Topical Query Decomposition

Francesco Bonchi
bonchi@yahoo-inc.com

Carlos Castillo
chato@yahoo-inc.com

Debora Donato
debora@yahoo-inc.com

Aristides Gionis
gionis@yahoo-inc.com

Yahoo! Research
Barcelona, Spain

ABSTRACT

We introduce the problem of query decomposition, where we are given a query and a document retrieval system, and we want to produce a small set of queries whose union of resulting documents corresponds approximately to that of the original query. Ideally, these queries should represent coherent, conceptually well-separated topics.

We provide an abstract formulation of the query decomposition problem, and we tackle it from two different perspectives. We first show how the problem can be instantiated as a specific variant of a set cover problem, for which we provide an efficient greedy algorithm. Next, we show how the same problem can be seen as a constrained clustering problem, with a very particular kind of constraint, i.e., *clustering with predefined clusters*. We develop a two-phase algorithm based on hierarchical agglomerative clustering followed by dynamic programming. Our experiments, conducted on a set of actual queries in a Web scale search engine, confirm the effectiveness of the proposed solutions.

Categories and Subject Descriptors

H.2.8 [Database Management]: Database Applications - *Data Mining*
H.4.3 [Information Systems Applications]: Communications Applications

General Terms

Algorithms

Keywords

Query recommendation, set cover, clustering.

1. INTRODUCTION

It has been consistently observed over the past years that users typically enter short queries in search engines [15]. One of the many reasons for typing short queries is perhaps the fact that users are looking for information for which they
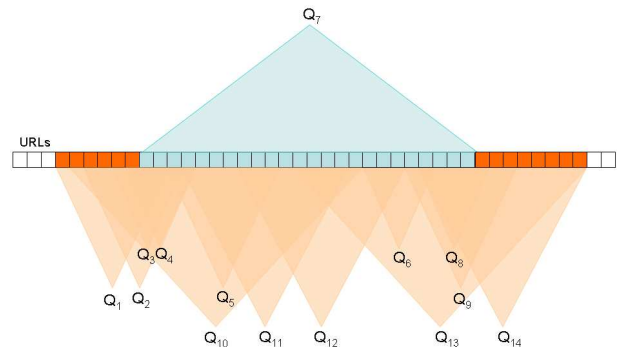
Figure 1: Graphical representation of the query decomposition problem.

do not have sufficient knowledge [5], and thus they may not be able to specify precisely their information need. In order to help the users locate information more effectively, most large-scale Web search engines offer *query recommendations* in response to the queries they receive. These recommendations are typically queries similar to the original one, and they are obtained by analyzing the query logs, for instance, finding recommendations by clustering of queries [25], or by identifying frequent re-phrasings [2].

In this paper we address the problem of assisting the user in the information-seeking task from a novel point of view. Our main intuition is to explore the fact that query logs provide a wealth of queries, which are related to the original query in many different ways. Motivated by this observation we introduce a novel paradigm, namely *topical query decomposition*, where the goal is to assist users in finding the information they are looking for, by providing to them a suitable set of queries as part of the results to their queries. Ideally, these resulting queries should retrieve coherent, conceptually well-separated sets of documents, whose union should cover almost all documents associated to the original query. In other words, we aim at dissecting the different topical groups underlying a query and presenting them to the user.

On a high level, topical query decomposition has similar goals with both query recommendation and clustering of query results. However, it has important differences from both of these tasks. Similar to query recommendation, given a query, our objective is to return other queries. But while the results in query recommendation are a set of queries order by *relatedness* or *frequency*, in the case of topical query decomposition we aim at returning a set of queries that *cover*

Figure 2: A possible solution to the problem illustrated in Figure 1.

the answer set of the original query. In other words, while query recommendation can be seen as a clustering task at the queries level, query decomposition may involve clustering as well, but at the documents level.

Query decomposition is also different from clustering the results of a query and returning to the user the clusters: we do not simply return sets of documents grouped by similarity, but we group the documents in such a way that each group *pre-exists* in the query log, given that it has been previously retrieved by other users using the query that represents that group.

A simple graphical representation of our problem is shown in Figure 1. We consider a query log $\mathcal{L}$, which is a list of pairs $\langle q, D(q) \rangle$, where q is a query and $D(q)$ is its result, i.e., a set of documents that answer query q. We denote with $\mathcal{Q}(q)$ the maximal set of queries p_i, where for each p_i, the set $D(p_i)$ has at least one document in common with the documents returned by q, this is,

$$\mathcal{Q}(q) = \{p_i | \langle p_i, D(p_i) \rangle \in \mathcal{L} \ \wedge \ D(p_i) \cap D(q) \neq \emptyset\}.$$

In the example shown in Figure 1 we have that the issued query is $q_i = q_7$ and $\mathcal{Q}(q_i) = \{q_1, \ldots, q_{14}\}$. The goal is to compute a *cover*, i.e., selecting a subcollection $\mathcal{C} \subseteq \mathcal{Q}(q_i)$ such that it covers almost all of $D(q_i)$. As stated before, the queries in $\mathcal{C}$ should represent coherent, conceptually well-separated set of documents: they should have small overlap, and they should not cover too many documents outside $D(q_i)$. One possible solution to the problem instance in Figure 1 is shown in Figure 2. What is missing in this graphical representation is the topical coherence of each query, i.e., how compact is the set of documents it retrieves in the space of topics.

The problem we study in this paper, topical query decomposition, has many potential applications:

query filtering: it can be applied to an existing query recommendation system (e.g., [25, 9, 4, 2, 28, 12] among others) to filter out recommendations that are topically too close to each other;

query diversification: it can produce a diversified set of recommendations, as some topical group needed to produce a good cover may be not so immediately similar to the given query (w.r.t. the similarity measures used by query recommendation systems) but still relevant for the user;

query-set model: it can be used for selecting terms to represent a document set, following the query-set model [19, 20];

query results presentation: it can be used to present the results of a given query with a different structure, for instance by picking the top document(s) from each representative query in the cover.

These are just few examples in the context of web search applications, but we believe that topical query decomposition may find application in any information-seeking context where the users must be helped in better specifying what they are looking for.

Paper contribution and organization

The main contribution of this paper is the introduction of a novel problem, i.e, topical query decomposition and its general formulation. We also describe two alternative solutions to this problem:

Top-down approach, based on set-covering. Starting from the queries in $Q(q)$, this approach tries to handle our problem as a special instance of the *weighted set covering problem*, where the weight of each query in the cover is given by: its internal topical coherence, the fraction of documents in $D(q)$, the amount of documents it retrieves that are not in $D(q)$, as well as its overlap with other queries in the solution.

Bottom-up approach, based on clustering. Starting from the documents in $D(q)$, this approach tries to build clusters of documents which are compact in the topics space. Since the resulting clusters are not necessarily document sets associated to queries existing in $\mathcal{L}$, a second phase in needed, in which the clusters found in the first phase are "matched" to the sets that correspond to queries in the query log.

Finally we report the empirical analysis performed on a real-world query log from a Web scale search engine, aimed at assessing on the effectiveness of the proposed algorithms for topical query decomposition.

The next section describes previous work related to ours. Section 3 defines the problem which is tackled from the perspective of set cover in Section 4 and of clustering in Section 5. Section 6 describes the empirical assessment of the methods proposed, and the last section presents some concluding remarks.

2. RELATED WORK

In the domain of our application, namely Web search engines, our work builds upon previous research on query recommendation, expansion, and clustering of query results. In the general formulation, we use ideas from set covering and constrained clustering. These are all topics that have attracted considerable attention in the research community. In this section we discuss some of the work developed in each of these topics.

Query recommendation is the task of suggesting to a user a set of queries related to the query he has just issued. Recently, researchers have used data mining techniques applied on the search engine query logs to build solutions for this problem. In [25] queries are clustered, by means of the density-based clustering algorithm DBSCAN [9] on the basis of four different notions of distance: based on keywords or

phrases of the query, based on string matching of keywords, based on common clicked URLs, and based on the distance of the clicked documents in some pre-defined hierarchy. Also the work in [4] proposes a query clustering technique based on common clicked URLs: the query log is represented as a bipartite graph with the vertices on one side representing queries and on the other side URLs. An agglomerative clustering is performed on the graph's vertices to identify related queries and URLs. The algorithm is *content agnostic*, as it makes no use of the actual content of the queries and URLs, but instead it only focuses on co-occurrences in the query log. As stated in [2], the distance measures discussed above have real-world practical limitations when it comes to identifying similar queries, because two related queries may output different URLs in the first places of their answer sets, thus inducing clicks in different URLs (given that the user clicks are affected by the ordering of the URLs [8]). Moreover, as empirically shown e.g., in [15], the average number of pages clicked per answer is very low. To overcome these limitations, the work in [2] clusters queries by representing them as term-weighted vectors obtained by aggregating the term-weighted vectors of their clicked URLs.

A different approach to query clustering for recommendation is in [28], where two different methods are combined. The first method is obtained by modeling search engine users' sequential search behavior, and interpreting this consecutive search behavior as client-side query refinement, that should form the basis for the search engine's own query refinement process. The second method is a traditional content-based similarity method used to compensate for the high sparsity of real query log data, and more specifically, the shortness of most query sessions. The two methods are combined together to form a similarity measures for queries. Association rule mining has also been used to discover related queries in [12]. The query log is viewed as a set of transactions, where each transaction represents a session in which a single user submits a sequence of related queries in a time interval.

Our proposal is completely orthogonal to the body of research discussed above. In fact, given a query we do not recommend similar queries, nor we return the answer set of documents, instead we return a set of queries that cover the answer set of the original query. In our context clustering is used but at the level of documents, not at the level of queries.

Query expansion is another approach adopted by search engines to suggest related queries [26, 13]. The idea here is to reformulate the query such that it gets closer to the term-weight vector space of the documents the user is looking for. Also in this case our approach is different as we look for other queries that are present in the query log, and thus that have been issued by other users, while query expansion methods construct artificial queries.

Clustering of query results is a common technique used to organize the set of query results, usually the top-ranked ones, into clusters. While query suggestion and query expansion have the ambitious goal of providing the users with better formulated queries, the main application of this technique is to facilitate users' browsing through search results. This task is usually associated with the problem of extracting meaningful phrases, i.e., snippets that summarize the contents of each cluster. Various different approaches for clustering search results have been proposed [14, 27, 17, 11]. Our work is substantially different from all these previous approaches since our goal goes beyond a simple reorganization of the pages on the base of their similarity: we aim at dissecting a given query into topical "subqueries" by mining a query log.

Set covering. As we mentioned in the introduction, one of our approaches is based on the set cover problem. The set cover formulations and the adaptations of the greedy algorithm [7] we use are inspired by related variants of the set cover problem in the literature, such as the *red-blue* set cover problem [6, 18], and set cover with minimizing the overlap of sets [16].

Constrained clustering is a relatively new field of research, that is receiving a great deal of attention (see [3, 23] for an overview of the field). However, the kind of constraints usually taken in consideration are instance-based constraints, such as *must-link* (two objects must be placed into the same cluster) and *cannot-link* (two objects must not be placed into the same cluster) [24]. To the best of our knowledge, the problem of clustering by picking clusters from a set of predefined clusters has not been studied before.

3. PROBLEM STATEMENT

In this section we provide an abstract, general, formulation of our problem.

Each instance of the problem consists of a set U of base points, formed by n *blue* points $B = \{b_1, \ldots, b_n\}$, and m *red* points $R = \{r_1, \ldots, r_m\}$, that is, $U = \{b_1, \ldots, b_n, r_1, \ldots, r_m\}$. We write $p \in U$ when we do not want to make the distinction if the point p of U is blue or red.

A collection $\mathcal{S}$ of l sets over U is provided, so that $\mathcal{S} = \{S_1, \ldots, S_k\}$, with $S_i \subseteq U$. For every set $S_i \in \mathcal{S}$, we denote by S_i^B the blue points in S_i, and by S_i^R the red points, that is, $S_i^B = S_i \cap B$ and $S_i^R = S_i \cap R$. As in [6], one part of our goal is to find a subcollection $\mathcal{C} \subseteq \mathcal{S}$ that covers many blue points of U without covering too many red points.

In our application, as will be explained in Section 6, there are *weights* associated with the set of blue points; each blue point $b \in B$ has a weight $w(b)$ that indicates the relative importance of covering point b. Accordingly, we define the weighted cardinality of sets to be the total weight of the blue points they contain: for each set S with blue and red points we define $|S|_w = \sum_{\{b \in S^B\}} w(b)$.

Another characteristic of our problem setting is that we consider a distance function $d(u, v)$, defined for any two points $u, v \in U$. A special case is when $U \subseteq \mathbb{R}^t$, and the distance function d is the Euclidean distance or any other L_p-induced distance.

We use the distance function d in order to define the notion of *scatter* $\mathrm{sc}(S)$ for the sets $S \in \mathcal{S}$. Given a S, we define the scatter of S to be

$$\mathrm{sc}(S) = \min_{u \in S} \sum_{v \in S} d(u, v)^2 .$$

The above definition of scatter corresponds to the notion of *1-mean*. Additionally, one can also define scatter using the notions of *1-median*, *diameter*, *radius*, or others. For our discussion we are also using the concept of *coherence*, which we do not define formally, but informally we refer to it as being the opposite of scatter. A set of high scatter has small coherence, and vice versa.

Our goal is to find a subcollection $\mathcal{C} \subseteq \mathcal{S}$ that covers almost all the blue points of U and has large coherence. More precisely, we want that $\mathcal{C}$ satisfies the following properties:

COVER-BLUE: $\mathcal{C}$ covers *almost* all blue points. The fraction of blue points covered is measured using the weights $w(b)$, defined on the blue points $b \in B$.

NOT-COVER-RED: $\mathcal{C}$ covers *as few* red points as possible.

SMALL-OVERLAP: The sets in $\mathcal{C}$ have small overlap among themselves.

COHERENCE: The sets in $\mathcal{C}$ have small scatter (large coherence).

At an intuitive level, the property of COHERENCE implies both SMALL-OVERLAP and NOT-COVER-RED. To give a geometric argument for the fact that COHERENCE implies SMALL-OVERLAP and NOT-COVER-RED, let us visualize the sets in $\mathcal{S}$ as balls in an euclidean space, and consider that coherence translates to balls of small size, e.g., radius. Then covering a predefined space $\mathcal{X}$ with balls of small radius forces the balls to have small overlap and not to cover much space outside $\mathcal{X}$. However, we explicitly state the properties of SMALL-OVERLAP and NOT-COVER-RED. The reason is that they are intuitive properties for the problems we consider, and second, trying explicitly to satisfy these properties can lead to solutions of better quality.

4. SET-COVER BASED METHOD

From the problem definition given in the previous section, it is clear that there is a mapping of our problem to the set-cover problem. Two of the most well-studied methods for solving many variants of the set-cover problem are the *greedy* approach and *Linear Programming* (LP). In this section, we discuss how to adapt these two general approaches for the particular problems we consider. In our experimental evaluation we have used only the greedy algorithm, however, we also discuss the LP-based algorithm, since we find it to be of theoretical interest.

4.1 Greedy algorithms

There is a simple and intuitive greedy algorithm for the set-cover problem [7], which can be adapted for many variants of it, and which achieves a $O(\log n)$ approximation ratio that matches the hardness of approximation lower bound [10].

The basic greedy algorithm forms the cover solution by adding one element at a time. At the i-th iteration, if not all elements of the base set have been covered, the algorithm maintains a partial solution consisting of $(i-1)$ sets, and it adds an i-th set by selecting the one that is locally optimal at that point. Local optimality is measured as a function of the costs of the candidate sets and the elements that have not been covered so far.

In order to instantiate such general algorithm to the topical query decomposition problem, we must take into account the fact that our set of points consists of blue and red points, that the blue points are weighted, the scatter scores $\text{sc}(S)$ of the sets, as well as the requirements of COVER-BLUE, NOT-COVER-RED, SMALL-OVERLAP, and COHERENCE. Given the above considerations, we reformulate the basic greedy algorithm as shown in Algorithm 1.

The cover parameter α controls the fraction of blue points that the algorithm aims at covering, and is measured in

---

Algorithm 1 Greedy
Input: Base set $U = B \cup R$, weights $w(b)$ of the blue points $b \in B$, set collection $\mathcal{S} = \{S_1, \ldots, S_l\}$, scatter costs $\text{sc}(S_1), \ldots, \text{sc}(S_l)$, cover parameter α
Ouput: A cover $\mathcal{C} \subseteq \mathcal{S}$
1: $V^B \leftarrow \emptyset$
2: $V^R \leftarrow \emptyset$
3: $\mathcal{C} \leftarrow \emptyset$
4: **while** $|V^B \cap B|_w < \alpha|B|_w$ **do**
5:     Select $S \in (\mathcal{S} \setminus \mathcal{C})$ that minimizes $s(S, V^B, V^R)$
6:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{S\}$
7:     $V^B \leftarrow V^B \cup S^B$
8:     $V^R \leftarrow V^R \cup S^R$
9: **end while**
10: Return $\mathcal{C}$

---

terms of the weights of the blue points. The score function $s(S, V^B, V^R)$ is used to evaluate each candidate set S with respect to the elements covered so far by the current solution. For the score function $s(S, V^B, V^R)$, we propose a function that combines three terms:

$$s(S, V^B, V^R) = \frac{\lambda_C \cdot \text{sc}(S) + \lambda_R \cdot |S^R|_w + \lambda_O \cdot |S^B \cap V^B|_w}{|S^B \setminus V^B|_w},$$

where λ_C, λ_R, λ_O are parameters that weight the relative importance of the three terms. Our score function $s(S, V^B, V^R)$ is motivated by the requirements of our problem and from approximation algorithms for the-set cover algorithm.

It is interesting to consider the three cases, where one of the parameters λ is equal to 1 and the other two equal to 0. For $\lambda_C = 1$ the problem corresponds to weighted set-cover, where the weight of each cost is the scatter $\text{sc}(S)$. In this case, the goal is only to minimize the cost of the sets selected in the cover. For $\lambda_R = 1$ the problem becomes finding a cover of the blue points while introducing as few red points as possible. The use of ratio $\frac{|S^R|}{|S^B \setminus V|}$ in this case is motivated by the approximation algorithm of Peleg [18] for the red-blue set cover problem. Finally, when $\lambda_O = 1$ the objective is to minimize the overlap among the sets in the solution. Again, the use of the ratio $\frac{|S \cap V|}{|S \setminus V|}$ is motivated by the approximation algorithm of Johnson [16], who considers set cover with minimizing the intersection of the sets selected in the solution.

Other combinations of the parameters λ_C, λ_R, and λ_O can be used to control the contribution of the three terms in the score function and fine-tune the results according to the application domain. In practice, for specific applications, a good combination of the parameters λ_C, λ_R, λ_O may be learned from training data, if available.

4.2 LP-based algorithms

We start by considering an Integer Programming (IP) formulation of the set cover problem: for each set $S \in \mathcal{S}$ we introduce a 0/1 variable x_S, and the task is to

$$\text{minimize} \quad \sum_{S \in \mathcal{S}} x_S \cdot \text{sc}(S) \quad\quad\quad (1)$$
$$\text{subject to} \quad \sum_{S \ni p} x_S \geq 1, \quad \text{for all } p \in B, \quad (2)$$
$$\text{where} \quad x_S \in \{0, 1\} \quad \text{for all } S \in \mathcal{S}. \quad (3)$$

The above integer program expresses the weighted version of set cover. A solution can be obtained by relaxing the

integrality constraints (3) to (3′): $\{0 \le x_S \le 1\}$, solving the resulting linear program, and then rounding the variables x_S obtained by the fractional solution. The resulting solution is a $O(\log n)$ approximation to the weighted set cover problem, see, e.g., [22].

One way to allow small overlaps among the sets belonging to the solution, is to require that each one of the blue points is covered by only a few sets. Such a constraint can be written as

$$\sum_{S \ni p} x_S \le c, \quad \text{for all } p \in B, \quad (4)$$

for some constant $c \ge 2$, enforcing that each point will be covered by at most c sets.

It can be shown that by solving the linear program $\{(1),(2),(4)\}$ and performing randomized rounding to obtain an integral solution provides again an $O(\log n)$ approximation algorithm, in which the constraint (4) is inflated by a factor of $\log n$, that is, each point in the final solution belongs to at most $c \log n$ sets. The proof, omitted due to space limitations, is an easy adaptation of the basic proof that shows the $O(\log n)$ approximation for the set cover problem via randomized rounding.

We also consider adding constraints to satisfy the NOT-COVER-RED property: for each red point $r \in R$ we introduce a 0/1 variable y_r. We then require that at most d red points are covered by

$$\sum_{r \in R} y_r \le d, \quad (5)$$

ensuring that whenever a set S is selected, the variables y_r for all red point $r \in S^R$ are set to 1, by

$$y_r \ge x_S, \quad \text{for all } r \in S^R. \quad (6)$$

The program $\{(1),(2),(4),(5),(6)\}$ can be either solved directly by an IP-solver, or again, relax the integrality constraints, solve the corresponding LP, and round the fractional solution.

5. CLUSTERING-BASED METHOD

Another approach to the topical query decomposition problem is based on clustering. At a high level of description, our clustering-based method is a two-phase approach. In the first phase, all points in the set B are clustered using a hierarchical agglomerative clustering algorithm. During this clustering phase, the points in B are clustered with respect to the distance function d, while the information about the sets in the collection $\mathcal{S}$, as well as the information about points in R is ignored. At any given level of the hierarchy the induced clustering intuitively satisfies the requirements of our problem statement: the clusters are non-overlapping, they have high coherence, they are covering the points in B, and no points in R. The problem, of course, is that those clusters are not necessarily corresponding to the sets of the collection $\mathcal{S}$. Thus, in the second phase our method attempts to match the clusters of the hierarchy produced by the agglomerative algorithm with the sets of $\mathcal{S}$.

A graphical representation of the two-phase method is shown in Figure 3. Next we describe the algorithm in more detail.

For the hierarchical clustering phase we adopt the method introduced in [29] (and available in the CLUTO toolkit[1]),

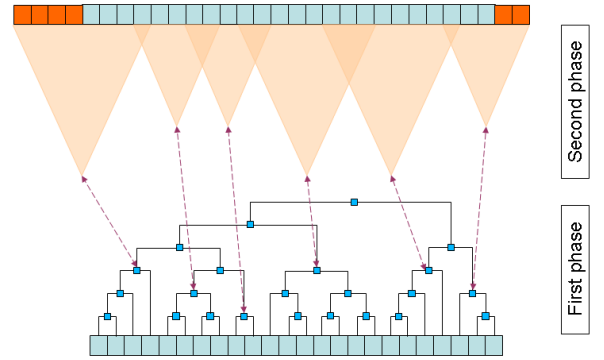[1] http://glaros.dtc.umn.edu/gkhome/views/cluto

Figure 3: Depiction of the clustering-based method.

that has been shown to outperform the traditional agglomerative algorithms when clustering document datasets. In this method, the agglomeration process is biased by a hierarchical divisive clustering solution that is initially computed on the dataset. This is done with the aim of reducing the impact of early-stage errors made by the agglomerative method, thus producing higher quality clustering. The method starts with a divisive clustering until $\sqrt{n}$ clusters are formed, where n is the number of objects to be clustered. Then, it augments the original feature space by adding $\sqrt{n}$ new dimensions, one for each cluster. Each object is then assigned a value to the dimension corresponding to its own cluster, and this value is proportional to the similarity between that object and its cluster-centroid. Now, given this augmented representation, the overall clustering solution is obtained by using the traditional agglomerative paradigm with the UPGMA (Unweighted Pair Group Method with Arithmetic mean) clustering criterion function [21].

Once this method has been performed over the set of points B, it produces a dendrogram $\mathcal{T}$ whose leaves are the points in B and every node $T \in \mathcal{T}$ corresponds to a cluster. Let $T(B)$ be the set of points in B that correspond to the cluster associated with node $T \in \mathcal{T}$, or in other terms, the leaves of the subtree rooted in T. Moreover, we denote by $child\_of(T)$ the list of children of T in $\mathcal{T}$.

The objective of the second phase of our method is to select the sets $\mathcal{C} \subseteq \mathcal{S}$ according to the requirements of our original problem statement — large coverage of B, small coverage of R, small overlap of sets in $\mathcal{C}$, and large coherence. We do this by exploiting the clustering produced in the first phase in to order to facilitate the selection of the sets $\mathcal{C}$. The main idea is to *match* sets of $\mathcal{S}$ into clusters of $\mathcal{T}$. In the following we describe how the matching is actually performed. For sake of simplicity, we first describe how to perform the matching in order to achieve complete coverage of B by means of *dynamic programming*. Then we modify the dynamic programming algorithm to handle the case of partial coverage.

Complete coverage. For each set $S \in \mathcal{S}$ and each node $T \in \mathcal{T}$ we define the *matching score* $m(T, S)$ between S and T to be as follows:

$$m(T, S) = \begin{cases} \text{sc}(S) & \text{if } T^B \subseteq S^B \\ \infty & \text{otherwise,} \end{cases}$$

that is, we match clusters T of $\mathcal{T}$ only to sets S that properly contain the clusters, and the cost is the scatter cost of S. Then given a cluster $T \in \mathcal{T}$, we denote by $m^*(T)$ the score of the best matching set in $\mathcal{S}$ In other words, we define:

$$m^*(T) = \min_{S \in \mathcal{S}} \{m(T, S)\}$$

Now we solve the assignment problem from nodes of $\mathcal{T}$ to sets in $\mathcal{S}$ by dynamic programming on the tree $\mathcal{T}$ in a bottom-up fashion. Let $M(T)$ be the optimal cost of covering the points of T^B with sets in $\mathcal{S}$. We have

$$M(T) = \min\{m^*(T), \sum_{R \in child\_of(T)} M(R)\}.$$

The meaning of the above equation, is that for each cluster T that we consider in a bottom-up fashion in $\mathcal{T}$, we either match T to a new covering set S—the one with the least cost—or we use the solutions obtained for the children of T to make up the covering for T. Among the two options, the one with the least cost is selected.

The motivation of the algorithm, in terms of the requirements of our problem statement, is as follows:

COVER-BLUE By assigning infinite costs to sets that do not contain clusters, any complete cover has lower cost than any partial cover.

NOT-COVER-RED: This requirement is achieved since sets that cover many red points tend to have higher scatter cost.

SMALL-OVERLAP: Again, sets with large overlap tend to contribute more to the scatter cost objective function.

COHERENCE: The objective function of the matching tries to minimize explicitly the total scatter cost.

Partial coverage. In almost all of the problem instances encountered in our dataset, it is not possible to cover all of the original set of blue points B, with the sets in $\mathcal{S}$. Furthermore, even if a complete cover was possible, it might not be the case that the clusters in the hierarchy tree $\mathcal{T}$ are covered by the sets in $\mathcal{S}$. Therefore, we need to adjust the matching algorithm in order to make it work with partial coverage.

The main idea is to relax the constraint that each cluster should be properly contained in the sets of $\mathcal{S}$ by adding a penalization term for the z points that are left uncovered. In particular, we define

$$m(T, S) = \text{sc}(S) + \lambda_U \cdot \left(|T^B \setminus S^B| \right)^2,$$

for all sets $T \in \mathcal{T}$ and $S \in \mathcal{S}$. For the cases of proper containment, $T^B \subseteq S^B$, the above matching score gives $m(T, S) = \text{sc}(S)$, as in the case of complete coverage. However, if $T^B \not\subseteq S^B$, the above score function penalizes gradually for the points of T^B not covered by S^B. Penalizing according to the square of the number of uncovered points was chosen among other choices by subjectively reviewing the results of the algorithm on our dataset. The parameter λ_U weights the relative importance between the two terms, the scatter cost of the sets S and the number of uncovered points. Again, as for the parameters λ of the greedy set cover algorithm, the value of λ_U needs to be selected heuristically, ideally to be learned via training data for a specific application at hand. In our experiments we study the behavior of the algorithm for various measures of interest as a function of the control parameter λ_U.

Given the modified definition of $m(T, S)$, the dynamic programming algorithm for the case of partial coverage is identical to the case of complete cover.

6. APPLICATION TO QUERY DECOMPOSITION

In this section we describe how to apply the methods introduced in this paper to a real query log. We use a sample from an in-house query log $\mathcal{L}$ including 2.9 million distinct queries. A majority of users only looks at the first page of results (as observed by [15] among others), while few users request more result pages. For each query q we record the maximum result page to which any user asking for q in the query log navigated, and consider the set of result documents for the query, which we denote by $D(q)$, We emphasize that in contrast to most of the research on query log mining, we use all the documents that are shown to the users, and not only the ones they click upon.

Overall, we have 24 million distinct documents seen by the users. This means that there is certain overlap between the result sets of different queries; otherwise, given that users see at least 10 documents per query, we would have at least 29 million distinct documents if there were no overlap.

6.1 Candidate queries for the cover

For each query q, we build a set of *candidate queries* for q. The candidate queries $\mathcal{Q}_k(q)$ are the ones that have sufficient overlap with the original query, namely:

$$\mathcal{Q}_k(q) = \{p_i | \langle p_i, D(p_i) \rangle \in \mathcal{L} \wedge |D(p_i) \cap D(q)| \geq k\}.$$

In the following, we set $k = 2$ meaning that each candidate query p_i should have at least 2 documents in common with the original query q.

A first question is whether there are enough candidates in the query log $\mathcal{L}$ for a given query q. In practice, the answer depends basically on the size of $|D(q)|$, as shown in Figure 4. The figure shows the average number of candidate queries in the data, for a sample of 200 queries having a certain number of documents seen by users. We observe that there are about $|D(q)|/2$ candidates for each query returning $|D(q)|$ documents. In practice this is sufficiently large to represent different topical aspects on each query.
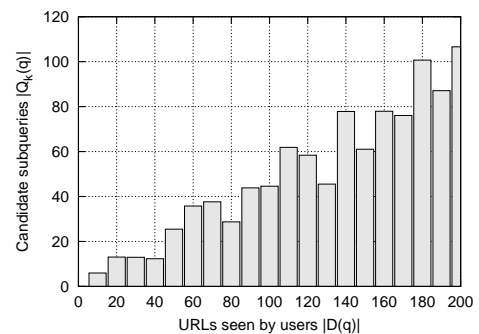

Figure 4: Number of candidates available for queries having different number of documents seen.

We also checked the size of the maximum cover attainable with this set of candidates. According to our observations, this is a fairly stable fraction of about 60%-70% across all queries that have at least 20 documents seen.

6.2 Costs for the candidate queries

Next we compute for each candidate query p_i its scatter $sc(D(p_i))$ as

$$sc(D(p_i)) = \min_{u \in D(p_i)} \sum_{v \in D(p_i)} d(u,v)^2 .$$

For defining the distance between two documents $d(u,v)$ in the result set of a candidate query there are many choices. Given that there is a potentially large set of candidate queries p_i for any query q, each one of them having potentially many documents, and given that we are interested only on an aggregate of the distances, we decided to use a coarse-grained metric. Our choice was to use a text classifier to project each document into a space of topics (100 distinct topics), and then use as $d(\cdot, \cdot)$ the Euclidean distance between the topic vectors.

For the distance between two documents $d(u,v)$ in the result set of the original query q, we use a more fine-grained metric. We remove stopwords, do stemming, and compute tf·idf weights for each term in each document [1]. Using this document representation, we use the standard cosine measure for measuring document similarity during the agglomerative clustering process.

Finally, the weight $w(d)$ of a document $d \in D(q)$ is given by the number of clicks the document has received when presented to the users in response to query q. The distribution of clicks is very skewed [8] and many documents that are seen by the users have no clicks, so we used the following weighting function:

$$w(d) = \log_2\left(1 + clicks(q,d)\right) + 1,$$

where $clicks(q,d)$ is the number of clicks received by document d when shown in the result set of query q.

6.3 Results

Next we picked uniformly at random a set of 100 queries out of the top 10,000 queries submitted by users, and ran the algorithms proposed in this paper over those queries. Given that the greedy algorithms stops when it reaches the maximum coverage possible and queries have different cover sizes, we fixed a cover set size k and evaluated the results of the top-k queries picked by each algorithm, using the following measures:

- Cost at k: sum of costs of the k queries in the cover.
- Red points at k: the number of documents included outside the set $D(q)$ in the solution, as a fraction of the total number of documents outside the set $D(q)$.
- Overlap at k: average number of queries covering each element in the solution.
- Coverage at k: coverage after the top k candidates have been picked.

The average results for the set cover method described in Section 4 are summarized in Table 1 for several parameter settings.

From the results of set-cover shown in Table 1, we observe that penalizing *only the overlap* does not yield good results, and either the scatter of the queries or the red points have to be taken into account.

For the clustering-based method described in Section 5, results are summarized in Table 2. Here, the size of the cover varies with the parameter λ_U. For small values of λ_U, there is not sufficient penalization for partial coverage, and thus

Table 1: Average results for the greedy algorithm at cover size $|\mathcal{C}| = 5$.

Parameters			Sum of	Red	Inter-query	
λ_C	λ_R	λ_O	costs	fraction	overlap	Coverage
0	0	1	0.11	0.15	1.07	0.47
0	1	0	0.06	0.04	1.53	0.48
0	1	1	0.06	0.06	1.11	0.44
1	0	0	0.03	0.06	1.32	0.43
1	0	1	0.04	0.08	1.10	0.40
1	0	10	0.05	0.09	1.09	0.39
1	1	0	0.05	0.04	1.41	0.47
1	1	1	0.05	0.07	1.13	0.44
1	10	0	0.06	0.04	1.51	0.47
1	10	10	0.05	0.06	1.12	0.44
10	0	1	0.04	0.08	1.17	0.42
10	1	0	0.03	0.05	1.33	0.44
10	1	1	0.04	0.07	1.16	0.43
				max.		0.61

Table 2: Average results for the clustering-based algorithm.

Parameter	Size	Sum of	Red	Inter-query	
λ_U	$|\mathcal{C}|$	costs	fraction	overlap	Coverage
2^0	1.00	0.00	0.01	1.00	0.06
2^6	2.15	0.01	0.02	1.13	0.12
2^7	2.78	0.01	0.03	1.21	0.14
2^8	3.56	0.01	0.03	1.25	0.16
2^9	4.52	0.02	0.04	1.31	0.20
2^{10}	5.63	0.02	0.05	1.38	0.23
2^{11}	7.70	0.03	0.07	1.55	0.29
2^{12}	10.11	0.05	0.09	1.68	0.34
2^{13}	14.48	0.08	0.14	1.90	0.43
2^{14}	18.06	0.13	0.18	2.06	0.50
				max.	0.61

the resulting solutions tend to involve only few queries that do not cover well the set $D(q)$. As the value of λ_U increases more sets are selected in the cover solution. We observe that the results of the clustering method are worse than the ones obtained by the set-cover method. If we observe Table 2 for average cover sizes $|\mathcal{C}|$ between 4.52 and 5.63, the coverage reached is about half of the coverage than the set-cover method at 5 obtains in Table 1, at a comparable level of cost for the solution.

Real examples of the queries returned by the algorithms are shown in Table 3. In the table we have included the top 10 queries returned by the algorithms, as well as they original position in the list ordered by overlap with respect to the issued query. The table provides a qualitative idea of the kind of anatomical dissection of a query obtained with our methods.

As a general observation, the results show that the query decomposition algorithms do not simply follow the original ordering by overlap, but pick queries in various positions. Also, extreme settings for the set-cover algorithm that consider only one aspect of cost at a time tend to be reflected qualitatively in the result set. For instance, if the scatter of the queries is given all the cost ($\lambda_C = 1, \lambda_R = 0, \lambda_O = 0$), the queries tend to be more narrow and specific.

Table 3: Example results showing the top 10 queries selected by the algorithms. The numbers on the left are the position of the candidate query in the original list of candidates when sorted by overlap with the given query.

Decomposition of query "border collie"

	Clustering $\lambda_U = 2^{14}$		Set cover $\lambda_C = 0$ $\lambda_R = 0$ $\lambda_O = 1$ (Low-overlap queries)
1	border collie rescue	1	border collie rescue
2	border collie pedigree	2	border collie pedigree
4	breeding border collie	9	border collie allergies
9	border collie allergies	14	border collie puppies
8	border collie rescue uk	12	caring for a border collie
7	border collie to good home	20	border collies skin problems
6	information on blue eyed border collies	16	nice of you to come bye*
5	rescue collies	4	breeding border collie
18	border collies free to good home	26	border collie association
12	caring for a border collie	23	border collie puppies cornwall

	Set cover $\lambda_C = 0$ $\lambda_R = 1$ $\lambda_O = 0$ (Few documents outside original set)		Set cover $\lambda_C = 1$ $\lambda_R = 0$ $\lambda_O = 0$ (More specific queries)
2	border collie pedigree	4	breeding border collie
4	breeding border collie	2	border collie pedigree
9	border collie allergies	1	border collie rescue
1	border collie rescue	6	information on blue eyed border collies
6	information on blue eyed border collies	22	border collie pups for sale
14	border collie puppies	9	border collie allergies
12	caring for a border collie	14	border collie puppies
16	nice of you to come bye	12	caring for a border collie
25	border collie boards	25	border collie boards
8	border collie rescue uk	11	collie

*This is the name of well-known border collie breeders.

Decomposition of query "coffee tables"

	Clustering $\lambda_U = 2^{14}$		Set cover $\lambda_C = 0$ $\lambda_R = 0$ $\lambda_O = 1$ (Low-overlap queries)
3	glass side tables uk	1	oak coffee tables
6	coffee tables modern	2	contemporary coffee table
7	designer coffee tables	3	glass side tables uk
9	coffee tables, large	11	contemporary glass coffee tables
11	contemporary glass coffee tables	29	london furniture coffee table shop
16	glass and wood coffee table	36	large coffee tables
18	glass table	41	reproduction mahogany nest of tables
21	beech coffee tables	26	aquarium tables
23	oak tables	39	handmade garden tables
24	coffee tables, large, wooden	58	chinese coffee dvd

	Set cover $\lambda_C = 0$ $\lambda_R = 1$ $\lambda_O = 0$ (Few documents outside original set)		Set cover $\lambda_C = 1$ $\lambda_R = 0$ $\lambda_O = 0$ (More specific queries)
6	coffee tables modern	6	coffee tables modern
7	designer coffee tables	61	contemporary white dining table uk
2	contemporary coffee table	7	designer coffee tables
12	glass coffee table	41	reproduction mahogany nest of tables
9	coffee tables, large	50	coffee table ergonomic
16	glass and wood coffee table	32	contemporary tv tables
29	london furniture coffee table shop	3	glass side tables uk
21	beech coffee tables	2	contemporary coffee table
4	oak coffee table	11	contemporary glass coffee tables
41	reproduction mahogany nest of tables	16	glass and wood coffee table

7. CONCLUSIONS

We introduced topical query decomposition, a novel problem that stands in between query recommendation and clustering the results of a query, while having similarities and important differences with both. We provided a general formulation and two elegant solutions, namely red-blue metric set cover, and clustering with predefined clusters. The set-cover formulation provides solutions of better quality, and we are currently exploring methods of improving the results of the clustering-based algorithm, by employing other clustering algorithms and/or using queries from query logs over larger periods of time. We are currently investigating the applicability of our proposal in other contexts such as tag-based querying of pictures and multimedia. We also consider performing large-scale experiments involving user studies in order to tune the parameters of the algorithms.

Code and data for reproducing the results shown in Table 3 is available at `http://www.yr-bcn.es/querydecomp/`.

8. REFERENCES

[1] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[2] R. A. Baeza-Yates, C. A. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. In *Proc. of EDBT Workshops*, pp. 588–596, 2004.

[3] S. Basu, I. Davidson, and K. Wagstaff, editors. *Constrained Clustering: Advances in Algorithms, Theory and Applications*. Chapman & Hall/CRC Press, Data Mining and Knowledge Discovery Series, 2008. (In press).

[4] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proc. of ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 2000.

[5] N. J. Belkin. The human element: helping people find what they don't know. *Communications of the ACM*, 43(8), 2000.

[6] R. D. Carr, S. Doddi, G. Konjevod, and M. V. Marathe. On the red-blue set cover problem. In *Proc. of Symposium on Discrete Algorithms*, 2000.

[7] V. Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4:233–235, 1979.

[8] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *Proc. of Int. Conf. on Web Search and Web Data Mining (WSDM)*, 2008.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of ACM Int. Conf. on Knowledge Discovery and Data Mining (KDD)*, 1996.

[10] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

[11] P. Ferragina and A. Gulli. A personalized search engine based on web-snippet hierarchical clustering. In *Proc. of World Wide Web Conf. (WWW)*, 2005.

[12] B. M. Fonseca, P. B. Golgher, E. S. de Moura, B. Pôssas, and N. Ziviani. Discovering search engine related queries using association rules. *Journal of Web Engineering*, 2(4), 2004.

[13] B. M. Fonseca, P. B. Golgher, B. Pôssas, B. A. Ribeiro-Neto, and N. Ziviani. Concept-based interactive query expansion. In *Proc. of ACM Int. Conf. on Information and Knowledge Management (CIKM)*, 2005.

[14] M. A. Hearst and J. O. Pedersen. Reexamining the cluster hypothesis: Scatter/gather on retrieval results. In *Proc. of ACM Int. Conf. on Information Retrieval (SIGIR)*, 1996.

[15] B. J. Jansen and A. Spink. How are we searching the world wide web? a comparison of nine search engine transaction logs. *Information Processing & Management*, 42(1):248–263, January 2006.

[16] D. Johnson. Approximation algorithms for combinatorial problems. In *Proc. of the ACM Symposium on Theory of Computing*, 1973.

[17] K. Kummamuru, R. Lotlikar, S. Roy, K. Singal, and R. Krishnapuram. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *Proc. of World Wide Web Conf. (WWW)*, 2004.

[18] D. Peleg. Approximation algorithms for the label-covermax and red-blue set cover problems. *Journal of Discrete Algorithms*, 5(1):55–64, 2007.

[19] B. Poblete and R. Baeza-Yates. Query-sets: Using implicit feedback and query patterns to organize web documents. In *Proc. of World Wide Web Conf. (WWW)*, 2008.

[20] B. Pôssas, N. Ziviani, W. Meira, and B. Ribeiro-Neto. Set-based model: a new approach for information retrieval. In *Proc. of ACM Int. Conf. on Information Retrieval (SIGIR)*, 2002.

[21] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[22] V. Vazirani. *Approximation Algorithms*. Springer, 2004.

[23] K. Wagstaff, S. Basu, and I. Davidson. When is constrained clustering beneficial, and why? In *Proc. of AAAAI'06*, 2006.

[24] K. Wagstaff, C. Cardie, S. Rogers, and S. Schrödl. Constrained k-means clustering with background knowledge. In *Proc. of Int. Conf. on Machine Learning (ICML)*, 2001.

[25] J.-R. Wen, J.-Y. Nie, H.-J. Zhang, and H.-J. Zhang. Clustering user queries of a search engine. In *Proc. of World Wide Web Conf. (WWW)*, 2001.

[26] J. Xu and W. B. Croft. Improving the effectiveness of information retrieval with local context analysis. *ACM Transact. on Information Systems*, 18(1):79–112, 2000.

[27] H. Zeng, Q. He, Z. Chen, and W. Ma. Learning to cluster web search results. In *Proc. of ACM Int. Conf. on Information Retrieval (SIGIR)*, 2004.

[28] Z. Zhang and O. Nasraoui. Mining search engine query logs for query recommendation. In *Proc. of World Wide Web Conf. (WWW)*, 2006.

[29] Y. Zhao and G. Karypis. Evaluation of hierarchical clustering algorithms for document datasets. In *Proc. of ACM Int. Conf. on Information and Knowledge Management (CIKM)*, pp. 515–524, 2002.