A Study and Implementation on User Search Histories

Shabuddin Sheik[*], B. Nandana Kumar[#]
[*#]Student, Asst.Professor
[*#]Dept of CSE, Swarnandhra College of Engg & Technology, Seetharampuram,WGDT,AP

Abstract: Organizing the user search logs is rapidly increasing in the field of data mining for finding the user interestingness and organizing the user search requirements in a proper way .Daily billions of queries can be passed to the server for relevant information, most of the search engines retrieves the information based on the query similarity score or related links with respect to the given query.In this paper we are proposing an efficient clustering mechanism for group up the similar type of query that helps in organizing user search histories.

I.INTRODUCTION

As of today, the indexed web contains at least 30 billion pages [1]. In fact, the overall web may consist of over 1 trillion unique URLs, more and more of which is being indexed by search engines every day. Out of this morass of data, users typically search for the relevant information that they want by posing search queries to search engines. The problem that the search engines face is that the queries are very diverse and often quite vague and/or ambiguous in terms of user basic inputs. Most of the individual queries may refer to a single concept, while a single query may correspond to several techniques. For organize and bring some order to this massive unstructured dataset, search engines cluster these queries to group similar items together. To increase usability, most commercial search engines and also augment their search facility through additional services such as query recommendation or query suggestion. These services make it more convenient for users to issue queries and obtain accurate results from the websearch engine, and thus it is quite valuable. From the search engine view, efficient group of search queries is a necessary pre-requisite for these services to function well. As the size and richness of information on web increases, because does the variety and the complexity of tasks the users try to complete online. End users are no longer content with issuing simple navigational queries. Various studies on query logs (e.g., Yahoo's [1] and AltaVista's [2]) reveal that only about 20 percent of queries are Hierarchal. The remaining are informational or transactional . since users now pursue much broader informational and task oriented goals such as arranging for travel of future , managing their finances, or planning their purchase plans.[1]-[3].

However, the primary means of accessing information online is still through keyword queries to a websearch engine. A typical task such as travel arrangement has to be broken down into a number of codependent parts over a span of period. For a time, users may initial search on possible paths, timestamps, events, etc .then deciding planning of the schedule, the user may then search for the most suitable arrangements for basic requirements(air tickets, rental cars, lodging, meals, etc.) Each step requires one or more queries, and each query results in one or more clicks on relevant pages.[4]

The K-means algorithm is one of the most frequently used investigatory algorithms in data analysis. The proposed approach of algorithm finds to locate K models or averages throughout a data set in such a way that the K prototypes in some way best represents the data. The algorithm is one of the rest which a data analyst will use to investigate a new data set because it is algorithmically simple, relatively robust and gives `good enough' answers over a wide variety of data sets: it will often not be the single best algorithm on any individual data set but it may be close to the optimal over a wide range of data sets. However the algorithm is known to suffer from the defect that the means or prototypes found depend on the initial values given to them at the start of the simulation: a typical program will converge to a local optimum. There are a number of heuristics in the literature which attempt to address this issue but, at heart, the fault lies in the performance function on which K-means is based.In this paper we are introducing an enhanced dbscan algorithm for organizing the user search histories[5][6].

II.RELATED WORK

The concept of query similarity was originally used in information retrieval studies [2]: measuring the similarity between the content-based keywords of two queries. However, the problem with using this in the query log environment is that users' search interests are not always the same even if the issued queries contain the same keywords. For instance, the keyword "Apple" may

represent a popular kind of fruit whereas it is also the keyword of a popular company "Apple Inc.". Hence, the use of content-based keywords descriptor is rather limited for this purpose. Subsequently, to measure similarity between two queries, the query representation of a vector of URLs in a clickthrough bipartite graph [3][4][5] has been adopted. Nevertheless, no matter how large the query log data set is, it is possible that the complete search intent of some queries may not be adequately represented by the available click-through information. For instance, in a particular large-scale query log, there may be no clicked URL for the query "Hoonda vs. Toyotaa". since, even if it is clearly relevant to the query "Hondaa", on the basis of this click-through data, there is no similarity. Therefore, existing query log data is not accurate enough for analyzing users' search intent, especially for those queries without any clicked URL. Another reason that causes inaccuracy is that the query log data comprise users' click-through information in a specific period, while search interests might even change over time. If we utilize an aggregated query logs collected in a long period to compare and grouping queries, final accuracy may be calculated.

A) Using Query Keywords :

The first group of related clustering approaches is certainly those that cluster documents using the keywords it. In proposed approaches, in general, a document is represented as a vector in a vector space generated by all the keywords]. Academicians have been concerned mostly with the following two aspects:

— similarity function
— algorithms for the clustering process

Typically, people use either cosine-similarity, Jaccard-similarity or Dicesimilarity [Salton and McGill 1983] as similarity functions. The edit-distance is also used in some other approaches [Gusfield 1997]. As to clustering algorithms, there are mainly two availble groups: hierarchical and non-hierarchical. Hierarchical agglomerative clustering (HAC) algorithm and k-means are representatives of the two groups [Dubes and Jain 1988].
Keyword-based document clustering has provided interesting results. One contributing factor is the large number of keywords contained in documents. Even if some of the keywords of two similar documents are different, there are still many others that can make the documents similar in the similarity calculation. However, because, specifically the queries submitted to the websearch engines ,usually are very short, in many cases it is hard to deduce the semantics from the query itself. Therefore, keyword alone do not provide a reliable basis for clustering queries effectively. In addition, words such as "where" and "who" are treated as stop words in traditional IR methods. For questions, however, these words (if they occur) encode important information about the user's requirement, specifically in the new-generation web search engines such as AskJeeves. Considering an example, along a "who" input parameter, the user intends to find information about a person. So even if a keyword-based approach is used in query clustering, it should be modified from that used in traditional document clustering. Special attention is paid to such words in question answering (QA) [Kulyukin et al. 1998] [Srihari and Li 1999], where they are used as prominent indicators of type of question. The entire question is represented as a template in accordance
with the question type. During input question evaluation, the input requirement template may be elobrated using a thesaurus (e.g. WordNet [Miller 1990]), or morphological changes, in our proposed case, we found that well-formed natural language input questions represented only a small part of queries. Most probable queries are simply short parts of phrases or keywords (e.g. "population of U.S."). The approach used in QA is therefore not completely applicable to our method. Even though, if keywords denoting a question type do appear in a complete input question, those words should be taken into consideration.

B) Using Hyperlinks:

Because of the limitations of keywords, people have been looking for additional criteria for document clustering or grouping, One of them is the hyperlinks between group of documents. The hypothesis is that hyperlinks connect similar documents. The proposed idea had been used in some early studies in IR [Garfield 1983] [Kessler 1963]. More recent examples are Google (http://www.google.com) and the authority/hub calculation of Kleinberg [1998]. Although Google does not perform document clustering explicitly, its PageRank algorithm still results in a weighting of hyperlinks to a document, it is then direct to know the documents that are the most strongly related to it according to the weights of the hyperlinks to/from the document. Since Therefore, we can see Page Ranking as an implicit clustering approach. Google's use of hyperlinks has been mostly successful, organizing and making it one of the best search engines currently available. The same idea is difficult to apply to query clustering, however, because there is no link between queries

III.PROPOSED WORK

To determine an appropriate clustering method, one first has to choose an appropriate clustering algorithm. There are many clustering algorithms available to us. The main characteristics that guide our choice are the following: As query logs usually are more, the algorithm approach should be capable of handling a large data set within reasonable time and space constraints. The algorithm should not require manual setting of the resulting form of the clusters, for consideration, number of maximal size of final clusters. It is not reasonable to determine these parameters in advance.
—because we only need to find Frequently asked questions, the proposed algorithm should filter out those queries with low frequencies.
—Due to t that reason the log data changes daily, the algorithm should be incremental.

The density-based clustering method DBSCAN [Ester et al. 1996] and its incremental version Incremental DBSCAN [Ester et al. 1998] satisfy the above requirements. Our proposed DBSCAN does not require the number of clusters as an input parameters, the cluster consists of at least the min number of points—MinPts (to eliminate very small clusters as noise); and for each point in the cluster and there is another point in the same cluster whose distance is less than the distance threshold Eps (points are densely located).this approach makes use of a spatial indexing structure (R*-tree) to locate points within the Eps distance from the core points of the clusters. Total clusters consisting of less than the minimum number of points are considered as "noise" and had been discarded. average time complexity of this DBSCAN algorithm is O(n*logn). Previous experiments showed that DBSCAN outperforms CLARANS [Ng and Han 1994] by a factor of between [250 and 1900], it increases with the size of the data set. During our experiments, it only requires 3 minutes to deal with one-day user logs of thousands of queries. its ability of Incremental DBSCAN to update incrementally is due to the density-based nature of the DBSCAN approach , which the insertion or deletion of an object only affects the neighborhood of this entity and based on the formal definition of clusters, it has been proven that the incremental algorithm yields the same results as DBSCAN. Performance evaluations show Incremental DBSCAN to be more efficient than the basic DBSCAN algorithm

To find a cluster, DBSCAN starts with an arbitrary point p and retrieves all points density-reachable from p with respect to. Min distance(Ep_s) and minimum number of points (MinPts). If p is a core point, it procedure produces a cluster wrt. Eps and MinPts (see Lemma 2). If p consider is a border point p and no points are density-reachable from p and DBSCAN visits the next point of the database therefor we use global values for (Min distance)Eps and MinPts ,the algorithm DBSCAN may merge two clusters according to definition 5 into single cluster, if two clusters of different density are "close" to each other. Let us consider the *distance between two sets of points* S1 and S2 be defined as dist (S1, S2) = min {dist(p,q) | p∈S1, q∈S2}. Then, two sets of points having at least the density of the thinnest cluster will be separated from each other only if the distance between the two sets is larger than min distance and Consequently, a recursive call of DBSCAN may be necessary for the detected clusters with a higher value for MinPts and however, no disadvantage because the recursive application of DBSCAN yields an elegant and very efficient basic approach. Furthermore and recursive clustering of the points of a cluster is only necessary under conditions that can be easily detected and the following, we present a basic version of DBSCAN omitting details of data types and generation of additional information about clusters:

```
DBSCAN (SetOfPoints, Eps, MinPts)
// SetOfPoints is UNCLASSIFIED
ClusterId := nextId(NOISE);
FOR i FROM 1 TO SetOfPoints.size DO
   Point := SetOfPoints.get(i);
 IF Point.ClId = UNCLASSIFIED THEN
   IF ExpandCluster(SetOfPoints, Point,
        ClusterId, Eps, MinPts) THEN
     ClusterId := nextId(ClusterId)
   END IF
 END IF
END FOR
END; // DBSCAN
```

SetOfPoints is either the whole database or a discovered cluster from a previous minimum distance(Eps) and MinPts are the global density parameters determined either manually or according to the heuristics presented. The function Set_Of_Points.get(i) returns the i-th element of Set_Of_Points and most important function used by DBSCAN is Expand_Cluster which is presented below:

```
Expand_Cluster(Set_Of_Points, Point, ClId, Eps,
         MinPts) : Boolean;
 seeds:=Set_Of_Points.regionQuery(Point,Eps);
 IF seeds.size<MinPts THEN // no core point
   SetOfPoint.changeClId(Point,NOISE);
   RETURN False;
ELSE // all points in seeds are density-
     // reachable from Point
    Set_Of_Points.changeClIds(seeds,ClId);
    seeds.delete(Point);
 WHILE seeds <> Empty DO
  currentP := seeds.first();
  result := Set_Of_Points.regionQuery(currentP,
                                Eps);
IF result.size >= MinPts THEN
    FOR i FROM 1 TO result.size DO
       resultP := result.get(i);
    IF resultP.ClId
      IN {UNCLASSIFIED, NOISE} THEN
     IF resultP.ClId = UNCLASSIFIED THEN
      seeds.append(resultP);
     END IF;
    Set_Of_Points.changeClId(resultP,ClId);
    END IF; // UNCLASSIFIED or NOISE
  END FOR;
END IF; // result.size >= MinPts
seeds.delete(currentP);
  END WHILE; // seeds <> Empty
   RETURN True;
  END IF
END; // ExpandCluster
```

A call of Set_Of_Points.regionQuery(Point,Eps)returns the Eps-Neighborhood of

Point in Set_Of_Points as a list of points. Region queries can be supported efficiently by spatial access methods such as R*-trees (Beckmann et al. 1990) which are assumed to be available in a SDBS for efficient processing of several types of spatial queries (Brinkhoff et al. 1994). The height of an R*-tree is O(log n) for a database of n points in the worst case and a query with a

"small" query region has to traverse only a limited number of paths in the R*-tree. Since the Eps- Neighborhoods are expected to be small compared to the size of the whole data space complexity and average run time complexity of a single region query as O(log n) is defined for each of the n points of the data, we had at most one region query and Thus, average run time complexity of DBSCAN is as O(n * log n).

The ClId (clusterId) of points which have been marked to be NOISE may be changed later, when they are density-reachable from some other point of the data and This happens for border points of a cluster and these points are not added to the seeds-list because we already know that a point with a ClId of NOISE is not a core point and by Adding those points to seeds would only result in additional region queries which would yield no generated answers. when two clusters C1 and C2 are very close to each other, it may occurs that some point p belongs to both, C1 and C2 and then p must be a border point in both clusters because otherwise C1 would be equal to C2 since we use global constants, In this case, point p will be assigned to the cluster discovered initially and Excluding from these rare situations, the result of DBSCAN is independent of the order in which the points of the database are visited due to Lemma 2.

The basic approach of how to determine the parameters *Eps* and *MinPts* is to look at the behavior of the distance from a point to its **k**th nearest neighbor and which is called **k**-dist and these **k**-dists are computed for all the data points for some number of k, points sorted in ascending order, and then plotted using the sorted values, as a result leads to a sharp change is expected to see. The sharp change at the value of **k**-dist corresponds to a suitable value of *Eps*. Note that the value of *Eps* that is determined in this way depends on **k**, but does not change dramatically as **k** changes. Because DBSCAN uses a density-based definition of a cluster and it is relatively resistant to noise and can handle clusters of different shapes and sizes. Thus, DBSCAN can find many clusters that could not be found using some other clustering algorithms like K-means, always the main weakness of DBSCAN is that it has trouble when the clusters have greatly density varies To sweep over the limitations of DBSCAN, VDBSCAN is acquainted. Firstly, VDBSCAN calculates and stores k-dist for each project and partition k-dist plots. Secondly, the number of densities is given intuitively by k-dist plot. Thirdly, choose parameters *Epsi* automatically for each density. Fourthly, scan the dataset and cluster different densities using corresponding *Epsi*. And finally, display the valid clusters corresponding with varied densities.

IV.CONCLUSION

In this paper we enhanced the mechanism of organizing the user search histories by providing the improved dbscan algorithm, it removes the unnecessary data points (Query url links).It is variable length and we need not to specify the number of clusters prior clustering. In this improved dbscan algorithm density factor is depends on k-dist plot. Here that generates the optimal clusters

REFERENCES

[1] http://www.worldwidewebsize.com/. [2] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, Inc., 1986.

[3] D. Beeferman and A. L. Berger, "Agglomerative clustering of a search engine query log," in *KDD*, pp. 407–416, 2000.

[4] J.-R. Wen, J.-Y. Nie, and H. Zhang, "Query clustering using user logs," *ACM Trans. Inf. Syst.*, vol. 20, no. 1, pp. 59–81, 2002.

[5] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li, "Context-aware query suggestion by mining click-through and session data," in *KDD*, pp. 875–883, ACM, 2008.

[6] T. Joachims, "Optimizing search engines using click through data," in *KDD*, pp. 133–142, ACM, 2002.

[7] E. Agichtein, E. Brill, and S. T. Dumais, "Improving web search ranking by incorporating user behavior information," in *SIGIR*, pp. 19–26, 2006.

[8] U. Irmak, V. von Brzeski, and R. Kraft, "Contextual ranking of keywords using click data," in *ICDE*, pp. 457–468, 2009.

[9] F. Radlinski and T. Joachims, "Query chains: learning to rank from implicit feedback," in *KDD*, pp. 239–248, 2005.

[10] T. Joachims, L. A. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay, "Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search," *ACM Trans. Inf. Syst.*, vol. 25, no. 2, 2007.

[11] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing top k lists," *SIAM J. Discrete Math.*, vol. 17, no. 1, pp. 134–160, 2003.

[12] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[13] M. Barbaro and T. Z. Jr., "A face is exposed for aol searcher no. 4417749," Augest 9, 2006. (New York Times).

[14] K. Hafner, "Researchers yearn to use aol logs, but they hesitate," Augest 23, 2006. (New York Times).

[15] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma, "Query expansion by mining user logs," *IEEE Trans. Knowl. Data Eng.*, vol. 15, no. 4, pp. 829–839, 2003.

[16] R. Baeza-Yates, C. Hurtado, and M. Mendoza, "Query recommendation using query logs in search engines," in *EDBT*, 2004.

[17] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang, "Clustering user queries of a search engine," in *WWW '01*.

[18] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density based algorithm for discovering clusters in large spatial databases with noise," in *KDD*, pp. 226–231, 1996.

[19] E. Yilmaz, J. A. Aslam, and S. Robertson, "A new rank correlation coefficient for information retrieval," in *SIGIR*, pp. 587–594, 2008.

[20] D. L. Davies and D. W. Bouldin, "A cluster separation

measure," *IEEE Transactions on In Pattern Analysis and Machine Intelligence*, vol. PAMI-1, pp. 224–227, Nov. 1977.
[21] S. Saitta, B. Raphael, and I. F. C. Smith, "A bounded index for cluster validity," in *MLDM*, pp. 174–187, 2007.

BIOGRAPHIES

Shabuddin Sheik completed MCA in 2006 at Y.N.College,Narsapur,affiliated to AU. At present pursuing M.Tech in Swarnandhra College of Engineering & Technology,Seetharampuram,W.G.Dt

B Nandana Kumar completed M.Tech(CSE) at GITAM college,Vizag, affiliated to AU in 2009. 5 years of experience as Asst.Professor.Present working in Swarnandhra College of Engineering& Technology,Seetharampuram,W.G.Dt His Areas of Interest: Data Mining and Computer Graphics