Finding Dimensions for Queries*

Zhicheng Dou[1], Sha Hu[2], Yulong Luo[3], Ruihua Song[1], and Ji-Rong Wen[1]
[1]Microsoft Research Asia; [2]Renmin University of China; [3]Shanghai Jiaotong University
[1]{zhichdou, rsong, jrwen}@microsoft.com; [2]sally_555@163.com; [3]ronagon@gmail.com

ABSTRACT

We address the problem of finding multiple groups of words or phrases that explain the underlying query facets, which we refer to as query dimensions. We assume that the important aspects of a query are usually presented and repeated in the query's top retrieved documents in the style of lists, and query dimensions can be mined out by aggregating these significant lists. Experimental results show that a large number of lists do exist in the top results, and query dimensions generated by grouping these lists are useful for users to learn interesting knowledge about the queries.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Clustering, Information filtering, Query formulation*

General Terms

Algorithms, Experimentation, Management, Measurement

Keywords

Query Dimension, Query Facet, Faceted Search, Query Suggestion, Entity, Query Reformulation, Query Summarization, User Intent, List Extraction

1. INTRODUCTION

We address the problem of finding query dimensions. A *query dimension*, which is similar to a dimension in data warehouses [19, 9], is a set of items which describe and summarize one important aspect of a query. Here a *dimension item* is typically a word or a phrase. A query may have *multiple* dimensions that summarize the information about the query from different perspectives. Table 1 shows dimensions for some example queries. For the query "watches", its query dimensions cover the knowledge about watches in five unique aspects, including brands, gender categories, supporting features, styles, and colors. The query "visit Beijing" has a dimension about popular resorts in Beijing (tiananmen square, forbidden city, summer palace, ...) and a dimension on several travel related topics (attractions, shopping, dining, ...).

Query dimensions provide interesting and useful knowledge about a query and thus can be used to improve search experiences in many ways. **First**, we can display query dimensions together with the original search results in an appropriate way. Thus, users can understand some important facets of a query without browsing tens of pages. For example, a user could learn different brands and categories of watches. We can also implement a faceted search [4, 14, 13] based on query dimensions. User can clarify their specific intent by selecting dimension items. Then search results could be restricted to the documents that are relevant to the items. A user could drill down to women's watches if he is looking for a gift for his wife. These multiple groups of query dimensions are in particular useful for vague or ambiguous queries, such as "apple". We could show the products of Apple Inc. in one dimension and different types of the fruit apple in another. **Second**, query dimensions may provide direct information or instant answers that users are seeking. For example, for the query "lost season 5", all episode titles are shown in one dimension and main actors are shown in another. In this case, displaying query dimensions can save browsing time. **Third**, query dimensions may also be used to improve the diversity of the ten blue links. We can re-rank search results to avoid showing the pages that are near-duplicated in query dimensions at the top. Query dimensions also contain structured knowledge covered by or related to the input keywords of a query, and thus they can be used in many other fields besides traditional web search, such as semantic search or entity search [10, 3, 5, 29, 35, 6]. There has been a lot of recent work on automatically building knowledge ontology on the Web [6, 29]. Query dimensions can become a possible data source for this.

We observe that important pieces of information about a query are usually presented in list styles and repeated many times among top retrieved documents. Thus we propose aggregating frequent lists within the top search results to mine query dimensions and implement a system called **QDMiner**. More specifically, QDMiner extracts lists from free text, HTML tags, and repeat regions contained in top search results, and groups them into clusters based on the items they contain. Compared to previous works on building facet hierarchies [4, 14, 13, 23, 11, 22, 31], our approach

[*] The work was done when the second and third author were visiting Microsoft Research Asia

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.
Copyright 2011 ACM 978-1-4503-0717-8/11/10 ...$10.00.

Table 1: Example query dimensions (automatically mined by our proposed method in this paper). Items in each dimension are seperated by commas.

query: **watches** 1. cartier, breitling, omega, citizen, tag heuer, bulova, casio, rolex, audemars piguet, seiko, accutron, movado, fossil, gucci, . . . 2. men's, women's, kids, unisex 3. analog, digital, chronograph, analog digital, quartz, mechanical, manual, automatic, electric, dive, . . . 4. dress, casual, sport, fashion, luxury, bling, pocket, . . . 5. black, blue, white, green, red, brown, pink, orange, yellow, . . .
query: **lost** 1. season 1, season 6, season 2, season 3, season 4, season 5 2. matthew fox, naveen andrews, evangeline lilly, josh holloway, jorge garcia, daniel dae kim, michael emerson, terry o'quinn, . . . 3. jack, kate, locke, sawyer, claire, sayid, hurley, desmond, boone, charlie, ben, juliet, sun, jin, ana, lucia . . . 4. what they died for, across the sea, what kate does, the candidate, the last recruit, everybody loves hugo, the end, . . .
query: **lost season 5** 1. because you left, the lie, follow the leader, jughead, 316, dead is dead, some like it hoth, whatever happened happened, the little prince, this place is death, the variable, . . . 2. jack, kate, hurley, sawyer, sayid, ben, juliet, locke, miles, desmond, charlotte, various, sun, none, richard, daniel 3. matthew fox, naveen andrews, evangeline lilly, jorge garcia, henry ian cusick, josh holloway, michael emerson, . . . 4. season 1, season 3, season 2, season 6, season 4
query: **flowers** 1. birthday, anniversary, thanksgiving, get well, congratulations, christmas, thank you, new baby, sympathy, fall 2. roses, best sellers, plants, carnations, lilies, sunflowers, tulips, gerberas, orchids, iris 3. blue, orange, pink, red, purple, white, green, yellow
query: **what is the fastest animals in the world** 1. cheetah, pronghorn antelope, lion, thomson's gazelle, wildebeest, cape hunting dog, elk, coyote, quarter horse 2. birds, fish, mammals, animals, reptiles 3. science, technology, entertainment, nature, sports, lifestyle, travele, gaming, world business
query: **the presidents of the united states** 1. john adams, thomas jefferson, george washington, john tyler, james madison, abraham lincoln, john quincy adams, william henry harrison, martin van buren, james monroe, . . . 2. the presidents of the united states of america, the presidents of the united states ii, love everybody, pure frosting, these are the good times people, freaked out and small, . . . 3. kitty, lump, peaches, dune buggy, feather pluckn, back porch, kick out the jams, stranger, boll weevil, ca plane pour moi, . . . 4. federalist, democratic-republican, whig, democratic, republican, no party, national union, . . .
query: **visit beijing** 1. tiananmen square, forbidden city, summer palace, temple of heaven, great wall, beihai park, hutong 2. attractions, shopping, dining, nightlife, tours, travel tip, transportation, facts
query: **cikm** 1. databases, information retrieval, knowledge management, industry research track 2. submission, important dates, topics, overview, scope, committee, organization, programme, registration, cfp, publication, programme committee, organisers, . . . 3. acl, kdd, chi, sigir, www, icml, focs, ijcai, osdi, sigmod, sosp, stoc, uist, vldb, wsdm, . . .

is unique in two aspects: (1) **Open domain**: we do not restrict queries in a specific domain, like products, people, etc. Our proposed approach is generic and does not rely on any specific domain knowledge. Thus it can deal with open-domain queries. (2) **Query dependent**: instead of a same pre-defined schema for all queries, we extract dimensions from the top retrieved documents for each query. As a result, different queries may have different dimensions. For example, although "lost" and "lost season 5" in Table 1 are both TV program related queries, their mined dimensions are different.

As the problem of finding query dimension is new, we cannot find publicly available evaluation datasets. Therefore, we create two datasets, namely UserQ, containing 89 queries that are submitted by QDMiner users, and RandQ, containing 105 randomly sampled queries from logs of a commercial search engine, to evaluate mined dimensions. We use some existing metrics, such as purity and normalized mutual information (NMI), to evaluate clustering quality, and use NDCG to evaluate ranking effectiveness of dimensions. We further propose two metrics to evaluate the integrated effectiveness of clustering and ranking.

Experimental results show that the purity of query dimensions generated by QDMiner is good. Averagely on UserQ dataset, it is as high as 91%. The dimensions are also reasonably ranked with an average NDCG@5 value 0.69. Among the top five dimensions, 2.3 dimensions are good, 1.2 ones are fair, and only 1.5 are bad. We also reveal that the quality of query dimensions is affected by the quality and the quantity of search results. Using more of the top results can generate better query dimensions.

The remainder of this paper is organized as follows. We briefly introduce related work in Section 2. Following this, we propose QDminer, our approach to generate query dimensions by aggregating frequent lists in top results, in Section 3. We discuss evaluation methodology in Section 4 and report experimental results in Section 5. Finally we conclude the work in Section 6.

2. RELATED WORK

Finding query dimensions is related to several existing research topics. In this section, we briefly review them and discuss the difference from our proposed method QDMiner.

2.1 Query Reformulation

Query reformulation is the process of modifying a query to get search results that can better satisfy a user's information need. It is an important topic in Web search. Several techniques have been proposed based on relevance feedback, query log analysis, and distributional similarity [1, 25, 2, 33, 28, 34, 36, 17]. The problem of mining dimensions is different from query reformulation, as the main goal of mining dimensions is to summarize the knowledge and information contained in the query, rather than to find a list of related or expanded queries. Some query dimensions include semantically related phrases or terms that can be used as query reformulations, but some others cannot. For example, for the query "what is the fastest animals in the world" in Table 1, we generate a dimension "cheetah, pronghorn antelope, lion, thomson's gazelle, wildebeest, ..." which includes animal names that are direct answers rather than query reformulations to the query.

2.2 Query-based Summarization

Query dimensions can be thought as a specific type of summaries that briefly describe the main topic of given text. Several approaches [8, 27, 21] have been developed in the area of text summarization, and they are classified into different categories in terms of their summary construction methods (abstractive or extractive), the number of sources for the summary (single document or multiple documents), types of information in the summary (indicative or informative), and the relationship between summary and query (generic or query-based). Brief introductions to them can be found in [16] and [12]. Similar to existing summarization systems, QDMiner aims to offer the possibility of finding the main points of multiple documents and thus save users' time on reading whole documents. The difference is that most existing summarization systems dedicate themselves to generating summaries using **sentences** extracted from documents, while we generate summaries based on frequent **lists**. In addition, we return **multiple groups** of semantically related items, while they return a flat list of sentences.

2.3 Entity Search

The problem of entity search has received much attention in recent years [10, 3, 5, 29, 15, 20, 32, 26]. Its goal is to answer information needs that focus on entities. The problem of mining query dimensions is related to entity search as for some queries, dimension items are kinds of entities or attributes. Some existing entity search approaches also exploited knowledge from structure of webpages [30, 35, 6, 15]. Finding query dimensions differs from entity search in the following aspects. Firstly, finding query dimensions is applicable for all queries, rather than just entity related queries. Secondly, they tend to return different types of results. The result of an entity search is usually a list of entities, their attributes, and associated homepages, whereas query dimensions are comprised of multiple lists of items, which are not necessarily entities.

2.4 Faceted Search

Faceted search is a technique for allowing users to digest, analyze, and navigate through multidimensional data. It is widely applied in e-commerce and digital libraries. A robust review of faceted search is beyond the scope of this paper. Most existing faceted search and facets generation systems [4, 14, 13, 23, 11, 22, 31] are built on a specific domain (such as product search) or predefined facet categories. For example, Dakka and Ipeirotis [11] introduced an unsupervised technique for automatic extraction of facets that are useful for browsing text databases. Facet hierarchies are generated for a whole collection, instead of for a given query. Li et al. proposed Facetedpedia [23], a faceted retrieval system for information discovery and exploration in Wikipedia. Facetedpedia extracts and aggregates the rich semantic information from the specific knowledge database Wikipedia. In this paper, we explore to automatically find *query-dependent* dimensions for *open-domain* queries based on a general Web search engine. Dimensions of a query are automatically mined from the top web search results of the query without any additional domain knowledge required. As query dimensions are good summaries of a query and are potentially useful for users to understand the query and help them explore information, they are possible data sources that enable a general open-domain faceted exploratory search.

3. OUR APPROACH

As the first trial of mining query dimensions, we propose to automatically mine query dimensions from the top retrieved documents and develop a system called **QDMiner**. QDMiner discovers query dimensions *by aggregating frequent lists* within the top results. We propose this method based on the following observations:

(1) Important information is usually organized in *list* formats by websites. They may repeatedly occur in a sentence that is separated by commas, or be placed side by side in a well-formatted structure (e.g., a table). This is caused by the conventions of webpage design. Listing is a graceful way to show parallel knowledge or items and is thus frequently used by webmasters.

(2) Important lists are commonly supported by relevant websites and hence repeat in the top search results, whereas unimportant lists just infrequently appear in results. This makes it possible to distinguish good lists from bad ones, and to further rank dimensions in terms of importance.

Experimental results in Section 5 confirm the above observations and demonstrate that the query dimensions mined by aggregating them are meaningful.

3.1 System Overview

We illustrate QDMiner in Figure 1. In QDMiner, given a query q, we retrieve the top K results from a search engine and fetch all documents to form a set R as input. Then, query dimensions are mined by the following four steps:

1. List Extraction Several types of lists are extracted from each document in R. "men's watches, women's watches, luxury watches, ..." is an example list extracted.

2. List Weighting All extracted lists are weighted, and thus some unimportant or noisy lists, such as the price list "299.99, 349.99, 423.99, ..." that occasionally occurs in a page, can be assigned by low weights.

3. List Clustering Similar lists are grouped together to compose a dimension. For example, different lists about watch gender types are grouped because they share the same items "men's" and "women's".

4. Dimension and Item Ranking Dimensions and their items are evaluated and ranked based on their importance. For example, the dimension on brands is ranked higher than the dimension on colors based on how frequent the dimensions occur and how relevant the supporting documents are. Within the dimension on gender categories, "men's" and "women's" are ranked higher than "unisex" and "kids" based on how frequent the items appear, and their order in the original lists.

In the remaining part of this section, we will describe the four modules in detail.

3.2 List Extraction

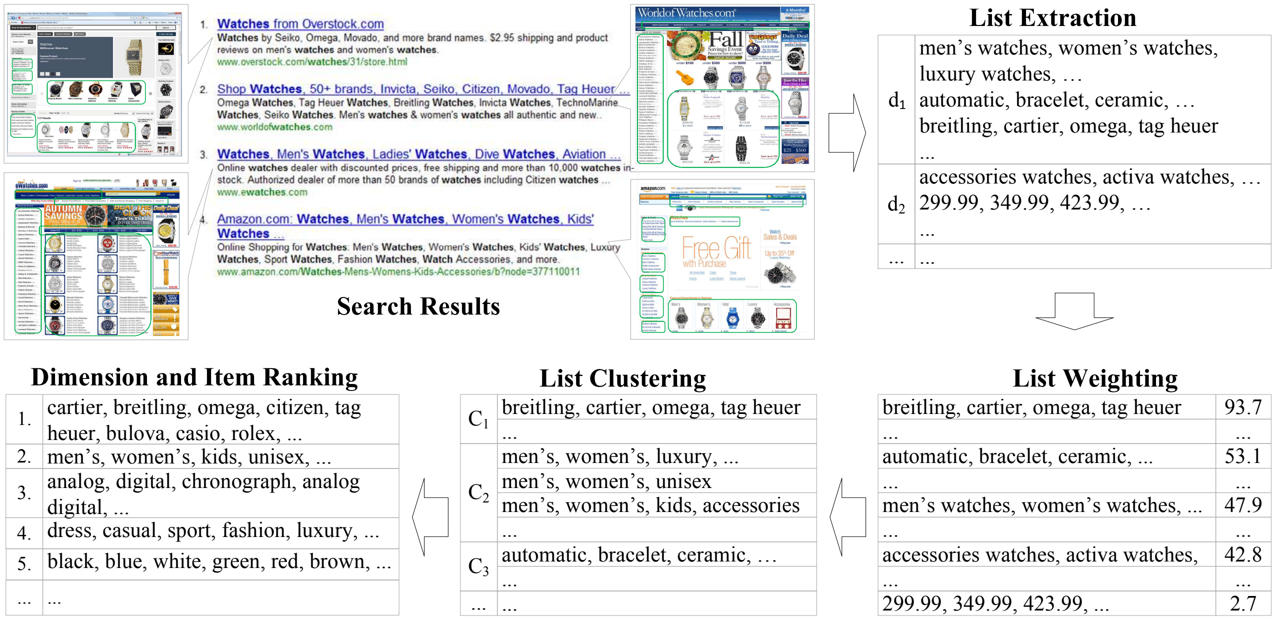
From each document d in the search result set R, we extract a set of lists $L_d = \{l'\}$ from the HTML content of d based on three different types of patterns, namely free text patterns, HTML tag patterns, and repeat region patterns.

3.2.1 Free text patterns

We extract all text within document d and split it into sentences. We then employ the pattern **item{, item}* (and|or) {other} item**, which is similar to that in [35], to extract all matched items from each sentence. In Example 1, the items in italic font are extracted as a list.

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

We further use the pattern **{^item (:|-) .+$}+** to extract lists from some semi-structured paragraphs. It extracts lists from continuous lines that are comprised of two parts separated by a dash or a colon. The first parts of these lines



List Extraction
d1
men’s watches, women’s watches, luxury watches, …
automatic, bracelet, ceramic, …
breitling, cartier, omega, tag heuer
...
d2
accessories watches, activa watches, …
299.99, 349.99, 423.99, …
...
... ...
1. Watches from Overstock.com
Watches by Seiko, Omega, Movado, and more brand names. $2.95 shipping and product reviews on men's watches and women's watches.
www.overstock.com/watches/31/store.html
2. Shop Watches, 50+ brands, Invicta, Seiko, Citizen, Movado, Tag Heuer ...
Omega Watches, Tag Heuer Watches, Breitling Watches, Invicta Watches, TechnoMarine Watches, Seiko Watches. Men's watches & women's watches all authentic and new..
www.worldofwatches.com
3. Watches, Men's Watches, Ladies' Watches, Dive Watches, Aviation ...
Online watches dealer with discounted prices, free shipping and more than 10,000 watches in-stock. Authorized dealer of more than 50 brands of watches including Citizen watches ...
www.ewatches.com
4. Amazon.com: Watches, Men's Watches, Women's Watches, Kids' Watches ...
Online Shopping for Watches: Men's Watches, Women's Watches, Kids' Watches, Luxury Watches, Sport Watches, Fashion Watches, Watch Accessories, and more.
www.amazon.com/Watches-Mens-Womens-Kids-Accessories/b?node=377110011
Search Results
Dimension and Item Ranking
1. cartier, breitling, omega, citizen, tag heuer, bulova, casio, rolex, ...
2. men’s, women’s, kids, unisex, ...
3. analog, digital, chronograph, analog digital, ...
4. dress, casual, sport, fashion, luxury, ...
5. black, blue, white, green, red, brown, ...
... ...
List Clustering
C1 breitling, cartier, omega, tag heuer
...
C2 men’s, women’s, luxury, ...
men’s, women’s, unisex
men’s, women’s, kids, accessories
...
C3 automatic, bracelet, ceramic, …
...
... ...
List Weighting
breitling, cartier, omega, tag heuer 93.7
... ...
automatic, bracelet, ceramic, ... 53.1
... ...
men’s watches, women’s watches, ... 47.9
... ...
accessories watches, activa watches, 42.8
... ...
299.99, 349.99, 423.99, ... 2.7

Figure 1: System overview of QDMiner

Table 2: Example HTML sources that contain lists

SELECT:

```
<select name=”ProductFinder2” id=”ProductFinder2” >
<option value=”WatchBrands.htm”>Watch Brands</option>
<option value=”Brands-Accutron.htm”>Accutron</option>
<option value=”Brands-Bulova.htm”>Bulova</option>
<option value=”Brands-Caravelle.htm”>Caravelle</option>
<option value=”Brands-Seiko.htm”>Seiko</option></select>
```

UL:

```
<ul><li><a href=”/rst.asp?q=dive”>Dive</a></li>
<li><a href=”/rst.asp?q=titanium”>Titanium</a></li>
<li><a href=”/rst.asp?q=automatic”>Automatic</a></li>
<li><a href=”/rst.asp?q=quartz”>Quartz</a></li>
<li><a href=”/rst.asp?q=gold”>Gold</a></li></ul>
```

TABLE:

```
<table width=”100%”>
<tr><td width=”10%”></td><td>White</td></tr>
<tr><td></td><td height=”20”>Red</td></tr>
<tr><td></td><td height=”20”>Black</td></tr>
<tr><td></td><td height=”20”>Pink</td></tr>
<tr><td height=”4” colspan=”2”></td></tr></table>
```

are extracted as a list. For instance, we extract all text in italic font in Example 2 as a list.

Example 2 ... are highly important for following reasons:
Consistency - every fact table is filtered consistently res...
Integration - queries are able to drill different processes ...
Reduced development time to market - the common dimensions are available without recreating the wheel over again.

3.2.2 *HTML tag patterns*

We extract lists from several list-style HTML tags, including SELECT, UL, OL, and TABLE. Example sources of these HTML tags can be found in Table 2. Extracted items are in italic.

SELECT For the SELECT tag, we simply extract all text from their child tags (OPTION) to create a list. Moreover, we remove the first item if it starts with some predefined text, such as “select” or “choose”.

UL/OL For these two tags, we also simply extract text within their child tags (LI).

TABLE We extract one list from each column or each row. For a table containing m rows and n columns, we extract at most $m+n$ lists. For each column, the cells within THEAD or TFOOT tags are regarded as table headers and are dropped from the list. We also drop the first cell of each column when its cascading style[1] is different from other cells.

3.2.3 *Repeat region patterns*

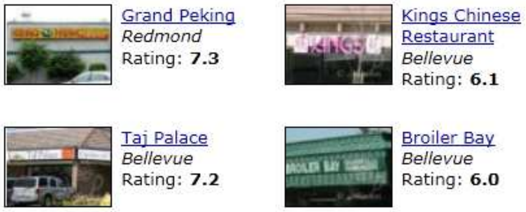
We observe that peer information is sometimes organized in well-structured visual blocks in webpages. Figure 2 shows a repeat region of four blocks in repeated style. Each block contains a restaurant record that is comprised of four attributes: a picture, a restaurant name, a location description, and a rating. We can extract three lists from this region: a list of restaurant names, a list of location descriptions, and a list of ratings. Note that images are simply ignored in this paper.

To extract these lists, we **first** detect repeat regions in webpages based on vision-based DOM trees [7]. Here a repeat region is the region that includes more than one block, e.g., M blocks, with similar DOM and visual structures. We **then** extract all leaf HTML nodes within each block, and group them by their tag names and display styles. In the above example, all restaurant names have the same tag name (`<a>`) and displaying style (in blue color), and hence can be grouped together. Each group usually contains M nodes. Each two of them are from different blocks. **At last**, for each group, we extract all text from its nodes as a list.

3.2.4 *Post-processing*

We further process each extracted list l' as follows. We first normalize all items by removing useless symbol charac-

[1] http://www.w3.org/Style/CSS/



Grand Peking
Redmond
Rating: **7.3**

Kings Chinese Restaurant
Bellevue
Rating: **6.1**

Taj Palace
Bellevue
Rating: **7.2**

Broiler Bay
Bellevue
Rating: **6.0**

Figure 2: An example repeat region in a webpage

Table 3: Less informative list examples

	Items (separated by commas)
1	we recommend, my account, help
2	home, customer service, my account, tracking, faq's
3	read, edit, view history
4	movado 605635 luno two tone... 547.50 717.00 1 rating 1 review, movado museum strap 0690299... 225.00 395.00 1 rating, citizen calibre 2100 av0031... 299.00 350.99 11 ratings

ters, such as '[' and ']', and converting uppercase letters to lowercase. We then remove long items which contain more than 20 terms. At last, we remove all lists that contain less than two unique items or more than 200 unique items.

3.3 List Weighting

Some of the extracted lists are not informative or even useless. Some of them are extraction errors. Table 3 shows some sample lists for the query "watches". The first three lists are navigational links which are designed to help users navigate between webpages. They are not informative to the query. The fourth list is actually an extraction error: several types of information are mixed together.

We argue that these types of lists are useless for finding dimensions. We should punish these lists, and rely more on better lists to generate good dimensions. We find that a good list is usually supported by many websites and appear in many documents, partially or exactly. A good list contains items that are informative to the query. Therefore, we propose to aggregate all lists of a query, and evaluate the importance of each *unique* list l by the following components:

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

- s_d^m **is the percentage of items contained in** d. A list l is supported by a document d, if d contains some or all items of l. The more items d contains, the stronger it supports l. Suppose $|d \cap l|$ is the number of shared items in d and l, and $|l|$ is the number of items contained in list l, we let $s_d^m = \frac{|d\cap l|}{|l|}$.
- s_d^r **measures the importance of document** d. It is derived from ranks of documents in this paper. The documents ranked higher in the original search results are usually more relevant to the query, and 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., $S_{\text{IDF}} = \frac{1}{|l|} \cdot \sum_{e\in l} idf_e$. 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[2], which includes about one billion webpages, as our reference corpus in counting N_e and N.

We combine these two components, and evaluate the importance of a list l by Equation (1).

$$S_l = S_{\text{DOC}} * S_{\text{IDF}} \tag{1}$$

Finally, we sort all lists by final weights for the given query. The first three lists in Table 3 are assigned low weights as they have low average invert document frequencies. The weight of the fourth list is also low. Its most items just appear in one document in top results; hence it has a low document matching weight.

3.4 List Clustering

We do not use individual weighted lists as query dimensions because: (1) An individual list may inevitably include noise. For example, the first item of the first list in Table 2, i.e., "watch brands", is noise. It is difficult to identify it without other information provided; (2) An individual list usually contains a small number of items of a dimension and thus it is far from complete; (3) Many lists contain duplicated information. They are not exactly same, but share overlapped items. To conquer the above issues, we group similar lists together to compose dimensions.

Two lists can be grouped together if they share enough items. We define the distance $d_l(l_1, l_2)$ between two lists l_1 and l_2 as $d_l(l_1, l_2) = 1 - \frac{|l_1 \cap l_2|}{\min\{|l_1|,|l_2|\}}$. Here $|l_1 \cap l_2|$ is the number of shared items within l_1 and l_2. We use the complete linkage distance $d_c(c_1, c_2) = \max_{l_1\in c_1, l_2\in c_2} d_l(l_1, l_2)$ to compute the distance between two clusters of lists. This means that two groups of lists can only be merged together when every two lists of them are similar enough.

We use a modified QT (Quality Threshold) clustering algorithm [18] to group similar lists. 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 do not 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 QT, the number of clusters is not required to be specified.

The QT algorithm assumes that all data is equally important, and the cluster that has the most number of points is selected in each iteration. In our problem, lists are not equally important. Better lists should be grouped together first. We modify the original QT algorithm to first group highly weighted lists. The algorithm, which we refer to as WQT (**Q**uality **T**hreshold with **W**eighted data points), is described as follows.

1. Choose a maximum diameter Dia_{max} and a minimum weight W_{min} for clusters.

2. Build a candidate cluster for the *most important point* by iteratively including the point that is closest to the group, until the diameter of the cluster surpasses the

[2] http://boston.lti.cs.cmu.edu/Data/clueweb09/

threshold Dia_{max}. Here the most important point is the list which has the highest weight.

3. Save the candidate cluster if the total weight of its points w_c is not smaller than W_{min}, and remove all points in the cluster from further consideration.

4. Recurse with the reduced set of points.

Recall that the main difference between WQT and QT is that WQT tries to get more neighbors for *important* points, and hence generated clusters are biased towards important data points. Suppose we have six lists: l_1 =(cartier, breitling, omega, citizen), l_2 =(breitling, omega, citizen, tag heuer), l_3 =(breitling, omega, citizen, movie, music, book), l_4 =(movie, music, book), l_5 =(music, book, radio), and l_6 =(movie, book, radio). Their corresponding weights satisfy: $S_{l1} > S_{l2} > S_{l3} > S_{l4} > S_{l5} > S_{l6}$. QT ignores their weights and generate a cluster (l_3, l_4, l_5, l_6) in the first iteration with $Dia_{max} = 0.6$, whereas WQT will generate a cluster (l_1, l_2, l_3) for list l_1. We prefer the second result, especially when S_{l_1} is much larger than S_{l_3}. In addition, WQT is more efficient than QT, as it just builds one candidate cluster while QT builds a candidate cluster for each remaining point.

In this paper, the weight of a cluster is computed based on the number of websites from which its lists are extracted. More specifically, $w_c = |Sites(c)|$ where $Sites(c)$ is the set of websites that contain lists in c. Note we use websites instead of webpages because webpages from the same website usually share the same page templates and contribute duplicated lists. We empirically set $Dia_{max} = 0.6$ and $W_{min} = 3$. $W_{min} = 3$ means that the lists of a qualified cluster are from at least three unique websites.

After the clustering process, similar lists will be grouped together to compose a candidate query dimension.

3.5 Dimension and Item Ranking

In this section, we evaluate the importance of dimensions and items, and rank them based on importance.

Based on our motivation that a good dimension should frequently appear in the top results, a dimension c is more important if: (1) The lists in c are extracted from more unique websites; and (2) the lists in c are more important, i.e., they have higher weights. We define S_c, the importance of dimension c, as follows.

$$S_c = \sum_{s \in Sites(c)} \max_{l \in c, l \in s} S_l \qquad (2)$$

Here S_l is the weight of a list l. We sum up the maximum list weight from each website as dimension importance.

In a dimension, the importance of an item depends on how many lists contain the item and its ranks in the lists. As a better item is usually ranked higher by its creator than a worse item in the original list, we calculate $S_{e|c}$, the weight of an item e within a dimension c, by:

$$S_{e|c} = \sum_{s \in Sites(c)} w(c, e, s) = \sum_{s \in Sites(c)} \frac{1}{\sqrt{AvgRank_{c,e,s}}} \qquad (3)$$

where $w(c, e, s)$ is the weight contributed by a website s, and $AvgRank_{c,e,s}$ is the average rank of e within all lists extracted from website s. Suppose $L(c, e, s)$ is the set of all lists in c that contain item e and are extracted from website s, we have $AvgRank_{c,e,s} = \frac{1}{|L(c,e,s)|} \sum_{l \in L(c,e,s)} rank_{e|l}$.

Table 4: Statistics about human created dimensions

Item	UserQ			RandQ		
	Bad	Fair	Good	Bad	Fair	Good
#DimemsionsPerQ	4.4	5.3	4.9	2.1	2.1	2.9
#ItemsPerQuery	135	151	219	61	86	98
#ItemsPerDimension	31	29	45	30	40	33

$w(c, e, s)$ gets the highest score 1.0 when the item e is always the first item of the lists from s.

We sort all items within a dimension by their weights. We define an item e is a *qualified* item of dimension c if $S_{e|c} > 1$ and $S_{e|c} > \frac{|Sites(c)|}{10}$. Here $S_{e|c} > 1$ means that e is qualified if it is once the first item of lists from one website and also occurs in lists from at least another website. $S_{e|c} > \frac{|Sites(c)|}{10}$ means that it should be supported by at least 10% of all websites in this dimension. We only output qualified items by default in QDMiner.

4. EVALUATION METHODOLOGY

4.1 Data

We do not find any existing dataset available for evaluating the quality of query dimensions. Therefore, we build two datasets from scratch. First, we build an online service for finding dimensions, and invite some human subjects to issue queries about topics they know well. We collect 89 queries issued by the subjects, and name the set of queries as "UserQ". As asking users to produce queries concerning a topic that they are familiar with might induce a bias towards topics in which lists are more useful than general web queries, we further randomly sample another set of 105 English queries from a query log of a commercial search engine, and name this set of queries as "RandQ".

For each query, we first ask a subject to manually create dimensions and add items that are covered by the query, based on his/her knowledge after a deep survey on any related resources (such as Wikipedia, Freebase, or official web sites related to the query). We then aggregate the *qualified* items in the top *five* dimensions returned by all algorithms we want to evaluate, and ask the subject to assign unlabeled items into the created dimensions. New dimensions will be created for the items that are not covered by the existing dimensions.

For each human created dimension, we ask the subject who has created the dimension and four additional subjects to rate its usefulness in the following three levels:

`[Good/2]` - It is very useful and I like it;
`[Fair/1]` - It is just so so;
`[Bad /0]` - It is useless and I don't like it.

The rating that is most chosen by subjects is regarded as the final rating of the dimension. The higher one is used if two ratings are selected by the same number of subjects.

Table 4 shows the statistics about human labeled query dimensions. There are on average about 4.9 good dimensions and 5.3 fair dimensions for each query in the UserQ collection, while there are about 2.9 good dimensions and 2.1 fair dimensions for the RandQ collection. There are more dimensions and items in UserQ than in RandQ. This is because the queries in RandQ are randomly sampled from query logs. Some of them are too specific or noisy to have meaningful dimensions.

We find that the assessment of query dimensions is timeconsuming and costly, even if we only assess the top five dimensions for each query. Each subject may spend up to an hour to completely assess a query.

4.2 Evaluation Metrics

The quality of query dimensions can be measured in the following two aspects:

Quality of clustering - Ideally, each dimension should only contain items reflecting the same facet of the query, and the items referring to the same information should not be separated into multiple dimensions. In this paper, we use several existing metrics [24], including Purity, NMI (Normalized Mutual Information), RI (Random Index), and F measure, to evaluate the quality of clusters.

Ranking effectiveness of dimensions - Obviously we aim to rank good dimensions before bad dimensions when multiple dimensions are found. As we have multi-level ratings, we adopt the nDCG measure (Normalized Discounted Cumulative Gain), which is widely used in information retrieval, to evaluate the ranking of query dimensions. Suppose that each output dimension c_i is assigned to a manually labeled class c'_i which covers the maximum number of items in c_i. The ranking quality of the top p dimensions is calculated by $\text{nDCG}_p = \frac{DCG_p}{IDCG_p}$ where $DCG_p = \sum_{i=1}^{p} DG_i$ and $IDCG_p$ is the ideal cumulative gain which is produced by the perfect ordering. DG_i is the discounted gain of the i^{th} dimension. $DG_i = \frac{2^{r_i}-1}{\log_2(1+i)}$ if the rating of c'_i is r_i. In our problem, a ground truth class may be divided into multiple dimensions in automatic results. Hence DCG_p may exceed IDCG_p and nDCG may exceed 1 in some cases. To solve this problem, for each ground truth class c'_i, we only credit the *first* dimension that is assigned to it, and skip all later ones.

nDCG does not consider the quality of clustering which does influence user satisfaction. To evaluate the integrated effectiveness, we let $DCG_p = \sum_{i=1}^{p}(w_i \cdot DG_i)$ where w_i is a weight for each automatic dimension c_i, and propose two alternative nDCG measurements.

fp-nDCG *- purity aware nDCG.* fp-nDCG is also calculated based on the first appearance of each class. Different from the original nDCG, we further consider the purity of each dimension c_i by multiplying DG_i by the percentage of correctly assigned items, i.e., we let $w_i = \frac{|c'_i \cap c_i|}{|c_i|}$.

rp-nDCG *- recall and purity aware nDCG.* rp-nDCG is calculated based on all output dimensions. We weight each dimension by $w_i = \frac{|c'_i \cap c_i|}{|c_i|}\frac{|c'_i \cap c_i|}{|c'_i|}$. Here $\frac{|c'_i \cap c_i|}{|c'_i|}$ is the percentage of items in c'_i matched by the current output dimension c_i. rp-nDCG ranges from 0 to 1, and the best value 1 is achieved when all items are correctly classified into the right dimensions.

We calculate the above metrics based on the top five dimensions for each query, and then average them over all queries. We argue that ranking quality is generally more important than clustering quality for query dimensions. We prefer to generating useful dimensions that may contain a little noise rather than pure but useless dimensions.

5. EXPERIMENTAL RESULTS

5.1 Overall Results

We mine query dimensions based on top 100 results from a commercial search engine. Some samples of query dimensions have been shown in Table 1. We find that our generated top dimensions are usually meaningful and useful for users to understand underlying facets of queries.

Table 5: Statistics about mined query dimensions

Desc.	UserQ	RandQ
#queries	89	105
#results per query	99.8	99.5
#lists per document	44.1	37.0
#Items per list	9.7	10.1
#dimensions per query	32.1	21.6
#lists per dimension	7.1	6.4
#items/qualified items per dimension	20.8/7.5	23.7/8.7
#good/fair dimensions among top five	2.3/1.2	1.7/1

Table 6: Quality of mined query dimensions

Dataset	Purity	RI	F1	F5
UserQ	0.910	0.891	0.803	0.791
RandQ	0.922	0.849	0.770	0.738
	NMI	nDCG5	fp-nDCG5	rp-nDCG5
UserQ	0.818	0.691	0.636	0.212
RandQ	0.770	0.679	0.627	0.248

Table 5 shows some statistics about the generated query dimensions. On average for each query in UserQ, there are about 32.1 dimensions generated. Each dimension contains about 20.8 unique items, and 7.5 of them are classified as qualified ones. Among the top five dimensions, about 2.3 dimensions are labeled as good, and 1.2 ones are labeled as fair. The queries in RandQ have less lists and dimensions than UserQ. As we mentioned in Section 4, this is because some randomly sampled queries in RandQ are too specific or not well-formed.

We evaluate the top five query dimensions for each query, and show the results in Table 6. We find that:

(1) Clustering quality on the UserQ collection is good, with a high purity score (0.910) and reasonable scores of NMI, RI, F1, and F5. RandQ has a higher purity of 0.922 but a lower NMI of 0.770 than UserQ. This indicates that more small dimensions, which are from the same ground truth classes, are generated in RandQ than in UserQ. This may happen when the quality of search results is not good and there is not enough evidence to group similar lists.

(2) Rankings of query dimensions on both datasets are effective in terms of nDCG and fp-nDCG. rp-nDCG@5 values are relatively low on both datasets, which indicates that only a small percentage of human labeled items are returned in the output dimensions. This is caused by the following reasons. **Firstly**, some items do not appear in the top search results, and some of them are not presented in list styles. **Secondly**, we just evaluate *qualified* items for each dimension. Table 5 shows that only about 1/3 of items are predicted to be qualified. This may cause low item recall for all methods. **Thirdly**, it is difficult to enumerate all items. For example, there are hundreds of items within the labeled dimension "watch brands" for the query "watches". These items are collected from a large number of results generated by different algorithms, and it is not easy for one specific algorithm to enumerate all of them. **Fourthly**, some items in the labeled dimensions are actually variants of the same meaning. For example, "season one" and "season 1" are ex-

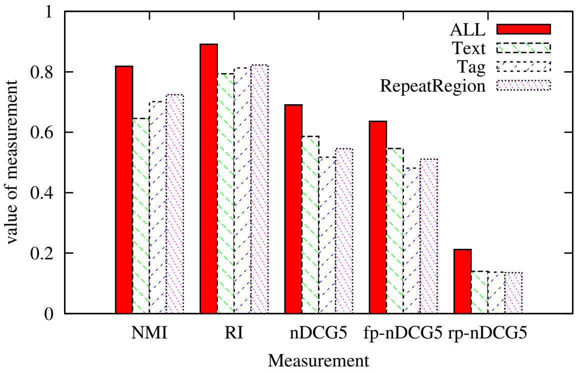


Figure 3: Effectiveness of different types of lists

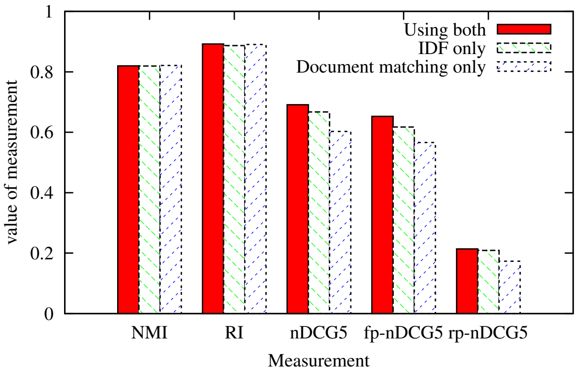


Figure 4: Experiments with list weighting methods

actly the same in meanings. These duplicated items may cause low recall if one algorithm can only return one of them. To overcome the issue, we plan to identify similar items in labeling dimensions in the future.

In the remaining parts of this paper, we only report the results on UserQ due to space limitations. In most experiments, we get the same conclusions on RandQ and UserQ.

5.2 Experiments with Different Types of Lists

As introduced in Section 3.2, we use three different types of patterns to extract lists from webpages, namely free text patterns (Text), HTML tag patterns (Tag), and repeat region patterns (RepeatRegion). In this section, we experiment with different types of lists, and investigate whether they are useful. Experimental results are shown in Figure 3. The figure indicates that the sole use of any type of list yields reasonable results, but a combination of them (series ALL) performs the best. This is intuitive because more lists provide more evidence for weighting important lists and generating better dimensions.

The repeat region based and HTML tag based query dimensions have *better clustering quality but worse ranking quality* than the free text based ones. By analyzing the data, we find that many lists appear in the header or the left region of webpages in well formatted HTML structures. They are originally designed to help users navigate between webpages, and hence we call them "navigational lists" in this paper. These navigational lists are easily extracted by HTML tag based or repeat region based patterns with high precision, but they are usually irrelevant to the query. The first two lists in Table 3 are such kinds of lists. Although we have punished them in Section 3.3, there are still some useless dimensions generated by aggregating them. The lists within free text of webpages, which are extracted based on simple sentence patterns, are short and sometimes noisy. However, they are generally more informative, and are more useful to users.

5.3 Experiments with List Weighting Methods

We integrate two different components, i.e., document matching weight and average invert document frequency (IDF), in evaluating the importance of a list in Section 3.3. In this section, we investigate whether these components are necessary and effective. We experiment with using only of them, and show the experimental results in Figure 4. The results indicate that clustering quality (in terms of NMI and RI) does not significantly change if each component is used or not. This confirms our previous conclusion that our WQT clustering algorithm performs well in most cases. In terms of ranking quality, we have the following findings:

(1) The quality of query dimensions significantly drops when IDF is not used (document matching only), which indicates that the average invert document frequency of items is an important factor. A list that contains common items in a corpus usually gets a high document matching score because most items also occur frequently in the top results of the query. It would be ranked high and generate a useless dimension if we did not consider IDF.

(2) Document matching weight is also helpful to improve the quality of query dimensions. Document matching weight does not affect the ranking of query dimensions as big as IDF. This is because in Equation (2) we use the number of websites (lists) in ranking query dimensions. Thus the function of document matching weight is partially overlapped.

5.4 Experiments with Clustering Algorithms

As introduced in Section 3.4, we use a modified QT clustering algorithm named WQT to group similar lists. In this section, we verify whether WQT is able to generate better query dimensions than QT. We generate query dimensions using both algorithms with different settings of maximum diameter Dia_{max}, and plot experimental results in Figure 5. This figure shows that WQT consistently outperforms QT with four different settings of maximum diameter. As WQT first generates clusters for important lists, it is proved to be able to generate better query dimensions than QT. The figure also indicates that our setting $Dia_{max} = 0.6$ is good for the WQT algorithm, as a lower value may generate less dimensions and a higher value may induce more noisy dimensions.

5.5 Experiments with Search Result Quantity

The top 100 search results are used in the above experiments. In this section, we experiment with various numbers of top results, ranging from 10 to 100, to investigate whether the quality of query dimensions is affected by the quantity of search results. Experimental results are shown in Figure 6. This figure shows that the number of results does affect the quality of query dimensions. Query dimensions become better when more search results are used. This is because more results contain more lists and hence can generate more dimensions. More results also provide more

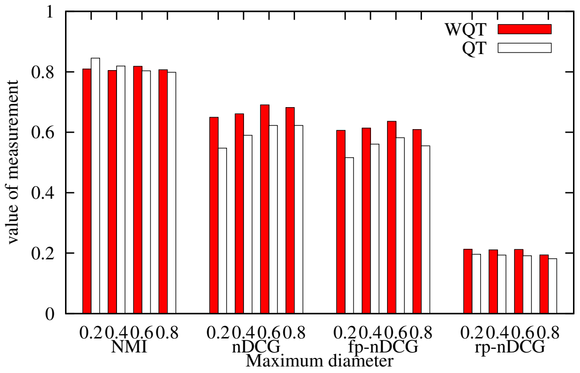


Figure 5: Effectiveness of clustering algorithms

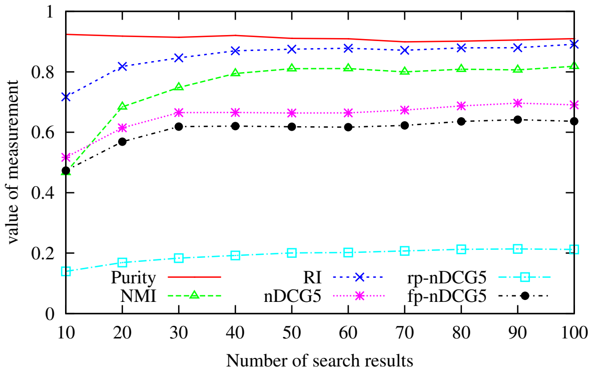


Figure 6: Experiments with search result quantity

evidence for voting the importance of lists, and hence can improve the quality of query dimensions. The figure also shows that the improvement of clustering quality becomes subtle when the number of results is larger than 50. Additional results may help discover more dimension items, but has less impact on the quality of query dimensions. Using top 50 results is already enough for the WQT algorithm to group similar lists into correct dimensions, and most valuable dimensions have already been found. Furthermore, the relevance of later search results also decreases, and the documents may contain less relevant or less useful lists.

In addition, we find that the purity of query dimensions (the red series in Figure 6) decreases a little bit when more results are used. This is because when a small number of results are used, some lists are not merged together but each of them individually has high purity. When similar lists are grouped based on more results, a part of purity may be sacrificed and the generated dimensions may inevitably include some noise from new lists.

5.6 Experiments with Search Result Quality

QDMiner is based on the assumption that most top results of a query are relevant. In this section, we investigate whether our dimension mining algorithms are significantly affected by the quality of search results. We experiment with the following configurations: (1) Top - using the original top K results; (2) TopShuffle - randomly shuffling the top K results; (3) Random - randomly selecting K results from the original 100 results and then shuffling them. In general, the Random method generates worse ranking than TopShuffle, and both perform worse than Top in ranking effectiveness.

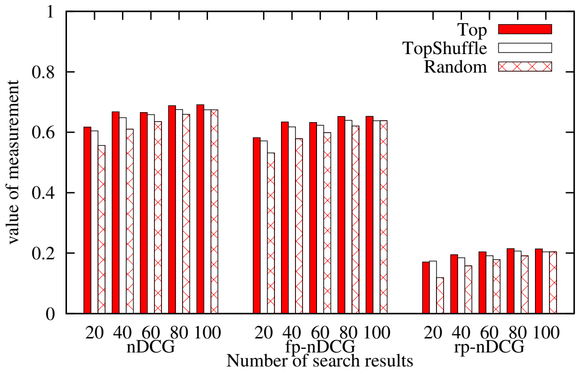


Figure 7: Results based on shuffled search results

Figure 7 shows that method Random is the worst among the three approaches. We find that Random generates much less dimensions than Top and TopShuffle. Consequently, the generated dimensions are usually less relevant to the query, and they also contain less qualified items. Some documents used by the Random method may have drifted to other irrelevant topics. Moreover, Figure 7 shows that shuffling the top results (TopShuffle) harms the quality of query dimensions. We assign larger weights for the lists that are extracted from the top-ranked documents in Equation (1). The TopShuffle method may cause the lists extracted from less relevant documents to be given higher weights, which finally affects the quality of query dimensions. All these results indicate that the quality of search results does affect query dimensions. Note that their clustering quality is comparable, and we skip them due to space limitations.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we study the problem of finding query dimensions. We propose a systematic solution, which we refer to as QDMiner, to automatically mine query dimensions by aggregating frequent lists from free text, HTML tags, and repeat regions within top search results. We create two human annotated data sets and apply existing metrics and two new combined metrics to evaluate the purity and the ranking quality of query dimensions. Experimental results show that a large number of lists exist in the top search results. Meaningful and useful query dimensions are mined by aggregating these lists using QDMiner.

As the first approach of finding query dimensions, QDMiner can be improved in many aspects. For example, some semi-supervised bootstrapping list extraction algorithms can be used to iteratively extract more lists from the top results. Specific website wrappers can also be employed to extract high-quality lists from authoritative websites. Adding these lists may improve both accuracy and recall of query dimensions. Part-of-speech information can be used to further check the homogeneity of lists and hence improve the quality of query dimensions. We will explore these topics to refine QDMiner in the future.

We will also investigate some other related topics to finding query dimensions. Good descriptions of query dimensions may be helpful for users to better understand the dimensions. How to automatically generate meaningful descriptions is an interesting research topic. We also plan to utilize query dimensions to improve Web search. Some candidate directions include using query dimensions to improve search result diversity, using dimensions to provide structured query suggestions, and building a general faceted search system based on QDMiner.

7. REFERENCES

[1] J. Allan. Relevance feedback with too much data. In *Proceedings of SIGIR '95*, pages 337–343, 1995.

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

[3] K. Balog, E. Meij, and M. de Rijke. Entity search: building bridges between two worlds. In *Proceedings of SEMSEARCH '10*, pages 9:1–9:5, 2010.

[4] 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.

[5] M. Bron, K. Balog, and M. de Rijke. Ranking related entities: components and analyses. In *Proceedings of CIKM '10*, pages 1079–1088, 2010.

[6] M. J. Cafarella, A. Halevy, D. Z. Wang, E. Wu, and Y. Zhang. Webtables: exploring the power of tables on the web. *VLDB*, 1:538–549, August 2008.

[7] D. Cai, S. Yu, J.-r. Wen, and W.-y. Ma. Vips: a vision-based page segmentation algorithm, 2003.

[8] Y. Chali and S. R. Joty. Unsupervised approach for selecting sentences in query-based summarization. In *FLAIRS Conference*, pages 47–52, 2008.

[9] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Rec.*, 26:65–74, March 1997.

[10] T. Cheng, X. Yan, and K. C.-C. Chang. Supporting entity search: a large-scale prototype search engine. In *Proceedings of SIGMOD '07*, pages 1144–1146, 2007.

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

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

[13] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic faceted search for discovery-driven analysis. In *CIKM '08*, 2008.

[14] 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.

[15] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in knowitall: (preliminary results). In *Proceedings of WWW '04*, pages 100–110, 2004.

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

[17] 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.

[18] L. J. Heyer, S. Kruglyak, and S. Yooseph. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Research*, 9(11):1106–1115, November 1999.

[19] W. H. Inmon. *Building the Data Warehouse,3rd Edition.* John Wiley & Sons, Inc., 3rd edition, 2002.

[20] Z. Kozareva, E. Riloff, and E. H. Hovy. Semantic class learning from the web with hyponym pattern linkage graphs. In *Proceedings of ACL'08*, 2008.

[21] B. Y.-L. Kuo, T. Hentrich, B. M. . Good, and M. D. Wilkinson. Tag clouds for summarizing web search results. In *Proceedings of WWW '07*, 2007.

[22] K. Latha, K. R. Veni, and R. Rajaram. Afgf: An automatic facet generation framework for document retrieval. In *Proceedings of ACE '10*, 2010.

[23] 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*, pages 651–660. ACM, 2010.

[24] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[25] M. Mitra, A. Singhal, and C. Buckley. Improving automatic query expansion. In *Proceedings of SIGIR '98*, pages 206–214, 1998.

[26] V. Nastase and M. Strube. Decoding wikipedia categories for knowledge acquisition. In *Proceedings of AAAI '08*, pages 1219–1224, 2008.

[27] D. R. Radev, H. Jing, M. Stys, and D. Tam. Centroid-based summarization of multiple documents. *Information Processing and Management*, 40(6):919 – 938, 2004.

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

[29] 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*, pages 993–1001, 2010.

[30] K. Shinzato and T. Kentaro. A simple www-based method for semantic word class acquisition. In *RANLP '05*, 2005.

[31] E. Stoica and M. A. Hearst. Automating creation of hierarchical faceted metadata structures. In *NAACL HLT '07*, 2007.

[32] R. C. Wang and W. W. Cohen. Iterative set expansion of named entities using the web. In *Proceedings of ICDM '08*, pages 1091–1096, 2008.

[33] R. W. White, M. Bilenko, and S. Cucerzan. Studying the use of popular destinations to enhance web search interaction. In *Proceedings of SIGIR '07*, 2007.

[34] J. Xu and W. B. Croft. Query expansion using local and global document analysis. In *Proceedings of SIGIR '96*, pages 4–11, 1996.

[35] 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.

[36] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *Proceedings of WWW '06*, pages 1039–1040, 2006.