2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology

Search Engine Query Clustering using Top-k Search Results

Yuan Hong, Jaideep Vaidya and Haibing Lu
MSIS Department and CIMIC, Rutgers University
One Washington Park, Newark, NJ 07102, USA
{yhong, jsvaidya, haibing}@cimic.rutgers.edu

***Abstract*—Clustering of search engine queries has attracted significant attention in recent years. Many search engine applications such as query recommendation require query clustering as a pre-requisite to function properly. Indeed, clustering is necessary to unlock the true value of query logs. However, clustering search queries effectively is quite challenging, due to the high diversity and arbitrary input by users. Search queries are usually short and ambiguous in terms of user requirements. Many different queries may refer to a single concept, while a single query may cover many concepts. Existing prevalent clustering methods, such as K-Means or DBSCAN cannot assure good results in such a diverse environment. Agglomerative clustering gives good results but is computationally quite expensive. This paper presents a novel clustering approach based on a key insight – search engine results might themselves be used to identify query similarity. We propose a novel similarity metric for diverse queries based on the ranked URL results returned by a search engine for queries. This is used to develop a very efficient and accurate algorithm for clustering queries. Our experimental results demonstrate more accurate clustering performance, better scalability and robustness of our approach against known baselines.**

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 requirements. Many different queries may refer to a single concept, while a single query may correspond to many concepts. To 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, such as Google, Yahoo!, Bing, and Ask 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 search engine, and thus are quite valuable. From the search engine perspective, effective clustering of search queries is a necessary pre-requisite for these services to function well.

Due to all of these reasons, clustering of search engine queries has attracted significant attention in recent years. However, existing prevalent clustering methods, such as K-Means or DBSCAN cannot assure good results in such a diverse environment. There are several challenges posed by the unique nature of the environment. The primary issue is to determine how to measure similarity between queries. To enable more precise information retrieval, a representative and accurate descriptor is indispensable for computing the similarity between queries.

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 click-through 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 "Honda vs Toyota". Therefore, even if it is clearly relevant to the query "Honda", 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 cluster queries, the accuracy may be impacted.

Therefore, describing queries only by content-based keywords or purely through click-through data is not always accurate for search engine query clustering. In this paper, our main contribution is to propose a novel query descriptor for comparing queries. This is based on a key insight – *search engine results might themselves be used to identify query similarity*; and thus incorporate both content and click-

978-0-7695-4513-4/11 $26.00 © 2011 IEEE
DOI 10.1109/WI-IAT.2011.224

through information [6][7][8][9]. Based on this, we also define a new similarity metric that can be used in any distance based clustering algorithm. Due to the diversity of queries and the curse of dimensionality, current clustering algorithms have high computational cost. We also propose an efficient clustering algorithm to reduce computational cost. We compare the query clustering results of our approach with several existing state of the art methods and show that our algorithm provides good cohesion, separation on clustered queries and significantly reduced runtime.

The remainder of this paper is organized as follows. In Section II, we propose a query similarity measuring approach – using their top-k search results. Then, we present our efficient clustering approach in Section III. Section IV presents the evaluation on clustered queries, runtime scalability and parameter tuning. We review the related literature in Section V. Section VI concludes the paper and discusses future work.

II. Similarity between Top-k Search Results

A. Search Results List

Table I
Query Log and Search Results Data

(a) AOL query log

Query	*Rank* and *Clicked URL*
Honda	1 http://www.honda.com 1 http://powersports.honda.com 2 http://powersports.honda.com 3 http://automobiles.honda.com 3 http://powersports.honda.com 6 http://www.hondapowerequipment.com
Honda accord	6 http://auto.consumerguide.com

(b) Google search results in 2009

Query	*Rank* and *Clicked URL*
Honda	1 http://www.honda.com 2 http://automobiles.honda.com 3 http://powersports.honda.com 4 http://www.hondapowerequipment.com
Honda accord	1 http://automobiles.honda.com/ 2 http://automobiles.honda.com/accord 3 http://www.edmunds.com/honda/accord 4 http://en.wikipedia.org/wiki/Honda_Accord

Since each query q has a list of search results, an associated (possibly time-varying) ranked list of search results, we define these ranked search results as a list of URLs $\mathcal{L}$. We consider only the top-k search results for a query instead of the entire list because the lower ranked URLs may introduce noise due to imprecise information provided by search engines. Thus, we can denote the top-k search results of a query q as below:

Definition 1: (Top-k List $\mathcal{L}(k)$) *For any query q with a list of search results $\mathcal{L}_q$, we define q's top-k search results as $\mathcal{L}_q(k) = \{d_i, \forall i \in [1,k]\}$ where d_i is q's ith URL.*

Table I shows the click-through information and top-k search results of two queries "Honda" and "Honda accord" extracted from the AOL query log and from Google respectively. Clearly, two queries are absolutely dissimilar if we compare their clicked URLs. Alternatively, to measure the similarity or distance between any two queries, we can compare their top-k search results. In general, the top ranked URLs are more relevant to the corresponding search/query (this can be identified by every robust search engine). To incorporate this, we can assign a weight $\omega(i)$ to denote the relevance of the ith URL in the top-k list of a query. This sequence of relevance weights is defined as $\Omega = \{\omega(i), \forall i \in [1,k]\}$.

Observation 1 ($\omega(i)$ is anti-monotonic on i): Given a sequence of relevance weights Ω for the top-k lists of some queries, $\forall i,j \in [1,k], \omega(i) > \omega(j)$ if $i < j$.

This can be observed from the search engines and users' intents: First, the search engine ranking algorithms sort the URLs according to the relevances beforehand. Second, if a URL ranks higher, it attracts significantly more clicks [10]. So we can assume that the weight value $\omega(i)$ decreases sharply as the rank i increases. Thus, we can describe any query q, for the purpose of comparison, using an association of its top-k URLs list $\mathcal{L}_q(k)$ and the assigned sequence of relevance weights Ω.

B. Transition Similarity with Top-k URLs

The key procedure for clustering queries is measuring the similarity or distance between all queries. Since the top-k list along with the weight sequence Ω gives a top-k ranked list, we could use the *Kendall's tau* [11] as the underlying distance metric. However, Kendall's tau is not that effective if the top-k lists have little overlap, which is quite often for search engine query URL lists. Additionally, scalability is an essential concern for search engine query clustering. Computing Kendall's tau is not very efficient in large number of top-k lists' comparisons, which is necessary in clustering queries. To improve this, we propose a new similarity measure for comparing top-k lists. Section III presents an efficient algorithm for comparing two queries' top-k lists which can be well coupled in query clustering.

With top-k lists, the similarity between two different queries q_x and q_y can be measured through their two sorted URL sets $\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$. First, we identify the common URLs in $\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$. Since the maximum number of common URLs is limited to k, we denote these common URLs as $\{\forall d_i \in \mathcal{L}_{q_x}(k) \cap \mathcal{L}_{q_y}(k)$ where $i \in [1,m]$ and $m \leq k\}$. However, simply comparing the two lists through their common URLs such as by using the Jaccard coefficient [3], or Euclidean distance[5] is not sufficiently accurate. The similarity between two queries is determined not only in terms of the number of common URLs, but also according to the ranks of each common URL in two top-k lists. For instance, "Apple" and "Apples" have many common URLs in the search results. If we do not consider the influence resulting from the positions of each URL, the similarity between these two queries might be very large. However, the first few URLs of "Apple" are related to "Apple Inc." and fruit related URLs are ranked not so high, whereas the search results of "Apples" have an inverse order of these two

meanings. Actually, they are two totally different searches. Hence, the ranks of the same URL in both queries' top-k lists impact the precision of the similarity as well. As illustrated in Section II-A, a common URL in $\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$ should have different relevance weight according to their ranks. Hence, our similarity measure should consider two factors: the relevance weight in two top-k lists and the "rank transition" (the rank difference in two queries' top-k lists). We thus define the **Transition Similarity** between two queries q_x and q_y as follows:

$$Sim(q_x, q_y) = \frac{1}{2}\sum_{i=1}^{m}\frac{\omega[r_x(i)] + \omega[r_y(i)]}{|r_x(i) - r_y(i)| + 1} \quad (1)$$

Here, m is the number of common URLs in $\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$, $r_x(i)$ and $r_y(i)$ represent the rank of common URL d_i in $\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$, respectively, and $|r_x(i) - r_y(i)|$ denotes the rank difference of d_i. Since $|r_x(i) - r_y(i)|$ might be 0, and has an anti-monotonic relationship with the similarity, we use the inverse value of $|r_x(i) - r_y(i)| + 1$ to measure the similarity w.r.t. rank difference.

C. Properties of Transition Similarity

We now discuss some properties of transition similarity with respect to given top-k lists.

Proposition 1: For any pair of queries q_x and q_y, we have $0 \leq Sim(q_x, q_y) \leq \sum_{i=1}^{k}\omega(i)$.

Proof: If $\mathcal{L}_{q_x}(k) = \mathcal{L}_{q_y}(k)$, then $m = k$. If each URL also has the same rank in the two lists, then $\forall d_i \in \mathcal{L}_{q_x}(k) \cap \mathcal{L}_{q_y}(k)$, we have $|r_x(i) - r_y(i)| = 0$. Therefore, $\forall d_i \in \mathcal{L}_{q_x}(k) \cap \mathcal{L}_{q_y}(k)$, $\frac{1}{|r_x(i)-r_y(i)|+1} = 1$, and $\max\{Sim(q_x, q_y)\} = \sum_{i=1}^{k}\omega(i)$. If $\mathcal{L}_{q_x}(k) \cap \mathcal{L}_{q_y}(k) = \emptyset$, we have $Sim(q_x, q_y) = 0$.

Furthermore, if Ω satisfies $\sum_{i=1}^{k}\omega(i) \doteq 1$, for instance, $\omega(i) = \frac{1}{2^i}$, we thus have $lim_{k\to\infty}\sum_{i=1}^{k}\omega(i) = 1$. Then, the transition similarity is naturally normalized. ■

As illustrated above, weight sequence $\omega(i) = \frac{1}{2^i}$ can be used to approximately normalize the transition similarity. Otherwise, if $\sum_{i=1}^{k}\omega(i) \neq 1$, we can normalize the transition similarity by normalizing Ω as $\omega'(i) = \frac{\omega(i)}{\sum_{i=1}^{k}\omega(i)}$ where $\sum_{i=1}^{k}\omega(i)$ is the maximum value derived above. Hence, given two top-k lists ($\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$) and the weight sequence Ω, we have the normalized transition similarity:

$$Sim'(q_x, q_y) = \frac{1}{2\sum_{i=1}^{k}\omega(i)}\sum_{i=1}^{m}\frac{\omega[r_x(i)] + \omega[r_y(i)]}{|r_x(i) - r_y(i)| + 1} \quad (2)$$

In the following, we let $Sim(q_x, q_y)$ denote the normalized transition similarity and assume that each $\omega(i)$ is normalized as well.

III. Efficient Similarity Computation

Since computing transition similarity is the essential step for clustering, we first discuss how to efficiently do so, and then present our approach for clustering queries using it.

A. Partial Transition Similarity

Instead of computing the exact transition similarity, in many cases it is sufficient to know whether the similarity is over or under a particular threshold δ. We term this Partial Transition Similarity and now show how this can be efficiently computed.

Since $\omega(i)$ decreases as i increases, the higher ranked URLs essentially contribute a greater portion to the overall similarity. Making use of this, we can develop early terminations to compute whether the similarity between both queries' top-k lists is above or below the threshold. Intuitively, we can terminate the similarity computation if the first few URLs in two top-k lists have already made the accumulated transition similarity $a \geq \delta$. On the other hand, if the transition similarity $Sim(q_x, q_y)$ cannot achieve δ even though all the remaining URLs in $\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$ are identical, we can also terminate the similarity computation. Using this, we can derive *Partial Transition Similarity* based query clustering algorithm with minimum similarity threshold δ for each cluster. Thus we define two concepts – "Similarity Threshold URL" and "Dissimilarity Threshold URL".

Definition 2: (Similarity Threshold URL $\overrightarrow{d}$) *Given a weight sequence Ω for comparing top-k lists and a minimum similarity threshold δ for clustering, the nth URL ($n \leq k$) of q_x (or q_y) is called Similarity Threshold URL $\overrightarrow{d}$ for $Sim(q_x, q_y)$, if this is the first URL at which the accumulated transition similarity satisfies $a \geq \delta$.*

Definition 3: (Dissimilarity Threshold URL $\overleftarrow{d}$) *Given a weight sequence Ω for comparing top-k lists and a minimum similarity threshold δ for clustering, the nth URL ($n \leq k$) of q_x (or q_y) is called Dissimilarity Threshold URL $\overleftarrow{d}$ for $Sim(q_x, q_y)$ if this is the first URL at which the accumulated transition similarity satisfies $a < \delta - \boldsymbol{max}$ where **max** represents the maximum partial transition similarity for all the remaining URLs in $\mathcal{L}_{q_x}(k)$ and $\mathcal{L}_{q_y}(k)$.*

It is clear that only the first few URLs which rank higher than $\overrightarrow{d}$ or $\overleftarrow{d}$ need to be considered to identify whether the similarity of q_x and q_y is over the threshold or not. Since the density of the similarity matrix for web query clustering is typically very low (less than 2%), the similarity between most queries is "0". Frequently, we can discover $\overleftarrow{d}$ or $\overrightarrow{d}$ in the first few URLs of $\mathcal{L}(k)$, and move on to the next pair of queries. Thus, the computational cost is significantly reduced in either case.

B. Efficient Query Similarity Computation

To develop early termination, we can traverse any query's ranked URLs in top-down sequence. It is immaterial which of the two URL sequences is traversed. i.e., as long as all URLs from the first to the last in either top-k list are traversed, the overall transition similarity computed is the same since the individual transition similarity of each *com-*

mon URL in both sequences can be accumulated. Since the accumulated transition similarity a can be simply calculated by summing over the individual transition similarity of each URL in one top-k list, the "Similarity Threshold URLs" $\overrightarrow{d}$ can be easily discovered by terminating the process once $a \geq \delta$. However, the steps for discovering $\overleftarrow{d}$ is more complicated: we must somehow compute the maximum partial transition similarity (**max**) among the remaining URLs in the traversal route, and use it to check the satisfaction condition. Since finding **max** is an optimization problem, the computation is not that efficient if we try to solve the problem at each step. Instead, we can find an upper bound **max'**$\geq$**max** of the partial similarity among all the remaining untraversed URLs. Lemma 1 shows how to derive this.

Lemma 1: At q_x's nth URL (w.o.l.g. traversing q_x's top-k list), the partial transition similarity for untraversed URLs is bounded by $\sum_{i=n+1}^{k} \max\{\omega(i), \frac{\omega(i)+\omega(r_y)}{2(i-r_y+1)}\}$ where r_y is the current highest rank of all untraversed URLs in $\mathcal{L}_{q_y}(k)$.

Proof: Let the ith URL in $\mathcal{L}_{q_x}(k)$ be d_i and $Sim(q_x, q_y)_i$ represent the partial transition similarity from d_i where $i \in [n+1, k]$. We seek **max**, that is the sum of $\forall d_i, Sim(q_x, q_y)_i$, while the positions of d_i can be any untraversed position in $\mathcal{L}_{q_y}(k)$. We thus have **max**$= \max\{\sum_{i=n+1}^{k}\{Sim(q_x, q_y)_i\}\}$. Hence, the partial transition similarity for URLs in untraversed URLs of q_x and q_y is bounded by the objective value of this bipartite matching problem which is known to be $O(V^2 \log(V) + VE)$ where the bipartite graph is represented by $G = \{V = (X, Y), E\}$ [12]. In our case, the number of vertices in $\mathcal{L}_{qx}(k)$ is $k - n$ whereas the number of vertices in $\mathcal{L}_{q_y}(k)$ can be any number in $[k - n, k]$.

However, we can improve the efficiency by computing a slack upper bound max' instead of the exact value max. Let r_y be the current highest rank of all untraversed URLs in $\mathcal{L}_{q_y}(k)$. Since $Sim(q_x, q_y)_i = \frac{\omega(i)+\omega(j)}{2(|i-j|+1)}$ where d_i is ranked at the jth position of $\mathcal{L}_{q_y}(k)$, we have the local maximum partial transition similarity for d_i if $j = r_y$ or $j = i$ (according to Discussion 1 in the Appendix). That is $\max\{\omega(i), \frac{\omega(i)+\omega(r_y)}{2(i-r_y+1)}\}$ where $r_y \leq i$ because at most $i - 1$ positions in $\mathcal{L}_{q_y}(k)$ can be traversed prior to u_i. Therefore, $\forall i \in [n+1, k]$, we have $Sim(q_x, q_y)_i \leq \max\{\omega(i), \frac{\omega(i)+\omega(r_y)}{2(i-r_y+1)}\}$ and **max'**$= \sum_{i=n+1}^{k} \max\{\omega(i), \frac{\omega(i)+\omega(r_y)}{2(i-r_y+1)}\}$. Since **max'** still decreases rapidly in the traversal due to the decreasing weight and a small number of common URLs, we can adopt $\sum_{i=n+1}^{k} \max\{\omega(i), \frac{\omega(i)+\omega(r_y)}{2(i-r_y+1)}\}$ to approximately setup upper bound for untraversed URLs with complexity $O(n)$. ■

Since $\omega(i)$ drops down sharply, the comparison might be terminated in a very early stage. Essentially, we can derive a method to discover dissimilarity threshold URL and similarity threshold URL for efficient query comparison as shown in Algorithm 1.

Algorithm 1 Efficient Query Similarity Computation

Input: two queries q_x and q_y, δ and Ω;
Output: $Sim(q_x, q_y) \geq \delta$ at $\overrightarrow{d}$ or $Sim(q_x, q_y) < \delta$ at $\overleftarrow{d}$

1: $\{d_i$ is the ith URL of q_x where $i \in [1, k]\}$
2: **while** d_i is neither $\overrightarrow{d}$ nor $\overleftarrow{d}$ **do**
3: **if** d_i is the jth URL of q_y where $j \in [1, k]$ **then**
4: $a \leftarrow a + \frac{\omega(i)+\omega(j)}{2(|i-j|+1)}$;
5: **max'**$\leftarrow \sum_{s=i+1}^{k} \max\{\omega(s), \frac{\omega(s)+\omega(r_y)}{2(s-r_y+1)}\}; i++;$
$\{d_i$ is $\overrightarrow{d}$ if $a \geq \delta$, and is $\overleftarrow{d}$ if **max'**$+a < \delta$. r_y is the highest available rank in $\mathcal{L}_{q_y}(k)$.$\}$
6: return $Sim(q_x, q_y) \geq \delta$ or $Sim(q_x, q_y) < \delta$.

For example in Figure 1, we traverse URLs in q_x's top-k list. We can simulate **max'**$= 0.72, 0.47, 0.34, 0.19, \ldots$ as shown in Figure 1. Even though 0.72 or 0.47 cannot be

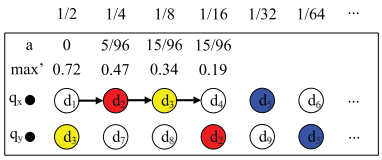


Figure 1. Early Termination ($\delta = 0.35$ and $k = 10$)

achieved, our algorithm can greatly reduce the complexity and assure the accuracy of the clustering results. Then, the $4th$ URL in $\mathcal{L}_{q_x}(k)$ can be identified as the dissimilarity threshold URL $\overleftarrow{d}$ (since $a + max' < \delta$). On the other hand, if we let $\delta = 0.15$ in the example, the similarity threshold URL $\overrightarrow{d}$ is detected at the $3rd$ URL and the similarity computation terminates. Since we traverse one query's top-k list and derive termination conditions in our approach, we can also traverse the URLs across two queries' top-k lists rank by rank. The early termination condition can be similarly derived.

To sum up, we can easily apply our efficient similarity computation to the agglomerative query clustering algorithm with a given similarity threshold δ. This algorithm is thus proposed as our efficient query clustering approach (this partial transition similarity based agglomerative clustering algorithm is simply denoted as partial TS in experiments). Besides this partial transition similarity based query clustering algorithm, our transition similarity can equally well be used as a similarity metric in some traditional query clustering methods such as K-Medoids (without early termination).

IV. Experimental Evaluation

In this section, we compare the performance of all metrics and two different data sets using agglomerative clustering. Specifically, we first compare the query clustering results of all the metrics on top-k lists, and then we compare the clustering results generated by top-k lists versus click through logs. In addition, we evaluate the performance on runtime scalability and parameter tuning.

A. Experimental Setup

Data sets. We make use of two real data sets for query clustering. The AOL query log [13], [14] includes 10 equally partitioned subsets of real data. We pick one partition as our click-through data. Since we should cluster the same queries for comparing the clustering results of top-k lists versus click-through data, we randomly pick $50,000$ distinct queries $\mathcal{Q}$ from the AOL query log and collect the top-10 search results for $\mathcal{Q}$ from Google in 2009. Clearly, all of the search engines can internally query themselves to get the results. The characteristics of the data sets are shown in Table II.

We fix the query set by picking the same subsets of queries from $\mathcal{Q}$ for testing all the metrics (algorithms), and cluster these queries using their click-through data and top-k search results.

Table II
CHARACTERISTICS OF THE DATA SETS

	AOL	AOL ($\mathcal{Q}$)	Google Top-10 List ($\mathcal{Q}$)
# queries (with clicks)	$1,864,860$	$50,000$	$50,000$
# distinct queries (with clicks)	$583,084$	$50,000$	$50,000$
# distinct query and URL pairs	$1,190,491$	$107,761$	$500,000$
# distinct URLs	$373,837$	$56,904$	$215,083$

Parameter and Algorithm Selection. We let $\omega(i) = \frac{1}{2^i}$ and normalize the transition similarity for Ω where i represents the numeric item rank number of each query. We apply Jaccard similarity, Euclidean distance, Cosine distance, Transition similarity and Kendall's tau to Beeferman et al.'s agglomerative clustering algorithm [3] using the same top-k lists data. We also apply Jaccard similarity, Euclidean distance, Cosine distance to the same algorithm [3] using click-through data with the same set of distinct queries as the top-k lists data. In both group of experiments, all the metrics are normalized into $[0, 1]$.

Experimental Variables. We select k from the range $[1, 10]$ for top-k search results in robustness testing. In other testing, all the results are based on top-5 search results. We test the clustering results by varying the similarity threshold δ in $\{0.1, 0.3, 0.5, 0.7\}$ for all metrics.

Experimental Platform. All the experiments are performed on a HP machine with Intel Core 2 Duo CPU E8400 3GHz and 3G RAM running Windows XP Professional.

B. Cohesion and Separation

Clusters can be evaluated by either some internal measures such as cohesion and separation or some external measures such as entropy and purity. Since users' submitted queries are highly diverse, we do not always categorize query clusters. Generally, we evaluate queries clusters using internal measures. The Silhouette Coefficient can be used as a composite measure that judges the quality of cohesion and separation of the clustering. Note that we can evaluate the Silhouette Coefficient using any underlying distance metric such as Euclidean distance, Jaccard distance, or the distance of our transition similarity.

In all cases, the distance/similarity for every metric are related as $||q_x - q_y|| = 1 - Sim(q_x, q_y)$. Now, we can define a as the average distance between queries to their centroid query in corresponding clusters and b as the minimum distance between every pair of centroid queries (the query which holds the greatest global similarity to all other queries in the same cluster is chosen as the centroid query). Given this, the silhouette coefficient is defined as:

$$\widehat{SC} = \frac{b - a}{\max\{a, b\}}. \quad (3)$$

Clearly, larger $\widehat{SC}$ shows better clustering results that queries are separated with large inter-cluster distance and small internal distances in the same clusters.

1) Comparing Metrics on Search Results: Since any pair of queries' top-k lists are two ranked sets of URLs, we can apply either our transition similarity or Kendall's tau or some other distance/similarity measure (such as Jaccard similarity, Euclidean distance and Cosine distance) to compare the metrics applied to search results data. To simplify notation, we denote them as "TS", "KT", "Cos", "JS" and "ED".

As discussed above, for the query clustering validation, without subjective evaluation, the notion of cohesion and separation is the standard objective way of judging the quality of clustering. However, since we have to specify parameter values for all the clustering algorithms, the clustering results cannot always be directly compared even if they have the same parameter. For example, agglomerative clustering algorithm requires a similarity threshold δ. Even if we use the same δ for TS, KT, JS, Cos and ED (for distance metrics, we transform them into similarity), they are defined in different manners so that δ means different levels of minimum similarity threshold for them. Hence, we should test all five clustering results' silhouette coefficient $\widehat{SC}$ for a varying threshold δ. We let $\delta = \{0.1, 0.3, 0.5, 0.7\}$.

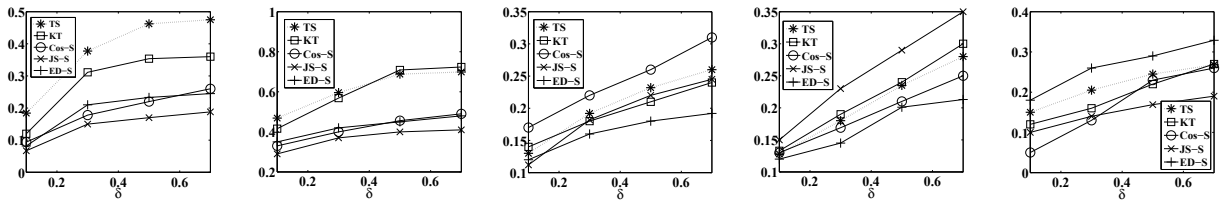
Since we intend to compare the clustering results of five metrics on the same top-k lists data, we first collect the clustering results as follows: *We apply five metrics into the same algorithm (agglomerative clustering), run them on the same search results data with a similarity threshold* $\delta = \{0.1, 0.3, 0.5, 0.7\}$ *and finally collect* 5 *groups of clustering results where each group includes* 4 *different clustering results for the same metric with different* δ.

To evaluate the cohesion and separation of each of these five groups of clustering results, we have to compute the distance between queries in each result. At this time, we again use each of these five metrics to calculate $\widehat{SC}$ based on the five different distances. As shown in Figure 2[1], we have five groups of clustering results' $\widehat{SC}$ w.r.t. five different distances on a range of δ. It is worth noting that we cannot vertically compare $\widehat{SC}$ for each given parameter i.e. 0.1 or 0.3, because the normalized δ is incomparable across metrics. Instead, we compare the entire five curves using five different distances. The larger $\widehat{SC}$ is, the queries are better separated with larger inter-cluster distance and smaller internal distances in the same clusters, we thus summarize the results from Figure 2 (Note: Cos-S, JS-S and ED-S represent the metrics based algorithm running on the top-k search results data in the figure; The dashed line represents TS metric) into a comparison table as below:

As described in Table III, we have following conclusions on cohesion and separation validation of the generated five groups of clustering results as below:

- Since TS and KT are the best or second best metric in most cases, TS and KT are more appropriate in

[1]Experimental data are chosen from two datasets with the same set of 10,000 distinct queries



(a) $\widehat{SC}$ (TS on Top-k) (b) $\widehat{SC}$ (KT on Top-k) (c) $\widehat{SC}$ (Cos on Top-k) (d) $\widehat{SC}$ (JS on Top-k) (e) $\widehat{SC}$ (ED on Top-k)

Figure 2. $\widehat{SC}$ for clustering results of TS, KT, Cos-S, JS-S and ED-S (validated by different distances on Top-k search results, # distinct queries=10^4)

Table III
CLUSTERING RESULTS COMPARISON ON COHESION AND SEPARATION
(CLUSTERING BY SEARCH RESULTS)

Distances for $\widehat{SC}$	Best	Second Best
TS Distance	TS	KT
KT Distance	TS, KT	-
Cos Distance (S)	Cos-S	TS, KT
JS Distance (S)	JS-S	KT, TS
Euclidean Distance (S)	ED-S	TS

comparing queries with their top-k lists.

- In most cases, TS is better than KT when we are comparing queries using top-k search results, except when the distance is measured by JS, KT (with very large δ) and Cos (with very small δ).

2) Search Results versus Click-through Data: Besides comparing all the metrics on search results, we conduct another group of experiment by applying metrics to two kinds of data sets in this section. Since we have concluded that TS and KT perform clearly better than JS, Cos and ED in clustering queries by Top-k search results, we choose the query clustering results of TS and KT to compare with the generated results by applying JS, ED and Cos to click-through data (we cannot directly apply TS and KT to click-through data). Similarly to the experiments in Section IV-B1, we collect another three groups of query clustering results as follows: *We apply JS, Cos and ED to the same agglomerative clustering algorithm, run them on the complete click-through data of the same distinct queries as top-k search results with an identical group of* $\delta = \{0.1, 0.3, 0.5, 0.7\}$ *and finally collect* 3 *groups of clustering results for JS, Cos and ED with different* δ.

Thus, we follow a similar group of cohesion and separation validation as in Section IV-B1. We have another five groups of results (denote them as TS, KT, JS-C, Cos-C, ED-C). We match the five clustering results to two corresponding data sets which are decided by validation distances. i.e. If we measure the distance using KT, we calculate the distance between queries (for all five groups of results) using search results data. On the contrary, if we measure the distance using JS, we calculate the distance between queries (for all five groups of results) using click-through data. We thus plot $\widehat{SC}$ for all validations in Figure 3[1] (Note: Cos-C, JS-C and ED-C represent the metrics based algorithm running on the click-through data in the figure; Dashed lines represent two metrics for clustering queries by Top-k search results). Analogously, we summarize a metric comparison in Table IV. We thus have:

Table IV
CLUSTERING RESULTS COMPARISON ON COHESION AND SEPARATION
(CLUSTERING BY SEARCH RESULTS AND CLICK-THROUGH DATA)

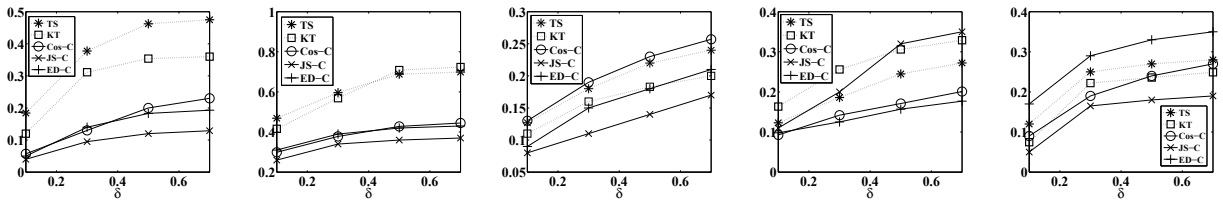
Distances for $\widehat{SC}$	Best	Second Best
TS Distance	TS	KT
KT Distance	TS, KT	-
Cos Distance (C)	Cos-C	TS
JS Distance (C)	KT, JS-C	-
Euclidean Distance (C)	ED-C	TS

- Since TS and KT are the best or second best metric in most cases, top-k lists exhibit better clustering results than click-through data.
- In most cases, TS is better than KT when we are comparing queries using top-k search results, only except when the distance is measured by JS or KT (with large δ).
- Neither JS, Cos nor ED on click-through data is good enough for query clustering simply because they only perform well while evaluating distances using metric by themselves.

C. Scalability and Robustness

1) Scalability: The *"curse of dimensionality"* problem has attracted increasing attention in query clustering. Compared to existing work, our efficient similarity computation significantly reduce the computation cost. Specifically, Figure 4 demonstrates the runtime of our algorithm, compared with agglomerative clustering and K-Medoids using other two metrics on corresponding data sets (KT on top-k lists and ED-C, the required computation costs of Cos-C and JS-C are quite close to ED-C, so we ignore them here). Clearly, the runtime of our algorithm is remarkably low in seconds as against hours.

As the size of testing data increases, agglomerative clustering (simplified as AC) algorithms generally increases in a nonlinear trend as seen in Figure 4. Compared with AC approaches, K-Medoids (KM) clustering performs better on time complexity (but still requires greater runtime than our algorithm) but assuming that "K" is given. In general, this is a difficult problem. In any case, our partial similarity based agglomerative clustering algorithm requires the least



(a) $\widehat{SC}$ (TS on Top-k) (b) $\widehat{SC}$ (KT on Top-k) (c) $\widehat{SC}$ (Cos on QL) (d) $\widehat{SC}$ (JS on QL) (e) $\widehat{SC}$ (ED on QL)

Figure 3. $\widehat{SC}$ for clustering results of TS, KT, Cos-C, JS-C and ED-C (validated by different distances on two data sets, # distinct queries=10^4)

amount of time. Figure 4 demonstrates a nominal increase on runtime as the dataset size increases. In conclusion, our algorithm is efficient and presents great scalability.

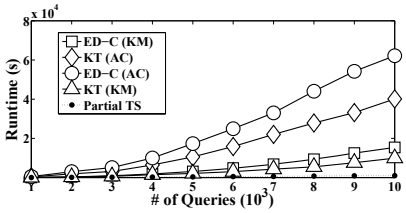


Figure 4. Agglomerative Clustering, K-Medoids (K=100) and Partial TS

2) Parameter Tuning: We now discuss how to select appropriate parameters, k (for top-k URLs sequence $\mathcal{L}(k)$) and the similarity threshold δ. To do this, we check the robustness of the approach to different parameter values.

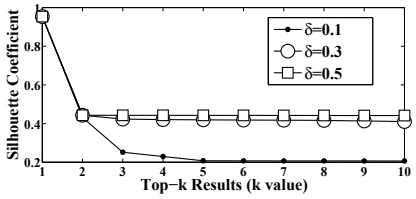


Figure 5. $\widehat{SC}$ on Varying k

Specifically, we plot the silhouette coefficient on varying k and $\delta = \{0.1, 0.3, 0.5\}$. Figure 5 shows that for any given δ, $\widehat{SC}$ reduces as k increases, quickly converging to its lowest value. This can be explained since for small k, the similarity between all the queries in the same cluster are far greater than δ. Note however, that smaller k leads to a smaller fraction of the queries being clustered. Thus, you should choose a small enough k that still leads to a reasonable number of queries being clustered. Similarly, while choosing a higher δ gives better clustering results, it also leads to a smaller fraction of queries being clustered.

Finally, we need to specify a specific weight sequence Ω. As we have discussed, the only constraint is that $\omega(i)$ should decrease as i increases. In fact, for any arbitrary decreasing sequence, the efficient query similarity computation can be implemented giving accurate results in a scalable fashion.

V. Related Work

Query clustering has its roots in keywords-based information retrieval research [2]. Since most of the keywords are ambiguous, analyzing the content of query keywords or phrases by traditional information retrieval techniques has many limitations. Following that, click-through query logs have been mined to yield similar queries [15]. Beeferman and Berger [3] first introduced the agglomerative clustering method to discover similar queries using query logs but with limitations (noise and small number of common clicks). The query clustering approach adopted in [16] uses a K-Means clustering approach. K-Means algorithm cannot adapt well in query clustering case due to the difficulty on specifying k. Wen et al. [17] analyzed both query contents and click-through bipartite graph and applied a density-based algorithm DBSCAN [18] to group similar queries. Similar to agglomerative query clustering, DBSCAN algorithm adopted in [17] requires high computation cost. Meanwhile, Wen et al. linearly combine measures on content-based similarity and cross-references based similarity but it's difficult to set parameters for linear combination of two similarity metrics. However, our ranked search results data (enforced by [6][7][8][9]) naturally consider both factors. Moreover, we have compared the accuracy of our approach with the Euclidean distance based query log clustering [5].

Furthermore, *Kendall's tau* [11] and some relevant measures [19] can be effective in measuring the accuracy of clustered queries. Since our most significant contribution is in observing the fact that search results can be used to perform query clustering, they might be effective metrics as well. However, since search queries clustering is a well known hard problem, there is no efficient algorithm that has been proposed for clustering queries with respect to existing top-k lists comparing measures.

Finally, we employ silhouette coefficient to measure the cohesion and separation of query clustering results. Greater Silhouette coefficient indicates large inter-cluster distances and small intra-cluster distances. Davies-Bouldin index [20][21] seeks the same objective on clustering validation. The distinction between silhouette coefficient is that – minimized index generates the best clustering results.

VI. Conclusion and Future Work

To extract maximum value from search queries, search engines must develop efficient approaches for generating more precise and multi-functional clusters of similar queries. However, most prevalent clustering methods all suffer from

certain limitations on clustering this highly diverse set of queries. In this paper, we have presented a novel query comparing descriptor for measuring the similarity of queries, that incorporates both content and click-through information. This is based on a key insight – search engine results might themselves be used to identify query similarity, and may thus best combine the complementary strategies of content-ignorant and content-aware clustering. With it, we define a new similarity metric that can be used in any distance based clustering algorithm. Taking use of this new metric, an efficient query similarity computation method is developed to reduce runtime for query clustering. The extensive experimental evaluation compares the clustering results of our metric and algorithms with several other existing metrics and approaches, and shows us two facts. First, top-k lists produce query clusters with better cohesion and separation than click-through data. Second, our metric and derived query clustering algorithms provide significantly reduced computational cost and excellent scalability.

There are several directions for extending the work in the future. Instead of implicitly considering the content and click-through data, we can personalize the method to provide recommendations by considering the relations between users. We can also evaluate our approach for specific applications and use accurate weights instead of estimated weight numbers.

VII. Acknowledgements

This work is supported in part by the National Science Foundation under Grant No. CNS-0746943 and the Trustees Research Fellowship Program at Rutgers, the State University of New Jersey.

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 clickthrough 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.

VIII. Appendix

Discussion 1 (Similarity on Rank Difference): For a common URL d_i with a fixed position $r_x(i)$ on q_x's top-k list, we have $Sim(q_x, q_y)_i = \frac{\omega(r_x(i))+\omega(r_y(i))}{2(|r_x(i)-r_y(i)|+1)}$. Without loss of generality, we let $r_x(i) \geq r_y(i)$, $s = r_x(i) - r_y(i)$ and $\omega(r_y(i)) = (1+g(s))\omega(r_x(i))$ where $g(s)$ is a function with the variable $s \in [0, r_x(i)-1]$. Hence, $Sim(q_x, q_y)_i = \omega(r_x(i)) \cdot \frac{1+g(s)}{2(s+1)}$. The derivative $\frac{d[\omega(r_x(i)) \cdot \frac{1+g(s)}{2(s+1)}]}{d_s} = \omega(r_x(i)) \cdot \frac{\frac{d}{d_s} g(s) \cdot 2(s+1) - 2g(s)}{4(s+1)^2}$.

If $\omega(i) = \frac{1}{2^i}$, we have $g(s) = 2^s$ and the derivative is $\omega(r_x(i)) \cdot \frac{\ln 2 \cdot 2^s(s+1) - 2^s}{2(s+1)^2}$. For $s \geq 1$, it is greater than 0, so $Sim(q_x, q_y)_i$ is monotonic on s and $Sim(q_x, q_y)_i$ is maximized as $s = r_x(i) - 1$. There exists $s \in (0, 1)$ such that the derivative is less than 0 while we have $Sim(q_x, q_y)_i = \omega(i)$ for $s = 0$. Hence, $\max\{Sim(q_x, q_y)_i\} = max\{\omega(i), \frac{\omega(i)+\omega(r_y)}{2(i-r_y+1)}\}$ where r_y is the current highest rank of all untraversed URLs in q_y's top-k list.

For other weight sequences, we can also prove $\max\{Sim(q_x, q_y)_i\} = max\{\omega(i), \frac{\omega(i)+\omega(r_y)}{2(i-r_y+1)}\}$ or find an alternative slack upper bound in similar discussion.