. QDM$_{\mathrm{KB}}$ significantly outperforms QF-J, in terms of most metrics on both datasets.

(4) QF-J outperforms QF-I in terms of fp-nDCG and NMI, whereas QF-I outperforms QF-J in terms of rp-nDCG, F1-nDCG, PRF, and wPRF. These results indicate that QF-I has much higher facet item recall than QF-J while QF-J has much higher precision. This may be because that we tune QF-I based on rp-nDCG during the cross validation. There might be different results if QF-I is tuned based on fp-nDCG or other metrics. Note that QF-J does not have parameters to be tuned, hence we have no way to improve recall for QF-J.

(5) QDM$_{\mathrm{KB}}$ significantly outperforms SEAL, and SEAL underperforms QDMiner in terms of rp-nDCG, F1-nDCG, and fp-nDCG. We further experiment with different numbers of seeds, and show the results on UserQ in Figure 3 (Similar results are obtained on RandQ). Note that, number of seeds in QDM$_{\mathrm{KB}}$ means the number of items of each initial facet used for expansion (both type-based and property-based). QDM$_{\mathrm{KB}}$ consistently outperforms SEAL, no matter how many seeds are used. Typical set expansion method can't work well for facet expansion.

5.2 Experiments with Different Components

In this paper, we use two different approaches to construct facet candidates, namely facet generation and facet expansion. Facet expansion includes two parts, namely property based and type based expansion. We rename facet generation as **dir-property**. We rename property based facet expansion as **ex-property** and type based facet expansion as **ex-type**. Different components may play different roles within QDM$_{\mathrm{KB}}$. In this section, we experiment with one or combinations of these components. Experimental results are shown in Table 9 and 10. Note that **dir-property+ex-property** is the combination of two components involving using property and **ex-property+ex-type** combines two expansion components.

(1) QDM$_{\mathrm{KB}}$ with all components gets the best results in terms of rp-nDCG, F1-nDCG, PRF, and wPRF on both

TABLE 8
Precision case study. The precision of the first case rises as opposed to the second case.

	query: **mobile phones** facet: **nokia, samsung, apple, blackberry, htc, motorola, lg, sony ericsson, acer, alcatel, palm, iphone, dell** (12 total)
result	*Nokia, Motorola, HTC, Panasonic, Samsung, Nexus One, NTT DoCoMo, BlackBerry, Siemens, Hewlett-Packard, Apple Inc., T-Mobile, ...* (21 total)
way	use type *Mobile Phone Brand(/user/robert/mobile_phones/mobile_phone_brand)*
	query: **mobile phones** facet: **bluetooth, gps, camera, fm radio, microsd card slot** (5 total)
result	*Handheld GPS, Radio, Electric blanket, Lexmark, Chainsaw, Can opener, DVD player, Swiss Army knife, Spincast Fishing Reel, Flashlight, ...*(93 total)
way	use entity *Mobile Phone*'s property /award/ranked_item/appears_in_ranked_lists → /award/ranked_list/ranked_list_items

TABLE 9
Experimental results of different components on UserQ.

Metric	NMI	fp-nDCG	**rp-nDCG**	F1-nDCG	PRF	wPRF
QDM_{KB}	.841	.613^{①}	**.239**$^{①②③}_{④⑤\star}$	**.438**$^{①②}_{⑤\star}$	**.410**$^{①②}_{③\star}$	**.428**$^{①②③}_{④\star}$
QDMiner*	.820	.620	.183	.402	.333	.349
dir-property^{①}	**.883**	.484	.187	.333	.270	.283
ex-property^{②}	.827	.611	.205	.407	.381	.396
ex-type^{③}	.834	**.626**	.210	.418	.366	.383
dir-property+ ex-property^{④}	.829	.609	.215	.418	.392	.406
ex-property+ ex-type^{⑤}	.828	.605	.212	.412	.390	.406

TABLE 10
Experimental results of different components on RandQ.

Metric	NMI	fp-nDCG	**rp-nDCG**	F1-nDCG	PRF	wPRF
QDM_{KB}	.800$^{②③}_{④\star}$	.604^{①}	**.285**$^{①②③}_{④\star}$	**.502**$^{①②}_{\star}$	**.473**$^{①②③}_{④\star}$	**.494**$^{①②③}_{④\star}$
QDMiner*	.758	.611	.234	.463	.354	.377
dir-property^{①}	**.842**	.397	.182	.321	.224	.234
ex-property^{②}	.766	**.617**	.256	.478	.418	.438
ex-type^{③}	.781	.602	.264	.488	.394	.421
dir-property+ ex-property^{④}	.766	.605	.249	.475	.369	.392
ex-property+ ex-type^{⑤}	.785	.608	.276	.495	.444	.465

datasets. This means that each component can contribute to the final model. Different components may discover different facet items for different queries.

(2) Each single component could improve quality of query facets over QDMiner, except for dir-property. Dir-property underperforms QDMiner on both datasets in terms of F1-nDCG. This is because that direclty outputting properties may bring many noises.

(3) We get consistent rp-nDCG and F1-nDCG results, i.e., ex-type>ex-property>dir-property, on both datasets. This means that the type based expansion is better than the property based method, and both expansion methods are better than directly outputting properties. This may be because that the type based expansion method could work on more queries than the other two.

5.3 Experiments with Direct-entity Queries

We further experiment with different components on direct-entity queries. We denote queries that match entities in Freebase as direct-entity queries, such as "Nokia" and "Tom Cruise." 47 out of 89 queries in UserQ and 55 out of 105 queries in RandQ are direct-entity queries as shown in Table 5. Experimental results are shown in Table 11 and 12.

TABLE 11
Experiments with direct-entity queries on UserQ.

Metric	NMI	fp-nDCG	**rp-nDCG**	F1-nDCG	PRF	wPRF
QDM_{KB}	.856$^{\dagger}_{\circ}$	.631$^{①\dagger}_{\ddagger\circ}$	**.271**$^{③\star\dagger}_{\ddagger\circ}$	**.449**$^{①③\star}_{\dagger\ddagger\circ}$	**.430**$^{①③\star}_{\ddagger\circ}$	**.447**$^{①③\star}_{\ddagger\circ}$
dir-property^{①}	**.957**	.387	.207	.281	.193	.205
ex-property^{②}	.852	.628	.242	.423	.404	.421
ex-type^{③}	.844	**.654**	.219	.413	.352	.368
dir-property+ ex-property^{④}	.856	.624	.256	.443	.417	.432
ex-property+ ex-type^{⑤}	.849	.624	.248	.426	.415	.432
QDMiner*	.838	.644	.199	.413	.313	.330
QF-I†	.741	.345	.147	.328	.408	.436
QF-J‡	.839	.405	.128	.237	.292	.314
SEAL$^{\circ}$	.746	.344	.131	.381	.338	.388

TABLE 12
Experiments with direct-entity queries on RandQ.

Metric	NMI	fp-nDCG	**rp-nDCG**	F1-nDCG	PRF	wPRF
QDM_{KB}	.809$^{\star\dagger}_{\circ}$	.585$^{①\dagger}_{\ddagger\circ}$	.292$^{①\star\dagger}_{\ddagger\circ}$	.484$^{①\star\dagger}_{\ddagger\circ}$	**.533**$^{①③④\star}_{\dagger\ddagger\circ}$	**.557**$^{①③④\star}_{\dagger\ddagger\circ}$
dir-property^{①}	**.928**	.191	.136	.177	.114	.123
ex-property^{②}	.783	**.612**	.279	.478	.485	.513
ex-type^{③}	.782	.589	.256	.464	.392	.425
dir-property+ ex-property^{④}	.782	.587	.256	.465	.392	.425
ex-property+ ex-type^{⑤}	.796	.610	**.294**	**.488**	.508	.535
QDMiner*	.768	.600	.236	.449	.362	.396
QF-I†	.678	.230	.111	.273	.346	.382
QF-J‡	.776	.308	.093	.189	.213	.247
SEAL$^{\circ}$	.749	.289	.148	.390	.258	.318

Again, QDM_{KB} with all components gets almost the best results in terms of rp-nDCG, F1-nDCG, PRF, and wPRF on both datasets, which is consistent with the results on all queries. This further confirms each component has its unique contribution. On RandQ, QDM_{KB} is slightly worse than expansion component, which indicates that dir-property has high risk to include irrelevant items.

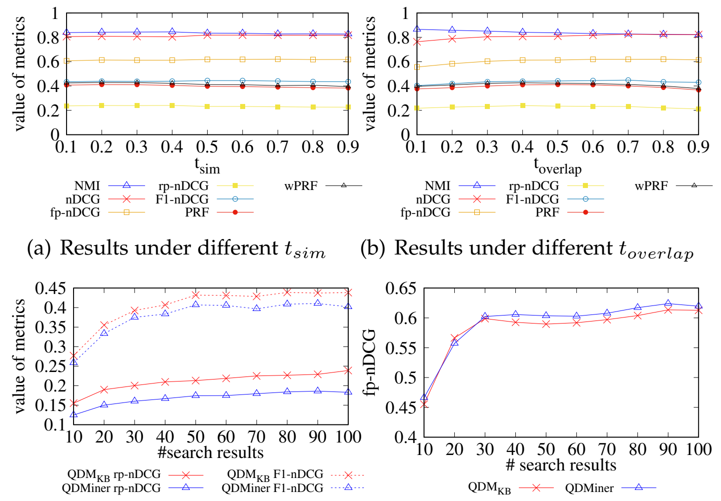
Dir-property yields the best performance on NMI. This is reasonable because a separate facet is created for each property, and hence the clustering quality of generated facets is higher than other methods.

TABLE 13
Experiments with general queries on UserQ.

Metric	NMI	fp-nDCG	**rp-nDCG**	F1-nDCG	PRF	wPRF
QDM$_{KB}$	**.824**$^{\dagger}_{\circ}$	**.596**$^{\dagger\ddagger}_{\circ}$	**.196**$^{\dagger\ddagger}_{\circ}$	**.420**$^{\dagger}_{\ddagger}$	.380°	.398
QDMiner*	.800	.592	.165	.390	.356	.368
QF-I†	.730	.322	.123	.304	**.399**	**.404**
QF-J‡	.801	.358	.108	.251	.341	.342
SEAL$^{\circ}$	.762	.312	.119	.377	.308	.374

TABLE 14
Experiments with general queries on RandQ.

Metric	NMI	fp-nDCG	**rp-nDCG**	F1-nDCG	PRF	wPRF
QDM$_{KB}$	**.806**†	.611$^{\dagger\ddagger}_{\circ}$	**.281**$^{\star\dagger}_{\ddagger\circ}$	**.521**$^{\star\dagger}_{\ddagger}$	**.397**$^{\star\ddagger}_{\circ}$	**.415**$^{\star\ddagger}_{\circ}$
QDMiner*	.747	**.623**	.232	.479	.344	.355
QF-I†	.636	.324	.167	.397	.351	.395
QF-J‡	.766	.329	.125	.264	.233	.242
SEAL$^{\circ}$	.788	.353	.190	.463	.264	.320



(a) Results under different t_{sim} (b) Results under different $t_{overlap}$

(c) rp/F1-nDCG with various #results (d) fp-nDCG with various #results

Fig. 4. Experimental results under different settings

5.4 Experiments with General Queries

We further conduct experiments on general queries. We denote queries that match no entities in Freebase as general queries, such as "chinese women tennis players." Note that only type-based facet expansion could be applied to general queries. Experiment results are shown in Table 13 and 14.

The results are consistent with the experiments on all queries: QDM$_{KB}$ yields the best result of rp-nDCG and F1-nDCG on both datasets. Overall, the improvement of rp-nDCG and F1-nDCG on general queries is smaller than that on direct-entity queries, this is because that additional dir-property and ex-property methods are applied to direct-entity queries. Another point worth noticing is that the gap of rp-nDCG between these two kinds of queries on UserQ (from 0.271 to 0.196) is more obvious than that on RandQ (from 0.292 to 0.281). A reasonable explanation may be that the queries in UserQ, which are issued by QDMiner users, are more entity-like, so methods specifically designed for direct-entity queries play an important role in improving facet quality. While for queries in RandQ, which are sampled from query logs and more general, the results on direct-entity queries and general queries are similar. Improving effectiveness on general queries still needs our further effort.

5.5 Experiments with Different Settings

In this section, we will conduct experiments under different settings including t_{sim}, $t_{overlap}$, and number of search results used. We change t_{sim} and $t_{overlap}$ from 0.1 to 0.9 by step 0.1 and the number of search results from 10 to 100 by step 10. We only report the results on UserQ because we obtain similar results on RandQ.

Figure 4(a) shows that the dependency of experiment results on t_{sim} setting is slight. Because our approach for weighting Freebase types and properties is based on voting, the final result is not sensitive to the number of entities retrieved from Freebase. Basically, lower t_{sim} value tends to yield slightly higher rp-nDCG, PRF, and wPRF, which indicates that using relatively more entities could lead to more reasonable voting result.

Figure 4(b) shows that as $t_{overlap}$ goes up, fp-nDCG and nDCG increase, NMI decreases, rp-nDCG, F1-nDCG, PRF, and wPRF increase at the beginning then decrease. These results are consistent with intuition. The higher $t_{overlap}$ we use, the more cautiously QDM$_{KB}$ expands the initial facets. So precision increases while recall decreases as $t_{overlap}$ rises. Metrics such as rp-nDCG, F1-nDCG, PRF, and wPRF take both precision and recall into consideration, so that the best scores of these metrics are achieved under moderate $t_{overlap}$ settings. Note that the dependency of experiment results on $t_{overlap}$ setting is still slight, which indicates that our method doesn't heavily rely on predefined parameters.

From Figure 4(c) and 4(d), we find that QDM$_{KB}$ consistently gets superior rp-nDCG and F1-nDCG and comparable fp-nDCG compared to QDMiner using different number of search results. As number of search results increases, performance also rises, which indicates the necessity of leveraging both knowledge bases and search results. Knowledge bases and search results are complementary to each other.

6 Conclusions

Existing query facet mining algorithms, including QDMiner, QF-I, and QF-J mainly rely on the top search results from the search engines. The coverage of facets mined using this kind of methods might be limited, because usually only a small number of results are used. We propose leveraging knowledge bases as complementary data sources. We use two methods, namely facet generation and facet expansion, to mine query facets. Facet generation directly uses properties in Freebase as candidates, while facet expansion intends to expand initial facets mined by QDMiner in property-based and type-based manners. Experimental results show that our approach is effective, especially for improving the recall of facet items.

Acknowledgments

Zhicheng Dou is the corresponding author of the paper. This work was supported by the National Key Basic Research Program (973 Program) of China under grant No.

2014CB340403, and the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China No. 15XNLF03, the National Natural Science Foundation of China (Grant No. 61502501).

REFERENCES

[1] H. Shum, "Bing dialog model: Intent, knowledge, and user interaction," http://research.microsoft.com/en-us/um/redmond/events/fs2010/presentations/Shum_Bing_Dialog_Model_RFS_71210.pdf.

[2] J. Niccolai, "Yahoo vows death to the '10 blue links'," http://www.pcworld.com/businesscenter/article/165214/yahoo_vows_death_to_the_10_blue_links.html.

[3] R. Browne, "No longer just 'ten blue links'," http://blogoscoped.com/archive/2007-06-28-n28.html.

[4] Z. Dou, S. Hu, Y. Luo, R. Song, and J.-R. Wen, "Finding dimensions for queries," in *Proceedings of CIKM'11*, 2011, pp. 1311–1320.

[5] W. Kong and J. Allan, "Extending faceted search to the general web," in *Proceedings of CIKM '14*, 2014, pp. 839–848.

[6] ——, "Extracting query facets from search results," in *Proceedings of SIGIR '13*, 2013, pp. 93–102.

[7] Z. Dou, Z. Jiang, S. Hu, J. Wen, and R. Song, "Automatically mining facets for queries from their search results," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 2, pp. 385–397, 2016.

[8] O. Ben-Yitzhak, N. Golbandi, N. Har'El, R. Lempel, A. Neumann, S. Ofek-Koifman, D. Sheinwald, E. Shekita, B. Sznajder, and S. Yogev, "Beyond basic faceted search," in *Proceedings of WSDM '08*, 2008, pp. 33–44.

[9] M. Diao, S. Mukherjea, N. Rajput, and K. Srivastava, "Faceted search and browsing of audio content on spoken web," in *Proceedings of CIKM '10*, 2010, pp. 1029–1038.

[10] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman, "Dynamic faceted search for discovery-driven analysis," in *CIKM '08*, 2008, pp. 3–12.

[11] C. Li, N. Yan, S. B. Roy, L. Lisham, and G. Das, "Facetedpedia: dynamic generation of query-dependent faceted interfaces for wikipedia," in *Proceedings of WWW '10*, 2010, pp. 651–660.

[12] W. Dakka and P. G. Ipeirotis, "Automatic extraction of useful facet hierarchies from text databases," in *Proceedings of ICDE '08*, 2008, pp. 466–475.

[13] W. Dakka, R. Dayal, and P. G. Ipeirotis, "Automatic discovery of useful facet terms," in *SIGIR Faceted Search Workshop*, 2006, pp. 18–22.

[14] W. Dakka, P. G. Ipeirotis, and K. R. Wood, "Automatic construction of multifaceted browsing interfaces," in *Proceedings of CIKM '05*, 2005, pp. 768–775.

[15] K. Latha, K. R. Veni, and R. Rajaram, "Afgf: An automatic facet generation framework for document retrieval," in *Proceedings of ACE '10*, 2010, pp. 110–114.

[16] E. Stoica and M. A. Hearst, "Automating creation of hierarchical faceted metadata structures," in *Proceedings of NAACL HLT '07*, 2007.

[17] S. Basu Roy, H. Wang, G. Das, U. Nambiar, and M. Mohania, "Minimum-effort driven dynamic faceted search in structured databases," in *Proceedings of CIKM '08*, 2008, pp. 13–22.

[18] J. Pound, S. Paparizos, and P. Tsaparas, "Facet discovery for structured web search: A query-log mining approach," in *Proceedings of SIGMOD '11*, 2011, pp. 169–180.

[19] E. Stoica, M. A. Hearst, and M. Richardson, "Automating creation of hierarchical faceted metadata structures," in *Proceedings of NAACL HLT '07*, 2007, pp. 244–251.

[20] "Google sets," http://labs.google.com/sets.

[21] R. C. Wang and W. W. Cohen, "Language-independent set expansion of named entities using the web," in *Proceedings of ICDM '07*, 2007, pp. 342–350.

[22] ——, "Iterative set expansion of named entities using the web," in *Proceedings of ICDM '08*, 2008, pp. 1091–1096.

[23] ——, "Character-level analysis of semi-structured documents for set expansion," in *Proceedings of EMNLP '09*, 2009, pp. 1503–1512.

[24] A. Carlson, J. Betteridge, R. C. Wang, E. R. Hruschka, Jr., and T. M. Mitchell, "Coupled semi-supervised learning for information extraction," in *Proceedings of WSDM '10*, 2010, pp. 101–110.

[25] H. Zhang, M. Zhu, S. Shi, and J.-R. Wen, "Employing topic models for pattern-based semantic class discovery," in *Proceedings of ACL-IJCNLP '09*, 2009, pp. 459–467.

[26] S. Shi, H. Zhang, X. Yuan, and J.-R. Wen, "Corpus-based semantic class mining: distributional vs. pattern-based approaches," in *Proceedings of COLING '10*, 2010, pp. 993–1001.

[27] R. Baeza-Yates, C. Hurtado, and M. Mendoza, "Query recommendation using query logs in search engines," in *Current Trends in Database Technology EDBT 2004 Workshops*, 2004, pp. 588–596.

[28] Z. Zhang and O. Nasraoui, "Mining search engine query logs for query recommendation," in *Proceedings of WWW '06*, 2006, pp. 1039–1040.

[29] L. Li, L. Zhong, Z. Yang, and M. Kitsuregawa, "Qubic: An adaptive approach to query-based recommendation," *J. Intell. Inf. Syst.*, vol. 40, no. 3, pp. 555–587, Jun. 2013.

[30] I. Szpektor, A. Gionis, and Y. Maarek, "Improving recommendation for long-tail queries via templates," in *Proceedings of WWW '11*, 2011, pp. 47–56.

[31] A. Herdagdelen, M. Ciaramita, D. Mahler, M. Holmqvist, K. Hall, S. Riezler, and E. Alfonseca, "Generalized syntactic and semantic models of query reformulation," in *Proceeding of SIGIR'10*, 2010, pp. 283–290.

[32] M. Mitra, A. Singhal, and C. Buckley, "Improving automatic query expansion," in *Proceedings of SIGIR '98*, 1998, pp. 206–214.

[33] P. Anick, "Using terminological feedback for web search refinement: a log-based study," in *Proceedings of SIGIR '03*, 2003, pp. 88–95.

[34] S. Riezler, Y. Liu, and A. Vasserman, "Translating queries into snippets for improved query expansion," in *Proceedings of COLING '98*, 2008, pp. 737–744.

[35] X. Xue and W. B. Croft, "Modeling reformulation using query distributions," *ACM Trans. Inf. Syst.*, vol. 31, no. 2, pp. 6:1–6:34, May 2013.

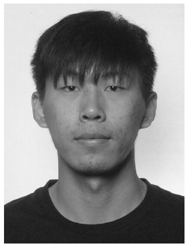
[36] L. Bing, W. Lam, T.-L. Wong, and S. Jameel, "Web query reformulation via joint modeling of latent topic dependency and term context," *ACM Trans. Inf. Syst.*, vol. 33, no. 2, pp. 6:1–6:38, Feb. 2015.

[37] J. Huang and E. N. Efthimiadis, "Analyzing and evaluating query reformulation strategies in web search logs," in *Proceedings of CIKM '09*, 2009, pp. 77–86.

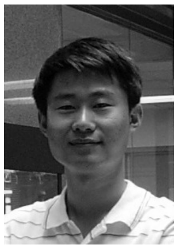
[38] S. Gholamrezazadeh, M. A. Salehi, and B. Gholamzadeh, "A comprehensive survey on text summarization systems," in *Proceedings of CSA '09*, 2009, pp. 1–6.

[39] M. Damova and I. Koychev, "Query-based summarization: A survey," in *S3T'10*, 2010.

[40] L. J. Heyer, S. Kruglyak, and S. Yooseph, "Exploring Expression Data: Identification and Analysis of Coexpressed Genes," *Genome Research*, vol. 9, no. 11, pp. 1106–1115, November 1999.



Zhengbao Jiang is a graduate student from School of Information at Renmin University of China. He received his B.S degrees in computer science from Renmin University of China in 2015. His research interests include natural language processing, information retrieval, and data mining.



Zhicheng Dou is an associate professor at School of Information, Renmin University of China. He received his Ph.D. and B.S. degrees in computer science and technology from the Nankai University in 2008 and 2003, respectively. He worked at Microsoft Research as a researcher from July 2008 to September 2014. His research interests include information retrieval, data mining, and big data analytics. His homepage is http://www.playbigdata.com/dou/.

Ji-Rong Wen is a professor in Renmin University of China. He received B.S. and M.S. degrees from Renmin University of China, and received his Ph.D. degree in 1999 from the Chinese Academy of Science. He is a senior researcher and research manager at Microsoft Research during 2000 and 2014. His main research interests are web data management, information retrieval (especially web IR), and data mining.