Stochastic Query Covering [*]

Aris Anagnostopoulos[1]
aris@dis.uniroma1.it

Luca Becchetti[1]
becchett@dis.uniroma1.it

Stefano Leonardi[1]
leon@dis.uniroma1.it

Ida Mele[1]
mele@dis.uniroma1.it

Piotr Sankowski[2]
sank@mimuw.edu.pl

[1]Sapienza University of Rome, Italy

[2]Warsaw University, Poland and Sapienza University of Rome, Italy

ABSTRACT

In this paper we introduce the problem of *query covering* as a means to efficiently cache query results. The general idea is to populate the cache with documents that contribute to the result pages of a large number of queries, as opposed to caching the top documents for each query. It turns out that the problem is hard and solving it requires knowledge of the structure of the queries and the results space, as well as knowledge of the input query distribution. We formulate the problem under the framework of stochastic optimization; theoretically it can be seen as a stochastic universal version of set multicover. While the problem is NP-hard to be solved exactly, we show that for any distribution it can be approximated using a simple greedy approach. Our theoretical findings are complemented by experimental activity on real datasets, showing the feasibility and potential interest of query-covering approaches in practice.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*; F.2.m [**Analysis of Algorithms and Problem Complexity**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*.

General Terms

Algorithms, Measurement.

Keywords

Query Covering, Stochastic Analysis, Caching.

*The research leading to these results has received funding from the EU FP7 Project N. 255403 – SNAPS, from the PRIN 2008 research projects COGENT (COmputational and GamE-theoretic aspects of uncoordinated NeTworks) and Mad Web (Models, Algorithms and Data structures for the Web and other behavioural networks) funded by the Italian Ministry of University and Research, and from the EU ERC StD project PAAl no. 259515.

1. INTRODUCTION

In this paper, we study the problem of *query covering*, a strategy for caching query results in a document search engine exploiting the fact that some documents are relevant to a large number of queries. We do not refer to a specific application, but we believe that the problem that we consider might be of potential interest in a number of different applications, some of which are briefly discussed in Section 1.1. We assume a framework in which users submit queries to a document retrieval system over time. The document retrieval system hosts a document collection and each document is either *relevant* or *irrelevant* for a given query. Whenever a query is submitted, the document retrieval system must return a set of at least k relevant documents for the query. To improve response time, the system uses a cache, hosting a collection of documents that are relevant for the queries that users are most likely to submit. When a query is submitted, the system either constructs a result page using a set of k documents in cache that are relevant for the query, or it incurs a *cache miss*, that is, a penalty reflecting the fact that constructing a result page will require a time-expensive operation, such as accessing secondary storage or retrieving contents from a back-end server.

Our goal is to use statistical information about past users' behavior, so as to populate the cache with a set of documents that in the average will maximize the number of user queries that are entirely served in cache over a certain period of interest. In more detail, we assume that time is split into *time periods* (e.g., one day or one week) and statistical information about the ith period is used to select a static collection of documents for the cache to serve queries submitted during the $(i+1)$th time period. Doing this assumes some amount of stationarity of the distribution of user queries over consecutive time periods. As previous experimental evidence suggests and as our results confirm, this is often the case in applications, at least as long as the granularity of time periods is not too coarse. For example, the analysis of Web search engines' query-logs shows [8] that the frequency distribution of queries is with good approximation stable on a daily basis, though important statistical changes may occur at critical points (e.g., day versus night or week versus weekend).

1.1 Motivating Scenarios

The general scenario we consider can be of interest in a number of applications. We briefly discuss four different scenarios below.

Web Search.

In this case, queries are those commonly placed by users, while documents are the snippets and satellite information forming page results returned to users in response to their queries. At present, Web search engines are typically large-scale distributed systems [4,7,11], in which the index is partitioned among multiple processing clusters and local queries are processed over smaller indices. In this way, users are typically served by local search sites that are geographically closer to them rather than a central site. This results in gains in network latency experienced by users and in overall bandwidth savings.

Serving a query entails a number of steps [7, 11]: a front-end machine receives the query and checks the result cache. If the query result (for example, the first page of results) is cached, the query is answered without any processing. Otherwise, the front-end has to submit the query to the local cluster. Each machine fetches from disk the inverted lists corresponding to query terms that are not already cached in memory. Then, documents in the inverted lists satisfying the query are selected and scored using various scoring techniques, and the top-scored documents are returned to the master. Finally, the master sorts all results in decreasing order of scores and returns the top k results to the user.

In our scenario, caching is performed at a snippet level. In particular, a document is a snippet plus its satellite information and the problem we consider becomes that of maintaining in cache a set of documents that maximizes the average number of queries for which we can build the first page of results entirely from information contained in cache.

Content Delivery Networks.

A content delivery network (CDN) [10] is a collection of network elements arranged for more effective delivery of content to end-users. A CDN provides better performance through caching or replicating content over some mirrored servers (often called surrogate servers), strategically placed at various locations in order to deal with the sudden spike in Web-content requests. The users are redirected to the surrogate server that meets a given optimality criterion, typically based on proximity in the network. This approach helps to reduce network impact on the response time of user requests. In the context of CDNs, content refers to any digital data resources and it includes static, dynamic and continuous media data (e.g. audio, video, documents, images, and Web pages).

A CDN can be viewed as a mediator between content providers and users. Content providers place all or part of their contents (e.g., their Web sites) on a CDN, while users' requests to content providers are transparently redirected to the CDN(s) hosting their contents. More generally, CDNs nowadays allow to efficiently publish a variety of Web applications and many important CDN providers exist on the market, such as Akamai[1] or Amazon's CloudFront[2].

A critical aspect in the operation of a CDN is how contents are placed on surrogate servers. Ideally, the contents placed on a surrogate server should be chosen, so as to maximize the amount of requests served locally, without interaction with back-end CDN servers or the origin server of the requested contents. A number of techniques are used to address this issue, with the CDN provider typically pushing contents to surrogate servers based on popularity, so that each surrogate server ideally hosts contents that have appeared to be the most popular among users in the region which it serves.

Although CDNs typically operate by serving user requests for specific contents, they can prove extremely useful in improving performance of search services over large document corpora. One can envision a CDN-based media service, where users can pose queries to the CDN and the CDN returns a collection of objects matching the query. While the top-k objects matching a given query should be fixed, the collection of objects cached locally at a surrogate server might depend on the statistical distribution of queries that have been served by the surrogate server in the recent past.

Computational Advertising.

Yet another potential application is in the area of computational advertising. Typically, when a user performs a Web-search query (in the case of sponsored search) or visits a content page (in the case of content match) the online service provider (OSP), such as Google or Yahoo!, selects a small number of ads to show to the user from a pool of several hundred million ads. Choosing the appropriate ads is a complicated procedure. Typically, there exists an information retrieval system, that given the query (which consists of the search query or the content page, information about the user, etc.) retrieves a number of relevant ads (e.g., 300 ads). Then these ads compete against each other through an auction process to deduce the winners that will be selected for display. In particular, the information retrieval process can be a rather expensive process, so there is need for ad caching [12]—in such a way, the OSP might be able to avoid the information-retrieval step and retrieve the ads to be sent to the auction marketplace directly from a cache. The ideas developed here can allow for efficient use of the system cache. Note that the exact application of the model presented here might not be perfectly suitable, and there are several issues that need to be taken care of (e.g., if an ad is being shown it might not be available in the future, and in that case it should be removed from the cache); nevertheless, the our ideas might be able to suggest ways to improve cache usage. We discuss more later in this and in the conclusions section.

Offline Search.

Finally, another application could be the creation of a service that would allow for offline searching and browsing.[3] Consider, for example, a tourist visiting a new city. She might be interested in various information while in the city, such as accommodation information, transportation information, cultural or sports events, restaurants, history information, practical information, related news, and so on. A useful application would be a service that would allow her to be able to search without connection (either because of unavailability due to lack of infrastructure, or because of high connection cost) and obtain the relevant information. This is exactly the problem we consider here: for the potential queries that the user might have (gathered from historic data) the application must have stored at least a number of

[1]http://www.akamai.com/.
[2]http://aws.amazon.com/cloudfront/.

[3]Such a service was offered by the Webaroo startup (http://webaroo.com) before the company changed its focus.

results; however, since the space is limited, the results must be selected in such a way, so as to cover the user's needs with high probability. Such an application could also use first this approach as a means to search offline and if the information retrieved is not sufficient it could attempt an online connection, incurring additional cost.

1.2 Related Work

We describe some areas related to query caching with respect to Web caching, query-log analysis, query caching, similarity caching and its potential use in content-match advertising, and some theoretical work on set cover and its stochastic variants.

Web Caching.

Caching is a useful technique on the Web: it allows to reduce the overall amount of utilized bandwidth, the workload on back-end servers, and the average latency perceived by users. The cache memory is both used at the client side and server side. For example, Web browsers can cache Web objects instead of retrieving them repeatedly. The cache memory is also exploited at proxy level, so that frequently requested Web objects are stored for subsequent requests [21]. Xie and O'Hallorn [25] analyzed query logs and they observed that many popular queries are shared by different users. This observation justifies the idea of exploiting a server-side caching system: servers can cache pre-computed answers or partial data used in computation of new answers. For more discussion on Web caching, issues such as static and dynamic caching, caching of terms or results, and so on, we refer the reader to [5] and the references therein.

Query Logs.

Early work suggested exploiting the rich information contained in search engines' query logs so as to cache results and boost system performance. Raghavan and Sever [22] exploit user query history in order to build a set of persistent "optimal" queries submitted in the past. This set is called "query base" and it is used to improve the retrieval effectiveness for similar future queries. Query logs are a valuable source of information and the study of their statistical properties has received considerable attention in the past. In many studies, the authors analyzed Query logs and they obtained statistical observations of user behavior. Silverstein et al. [23] analyzed a query log of AltaVista, containing about a billion queries submitted over more than a month. They analyzed users' query sessions, in particular the correlation among terms of the query. Their results show that the 85% of users visit only the first page, while 77% of users' sessions end up after just the first query. These aspects were also analyzed in [16], where the authors perform a thorough analysis of search engine users' behavior. Beitzel et al. [8] analyzed a query log from the AOL search engine, containing queries submitted by a population of ten millions users. They partitioned this query log into groups of queries submitted in different hours of the day and studied the temporal evolution of important aspects, such as popularity and uniqueness of topically categorized queries within the different groups. Further studies emphasized a degree of temporal locality in query logs [13, 19], in particular the fact that the interval of submission of the same query is short in many cases [13]. Finally, query popularity typically follows an inverse power-law distribution (see, for example [14]).

Query Caching.

The above-mentioned findings suggest the use of caching strategies to cache results for queries that are likely to be submitted by users in the next future. This aspect has received considerable attention in the recent past. Standard search engines' query caching strategies are LRU (Least Recently Used) or LFU (Least Frequently Used) based.

Fagni et al. [13] performed a statistical analysis of three large query logs. Based on their findings and noting that a simple, popularity-based caching policy may not address satisfyingly the issue of temporal locality, they proposed a new caching strategy, based on maintaining in cache both a static set of the most popular queries over a recent period of observation and a dynamic set, updated according to an LRU-like strategy, thus reflecting temporal changes in topic interest and so temporal locality. Combined or not with prefetching strategies, the proposed heuristic was shown to outperform standard ones on the datasets considered by the authors. Subsequent work [14] addressed the case in which queries have a weight, for instance reflecting the cost of serving a query, and the goal is maximizing the overall weight of queries served entirely in cache.

Similarity Caching.

Chierichetti et al. [12] studied a related problem in contextual online advertisements associated to Web pages. In this case, upon a user's visit to a Web site, the online service provider, e.g., Yahoo! or Google, must choose the most relevant advertisement to display to the user based on the user characteristics and the contents of the visited page. This process can be seen as a querying procedure, in which the user visiting a given page is the query and serving the query means returning an advertisement that is relevant for the (user, visited page) pair. The authors propose caching strategies for online advertisement information that are based on a notion of similarity between queries, so that a query p is satisfied in cache whenever the system returns an advertisement that was previously used for a query q that is sufficiently similar to query p.

Set Cover and Its Variants.

The notion of relevance determines a relationship between queries and documents, so that a query may be viewed as the set of documents that are relevant to it or, conversely, a document may be viewed as the set of queries to which it is relevant. In this way, the problem considered in this paper turns out to be a stochastic variant of the *set-multicover problem*, itself generalizing the *set-cover problem* [24]. Both problems are NP-hard and it is known that the natural greedy algorithm provides logarithmic approximations for these problems and this bound is tight [24]. This problem has been also studied in the online setting. As in the offline case, here the input is given by a universe of items and a collection of subsets thereof. In this case, items are released over time and the online algorithm must incrementally maintain a collection of sets that covers all items released so far. Azar et al. [2] proved an almost tight poly-logarithmic competitive ratio for this problem, a result that was slightly improved by Buchbinder and Naor [9]. Very often, items are released online according to some stochastic process. Grandoni et al. [15] studied the problem under the assumption that the underlying stochastic problem is stationary and they proposed online strategies that, on average, have performance that is at most a logarithmic factor away from the offline optimum.

1.3 Our Contribution

In this paper we propose and analyze online algorithms for stochastic set multicover and assess their effectiveness in practice as strategies to boost the performance of a document search engine. To this aim, we perform experiments on real data, namely on a large query log from the AOL search engine [20].

Note that we examine the problem of query covering under the simple information retrieval model [6] in which each document is either relevant or irrelevant to a query. While, of course, this model is not appropriate for some of the applications that we mention (e.g, Web search), where ranking is of paramount importance, we believe that the ideas presented here are a first step to show the potential of smart selection of documents to cover several queries; generalizing the algorithm and extending the theoretical and empirical studies is an area for future work.

Since our focus is on the power of query-covering we ignore issues such as cache policies and instead we consider the case of doing prefetching.

Next we describe the theoretical and empirical contributions of our work.

Stochastic Set Multicover.

On the theoretical side, we extend the work of [15] to *stochastic universal set multicover*. In particular, we prove that a heuristic based on the standard greedy strategy for set multicover [24] allows to achieve an expected competitive ratio $O(k \ln mn)$, where m and is the overall number of documents, n is the number of possible queries and k is the coverage requirement for each query. The theoretical analysis in this paper is much more involved than the one in [15] (one reason being that while there exist logarithmic approximation algorithms for the partial set cover problem, which are being used in [15], such an algorithm does not exist for the corresponding partial set multicover problem; in fact, the densest k-subgraph problem, for which finding a polylogarithmic approximation is a famous open problem and whose hardness is an assumption to some recent hardness results [3], can be reduced to the partial set multicover problem even with a coverage requirement of 2 sets).

Experimental Work.

Experimental evidence confirms that our approach allows for serving a large fraction of the queries presented in a time period by the use of a cache, where the cache is static and has been populated by documents based on the distribution observed during the previous time period. Further results show that (i) covering all queries from the previous time period is not necessary; in fact, even covering a smaller fraction (e.g., 40% or 50%) allows to achieve a good coverage of the current time period; (ii) the importance of this result is magnified by another one, showing a superlinear increase in the amount of memory required to cover increasing fractions of queries observed in the previous time period, when the fraction to cover grows above 40–50%. We do not explicitly compare the hit ratios of our approach with those of query caching policies, since the latter refer to a different problem, in which we have a hit if and only if we host in cache the entire result page for the query under consideration.

On the other hand, our approach relies on a coarse classification in which a document is either relevant or irrelevant to a query. So, the question arises, as to the average degree of relevance of the results returned by our heuristics to submitted queries. In particular, we want to assess to which degree, on average, our approach returns the top-k relevant results for a query. To this purpose, we conducted experimental work, in which the queries present in the AOL's query log were submitted to the Yahoo! search engine and the top-k results for each of them were collected. We then compare these with the results returned to the same queries by our heuristics. To this purpose, we used standard measures of accuracy in Information Retrieval, such as precision and recall. Experimental evidence suggests that, while bringing considerable savings in cache size, the results returned by our covering algorithms on average exhibit a decent degree of relevance, although relevance has to be incorporated to the problem formulation to achieve results close to the top-k, as we discuss in Section 6.

1.4 Roadmap

We discuss some preliminaries of our work in Section 2 and we present our model and assumptions in Section 3. We develop the algorithm and our theoretical findings in Section 4, while in Section 5 we present experimental work assessing the potential interest of our approach in practice. Finally, we conclude in Section 6 with some discussion on our findings and with ideas for future work.

2. PRELIMINARIES

We are not focusing on a particular application or platform. Therefore, we present a general description of a search engine that can capture situations such as web text searches, multimedia searching, web advertising, content delivery networks, and so on.

We assume a *corpus* of n *documents* that are indexed appropriately by a *search engine*. Documents can be retrieved by users by submitting *queries*. For each query q there is a number of documents in the corpus that are *relevant* to the query, while the rest are considered *irrelevant*[4]. To simplify the discussion, and without sacrificing generality, we furthermore assume that for each query there are at least k relevant (for some parameter k defined later) documents[5].

Users submit queries to the search engine. We assume that, using historic data, we possess rich statistical information about queries submitted in the past. When a user submits a query the system has to return k relevant documents to the user. The search engine can access a *cache* to retrieve k relevant documents for the query; if the cache contains fewer than k relevant documents to the query, the search engine has to perform a much more expensive operation, such as accessing secondary storage or a remote server, thus incurring a significantly higher response time and/or communication cost, depending on the particular platform.

There are two main cache architectures that a system can use, and often a combination of them is being employed. The first one is to cache queries and the entire set of their

[4]As we explain in Section 3 we do not consider the important case of different amounts of relevance of each document to a query; a document either is relevant or not.

[5]For queries that have fewer than k relevant documents we only consider the subset of queries that have at least k relevant documents in the corpus.

results. Then queries that are in cache can be answered very efficiently from the search engine. A second architecture is more document oriented: a cache is a collection of documents indexed appropriately, and to respond to a query a search engine can access first the cache index and subsequently, if required, the main memory. A drawback of the latter approach with respect to the former is that query cache response times will in general be higher, since result pages must be constructed from documents in cache. On the other hand, this approach presents the advantage that the same document can be used to serve multiple queries, so that the space is used more efficiently. As we mentioned, both architectures can be employed. So, for example, one can use a cache of the first type for the most frequent queries, while a second level of cache can be used if the result page for a query is not found in the first-level cache. Another way the two architectures can be combined is, for example, by using a cache of the first type for various queries that indicate the IDs of the results. Then a cache of the second type can be used to retrieve the contents of frequently used results (e.g., Web page snippets or photos) type can be used.

3. MODEL

Our formulation of the problem captures a few key elements. We make the reasonable assumption that we have full knowledge of the document set, and in particular we know what documents are relevant to what queries. Eventually, we might be interested in taking into account the extent to which a document is relevant to a query. Here, as a first approach in the study of query covering, we consider the simple information retrieval model [6], in which each document is either relevant or irrelevant to a query. Furthermore, we assume no knowledge about queries that are going to be submitted in the future, but we assume that we have a relatively good estimate of their distribution.

We assume a ground set Q of n queries and a set U of m documents. A sequence of t queries are being drawn independently (with replacement) from the Q queries according to a distribution $\mathcal{Q}$. Each query is *covered* by a set of documents, which are the set of documents that are relevant to the query. Conversely, each document $u \in U$ *covers* a set of queries Q_u, which are those to which it is relevant. When a user submits a query, we need to return k results. We want to create a universal map (offline, which will be computed, for example, during the night) from each query to a set of k relevant documents for the query, so that when a query is presented we can fetch the k documents returned from the map to a cache and we can use them to serve the current query as well as (possibly) future ones. We formalize this as follows:

Problem 1 (Query Multicover(t)). *Given a set Q of queries, a distribution $\mathcal{Q}$ on Q, a set of U documents, each covering a subset of the queries, and a sequence length t, compute a mapping ϕ from each query $q \in Q$ to a set $\phi(q) \subset U$ of size k such that for each $u \in \phi(q)$ we have $q \in Q_u$, and our choice minimizes*

$$\mathbf{E}\left[\left|\bigcup_{i=1}^{t} \phi(q_i)\right|\right],$$

where expectation is taken over all the sequences of t queries $q_1, q_2, \ldots, q_t$ drawn independently from $\mathcal{Q}$.

In other words, assuming that t queries are drawn independently from the distribution $\mathcal{Q}$, we need to populate the cache with a set of documents, so that for each sampled query we have at least k relevant documents. The goal is to create a mapping from queries to documents so as to minimize the expected number of documents in cache.

In practice, we have incomplete information for the distribution since we only have access to some samples from it, and this lack of information is especially manifested in rare and one-time queries. In the practical application of our universal stochastic algorithms, we will only be able to select a set of documents to cache that cover at least k times a fair percentage of the queries that will be submitted. In Section 4.2 we delve more into these issues.

Note also that the problem definition has the sequence length t as part of the problem input (so that the minimization problem is well defined); however, we would like to have policies that perform well even when the value of t is not known. Indeed, the algorithm that we present in the next section creates a mapping that provides a small cost (compared with the optimal solution) without any prior knowledge of t, that is, the same mapping is competitive for all values of t.

As we mentioned in the introduction, since our focus is on the power of query-covering we ignore issues such as cache policies and we consider the case of doing prefetching.

4. ALGORITHMS

The problem that we just described is a stochastic generalization of the set-cover problem, in which elements correspond to queries and documents correspond to sets. The stochastic online setting that we consider resembles the one by Grandoni et al. [15], who studied the problem of *stochastic universal set cover*. Observe that differently from the traditional set cover problem we need to define a fixed mapping from elements to covering sets without knowledge of the elements to cover, since we have no a priori knowledge of the queries that will be submitted. We mention here that in a deterministic setting we cannot achieve good results (as is shown in [17]). Instead, we prove that knowing the query distribution suffices to provide algorithms with logarithmic (expected) approximations. In particular, in Section 4.1 we present a simple and efficient greedy algorithm for the Query Multicover(t) problem and we prove that it achieves logarithmic approximation ratio when k is constant. Note that the proof in [15] for the set-cover problem cannot be applied to our setting and that we need more sophisticated arguments to prove that our algorithm achieves a good approximation.

4.1 Stochastic Query Multicover

Consider a sequence of t queries that are sampled from the distribution $\mathcal{Q}$. For a sequence $\omega = (q_1, \ldots, q_t)$, where q_is are iid (independently and identically distributed) samples from $\mathcal{Q}$, let $C^{\text{opt}}(\omega)$ be the optimal offline cost, and let $\bar{C}^{\text{opt}} = \mathbf{E}[C^{\text{opt}}(\omega)]$ be the expected offline cost. Similarly, define $C(\omega)$ and $\bar{C}$ as the cost and the expected cost, respectively, induced by Algorithm `multiCoverGreedy` in Figure 1. With respect to this setting, we are able to prove Theorem 2 below.

```
1.  Function multiCoverGreedy($k$)
2.      /* Initially all queries are uncovered. */
3.      foreach ($q \in Q$)
4.          $results(q) \leftarrow \emptyset$
5.      while (exists $q$ s.t. $|results(q)| < k$)
6.          /* If there exists an uncovered query, find the most
              cost-effective doc */
7.          $doc \leftarrow$ mostCostEffectiveDoc()
8.          foreach ($q \in queries(doc)$)
9.              $results(q) \leftarrow results(q) \cup \{doc\}$
10. end while

1.  Function mostCostEffectiveDoc()
2.      /* In the unweighted case, the most cost-effective doc
          covers most queries */
3.      foreach ($u \in U$)
4.          $costEffectiveness(u) \leftarrow$
              $1/|\{q \in Q : results(q) < k$ and $u \in q\}|$
5.          if ($costEffectiveness(u) < costEffectiveness(doc)$)
6.              $doc \leftarrow u$
7.      return $doc$
```

Figure 1: The greedy multicover algorithm.

THEOREM 2. *For any sequence of t queries the mapping created by Algorithm 1 satisfies*

$$\bar{C} = O(k \ln mn) \cdot \bar{C}^{opt}.$$

Due to space limitations we do not present the proof of the theorem, which will appear in the full version of the paper. The main part of the proof is showing the following lemma, which, as we see in Section 4.2, also provides us with a lot of intuition about the problem structure, and will be useful when applying the algorithm in practice.

LEMMA 3. *The greedy algorithm needs at most $96k\bar{C}^{opt} \ln nk$ documents to cover k times all but $8\frac{n}{t}\bar{C}^{opt} \ln mn$ queries from Q.*

4.2 Discussion and Practical Considerations

In this section we will discuss the knowledge obtained by our theoretical approach and we will leverage it to create a practical algorithm. First, we comment on the assumption that queries are drawn from a distribution. Note that to obtain a strong theoretical result the stochastic assumption is necessary to achieve a small approximation ratio; even for the simple universal set cover problem (where $k = 1$) there is a lower bound of $\Omega(\sqrt{n})$ if no stochastic assumptions for the input distribution are made [17]. On the other hand, assuming that queries are sampled from a distribution is a standard and reasonable assumption (describing the ensemble of queries from all users) and can provide us with much sharper bounds. An issue with the stochastic assumptions is that results are more demanding technically; for example the proof of Grandoni et al. [15] (who examine the stochastic universal set cover, that is, when $k = 1$), which is already technically involved, cannot be applied here and we need further technical work.

In practice, however, we do not know the distribution, instead we observe samples from it. In our experiments we will use the knowledge from a given period (it can be a day, or a given time period of a day) to obtain an estimate of the distribution and apply the algorithm for the next period (the same period next day, or the same period next week, etc.). Another issue is that the distribution changes over time, however for most of the queries, between two consecutive time periods we expect that most of the queries will have similar probability to appear. This is one of the reasons that to learn the distribution we use only the previous time period, although another reasonable option would be to accumulate queries over a longer period—we expect that qualitatively our findings will be the same. In our experimental results we study the overlap between queries in consecutive time periods; as expected, there is overlap between queries with high frequency, while queries in the tail of the distribution essentially do not overlap.

A careful examination of the proof can provide us also with some intuition of the underlying picture. In particular, Lemma 3 gives us information about the problem structure that can allow us to apply the greedy algorithm even in the case that the cache size is limited. At a high level, what the lemma states is that there exists a set of documents that covers a large fraction of the queries k times. Furthermore, the greedy algorithm is able to find it. Thus, even if the cache size is limited, we are able to apply the greedy algorithm and populate the cache with those documents, using prefetching. Note though that the proportion of queries that are not covered according to Lemma 3 depends on the value of the optimal solution (for t requests), $\bar{C}^{\text{opt}}$. If the ratio $\bar{C}^{\text{opt}}/t$ is small this implies that there is a large number of documents that can cover several queries. This is both because of overlap between the relevant documents among various queries, as well as because of the skewness of the query distribution. In practice, both of these situations are true: the result sets of web search queries such as "`hotel rome`," "`cheap hotel rome`," "`rome accommodation`," and so on, are expected to be highly overlapping, and similar is the situation in other settings where we can apply query-covering ideas (or at least where query covering is suitable). Also it is a well documented fact that query frequencies follow power law distributions (see for example [14,18]). Thus we expect the ratio $\bar{C}^{\text{opt}}/t$ to be small enough and thus the greedy algorithm to be able to cover many queries that will appear in the future by inserting documents in the cache. Our experimental findings, reported in Section 5, confirm the intuition obtained from our theoretical analysis.

5. EXPERIMENTAL RESULTS

We applied the greedy heuristic on real web query data, and we compare it with various baselines. Next we describe the dataset that we generated and our findings.

5.1 Dataset

In order to generate query requests we used the AOL query-log dataset [20]. It consists of about 36M query records, amounting to about 20M distinct queries submitted by 650K users over a period of three months (from March to May 2006).

The records are sorted by anonymous user ID and, for each ID, they are ordered by their submission times. Each entry in the data set contains the following information:

- *AnonID*: anonymous user ID
- *Query*: keywords submitted by the user

- *QueryTime*: time at which the search query was submitted
- *ItemRank* and *ClickURL*: they are present only if the user clicked on a search result, and represent the rank and the domain portion of the clicked result; entries are repeated if a user has clicked more than one result

In our experiments, we use the *Query* and the *QueryTime* fields. In particular, we globally sort query records by time and then analyze queries submitted during the first days of March. We observe that an average day contains a bit more than 250K queries, out of which the number of distinct queries is about 200K.

5.2 Description of the Experiments

We used the query logs of some days of the AOL dataset to assess the performance of our algorithms. The idea, as we mentioned in Section 4.2, is to use statistical information about queries submitted during a suitably defined time interval to learn the query distribution for the next time interval. For example, we can use the query log of a given week day to cache documents to serve queries submitted in the next day, or we can use queries submitted on a Saturday morning to serve queries submitted on the Saturday of the next week, the underlying hypothesis being that queries submitted at periodic (and not too distant) intervals will be similar. In our experiments, we used statistics over queries submitted on a day to select documents (URLs) to store in cache, as to serve queries submitted the next day. In particular, we used the first days of March 2006 (from March 1^{st} (Wednesday) to March 4^{th} (Saturday)), so that the query log for March i was used to learn the distribution and populate the cache to serve queries submitted on March $i+1$ ($i = 1, 2, 3$). All results we give are averaged over the three resulting experiments. We also conducted experiments with different choices of time intervals, obtaining similar results. For the sake of space, these will be reported in the full version of the paper. Some statistics about the queries submitted during these days are shown in Figure 2. In the rest of the section we call *training set* the set of queries used in one time interval to learn the distribution and populate the cache and we denote it by Q_{training}. We call *test set* the set of queries submitted in the next time interval, used to assess the performance of our algorithms and we denote this set by Q_{test}.

	1^{st}	2^{nd}	3^{rd}	4^{th}
Num Total Queries	267,887	272,935	246,136	266,687
Num Distinct Queries	197,651	202,926	183,567	198,063

Training	Test	Training	Test	Training	Test
1^{st}	2^{nd}	2^{nd}	3^{rd}	3^{rd}	4^{th}
overlap		**overlap**		**overlap**	
22,091		20,969		20,471	

Figure 2: Number of total and distinct queries during the period of observation. Overlap: number of distinct queries in training set that appear in test set.

Given a query of the training set, we retrieved its result list using the *Yahoo! Boss* interface. We assume that the first 30 results for each query are relevant and can be used to cover it, and that the rest of the documents are irrelevant. In Figure 3 we show some statistics about the documents collected.

Day	Total Docs	Distinct Docs
1^{st}	7,680,287	4,427,403
2^{nd}	7,810,571	4,511,187
3^{rd}	6,984,627	4,075,475
4^{th}	7,578,400	4,417,791
Sum	30,053,885	17,431,856
Average	7,513,471.25	4,357,964
Union of Distinct Docs		14,559,154

Training	Test	Training	Test	Training	Test
1^{st}	2^{nd}	2^{nd}	3^{rd}	3^{rd}	4^{th}
overlap		**overlap**		**overlap**	
8,112,469		7,925,203		7,818,313	

Figure 3: Number, average and sum of total and distinct documents during the period of observation. Overlap: number of distinct documents in training set that appear in test set.

We implemented the main greedy algorithm presented in Section 4 and we compared it with several variants. Variants of the main greedy algorithm are parameterized by two parameters, k and x:

- k: parameter k is the *coverage degree*. Given a coverage degree k, a query q of the training set is considered covered if the algorithm has selected at least k documents in the result list of q. We report for values of $k = 2$, 6, and 10.
- x: the parameter x specifies the percentage of the queries of the training set (i.e., the percentage of the queries that arrived during the first period) that are selected to be covered by inserting documents in the cache. The way in which a fraction x% of the queries is selected defines the variants of the algorithm we consider (see below).

The performance of the greedy approach (in its different versions) is compared against the simple approach that selects the first k results for each query in a set consisting of the x% of most frequent queries.

In more details, we tested the following algorithms (the names in the parentheses are the names used to denote these algorithms in the figures):

- **Greedy** (`greedy`): This is the greedy algorithm described in Section 4 with the modification that we stop adding new documents to the cache when we have covered x% of the queries. For $x = 100\%$, the algorithm is exactly the one described in Section 4, where the cache size is unlimited.
- **Top-k** (`top-`k): This simple approach selects the first k results of the x% most frequent queries in the training set. This algorithm, intuitively, should give an upper bound on the performance of our algorithm, although it is expected to be very costly in terms of space used as it completely ignores the query-document structure.

- **Greedy, most frequent queries** (`freq-queries`): Here we select the $x\%$ most frequent queries in the training set and we select the documents to insert to the cache greedily until we cover each of the queries k times.

- **Greedy, first queries** (`first-queries`): This is the same as the previous one, with the difference that we cover greedily the *first* (earliest) $x\%$ of the queries of the training set.

We evaluate our algorithms on the basis of three performance metrics:

- `recall`: this is the percentage of covered queries in the test set. For a given value of k, a query $q \in Q_{\text{test}}$ is considered covered if there are at least k documents in the cache that are relevant to q (i.e., if at least k out of the first 30 results of query q are in the cache).[6] Note that when computing the average recall, each query is considered multiple times if it appears multiple times in the test set.

- `num_doc`: this is the number of cached documents and it determines the cache size. We are interested in (i) how many documents are cached by each algorithm, (ii) the increase in memory as the coverage degree and the percentage of covered queries in Q_{training} increases, and (iii) the behavior of `recall` as the cache size changes for the different algorithms.

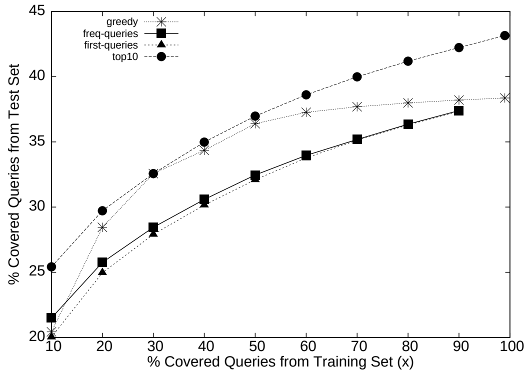
- `precision@n`: as we have mentioned, we consider that each document is either relevant or irrelevant, and the generalization of query covering to models that include ranking is an interesting open problem. Nevertheless, it would be interesting to evaluate the performance of the various algorithms taking the ranking into account. As a measurement we use the precision-at-n ($P@n$). For a given query in the test set, this is the proportion of the top n results of the query that are found in the cache. We then aggregate by averaging over all the covered queries in the test set.

5.3 Results

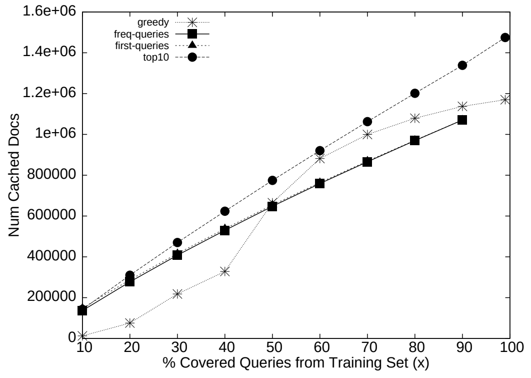
In this section we report our experimental results. We compare the performance of the greedy approach with the other baselines.

Algorithm comparison. First we compared the algorithms considered with respect to the extent to which they cover a given percentage of queries. Results are shown in Figure 4. In Figure 4(a), for each of the algorithms we fix the fraction x of queries that are covered (k times) in the training set, and we measure the number of covered queries in the test set. As expected, `top-`k has the best performance, however `greedy` is following closely. On the other hand, algorithms `freq-queries` and `first-queries` perform worse and, rather surprisingly, their behavior is very similar. This is probably due to the fact that the distribution of the queries is stationary enough, in a small time period, thus the statistics collected as time proceeds resemble those of the entire period, leading to the performance of `first-queries` being similar to that of `freq-queries`.

[6]We also considered a weaker notion of recall, which gives partial credit if fewer than k documents are in the cache; qualitatively the results are the same.



(a) Percentage of covered queries from test set vs. percentage of covered queries from training set (x).



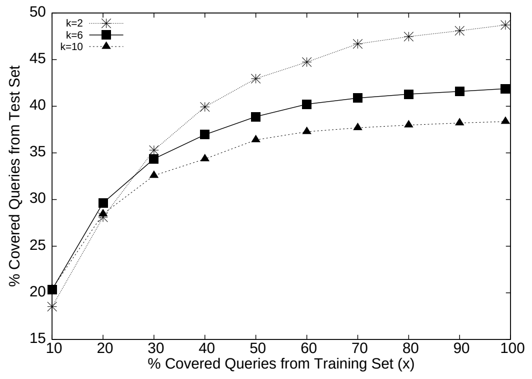
(b) Number of cached documents vs. percentage of covered queries from training set (x).

Figure 4: Comparison of the 4 algorithms for $k = 10$.

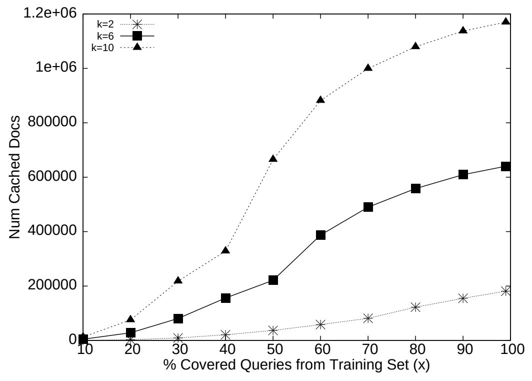
Figure 4(b) summarizes the amount of space required to cover the queries in the training set. Not surprisingly, `top-`k has to pay for its good performance: the space required grows linearly with the number of queries in the training set that are covered. Algorithm `freq-queries` and `first-queries` are also almost identical with respect to space usage. Instead, `greedy` has low space usage until coverage is 40% of the documents, after which we observe a steep increase. This is also in accordance with the theory developed in Section 4 and in particular with Lemma 3 and the discussion in Section 4.2: `greedy` is able to find a small number of sets that cover a good percentage of the queries.

Performance for different coverage degrees. To delve more into the behavior of `greedy` we show in Figure 5 its performance for different values of k. The curves plotted in Figures 5(a) and 5(b) show respectively the `recall` and the `num_doc` as x varies, each curve corresponding to a different value of the coverage degree k.

As we can observe in 5(a) if $x < 30\%$, the curves for $k = 6$ and for $k = 10$ are higher than the curve for $k = 2$. Regarding the dimension of the cache required, `num_doc` is limited fixing x to be less than 30%, while for different values of k we can get an indication of the value of the sweet-spot (the one to which the bound $8\frac{n}{t}\bar{C}^{\text{opt}}\ln mn$ in Lemma 3 refers to), which we cannot compute otherwise due to the hardness of computing the value $\bar{C}^{\text{opt}}$.



(a) Percentage of covered queries from test set vs. percentage of covered queries from training set (x).



(b) Number of cached documents vs. percentage of covered queries from training set (x).

Figure 5: `greedy` and different values of k.

Effectiveness in cache size use. To finish our comparison of the various algorithms, we integrate the two plots in Figure 4, and in Figure 6 we present the recall versus the cache size used (for $k = 10$). The graph makes clear (as expected) the superiority of `greedy` in the use of the available cache, especially for moderate cache sizes.

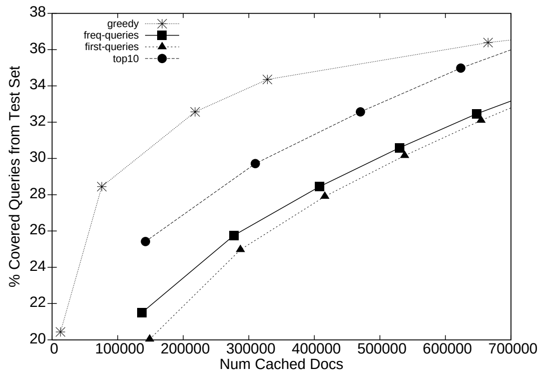


Figure 6: Percentage of covered queries from test set vs. number of cached documents for $k = 10$.

Precision@n. Finally, as we mentioned earlier in Section 5.2, we computed the precision achieved by three of the approaches (the results of `first-queries` are similar to those of `freq-queries` and they are omitted). In Figure 7 we depict the precision at 1, 3, 5 and 10. Of course, `top-`k achieves the highest precision for the same coverage x. This is expected, for example, for queries whose results are in the cache, the precision is 1. But we can also observe that `greedy`, although not optimized for precision, has very close performance in finding the top document, at the same time saving a significant amount of space. On the other hand, it performs quite worse for precision at higher values, indicating the need for explicitely optimizing for document relevance when the application requires it (such as in Web search). `freq-queries` (and `first-queries`) performs slightly worse than `greedy`.

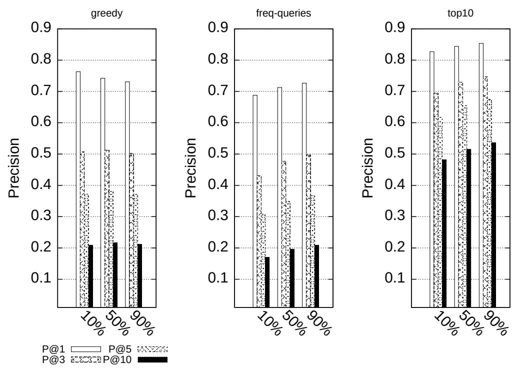


Figure 7: Precision@n with $k = 10$. The x axes denote the percentage of covered queries from the training set (x).

6. CONCLUSIONS AND OPEN PROBLEMS

In this paper we introduced *query covering* as a problem of potential interest to the purpose of effectively and efficiently caching query results. We formulated this problem in the framework of stochastic optimization. While this problem is NP-hard to solve exactly, we proved that a simple greedy approach can provide good expected approximation of the optimal solution. We also provided evidence of the potential practical effectiveness of our approach by extensive simulation on real datasets.

This work represents a first approach to the problem with a number of interesting open questions. The first and obvious one concerns a more general model in which query coverage is performed taking the ranking of the results into account. Our experimental findings indicate that the approach we consider partially addresses this issue. It would be interesting to design and analyze policies that explicitly considered this aspect. While a theoretical result such as the one presented in this paper would be ideal, even empirical heuristics would be very interesting, if they could achieve good performance.

Although we did not address this issue in the present paper, some of our experimental results (in particular, the fact that some heuristics can cover the same fraction of queries using considerably fewer documents) suggest that the techniques we propose or extensions thereof could prove useful to the purpose of diversification [1] of search results. This aspect deserves in our opinion further scrutiny.

Finally, we have assumed that the query distribution is stationary, and this is a reasonable assumption for most of the queries if the time periods are carefully chosen. Nevertheless, very often there are spikes due to sudden events that our approach handles only after the next time period arrives. A more dynamic caching policy that could detect spikes and change the caching and the mapping if necessary is of both theoretical and practical interest.

7. ACKNOWLEDGEMENTS

We would like to thank the authors of [15] for providing us with a lemma (omitted from this version) used for the proof of Theorem 2, and Michele Gigli and Marco Laurenza for providing us a preliminary version of the algorithm's implementation. Flavio Junqueira shared very helpful insights regarding the problem's motivation. Finally, the presentation was improved thanks to the comments of the anonymous reviewers.

8. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 5–14. ACM, 2009.

[2] N. Alon, B. Awerbuch, and Y. Azar. The online set cover problem. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 100–105. ACM, 2003.

[3] B. Applebaum, B. Barak, and A. Wigderson. Public-key cryptography from different assumptions. In *Proceedings of the 42nd ACM symposium on Theory of computing*, STOC '10, pages 171–180, New York, NY, USA, 2010. ACM.

[4] R. Baeza-yates, C. Castillo, F. Junqueira, V. Plachouras, and F. Silvestri. Challenges on distributed web retrieval. In *IEEE 23rd International Conference on Data Engineering*, 2007.

[5] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. Design trade-offs for search engine caching. *ACM Transactions on the Web*, 2(4):20:1–20:28, 2008.

[6] R. A. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[7] L. A. Barroso, J. Dean, and U. Hölzle. Web search for a planet: The google cluster architecture. *IEEE Micro*, 23(2):22–28, 2003.

[8] S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly analysis of a very large topically categorized web query log. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 321–328. ACM, 2004.

[9] N. Buchbinder and J. S. Naor. Online primal-dual algorithms for covering and packing. *Math. Oper. Res.*, 34(2):270–286, 2009.

[10] R. Buyya, M. Pathan, and A. V. (eds.). *Content Delivery Networks*. Springer, Berlin, 2008.

[11] B. B. Cambazoglu, V. Plachouras, and R. Baeza-Yates. Quantifying performance and quality gains in distributed web search engines. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 411–418. ACM, 2009.

[12] F. Chierichetti, R. Kumar, and S. Vassilvitskii. Similarity caching. In *Proceedings of the twenty-eighth ACM Symposium on Principles of database systems*, pages 127–136. ACM, 2009.

[13] T. Fagni, R. Perego, F. Silvestri, and S. Orlando. Boosting the performance of web search engines: Caching and prefetching query results by exploiting historical usage data. *ACM Trans. Inf. Syst.*, 24(1):51–78, 2006.

[14] Q. Gan and T. Suel. Improved techniques for result caching in web search engines. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 431–440. ACM, 2009.

[15] F. Grandoni, A. Gupta, S. Leonardi, P. Miettinen, P. Sankowski, and M. Singh. Set covering with our eyes closed. In *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 347–356. IEEE Computer Society, 2008.

[16] B. Jansen and A. Spink. How are we searching the world wide web? a comparison of nine search engine transaction logs. In *Inf. Proc. and Management*, volume 42, pages 248–263, 2006.

[17] L. Jia, G. Lin, G. Noubir, R. Rajaraman, and R. Sundaram. Universal approximations for TSP, Steiner tree, and set cover. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 386–395. ACM, 2005.

[18] R. Lempel and S. Moran. Predictive caching and prefetching of query results in search engines. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 19–28. ACM, 2003.

[19] E. P. Markatos. On caching search engine query results. *Computer Communications*, 24(2):137–143, 2001.

[20] G. Pass, A. Chowdhury, and C. Torgeson. A picture of search. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, page 1, New York, NY, USA, 2006. ACM.

[21] S. Podlipnig and L. Böszörmenyi. A survey of web cache replacement strategies. *ACM Comput. Surv.*, 35(4):374–398, 2003.

[22] V. V. Raghavan and H. Sever. On the reuse of past optimal queries. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 344–350. ACM, 1995.

[23] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz. Analysis of a very large web search engine query log. In *ACM SIGIR Forum*, pages 6–12, 1999.

[24] V. V. Vazirani. *Approximation algorithms*. Springer-Verlag New York, Inc., 2001.

[25] Y. Xie and D. O'Hallaron. Locality in search engine queries and its implications for caching. In *IEEE Infocom 2002*, pages 1238–1247, 2002.