Document Selection for Tiered Indexing in Commerce Search

Debmalya Panigrahi*
Microsoft Research
Redmond, WA 98052
depan@microsoft.com

Sreenivas Gollapudi
Search Labs, Microsoft Research
Mountain View, CA 94043
sreenig@microsoft.com

ABSTRACT

A search engine aims to return a set of relevant documents in response to a query, while minimizing the response time. This has led to the use of a tiered index, where the search engine maintains a small cache of documents that can serve a large fraction of queries. We give a novel algorithm for the selection of documents in a tiered index for commerce search (i.e. users searching for products on the web) that effectively exploits the superior structural characteristics of commerce search queries. This is in sharp contrast to previous approaches to tiered indexing that were aimed at general web search where queries are typically unstructured. We theoretically analyze our algorithms and give performance guarantees even in worst-case scenarios. We then complement and strengthen our theoretical claims by performing exhaustive experiments on real-world commerce search data, and show that our algorithm outperforms state-of-the-art tiered indexing techniques that were developed for general web search.

1. INTRODUCTION

With the astronomical growth of the internet over the last decade, *search engines* have been playing a pivotal role in guiding users to internet resources that they desire. Over the last few years, *commerce search*, i.e. users searching for products with the intention of purchasing them online, has come to occupy a prominent place in this domain. On being presented a user query, the goal of a commerce search engine is to output a set of relevant products from its online catalog. Since query response time is a key parameter in user experience, a natural approach is to use a *tiered index* where the search engine maintains a smaller index over a subset of documents (product descriptions) that can serve a large fraction of popular queries. The selection of these documents is a challenging task in general web search because of the *diversity* and *lack of structure* in user queries. However, in commerce search, queries typically comprise a list of desirable product features. Therefore, the query space, while being extremely large, is much more structured than in general web search. In this correspondence, we show that we can exploit this additional structure to design efficient algorithms for selecting the set of documents in a tiered index for commerce search with significantly better performance than the state-of-the-art in general web search. While we present our results in the context of commerce search, our techniques automatically extend to other structured search domains (e.g. travel, music, movies, etc.) as well.

As mentioned above, commerce search queries are characterized by a set of product features (we call these *attribute values* or *keywords*) that the user desires in the product she is searching for. This makes it feasible to categorize commerce search queries based on these features, and to decompose them into their constituent attribute values. For instance, it is relatively easy to identify that a query `Canon EOS black 12MP` was issued by a user searching for cameras, and that her desired attribute values are `Canon`, `EOS`, `black`, and `12MP` for attributes *manufacturer*, *model line*, *color*, and *resolution* respectively. Given that such categorizers exist (see e.g. [23]), the central question that we seek to address in this paper is the following: *can we use the structure in commerce search queries to select documents for a tiered search index aimed at faster query response?*

In response to a user query, a commerce search engine displays a set of relevant products from its online catalog. So, we need to understand how a search engine evaluates the relevance of a user query to a product. Since commerce search queries are a collection of attribute values, this boils down to finding the relevance of a document to an attribute value. For example, a `Nikon D7000` camera is likely to be more relevant to the keyword `Nikon` than to `Canon`. This is a rather straightforward assertion, but in general we might have more ambiguous scenarios: e.g. is a `blue` or a `red` camera more relevant to a user who is searching for `purple` cameras? Recently, we suggested a completely automated technique for inferring these *relevances* (or *similarities*) based on previous user behavior represented by browse trail data collected from toolbars of web browsers [19]. In this paper, we use the algorithm in [19] to obtain, for every keyword a, a list of relevance scores of a with every other keyword for the same attribute (e.g. between `Canon` and every other manufacturer). (We may note that most of these scores are 0 allowing for succinct representation.) For a product in the catalog (say a `Nikon D7000` camera), its relevance score for a user-specified attribute value (say `Canon`) is the relevance score of its corresponding attribute value (i.e. the similarity score of `Nikon` with `Canon`). We note that such a model of similarity between attribute values allows us to extend our techniques presented in this paper to indexes where the data is probabilistic in nature and the matching score of a record to the user query is some function of these probabilities.

*Part of this work was done as a graduate student supported, in part, by NSF contract CCF-1117381 at the Massachusetts Institute of Technology, Cambridge, MA 02139.

WSDM '13 Rome, Italy

The next step is to infer the relevance score of a product for a query q from its relevance scores for the constituent attribute values of q. A natural strategy is to say that a product is relevant to a query if it has a high relevance score for most attribute values in the query. There are multiple functions implementing this strategy, and we discuss some of them later in the paper. Once we have decided on such a function, the problem of constructing a tiered index of size b boils down to finding the set of b products that maximizes the weighted (by query frequency) fraction of queries that have at least k relevant products in the index, k being the number of search results returned to the user. Note that these are precisely the set of queries that can be exclusively served from the cache, and therefore represent *cache hits*. We formalize this problem later, and call it the INDEX SELECTIONproblem.

Caching posting lists v/s query results. As mentioned earlier, algorithms for selecting documents in a tiered index have been proposed in the literature for general web search. Two prominent strategies are: (1) include posting lists, i.e., list of documents, for relevant popular search terms (e.g. [3]), or (2) include search results for popular queries (e.g. [1]). Both strategies suffer from serious deficiencies. While the former strategy generalizes better since it does not restrict itself to only optimizing over the training set of queries, it is unable to differentiate between documents in the same posting list. For example, the posting list for `Canon` would typically contain documents that are relevant to many manufacturers as well as those that are specific to `Canon`, and the algorithm fails to preferentially select documents of the first type over those of the second type. In addition, the strategy of copying entire posting lists leads to wastage of valuable cache space since posting lists for related but distinct keywords tend to have substantial overlap. On the other hand, selecting the contents of the tiered index based solely on a training set of queries leads to poor generalization characteristics since the query space is enormous and typical query distributions have large support. Further, the algorithm in [1] uses an exhaustive search over the entire set of training queries, which hampers efficiency and scalability of the technique. In this paper, we present a novel solution that exploits the structure of commerce search queries to overcome the deficiencies of both these techniques.

Our solution. Before describing our solution, let us first understand the structure of the query space for commerce search. As we stated earlier, a query comprises a set of attribute values. For example, a user searching for digital cameras can opt for one of several manufacturers/model lines, various different resolutions, multiple colors, price ranges, lens types, and so on, and these choices are largely independent of each other. Note that typical search queries also contain "free text", i.e. textual description that does not correspond to any particular attribute. We strip queries of such free text at the very outset, and consider queries to be composed only of the identifiable attribute values in them. Even though the number of attribute values for each such attribute is not very large (say around 20), and there are a manageable number of different attributes (there are at most 10 important attributes for most categories of products), the total number of possible queries generated by combining these attribute values is enormous. Our solution bypasses the obvious computational challenges of this large query space by using its structural characteristics to avoid exhaustive searches.

At a high level, our solution borrows (from [1]) the idea of greedily selecting documents one at a time based on the incremental query coverage they offer. However, we want to ensure that the solution generalizes well to an arbitrary query test set. To this end, we derive marginal frequencies on individual attribute values from the training data and use the structure of the query space to estimate the overall query distribution from these marginal frequencies. This would naturally suggest a greedy algorithm on the entire query distribution (rather than the one observed in the training data). However, a naïve implementation of this idea would lead to an exhaustive search over the support of the query distribution. Our main technical contribution is an efficient implementation of this greedy strategy using algorithmic sampling techniques to reduce the sample space to a sequence of marginal distributions (rather than the overall query distribution) in order to obtain estimates on the incremental query coverage of individual documents.

It is worth mentioning that our algorithms apply not only to commerce search but to any other structured search domain where queries predominantly consist of a set of attribute values for which relevance scores as described above can be computed. However, our techniques do not naturally extend to general web search. In particular, our sampling technique is based on the query space being a Cartesian product of sets of attribute values for individual attributes. Unfortunately, the query space for general web search is substantially less structured and cannot be interpreted as the Cartesian product of a small number of small sets. We leave the extension of our ideas to general web search as an interesting and independent open question.

An additional algorithmic complication arises from the *dynamic* nature of the set of documents. To ensure that the tiered index reflects changes in the catalog, one option is to discard the contents of the index periodically and re-run the algorithm to generate a new set of documents. However, this is a rather expensive solution since it involves repeated re-runs of the algorithm on the entire set of documents. We propose simple algorithmic solutions that overcome this inefficiency.

To evaluate the performance of our algorithm in practice, we perform extensive experiments on the product catalog used by a commercial online shopping portal. A query is said to have a *cache hit* if at least k of the products returned by the commercial search engine on being issued the query are indeed in the index. We vary the parameters of the algorithm such as the size of the index, the relevance threshold, the value of k, etc. and observe the resulting changes in the cache hit ratio. For these parameter ranges, we compare our performance with those of standard algorithms in the literature for tiered indexing in general web search [1, 3].

Our Contributions. The following is a brief sketch of our main contributions in this paper:

- We formalize the problem of selecting products in a tiered index for commerce search (or any other structured search domain). We call this the INDEX SELECTIONproblem.
- We propose a randomized algorithm for the INDEX SELECTIONproblem, and show that it achieves a near-optimal approximation ratio (subject to standard assumptions in complexity theory).
- To address the practical consideration of changing catalogs, we give a natural extension of our algorithm that adjusts the contents of the index in response to the arrival of new products, without having to compute the entire contents of the index from scratch.
- We complement our theoretical results by performing extensive experiments on real-world data. Our experiments confirm that our algorithm is scalable, accurate, and efficient, and outperforms the state-of-the-art techniques in tiered indexing for general web search.

2. PROBLEM DEFINITION

Recall that commerce search queries can be classified into categories (such as *digital cameras*, *cellphones*, etc.), where each category is characterized by a set of attributes (e.g. *resolution* is an attribute for *digital cameras*). Each attribute, in turn, has a set of attribute values (e.g. `Canon`, `Nikon`, etc. for attribute *manufacturer* in the category *digital cameras*). Formally, suppose there are L categories, where category j has k_j attributes. Let $A_1^{(j)}$, $A_2^{(j)}$, $\ldots$, $A_{k_j}^{(j)}$ denote the sets of attribute values for these attributes. Further, for every attribute value a, we estimate its relative frequency in the set of queries, and denote it by f_a. On the other hand, let D be the set of documents,[1] and let w_{ad} denote the relevance score of attribute value a for document $d \in D$.

Each query comprises attribute values for a subset of attributes in its category (recall that we assume that we have stripped the query of its free text). For example, a query `Canon EOS 12MP` has attribute values `Canon`, `EOS`, and `12MP` for attributes *manufacturer*, *model line*, and *resolution* respectively but is missing attributes such as *color*. For ease of notation, we introduce the special element ϕ in the set of attribute values $A_i^{(j)}$ for every attribute i in every category j, and denote the augmented set of attribute values by $A_i^{(j,\phi)} = A_i^{(j)} \cup \{\phi\}$. A query missing attribute i is now assumed to have the special attribute value ϕ for attribute i, and the relative frequency of ϕ for attribute i is the fraction of queries that are in category j and are missing attribute i. The set of all queries in category j is then given by

$$Q_j = A_1^{(j,\phi)} \times A_2^{(j,\phi)} \times \ldots \times A_{k_j}^{(j,\phi)}.$$

In any such query $q \in Q_j$, the attribute value for attribute i is denoted by $q(i)$ (note that $q(i)$ could be ϕ). The relative frequency of a query q in category j is defined as $f_q = \prod_{i=1}^{k_j} f_{q(i)}$.

We are implicitly assuming independence over the set of attributes for any category. While this is accurate for attributes such as color, manufacturer, product specifications, etc. that have small correlation, it does not hold for highly correlated attribute combinations such as manufacturer and model line. To overcome this problem, following [19], we treat correlated subsets of attributes as single attributes, and leave the problem of obtaining a more refined treatment of attribute correlation as future work.

Relevant documents for a Query. As mentioned in the introduction, there are multiple options for defining the set of relevant documents for a query, based on the relevance scores of the documents for individual keywords in the query. Perhaps the simplest option is to define relevance of a document for individual keywords by setting a threshold on the relevance score. The document is now defined to be relevant for a query as a whole if it is relevant for *every* keyword in the query. However, this suffers from the shortcoming that even if a document is relevant for all but one keyword in a query, it is deemed to be irrelevant for the query as a whole. A natural relaxation is to call a document relevant if it is relevant to most (e.g. at least some fixed fraction) of the keywords in the query. Noting that this strategy does not differentiate between two documents that meet the relevance threshold for an attribute value but have very different relevance scores, we finally converge on the following definition of relevance: *A document is said to be* relevant *to a query if the average of the relevance scores of the document over all the keywords in the query is at least some fixed threshold* θ. Both the previous definitions of relevance can be simulated using the new definition, and therefore our algorithm can also handle either of the previous definitions.

[1]For commerce search, a document is a product.

$X \leftarrow \emptyset$
for i from 1 to b **do**
  $d^* \leftarrow \arg\max_{d \in D \setminus X}$ COUNT QUERY$(X \cup \{d\})$
  $X \leftarrow X \cup \{d^*\}$
end for
return X

Figure 1: The GREEDY SELECTION algorithm

If a document d is relevant for a query q, then we say that q is *covered* by d. Similarly, if at least one document in a set of documents X covers a query q, then we say that q is covered by X. Let $Q(d)$ denote the set of queries that are covered by document d, and let $Q(X)$ be the set of queries that are covered by a set of documents X. Corresponding, let $f_d = \sum_{q \in Q(d)} f_q$ and $f_X = \sum_{q \in Q(X)} f_q$.

Choice of index size.[2] A key input is the size of the index denoted by b. The choice of the relevance threshold θ is dictated by the size of the index (e.g. a large value of θ and a small index size might lead to most of the index remaining unused). In turn, the size of the index is a function of external parameters such as the desired cache hit ratio and the amount of memory and indexing infrastructure available to the search engine.

Objective Function. For simplicity, we set the number of results returned by the algorithm k to 1. (We will consider general values of k later.) Then, the objective of the algorithm is to output a set X of at most b documents that maximizes the sum of relative frequencies of queries covered by X, i.e. maximizes f_X.

We call this the INDEX SELECTION problem.

3. THE GREEDY SELECTION ALGORITHM

We propose a natural, greedy algorithm for the INDEX SELECTION problem, which we call the GREEDY SELECTION algorithm (see Figure 1). The algorithm has b iterations, where in each iteration, we add the document $d \in D \setminus X$ to X that maximizes the value of the objective function f_X. The primary challenge lies in the polynomial-time implementation of this algorithm. For every document $d \in D \setminus X$, the algorithm requires to compute the increase in the objective function if d were added to the index X. This entails computation of the sum of relative frequencies of queries in $Q(d) \setminus Q(X)$ for any document $d \in D \setminus X$. In a naïve implementation, this would take time proportional to the total number of queries, which is exponential in the size of the input. Our main technical contribution is a polynomial time implementation of the GREEDY SELECTION algorithm that proves the following theorem.

THEOREM 1. *For any $\epsilon > 0$, there is a randomized polynomial-time*[3] *algorithm (which we call the* GREEDY SELECTION *algorithm) for the* INDEX SELECTION *problem that achieves an approximation ratio of $1 - 1/e - \epsilon$ with high probability.*[4]

Since $Q(d) \setminus Q(X) = Q(X \cup \{d\}) \setminus Q(X)$, our problem boils down to estimating the sum of relative frequencies of queries cov-

[2]By index/cache size, we mean the number of unique documents in the index.

[3]The running time is polynomial in the input parameters as well as in $1/\epsilon$.

[4]A statement is said to hold *with high probability* (or whp) if the statement holds with probability $1 - o(1)$.

ered by a given subset of documents Y (here $Y = X \cup \{d\}$), i.e. estimating f_Y. We call this the COUNT QUERY problem.

Algorithm for the COUNT QUERY problem. We now describe an algorithm for estimating the sum of frequencies of queries that are covered by Y in a single category; the overall estimate is the sum of these estimates over all categories. Our algorithm will have an error parameter $\epsilon > 0$; as a preprocessing step, we round the relevance scores w_{ad} to multiples of a small enough value η such that the error due to the rounding can be absorbed in ϵ.

A simple idea would be to sample of a set of queries uniformly at random and use the fraction of sampled queries that are covered by Y as an estimator for the fraction of queries in $Q(Y)$. Unfortunately, the size of the sample required to control the sampling error is inversely proportional to the fraction that we are trying to estimate, which in our case implies that we might need exponentially many samples. To overcome this difficulty, we use a technique called *importance sampling* that was originally proposed by Karp, Luby, and Madras [12] for counting the number of truth assignments of a DNF[5] formula. The main idea is to sample queries from a *multiset* where each query appears as many times as the number of documents in Y that cover it, and then use a pre-determined order on the documents to estimate the ratio of the number of *distinct* queries in the multiset to the size of the multiset. It is easy to show that this ratio is always at least inverse polynomial, and can therefore we estimated using a polynomial number of samples.

We order the documents in Y in an arbitrary fixed order d_1, d_2, $\ldots$, $d_{|Y|}$. Let $N_r = f_{d_r}$ and $N = \sum_{r=1}^{|Y|} N_r$, which is the size of multiset formed by combining $Q(d_r)$ for all documents $d_r \in Y$. Our algorithm has two phases. In the first phase, we estimate the values of N_r (and therefore of N), and in the second phase, we perform the sampling procedure described above.

Phase 1. The first phase of our algorithm employs a dynamic program (DP) to estimate the values of N_r. Let I_q denote the set of attributes in query q (i.e. $q(i) \neq \phi$ iff $i \in I_q$). Further, let $S(j, \ell, \gamma)$ denote the sum of relative frequencies of queries q which have exactly ℓ attribute values, all of which are among the first j attributes, and have a sum of relevance scores over these attribute values of at least γ. Let $Q(d_r, j, \ell, \gamma)$ (for $\ell \leq j$) be the set of queries that have exactly ℓ attributes, all of which are among the first j attributes, and would have been covered by document d_r if the threshold on the average relevance score were some value γ/ℓ. Let $[n] = \{1, 2, \ldots, n\}$. Formally, $Q(d_r, j, \ell, \gamma) = \{q : |I_q| = \ell, I_q \subseteq [j], \sum_{i \in I_q} w_{q(i),d_r} \geq \gamma\}$, and $S(d_r, j, \ell, \gamma) = \sum_{q: q \in Q(d_r, j, \ell, \gamma)} f_q$. Note that $Q(d_r) = \cup_{\ell=1}^{k} Q(d_r, k, \ell, \theta\ell)$. Therefore, $N_r = \sum_{\ell=1}^{k} S(d_r, k, \ell, \theta\ell)$. (Recall that k is the number of attributes in the current category.) We will compute the values of $S(d_r, k, \ell, \theta\ell)$ using a DP that we describe below, which yields the values of N_r.

We make the following observations:

- If $j = \ell = 1$, then the queries in $Q(d_r, j, \ell, \theta\ell)$ are precisely the singleton queries containing attribute values $a \in A_1$ that satisfy $w_{ad_r} \geq \gamma$.
- If $j > \ell = 1$, then the queries in $S(d_r, j, \ell, \theta\ell)$ are also singleton queries, and contain attribute values $a \in A_t$ for all $t \leq j$ that satisfy $w_{ad_r} \geq \gamma$.
- If $j = \ell > 1$, then the queries in $S(d_r, j, \ell, \theta\ell)$ must contain an attribute value for every attribute in $[j]$.
- if $j > \ell > 1$, then the queries in $S(d_r, j, \ell, \theta\ell)$ either contain an attribute value for attribute j and $\ell - 1$ attribute values for attributes in $[j-1]$, **or** contain ℓ attribute values for attributes in $[j-1]$ and ϕ for attribute j.

These observations lead to the following DP:

$$
\begin{aligned}
S(d_r, 1, 1, \gamma) &= \sum_{a \in A_1 : w_{ad_r} \geq \gamma} f_a \\
S(d_r, j, 1, \gamma) &= f_\phi \cdot S(d_r, j-1, 1, \gamma) + \sum_{a \in A_j : w_{ad_r} \geq \gamma} f_a \\
&\quad \text{for } j \in [k], j > 1 \\
S(d_r, j, j, \gamma) &= \sum_{a \in A_j} f_a \cdot S(d_r, j-1, j-1, \max(\gamma - w_{ad_r}, 0)) \\
&\quad \text{for } j \in [k], j > 1 \\
S(d_r, j, \ell, \gamma) &= f_\phi \cdot S(d_r, j-1, \ell, \gamma) + \\
&\quad \sum_{a \in A_j} f_a \cdot S(j-1, \ell-1, \max(\gamma - w_{ad_r}, 0)) \\
&\quad \text{for } j \in [k], j > 1, \ell \in [j-1]
\end{aligned}
$$

Since the relevance scores are in multiples of η, which is polynomial in the error parameter ϵ, the running time of the this DP is polynomial.

Phase 2. In the second phase of the COUNT QUERY algorithm, we draw a set of n sample queries $q_1, q_2, \ldots, q_n$ (the value of n will be determined later). Each query $q_j \in cup_{r=1}^{|Y|} Q(d_r)$ is drawn i.i.d. using the following procedure: First, we sample a document d_r with probability $\frac{N_r}{N}$. Next, we sample the number of attributes ℓ in q_j with probability $\frac{S(k,\ell,\theta\ell)}{N_r}$. This restricts the set of queries to $Q(k, \ell, \theta\ell)$. Finally, we sample a query in $Q(k, \ell, \theta\ell)$ using a DP like the one in phase 1, the details of which are given in Figure 2. (In Figure 2, $q(-j)$ denotes $(q(1), q(2), \ldots, q(j-1))$ and f_Φ is the product of f_ϕ over all attributes $k < j$.)

Next, we need to define the estimator for f_Y that we generate using these sampled queries. Let

$$Q'(d_s) = \{q \in Q(d_s) : \forall t < s,\ q \notin Q(d_t)\}.$$

For each $j \in [n]$, we set $x_j = 1$ if $q_j \in Q'(d_r)$; otherwise, $x_j = 0$. The estimate for f_Y returned by the algorithm is

$$X = \left(\frac{\sum_{j=1}^{n} x_j}{n} \right) N.$$

Analysis. Our main technical lemma establishes the accuracy of the above algorithm for the COUNT QUERY problem.

LEMMA 1. *There is a randomized polynomial-time[6] approximation scheme for the* COUNT QUERY *problem that has a multiplicative error of* $1 - \epsilon$ *for any fixed* $\epsilon > 0$ *with high probability.*

Before proving this lemma, let us show that it is sufficient to prove Theorem 1. Let us denote the COUNT QUERY algorithm by $\mathcal{O}$ (we call it an *oracle*), and the GREEDY SELECTION algorithm by $\mathcal{A}$. Recall that in each iteration of $\mathcal{A}$, the document to be added to the index X is selected as follows: for every document $d \in D \setminus X$, we use oracle $\mathcal{O}$ to estimate $f_{X \cup \{d\}}$ and add the document for which this estimate is the maximum to X.

The next lemma maps the approximation ratio of $\mathcal{A}$ to that of $\mathcal{O}$.

[5]Disjunctive Normal Form

[6]The algorithm runs in time polynomial in the input parameters and in $1/\epsilon$.

$B \leftarrow \{a \in A_j : w_{ad} \geq \gamma\}$
For each $a \in A_j$, $r_a \leftarrow \frac{f_a}{\sum_{x \in B} f_x}$
if $\ell > 1$ **then**
  For each $a \in A_j$, $p_a \leftarrow \frac{f(a) \cdot S(d, j-1, \ell-1, \max(\gamma - w_{ad}, 0))}{S(d,j,\ell,\gamma)}$
end if
if $j > \ell$ **then**
  $p_\phi \leftarrow \frac{f_\phi \cdot S(d, j-1, \ell, \gamma)}{S(d,j,\ell,\gamma)}$
end if
Case 1 ($j = \ell = 1$):
  For each $a \in B$ with probability r_a, $q(j) \leftarrow a$
Case 2 ($j > \ell = 1$):
  For each $a \in B$ with probability $\frac{f_a \cdot f_\Phi}{S(d,j,\ell,\gamma)}$,
  $q(j) \leftarrow a$ and $q(-j) \leftarrow (\phi, \phi, \ldots, \phi)$;
  with the remaining probability,
  $q(j) \leftarrow \phi$ and $q(-j) \leftarrow \mathbf{Sample}(d, j-1, \ell, \gamma)$
Case 3 ($j = \ell > 1$):
  For each $a \in A_j$ with probability p_a, $q(j) \leftarrow a$ and
  $q(-j) \leftarrow \mathbf{Sample}(d, j-1, \ell-1, \max(\gamma - w_{ad}, 0))$
Case 4 ($j > \ell > 1$):
  For each $a \in A_j \cup \{\phi\}$ with probability p_a, $q(j) \leftarrow a$.
  if $q(j) = \phi$ **then**
    $q(-j) \leftarrow \mathbf{Sample}(d, j-1, \ell, \gamma)$;
  else
    $q(-j) \leftarrow \mathbf{Sample}(d, j-1, \ell-1, \max(\gamma - w_{a,d}, 0))$.
return q

Figure 2: The subroutine $\mathbf{Sample}(d, j, \ell, \gamma)$ used in phase 2 of the algorithm for the COUNT QUERY problem

LEMMA 2. *For any $\delta > 0$, there is a setting of ϵ for oracle $\mathcal{O}$ such that algorithm $\mathcal{A}$ has an approximation ratio of $1 - \delta$ whp, and has running time polynomial in $1/\delta$.*

To prove this lemma, we introduce some terminology. In any call to oracle $\mathcal{O}$, we say that the oracle is *successful* if the estimate produced is $(1 \pm \epsilon)$-accurate; otherwise, we say that the oracle failed. The next lemma quantifies the impact of a successful call to $\mathcal{O}$ on algorithm $\mathcal{A}$.

LEMMA 3. *Suppose in an iteration of algorithm $\mathcal{A}$, document d is selected, and let $d^* = \arg\max_{d \in D \setminus X} f_{X \cup \{d\}}$. Further, let $Z = f_{X \cup \{d\}}$ and $W = f_{X \cup \{d^*\}}$. If all the calls to oracle $\mathcal{O}$ in this iteration (with parameter ϵ) are successful, then $Z \geq \left(\frac{1-\epsilon}{1+\epsilon}\right) W$.*

PROOF SKETCH. Let $Y = f_X$; let $\widetilde{Y}$, $\widetilde{Z}$, and $\widetilde{W}$ denote the values of Y, Z, and W as estimated by oracle $\mathcal{O}$. Let $\rho = \frac{\widetilde{Y}}{Y} - 1$. Thus, $\rho \in [-\epsilon, \epsilon]$. Then,

$$Z - Y \geq \frac{\widetilde{Z}}{1+\epsilon} - \frac{\widetilde{Y}}{1+\rho} \geq \left(\frac{1-\epsilon}{1+\epsilon}\right) W - Y.$$

(Calculation details deferred to full version.) □

We use this lemma to prove Lemma 2.[7]

PROOF OF LEMMA 2. The probability that the oracle $\mathcal{O}$ fails in at least one call is $o(1)$, and is therefore absorbed in the failure probability of algorithm $\mathcal{A}$. Hence, we only need to show that if the oracle $\mathcal{O}$ is successful in every call of $\mathcal{A}$, then the approximation ratio of $\mathcal{A}$ is at most $1 - 1/e - \delta$.

Let X_i denote the selected documents in the first i iterations. Let the sum of relative frequencies of queries covered by an optimal solution be OPT. By averaging, there is at least one document $d \in D \setminus X_i$ such that

$$f_{X_i \cup \{d\}} - f_{X_i} \geq \frac{\text{OPT} - Y_i}{b}.$$

By Lemma 3, the success of oracle $\mathcal{O}$ with parameter ϵ (we will set the value of ϵ later) in iteration $i+1$ implies that

$$f_{X_{i+1}} \geq \rho \left(\frac{\text{OPT} - f_{X_i}}{b} + f_{X_i} \right),$$

where $\rho = \frac{1-\epsilon}{1+\epsilon}$. Since $X_0 = \emptyset$, the above recurrence yields

$$\begin{aligned} f_{X_b} &= \left(\frac{\rho}{b}\right) \text{OPT} \left(\sum_{i=0}^{b-1} \rho^i \left(1 - \frac{1}{b}\right)^i \right) \\ &= \left(\frac{\rho}{1 + (1-\rho)b}\right) \text{OPT} \left(1 - \rho^b \left(1 - \frac{1}{b}\right)^b \right). \end{aligned}$$

We set $\rho = 1 - \frac{c}{b}$ for some constant c that we will determine later. Then,

$$\begin{aligned} f_{X_b} &> \left(\frac{1 - c/b}{1+c}\right) \text{OPT} \left(1 - e^{-(c+1)}\right) \\ &\geq \left(\frac{1-c}{1+c}\right) \text{OPT} \left(1 - e^{-(c+1)}\right) \\ &> (1 - 2c) \text{OPT} \left(1 - \frac{1}{e}\right) \\ &= 1 - \frac{1}{e} - \delta, \end{aligned}$$

where we set $\epsilon = \frac{\delta}{b}$ to ensure that

$$c = \left(1 - \frac{1-\epsilon}{1+\epsilon}\right) b > \epsilon b = \delta > \frac{\delta}{2(1 - 1/e)}.$$

We used $\left(1 - \frac{x}{b}\right)^b < e^{-x}$ (first step), $b \geq 1$ (second step), and $\frac{1-c}{1+c} < 1 - 2c$ and $c > 0$ (third step) in the calculation. □

We now analyze the COUNT QUERY algorithm, and prove Lemma 1. The analysis comprises two claims: that the estimator X is unbiased, and that for a large enough value of n, X has small variance. The next lemma combines these claims.

LEMMA 4. *In the algorithm for the COUNT QUERY problem,*

$$\mathbb{P}\left[X \notin (1 \pm \epsilon) f_Y\right] = o(1).$$

PROOF. For any sample query q_j in the above algorithm,

$$\mathbb{E}[x_j] = \sum_{r=1}^{|Y|} \left(\frac{N_r}{N}\right) \cdot \left(\frac{\sum_{q \in Q'(d_r)} f_q}{N_r}\right) = \frac{\sum_{r=1}^{|Y|} \sum_{q \in Q'(d_r)} f_q}{N}.$$

Clearly, each query in $Q(Y)$ is present in $Q'(d_r)$ for exactly one document $d_r \in Y$. Therefore, $\mathbb{E}[x_j] = \frac{f_Y}{N}$, and hence, the estimator is unbiased.

Now, we employ Chernoff bounds (see e.g. [17]) to bound the probability of deviation of the estimator. Let $\zeta = \frac{f_Y}{N}$. Then

$$\mathbb{P}[X \notin (1 \pm \epsilon) f_Y] = \mathbb{P}\left[\sum_{i=1}^{n} x_i \notin (1 \pm \epsilon)\zeta n\right] \leq e^{-\frac{\epsilon^2 \zeta n}{3}},$$

which is $o(1)$ for $n = \omega\left(\frac{3}{\epsilon^2 \zeta}\right)$.

We now note that $\frac{1}{\zeta} = \frac{N}{f_Y}$, which in turn equals

$$\frac{\sum_{q \in Q(Y)} f_q |\{r : q \in Q(d_r)\}|}{\sum_{q \in Q(Y)} f_q} \leq \max_{q \in Q(Y)} |\{r : q \in Q(d_r)\}| \leq |Y|.$$

[7] The proof implicitly uses the fact that the objective function for the INDEX SELECTION problem is *submodular*.

Hence, n is polynomial in the input parameters. $\square$

Running Time. The overall running time of the algorithm is

$$O\left(b \cdot |D| \cdot n \cdot \max_j \left(\sum_{i=1}^{k_j} |A_i|\right) + \sum_j k_j \cdot \sum_{i=1}^{k_j} \frac{|A_i|}{\eta}\right),$$

which is markedly better than the running time of a naïve implementation of the greedy algorithm, $O(b \cdot |D| \cdot \sum_j \prod_{i=1}^{k_j} |A_i|)$. Further, we will now describe algorithmic and data structural optimization that makes our implementation of the algorithm much more efficient than that suggested by the worst-case running time above. Our first observation is that the sets of sample queries generated in various calls to the COUNT QUERY sub-routine need not be independent since we are using an union bound over the error probabilities. This leads to the first optimization: For every document $d \in D$, we use the **Sample** sub-routine to generate apriori a set of (mutually independent) sample queries from $Q(d)$ with probabilities proportional to their relative frequencies (we call the set $SQ(d)$). Whenever the COUNT QUERY algorithm requires a sample query from $Q(d)$, it is provided the first query in $SQ(d)$ that has not been used previously in the current run of the COUNT QUERY algorithm.

To improve the performance and scalability of the GREEDY SELECTION algorithm further, we design the following data structure. Let P be a max priority queue of all the documents $d \in D$ ordered according to $f_d \cdot (1 - g_d)$, where f_d is the sum of frequencies of all queries covered by d, and g_d is the fraction of queries in $SQ(d)$ that are already covered by the current index X. Clearly, the document at the front of this priority queue is the next document that should be inserted in X. However, this data structure presents two challenges:

- Every time a document is inserted in X, we incur the huge overhead of having to update the values of g_d for every document d in P.
- Since the set of documents D is enormous, it would take a large amount of memory space to store the query samples corresponding to all the documents in P.

We overcome these challenges by adopting a *lazy policy*: For every document d in P, we maintain a counter c_d indicating the last index in X for which the value of g_d has been updated for document d. Initially, $c_d = 0$ and $g_d = 1$ for every document $d \in D$. Further, no sample query is generated for any document at the outset. Every step of the algorithm now comprises the following operations: We dequeue the document at the front of the priority queue P; let this document be d. If $c_d = 0$, then we generate the set of sample queries $SQ(d)$. Now, we check if $c_d < |X|$; if so, then we update the value of g_d for the current contents of X, set c_d to $|X|$, and re-insert document d in P. (Note that d may no longer be at the front of the queue since the value of g_d might have increased). On the other hand, if $c_d = |X|$, then we add d to X. Clearly, in this case, even though some of the documents in P have stale (i.e. smaller than actual) values of g, document d is indeed the one that maximizes $f \cdot (1 - g)$. Note that this solves both problems: we are now updating the values of c and g only when required, and also generating sample queries only for documents that show up at the front of P. In fact, we observe that most documents have small values of f and therefore never appear at the front of P.

If we are required to handle multiple coverages (see the first extension in the next section), we keep a counter $n(q, d)$ for every sample query $q \in SQ(d)$ (rather than g_d for document d) that indicates the number of documents in $X_{c(d)}$ that cover document d. (Recall that X_i is the prefix of X containing the first i inserted documents.) The priority queue P is now ordered on $\sum_{q \in SQ(d)} \rho^{n(q,d)+1} f_q$. When document d appears at the front of P, we increase the counters $n(q, d)$ by the number of documents in $X \setminus X_{c(d)}$ that cover query q.

4. EXTENSIONS

In this section, we describe various extensions to the GREEDY SELECTION algorithm.

Multiple Coverage. As mentioned in the introduction, $k > 1$ in most applications, e.g. a search engine returns multiple results in response to a user query. We generalize the INDEX SELECTION problem to this scenario by introducing an utility function u that maps the number of documents relevant to a query to their aggregate benefit. The objective function now becomes

$$\sum_{j=1}^{L} \sum_{q \in Q_j} f_q u(N(X, q)),$$

where $N(X, q)$ is the number of documents in X that are relevant to query q.

Note that the GREEDY SELECTION algorithm should logically prefer a document that covers queries that already have a significant coverage in the index to a document that covers queries having low coverage. This follows from the fact that the index is useful for a query only if has at least k relevant documents. To reflect this bias, we define the utility function as $u(i) = \min(2^i, 2^k)$ in our experiments. We may note that this choice violates the submodularity of the overall utility function, but as we show in the experiments, this leads to better results than choosing $u(i) = \min(i, k)$ (which is submodular) as in [1].

Dynamic Set of Documents. Recall that in most practical situations, the set of documents typically change over time. First, we consider the modifications to the index on the removal of a document. If the document was not present in the index, then we do not need to do anything. On the other hand, if the document was in the index, then we replace it by a new document which is chosen by running a single iteration of the GREEDY SELECTION algorithm.

Now, let us consider the modifications to the index on the addition of a document. Note that we have two decisions in this case: first, do we add the document to the index; and second, if we do, which document do we evict from the index? Further investigation of the problem reveals that the new document may cause more drastic changes of the following kind: if the new document covers all queries covered by the documents currently in the index, then inserting the new document in the index would make the current documents redundant (for $k = 1$). In this case, we have to recompute the index from scratch.

However, note that the situation described above is rather pathological. In practice, we do not expect the index to change drastically on the addition of a single document. To quantify this intuition, we impose the additional constraint that the index cannot change by more than one document for every document arrival. This has the added benefit that auxiliary data structures such as the indexing mechanisms do not need frequent drastic changes. The following simple algorithm now solves this problem efficiently: For every document $x \in X$, we estimate $f_{X,x,d} = f_{(X \setminus \{x\}) \cup \{d\}}$ using COUNT QUERY calls, where d is the new document. If $f_X \geq f_{X,x,d}$ for all $x \in X$, then we do not include the new document d in the index; otherwise, we evict $x^* = \arg\max_{x \in X} f_{X,x,d}$ from the index and replace it by d. Note that the number of calls

to the COUNT QUERY sub-routine is now proportional to b rather than $|D|$.

Streaming Set of Documents. Finally, consider the scenario where the set of documents appear sequentially, and the algorithm needs to immediately decide whether to include a document or discard it on its arrival. We propose the following algorithm in this scenario: The algorithm has multiple *epochs*, where each epoch is characterized by a guessed value opt of the objective in an offline optimal solution. An epoch ends when the objective of the algorithmic solution exceeds opt, at which point we double our guess and start the next epoch. In any epoch, the algorithm includes every document that increases the objective value by at least $\frac{\mathrm{opt}}{b_r}$, where $b_r = b - |X|$. Note that the increase in the objective can be computed by making a single call to the COUNT QUERY sub-routine. If the set of documents have some desirable properties (e.g. are drawn i.i.d. from a distribution), then this simple algorithm can be shown to have a constant approximation factor.

5. EXPERIMENTS

In this section, we evaluate the GREEDY SELECTION algorithm on real-world data. Recall that this algorithm can be used to generate a tiered index for the set of products in the index of a commerce search engine. We will begin by describing the datasets used in our experiments.

5.1 Datasets

For our experiments, we built a prototype search engine and populated it with real data from the shopping vertical of a commercial search engine. To this end, we downloaded detailed descriptions for about 30 million products from a commercial online shopping catalog. These products were categorized into around 600 leaf-level categories under 32 top-level categories such as `electronics`, `camera and optics`, `clothing and shoes`, and so on. The GREEDY SELECTION algorithm does not interact between categories, and therefore can be scaled to large indexes by processing combinations of categories in parallel. For clarity and without loss of generality, we restrict our analysis to two top-level categories in the catalog: `electronics` and `cameras and optics` with around $150,000$ and $40,000$ products respectively. We note that there are 74 sub-categories under electronics (such as `televisions`, `equalizers`, and `GPS Receivers`) and 49 sub-categories under cameras (such as `digital cameras`, `telescopes`, and `lenses`).

Next, we describe how we set the relevant scores w_{ad} for the products in the dataset. Consider, for example, a `sony bravia xbr tv` (with product id d). Since its product description contains attribute `brand:sony`, we set $w_{\mathrm{sony},d} = 1.0$. Further, as mentioned in the introduction, we enrich the product with related attribute value information. For example, if `samsung` is a relevant brand for `sony` (i.e. users searching for `samsung` products previously have ended up buying `sony` products), then we set $w_{\mathrm{samsung},d}$ to some value between 0 to 1. For a detailed description of how such similarity scores are chosen, the reader is referred to [19].

The Query Set. For our experiments, we sampled around $100,000$ queries ($\mathcal{Q}$) from the query log of the same online shopping portal. We used uniform sampling over the queries in the query log (note that popular queries automatically get a bias because they appear more frequently in the query log). In order to analyze the effect of structure in the query on the performance of GREEDY SELECTION, we categorized these queries into five buckets depending on the degree of structure extracted from the query ranging from bucket 1 containing highly unstructured queries (mostly with one annotated keyword) to bucket 5 composed of many annotated tokens (sometimes as many as six). The number of queries in bucket 1 constituted around 20% of the queries in $\mathcal{Q}$. The number of queries in bucket 5 had a higher fraction of around 32%. In fact, the fraction of queries with at least 3 annotated keywords was close 70% of $\mathcal{Q}$. We note that annotation of queries is not the focus of this study. Toward this end, we used the query annotator described in [23].

In our experiments, we only considered the important attributes associated with each category. These are attributes that have a high *selectivity*, i.e. occur in a large fraction of product descriptions or user queries. We set the selectivity threshold to a conservative value of 0.5 and this yielded around six important attributes in each category, and for each selected attribute, we computed the relative frequencies of individual attribute values in a query log that spanned a six-month period.

5.2 Baseline Algorithms

We compare the performance of GREEDY SELECTION with the cache selection algorithms in Baeza-Yates *et al* [3] and Anagnostopoulos *et al* [1].

In the first algorithm, the goal is to populate the cache with a set of posting lists for important query terms. Each posting list is characterized by the frequency f of the associated term, and the size of the posting list s. The algorithm greedily selects the posting list with the minimum ratio s/f repeatedly until the entire cache is filled up. We refer to this algorithm as POSTING LISTS in our experiments.

The algorithm in [1] is also an iterative greedy algorithm, but selects, in each iteration, only the document that is relevant for the maximum number of queries not already covered by k cached documents. We refer to this algorithm as STOCHASTIC QUERY COVER in our experiments.

5.3 Experimental Results

Our experimental results can be categorized into two parts that we call *index generation* and *index serving*.

Index Generation Experiments. In the index generation experiments, we use around 10% of our query set $\mathcal{Q}$ (we call this the *training set* $\mathcal{T}$) for generating the index using all three algorithms separately. The performance of the algorithms in this step is measured by the fraction of queries in $\mathcal{T}$ covered by the index. This gives us a measure of the quality of the selection process used by the three algorithms in terms of coverage achieved on the training data itself. Note that this measure of effectiveness is extremely well-suited to STOCHASTIC QUERY COVER since it does not test the generalizability of the algorithm. Nevertheless, we show that GREEDY SELECTION has almost the same performance as STOCHASTIC QUERY COVER at small values of k and outperforms STOCHASTIC QUERY COVER for larger values of k. In addition, we show that we consistently outperform POSTING LISTS in this set of experiments.

We ran our experiments for cache sizes (or budgets) b ranging from 1% to 5% of the total index size, number of results k ranging from 6 to 14[8], and the relevance threshold θ ranging from 0.1 to 0.5. We also varied the number of samples in GREEDY SELECTION from 10 to 50, but observed that the sample size has negligible effect on the performance of the algorithm. We therefore only report the results for a sample size of 10 in all the experiments.

[8]We selected this range so that the typical value of $k = 10$ falls in the middle of the range.

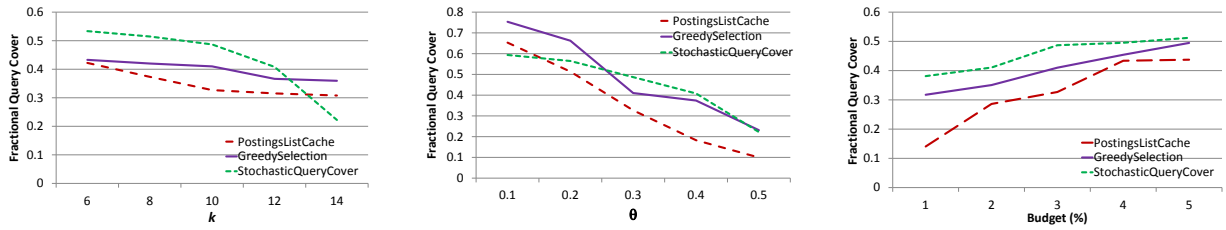


Figure 3: Performance of GREEDY SELECTION compared to POSTING LISTS and STOCHASTIC QUERY COVER using cache hit ratio on the training set of queries for various parameter ranges: (a) Effect of number of results k needed to cover a query, with $b = 3\%$ and $\theta = 0.3$, (b) Effect of relevance threshold θ, with $b = 3\%$ and $k = 10$, and (c) Effect of index size b with $\theta = 0.3$ and $k = 10$.

Fig. 3 illustrates the relative performance of GREEDY SELECTION compared to POSTING LISTS and STOCHASTIC QUERY COVER for all the aforementioned parameter values. In Fig. 3(a), all algorithms exhibit a drop in performance as k is increased since it becomes more difficult to cover a query. (Recall that a query is covered only if at least k relevant documents are present in the index.) Also, as expected, STOCHASTIC QUERY COVER performs better than GREEDY SELECTION at small values of k because (1) STOCHASTIC QUERY COVER optimizes over only the training query set whereas GREEDY SELECTION optimizes over the entire query space, and (2) STOCHASTIC QUERY COVER uses an expensive but accurate exhaustive search to make its greedy choices wheareas GREEDY SELECTION uses a much more efficient but slightly less accurate sampling technique to make its greedy choices. Interestingly, as k increases, the performance of STOCHASTIC QUERY COVER degrades rapidly. This can be attributed to the fact that STOCHASTIC QUERY COVER treats documents that cover queries already having significant cache coverage identically to documents that cover queries having no cache coverage. Clearly, the former set of documents should get precedence, and as mentioned in Section 4, GREEDY SELECTION ensures this by using an increasing utility function on coverage for making our greedy choices. Comparing with POSTING LISTS, we note that GREEDY SELECTION performs better than POSTING LISTS in the entire range of k since it is able to use cache space more judiciously by distinguishing between individual documents in a posting list based on whether they are relevant for other queries or not. In fact, GREEDY SELECTION does better than POSTING LISTS by almost 20% at the typical value of $k = 10$.

In Fig. 3(b), where we vary the relevance threshold θ, GREEDY SELECTION consistently outperforms POSTING LISTS by at least 20% and the gap in performance increases with the relevance threshold. Again, the greater selectivity of GREEDY SELECTION in terms of choosing documents gives it this advantage over POSTING LISTS. Furthermore, the performance of GREEDY SELECTION is either better than or comparable to that of STOCHASTIC QUERY COVER at all values of θ, even though the evaluation is only on the training query set.

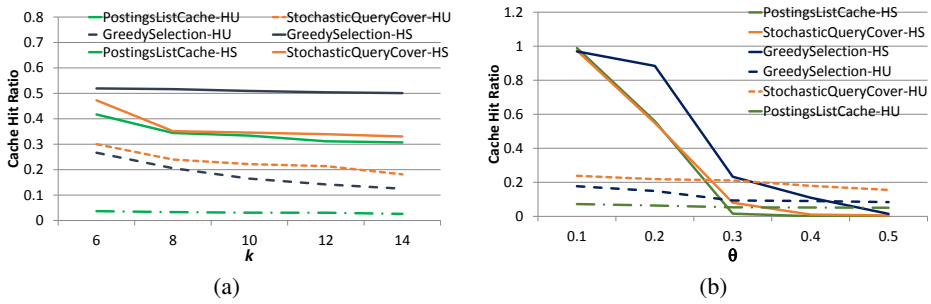
Finally, in Fig. 3(c), we observe that POSTING LISTS has a less smooth uptick in performance than GREEDY SELECTION with increasing index size. This is because it is always forced to include entire posting lists, and therefore often cannot use residual space in the cache. Such behavior causes pockets of relatively less progress with increase in budget, which is illustrated in Fig. 3(c). On the other hand, GREEDY SELECTION, with its ability to select one document at a time, exhibits a smoother increase in query coverage as the budget is increased. Further, GREEDY SELECTION gets closer to STOCHASTIC QUERY COVER as b increases and again, does not show a significant drop in performance when compared to STOCHASTIC QUERY COVER across the whole range of budgets.

To summarize, this experiment shows us that even when we restrict ourselves to evaluation on training data, GREEDY SELECTION exhibits performance that is comparable to (and sometimes better than) STOCHASTIC QUERY COVER. Further, for various ranges of parameters, we have exhibited that GREEDY SELECTION comfortably outperforms POSTING LISTS in this set of experiments.

Index Serving Experiments. In the next set of experiments, we measured the generalization capabilities of the algorithms by defining our test query set as $\mathcal{S} = \mathcal{Q} \setminus \mathcal{T}$ and measuring the hit ratios for different parameter values. We further analyzed the effect of the extent of structure in a query on the hit ratios. As described earlier, we classified the queries into five buckets ranging from highly unstructured ($= 1$) to highly structured ($= 5$). Again, we varied k from 6 to 14 and θ from 0.1 to 0.5, whereas the budget was fixed to 3% of the total index size. In Fig. 4, we illustrate the performance of the three algorithms as a function of these parameters.

As Fig. 4(a) shows, GREEDY SELECTION results in a much better hit ratio compared to POSTING LISTS and STOCHASTIC QUERY COVER when the queries are highly structured. However, when the queries become less structured, even though GREEDY SELECTION continues to dominate POSTING LISTS in performance, STOCHASTIC QUERY COVER starts performing marginally better than GREEDY SELECTION. Unstructured queries are less specific than structured queries and therefore, are often covered by more documents in the index. STOCHASTIC QUERY COVER performs well in this regime since the lack of specificity in unstructured queries implies that they are covered by many documents that also cover queries in the training set. On the other hand, structured queries tend to be more specific and are often covered by fewer documents in the index. Such queries are often missed by a typical LRU cache and therefore, almost always, rely on a tier-one index to be served efficiently. For such queries, GREEDY SELECTION outperforms POSTING LISTS and STOCHASTIC QUERY COVER by around 30% for typical values of $k = 10$. In fact, the relative performance of GREEDY SELECTIONincreases as the value of k increases. This reflects the powerful generalizing capability of GREEDY SELECTION to queries with unseen or rare combination of attribute values.

Fig. 4(b) shows the performance of the algorithms for different values of θ. Here, we observe that while the performance of all the algorithms drops as θ increases, GREEDY SELECTIONconsistently exhibits a better hit ratio than POSTING LISTS and STOCHASTIC



(a) (b)

Figure 4: Performance of GREEDY SELECTION compared to POSTING LISTS and STOCHASTIC QUERY COVER at index serve time: (a) Effect of the number of documents k needed to cover a query, with $b = 3\%$ and $\theta = 0.3$, (b) Effect of the relevance threshold θ, with $b = 3\%$ and $k = 10$. (HU = "Highly Unstructured", HS = "Highly Structured")

QUERY COVER , especially for the highly structured queries. For example, for $\theta = 0.3$, GREEDY SELECTIONalmost outperforms the other algorithms 2 to 1. This again highlights the difference in the generalizing capabilities of the algorithms. In order to explicitly measure the effectiveness of the algorithms on less frequent queries, we repeated the experiment using unique queries in $\mathcal{S}$ (by ignoring the query frequencies) and we observed a similar gap in performance between GREEDY SELECTION and the other algorithms.

Since one of the primary applications of the selected documents is for tiered indexing, we measured the quality of the resulting index built using these documents. The quality of a document is measured as its average relevance score over all attribute values. In Fig. 5, we note that GREEDY SELECTION includes more relevant documents in the index compared to POSTING LISTS. In fact, the document quality is almost 40% better when using GREEDY SELECTION (see Fig. 5(a)). This can be attributed to the fact that while POSTING LISTS does not distinguish between documents in the same posting list, GREEDY SELECTION selects one document at a time and therefore, selects only high quality documents in the cache.

6. RELATED WORK

Previous work on understanding user query distributions in terms of their relative frequencies and repetition (see e.g. [26, 6]) has observed on numerous occasions that users share many popular queries. This immediately motivates a caching infrastructure such as the one that we propose in this paper in order to improve the performance of search engines. In the context database applications, there has been extensive work on building efficient middle-tier caches [2, 5, 15]. Early work on caching and tiering techniques for search queries can be largely categorized into two broad areas — result caching [20, 3, 25, 10, 9] and tiered indexing [21, 16, 18, 25, 4, 14, 11, 1]. The basic problem behind caching is to understand the cost of serving a given query workload. For example, Raghavan and Sever [20] focused on popular queries in query logs. More recently, Gan and Suel [9] proposed a weighted caching scheme that includes the cost of processing the more frequent queries to a search engine. Risvik *et al* [21] introduced the concept of tiering in order to improve the performance of search engines. Ntoulas and Cho [18] studied term and document pruning strategies with the aim of reducing resources needed to handle a given query. Skobeltsyn *et al* [25] combined result caching with index pruning for better efficiency. Baeza-Yates *et al* [3] compared the impact of results caching and static caching of posting lists on the performance of web search engines. They concluded that static caching of posting lists outperforms all other variations of caching they considered in their study. Long and Suel [16] introduced a three-level caching architecture that includes on-disk caching of the posting lists for popular term combinations. Leung *et al* [14] studied the problem of tier selection for storing documents with the goal of minimizing the number of tier traversals for a given workload of queries. The work of Anagnostopoulos *et al* [1] is the closest to ours. They gave a greedy algorithm for document selection based on a training set to queries. In fact, we will compare the performance of our algorithm to those of [3] and [1].

Another line of work that incorporates relevance and similarity functions is on top-k queries (also referred to as k nearest neighbors (kNN))in databases [22]. More recent work focused on applying the kNN problem to searching over a database [27, 24]. Our work differs from this line of work in that we focus our work on generating a tiered index that addresses the low latency requirements often imposed in online search settings.

From a theoretical perspective, our algorithms fall in a class of well-studied combinatorial optimization problems called *covering* problems. Of particular relevance is the maximum k-coverage problem, where the goal is to select a collection of at most k sets in a universe of weighted elements such that the sum of weights of elements covered by these sets is maximized. This is a classical NP-complete problem, and it has been shown that the natural greedy algorithm for this problem is indeed optimal under standard assumptions in complexity theory [8]. Many variants and generalizations of this problem have also been studied in the literature (see e.g. [13, 7]), including ones where we have oracle access to the sets. Our problem falls in this category, and the main technical contribution of this paper is to design a suitable oracle for our setting by exploiting the structural properties of commerce search queries.

7. CONCLUSION AND FUTURE WORK

In summary, we proposed a technique for caching search results for structured search domains such as commerce search, and demonstrated the effectiveness of our proposed algorithm both theoretically and via experiments on real-world data. There are multiple important questions that this work raises. For what other search domains can such result caches be generated using our techniques? In particular, can our techniques be extended to unstructured domains such as general web search? How to handle arbitrarily correlated attributes and arbitrary relevance functions?

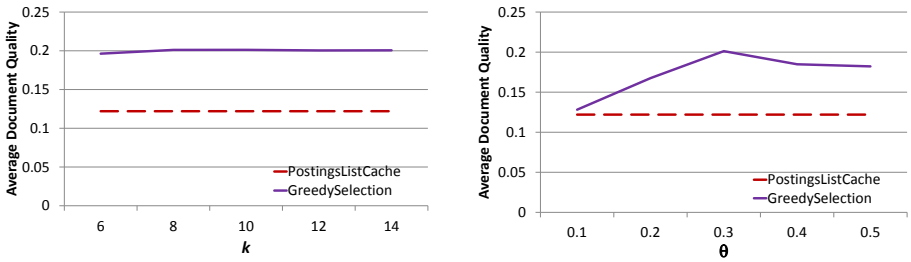


Figure 5: The quality of the index computed using GREEDY SELECTION compared to POSTING LISTS: (a) Effect of the number of documents k needed to cover a query, with $b = 3\%$ and $\theta = 0.3$, and (b) Effect of relevance threshold θ, with $b = 3\%$ and $k = 10$.

8. REFERENCES

[1] Aris Anagnostopoulos, Luca Becchetti, Stefano Leonardi, Ida Mele, and Piotr Sankowski. Stochastic query covering. In *WSDM*, pages 725–734, 2011.

[2] Jesse Anton, Lawrence Jacobs, Xiang Liu, Jordan Parker, Zheng Zeng, and Tie Zhong. Web caching for database applications with oracle web cache. In *SIGMOD Conference*, pages 594–599, 2002.

[3] Ricardo A. Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. The impact of caching on search engines. In *SIGIR*, pages 183–190, 2007.

[4] Ricardo A. Baeza-Yates, Vanessa Murdock, and Claudia Hauff. Efficiency trade-offs in two-tier web search systems. In *SIGIR*, pages 163–170, 2009.

[5] Christof Bornhövd, Mehmet Altinel, Sailesh Krishnamurthy, C. Mohan, Hamid Pirahesh, and Berthold Reinwald. Dbcache: Middle-tier database caching for highly scalable e-business architectures. In *SIGMOD Conference*, page 662, 2003.

[6] Andrei Z. Broder, Marcus Fontoura, Vanja Josifovski, Ravi Kumar, Rajeev Motwani, Shubha U. Nabar, Rina Panigrahy, Andrew Tomkins, and Ying Xu. Estimating corpus size via queries. In *CIKM*, pages 594–603, 2006.

[7] Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM*, pages 72–83, 2004.

[8] Uriel Feige. A threshold of *ln n* for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[9] Qingqing Gan and Torsten Suel. Improved techniques for result caching in web search engines. In *WWW*, pages 431–440, 2009.

[10] Charles Garrod, Amit Manjhi, Anastasia Ailamaki, Bruce M. Maggs, Todd C. Mowry, Christopher Olston, and Anthony Tomasic. Scalable query result caching for web applications. *PVLDB*, 1(1):550–561, 2008.

[11] Jun-Seok Heo, Junghoo Cho, and Kyu-Young Whang. The hybrid-layer index: A synergic approach to answering top-k queries in arbitrary subspaces. In *ICDE*, pages 445–448, 2010.

[12] Richard M. Karp, Michael Luby, and Neal Madras. Monte-carlo approximation algorithms for enumeration problems. *J. Algorithms*, 10(3):429–448, 1989.

[13] Samir Khuller, Anna Moss, and Joseph Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70(1):39–45, 1999.

[14] Gilbert Leung, Novi Quadrianto, Alexander J. Smola, and Kostas Tsioutsiouliklis. Optimal web-scale tiering as a flow problem. In *NIPS*, pages 1333–1341, 2010.

[15] Wen-Syan Li, Daniel C. Zilio, Vishal S. Batra, Calisto Zuzarte, and Inderpal Narang. Load balancing and data placement for multi-tiered database systems. *Data Knowl. Eng.*, 62(3):523–546, 2007.

[16] Xiaohui Long and Torsten Suel. Three-level caching for efficient query processing in large web search engines. In *WWW*, pages 257–266, 2005.

[17] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1997.

[18] Alexandros Ntoulas and Junghoo Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *SIGIR*, pages 191–198, 2007.

[19] Debmalya Panigrahi and Sreenivas Gollapudi. Result enrichment in commerce search using browse trails. In *WSDM*, pages 267–276, 2011.

[20] Vijay V. Raghavan and Hayri Sever. On the reuse of past optimal queries. In *SIGIR*, pages 344–350, 1995.

[21] Knut Magne Risvik, Yngve Aasheim, and Mathias Lidal. Multi-tier architecture for web search engines. In *LA-WEB*, pages 132–143, 2003.

[22] Nick Roussopoulos, Stephen Kelley, and Frédéic Vincent. Nearest neighbor queries. In *SIGMOD Conference*, pages 71–79, 1995.

[23] Nikos Sarkas, Stelios Paparizos, and Panayiotis Tsaparas. Structured annotations of web queries. In *SIGMOD Conference*, pages 771–782, 2010.

[24] Thomas Seidl and Hans-Peter Kriegel. Optimal multi-step k-nearest neighbor search. In *SIGMOD Conference*, pages 154–165, 1998.

[25] Gleb Skobeltsyn, Flavio Junqueira, Vassilis Plachouras, and Ricardo A. Baeza-Yates. Resin: a combination of results caching and index pruning for high-performance web search engines. In *SIGIR*, pages 131–138, 2008.

[26] Yinglian Xie and David R. O'Hallaron. Locality in search engine queries and its implications for caching. In *INFOCOM*, 2002.

[27] Wenjie Zhang, Xuemin Lin, Muhammad Aamir Cheema, Ying Zhang, and Wei Wang. Quantile-based KNN over multi-valued objects. In *ICDE*, pages 16–27, 2010.