Volume 2, Issue 2, February 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

A REVIEW ON QUERY CLUSTERING ALGORITHMS FOR SEARCH ENGINE OPTIMIZATION

Bhupesh Gupta [1]
CSE Deptt., MMUniversity
Mullana, India

Sandip Kumar Goyal [2]
CSE Deptt., MMUniversity
Mullana, India

Ashish Oberoi[3]
CSE Deptt., MMUniversity
Mullana, India

Abstract — **Classification of patterns into groups in unsupervised way represents clustering. Clustering can be done in many ways and by researchers in many disciplines, like clustering can be done on the basis of queries submitted to search engine. This paper provides an overview of algorithms which are helpful in search engine optimization. The algorithms discuss are BB's Graph Based Clustering Algorithm, Concept Based Clustering Algorithm and Personalized Concept based clustering algorithm. All the algorithm works on the basis of precision and recall values, which are helpful in determine the efficiency of search engine queries.**

Keywords — **Concept based clustering, Query Clustering, Search Engine, Personalization, Query Optimization.**

I. INTRODUCTION

Query clustering is a technique for discovering similar queries on a search engine. Also it is a class of techniques aiming at grouping users' semantically related, not syntactically related, queries in a query repository, which were accumulated with the interactions between users and the system. The driving force of the development of query clustering techniques comes recently from the requirements of modern web searching. The three main applications of query clustering are detection of frequently asked questions, index term selection and query reformulation. The query clustering comes out in following types: Content-based Query Clustering, Session-based Query Clustering, Graph-based Query Clustering, Concept-based Query Clustering and Personalized concept based query clustering.

II. RELATED WORK

Query clustering techniques have been developed in diversified ways. The very first query clustering technique comes from information retrieval studies [1]. Similarity between queries was measured based on overlapping keywords or phrases in the queries. Each query is represented as a keyword vector. Similarity functions such as cosine similarity or Jaccard similarity [1] were used to measure the distance between two queries. One major limitation of the approach is that common keywords also exist in unrelated queries.

Wen et al. [2] proposed a clustering algorithm combining both query contents and URL clicks. They suggested that two queries should be clustered together, if they contain the same or similar terms, and lead to the selection of the same documents. However, since Web search queries are usually short and common clicks on documents are rare (see discussion below), Wen et al.'s method may not be effective for disambiguating Web queries. In contrast, our approach relates the queries with a set of extracted concepts in order to identify the precise semantics of the search queries.

Joachims [4] proposed a method which employs preference mining and machine learning to model users'clicking and browsing behavior. Joachims' method assumes that a user would scan the search result list from top to bottom. If a user has skipped a document di at rank i before clicking on document dj at rank j, it is assumed that he/she must have scan the document di and decided to skip it. Thus, we can conclude that the user prefers document dj more than document di (i.e., dj <r0 di, where r0 is the user's preference order of the documents in the search result list).

Baeza-Yates et al. [7] proposed a query clustering method that groups similar queries according to their semantics. The method creates a vector representation Q for query q, and the vector Q is composed of terms from the clicked documents of q. Cosine similarity is applied to the query vectors to discover similar queries.

More recently, Zhang and Nasraoui [10] presented a method that discovers similar queries by analyzing users' sequential search behavior. The method assumes that consecutive queries submitted by a user are related to each other. The sequential search behavior is combined with a traditional content based similarity method to compensate for the high sparsity of real query log data.

On Web search engines, clickthrough data is a kind of implicit feedback from users. it is a valuable resource for capturing the user's interest for building personalized Web search systems [4]-[6], [8]-[9], [11]-[15]. Joachims [4] proposed a method that employs preference mining and machine learning to re rank search results according to user's personal preferences. Later on, Smyth et al. [8] suggested that user search behavior is repetitive and regular. They proposed to re rank search results such that the results that have been previously selected for a given query are promoted ahead of other search results.

Ng et al. [16] proposed an algorithm which combines a spying technique together with a novel voting procedure to determine users' document preferences from the clickthrough data. They also employed the RSVM algorithm to learn the user behavior model as a set of weight features. More recently, Agichtein et al. [17] suggested that explicit feedback (i.e., individual user behavior, clickthrough data, etc.) from search engine users is noisy. One major observation is the bias of user click distribution toward top ranked results. To resolve the bias, Agichtein suggested cleaning up the clickthrough data with the aggregated "background" distribution. RankNet [18], a scalable implementation of neural networks, is then employed to learn the user behavior model from the cleaned click through data.

III. QUERY CLUSTERING ALGORITHMS

How to choose query clustering algorithm Choosing an appropriate clustering algorithm is also very critical to the effectiveness and efficiency of the query clustering process. While choosing the clustering algorithm, the following things must be kept in mind:

The algorithm should be capable of handling a large data set within reasonable time and space constrained.

The algorithm should be easily extended to cluster new queries incrementally.

The algorithm should not require manual setting of the resulting form of the clusters.

If user interested in only finding FAQs, then algorithm should filter out those queries with low frequencies.

Based upon above assumptions we have following query clustering algorithms:

A. BB's Graph Based Clustering Algorithm

In BB's graph-based clustering [3], a query-page bipartite graph is first constructed with one set of the nodes corresponding to the set of submitted queries, and the other corresponding to the sets of clicked pages. If a user clicks on a page, a link between the query and the page is created on the bipartite graph. After obtaining the bipartite graph, an agglomerative clustering algorithm is used to discover similar queries and similar pages. During the clustering process, the algorithm iteratively combines the two most similar queries into one query node, then the two most similar pages into one page node, and the process of alternative combination of queries and pages is repeated until a termination condition is satisfied. The main reason for not clustering all the queries first and then all the pages next are that two queries may seem unrelated prior to page clustering because they link to two different pages but they may become similar to each other if the two pages have a high enough similarity to each other and are merged later. The example in Fig. 1 helps illustrate this scenario. To compute the similarity between queries or documents on a bipartite graph, the algorithm considers the overlap of their neighboring vertices as defined in the following equation:

$$\mathrm{Sim}(x,y)= |N(x) \cap N(y)|/|N(x) \cup N(y)| \quad \text{if } |N(x) \cup N(y)| > 0, \quad 0, \text{ otherwise}$$

where N(x) is the set of neighboring vertices of x, and N(y)is the set of neighboring vertices of y. Intuitively, the similarity function formalizes the idea that x and y are similar if their respective neighboring vertices largely overlap and vice versa.

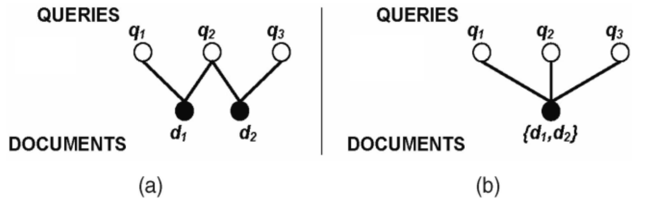

Fig.1 Queries q1 and q3 seem unrelated before document clustering.(b) After document clustering, queries q1 and q3 are then related to each other because they are both linked to the document cluster (d1; d2).

B. Concept Based Clustering Algorithm

1 Clustering on Query Concept Bipartite Graph: We now describe our concept-based algorithm (i.e.,BB's algorithm using query-concept bipartite graph) for clustering similar queries. Similar to BB's algorithm, our technique is composed of two steps: 1) Bipartite graph construction using the extracted concepts and 2) agglomerative clustering using the bipartite graph constructed in step 1.

Using the extracted concepts and clickthrough data, the first step of our method is to construct a query-concept bipartite graph, in which one side of the vertices correspond to unique queries, and the other corresponds to unique concepts. If a user clicks on a search result, concepts appearing in the web-snippet of the search result are linked to the corresponding query on the bipartite graph. Algorithm1 shows the first step of our method.

Algorithm 1 Bipartite Graph Construction

```
Input: Clickthrough data CT, Extracted Concepts E
Output: A Query-Concept Bipartite Graph G
1: Obtain the set of unique queries Q = {q1,q2,q3…….} from CT
2: Obtain the set of unique concepts C = {c1,c2,c3…….}from E
3: Nodes (G) = Q U C where Q and C are the two sides in G
4: If the web-snippet s retrieved using qi ϵQ is clicked by a user, create an edge e= (qi,cj) in G, where cj is a concept appearing in s.
```

After the bipartite graph is constructed, the agglomerative clustering algorithm is applied to obtain clusters of similar queries and similar concepts. The noise-tolerant similarity function (recall (2)) is used for finding similar vertices on the bipartite graph G. The agglomerative clustering algorithm would iteratively merge the most similar pair of white vertices and then merge the most similar pair of black vertices and so on. We present the details in Algorithm 2.

```
Algorithm 2 Agglomerative Clustering
Input: A Query-Concept Bipartite Graph G
Output: A Clustered Query-Concept Bipartite Graph $G^c$
1: Obtain the similarity scores for all possible pair's of queries in G using the noise-tolerant similarity function given in (2).
2: Merge the pair of queries ($q_i q_j$) that has the highest similarity score.
3: Obtain the similarity scores for all possible pair's of concepts in G using the noise-tolerant similarity f unction given in (2).
4: Merge the pair of concepts (ci; cj) that has the highest similarity score.
5. Unless termination is reached, repeat steps 1-4.
```

2 Personalized Concept based clustering: We now explain the essential idea of our personalized concept-based clustering algorithm with which ambiguous queries can be clustered into different query clusters. Personalized effect is achieved by manipulating the user concept preference profiles in the clustering process. An example is shown in Fig. 2a. We can see that the query "apple" submitted by users User1 and User3 become two vertices "appleUser1" and "appleUser3." If User1 is interested in the concept"apple store," as recorded in the concept preference profile, a link between the concept "apple store" and the query "apple(User1)" would be created. On the other hand, if User3is interested in the concept "fruit," a link between the concept "fruit" and "apple(User3)" would be created. After the personalized bipartite graph is created, our initial experiments revealed that if we apply BB's algorithm directly on the bipartite graph, the query clusters generated will quickly merge queries from different users together, thus losing the personalization effect. We found that identical queries, though issued by different users and having different meanings, tend to have some generic concept nodes such as "information" in common, e.g., "apple(User1)" and "apple(User3)" both connect to the "information" concept node in Fig. 2a. Thus, these query nodes will likely be merged in the first few iterations and cause more queries from different users to be merged together in subsequent iterations. Considering Fig. 2a again, if "apple(User1)" and "apple(User3)" are merged, the next iteration will merge the concept nodes "applestore," "fruit," and "information." When the clustering algorithm goes further, queries across users will be further clustered together. At the end, the resulting query clusters have no personalization effect at all. To resolve the problem, we divide clustering into two steps. In the initial clustering step, an algorithm similar to BB's algorithm is employed to cluster all the queries, but it would not merge identical queries from different users. After obtaining all the clusters from the initial clustering step, the community merging step is employed to merge query clusters containing identical queries from different users. We can see from Fig. 2d that "apple(User1)" and "apple(User3)" belong, correctly, to different clusters. Algorithm 3 shows the details of the personalized clustering algorithm. Similar to the BB's algorithm, a query-concept bipartite graph is created as input for the clustering algorithm. The bipartite graph construction algorithm is similar to Algorithm 1, except each individual query submitted by each user is treated as an individual vertex in the bipartite graph.

```
Algorithm 3 Personalized Agglomerative Clustering
Input: A Query-Concept Bipartite Graph G
Output: A Personalized Clustered Query-Concept
Bipartite Graph $G^p$
// Initial Clustering
1: Obtain the similarity scores in G for all possible pairs of queries using the noise-tolerant similarity function given in (2).
2: Merge the pair of most similar queries (qi, qj) that does not contain the same queries from different users.
3: Obtain the similarity scores in G for all possible pairs of concepts using the noise-tolerant similarity function given in (2).
4: Merge the pair of concepts (ci,cj) having highest similarity score.
5. Unless termination is reached, repeat steps 1-4.
// Community Merging
6. Obtain the similarity scores in G for all possible pairs of queries using the noise-tolerant similarity function given in (2).
7. Merge the pair of most similar queries (qi, qj) that contains the same queries from different users.
8. Unless termination is reached, repeat steps 6 and 7.
```

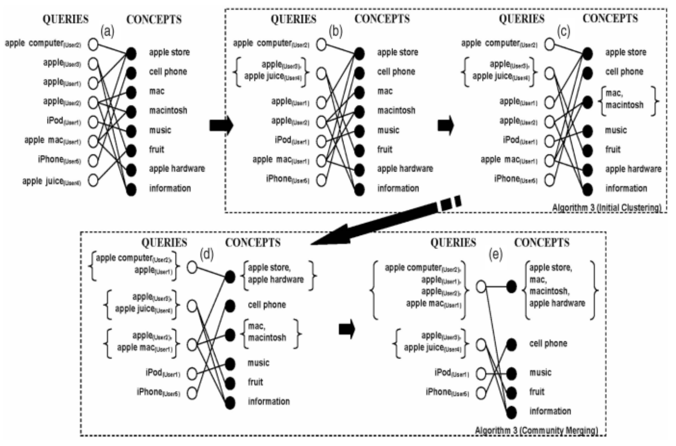

Fig. 2. Performing personalized concept-based clustering algorithm on a small set of clickthrough data. Starting from top left: (a) the original bipartite graph. (b), (c) initial clustering. (d), (e) Community merging.

Initial clustering (i.e., steps 1-5 of Algorithm 3) is similar to BB's agglomerative algorithm as already discussed. However, queries from different users are not allowed to be merged in initial clustering. Figs. 2b and 2c show examples of query and concept merging, respectively. Fig. 2d illustrates the result of initial clustering. In community merging (i.e., steps 6-8 of Algorithm 3), query clusters containing identical queries from different users are compared for merging. Figs. 2dand 2e show an example of query cluster merging. The query clusters {apple computer(User2); apple(User1)} and {apple(User2) and apple mac(User1)} both contain the query "apple" and are leading to the same concept "apple store." Therefore, they are merged in community merging as one big cluster.

IV CONCLUSION

The algorithms described in this paper are fully capable of clustering of search engine queries. If one performs experiment with these algorithms then he/she finds better precision and recall then the existing query clustering methods. Better precision and recall values increases the effectiveness of search engine queries, which does not add extra burden to the users. Also with the help of these algorithms one can improves prediction accuracy and computational cost. Future work can be extended by getting better precision and recall values for search engine queries.

V References

[1] G. Salton and M.J. Mcgill, *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[2] J. Wen, J. Nie, and H. Zhang, *"Query Clustering Using User Logs,"* ACM Trans. Information Systems, vol. 20, no. 1, pp. 59-81,2002.

[3] D. Beeferman and A. Berger, *"Agglomerative Clustering of a Search Engine Query Log,"* Proc. ACM SIGKDD, 2000.

[4] T. Joachims, *"Optimizing Engines Using Clickthrough Data,"* Proc. ACM SIGKDD, 2002.Search

[5] F. Liu, C. Yu, and W. Meng, *"Personalized Web Search for Improving Retrieval Effectiveness,"* IEEE Trans. Knowledge and Data Eng., vol. 16, no. 1, pp. 28-40, Jan. 2004.

[6] Q. Tan, X. Chai, W. Ng, and D.L. Lee, *"Applying Co-Training to Clickthrough Data for Search Engine Adaptation,"* Proc. Ninth Int'l. Conf. Database Systems for Advanced Applications (DASFAA), 2004.

[7] R.A. Baeza-Yates, C.A. Hurtado, and M. Mendoza, *"Query Recommendation Using Query Logs in Search Engines,"* Proc. EDBT Workshop, vol. 3268, pp. 588-596, 2004.

[8] B. Smyth et al., *"Exploiting Query Repetition and Regularity in an Adaptive Community-Based Web Search Engine,"* User Modeling and User-Adapted Interaction, vol. 14, no. 5, pp. 383-423, 2005.

[9] M. Speretta and S. Gauch, *"Personalized Search Based on User Search Histories,"* Proc. IEEE/WIC/ACM Int'l Conf. Web Intelligence (WI), 2005.

[10] Z. Zhang and O. Nasraoui, "*Mining Search Engine Query Logs for Query Recommendation,"* Proc. 15th Int'l World Wide Web Conf. (WWW), 2006.

[11] E. Agichtein, E. Brill, and S. Dumais, "*Learning User Interaction Models for Predicting Web Search Result Preferences*," Proc. 29th Ann. Int'l ACM SIGIR Conf. (SIGIR), 2006.

[12] E. Agichtein, E. Brill, S. Dumais, and R. Rango, *"Improving Web Search Ranking by Incorporating User Behavior Information*," Proc. 29th Ann. Int'l ACM SIGIR Conf. (SIGIR), 2006.

[13] L. Deng, W. Ng, X. Chai, and D.L. Lee, *"Spying Out Accurate User Preferences for Search Engine Adaptation,"* Advances in Web Mining and Web Usage Analysis, LNCS 3932, pp. 87-103, 2006.

[14] T. Joachims and F. Radlinski, "*Search Engines That Learn from Implicit Feedback*," Computer, vol. 40, no. 8, pp. 34-40, 2007.

[15] Z. Dou, R. Song, and J.R. Wen, "*A Large-Scale Evaluation and Analysis of Personalized Search Strategies,"* Proc. 16th Int'l World Wide Web Conf. (WWW), 2007.

[16] W. Ng, L. Deng, and D.L. Lee, *"Mining User Preference Using Spy Voting for Search Engine Personalization,*" ACM Tra , Internet Technology, vol. 7, no, 4 article 19,2007.

[17] E. Agichtein, E. Brill, and S. Dumais, *"Improving Web Search Ranking by Incorporating User Behavior Information,"* Proc. ACM SIGIR, 2006. ns. Internet Technology, vol. 7, no. 4, article 19, 2007.

[18] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "*Learning to Rank Using Gradient Descent*," Proc. Int'l Conf. Machine learning (ICML), 2005.