Result Enrichment in Commerce Search using Browse Trails

Debmalya Panigrahi*
Computer Science and Artificial Intelligence Laboratory, MIT
debmalya@mit.edu

Sreenivas Gollapudi
Microsoft Search Labs
Microsoft Research Silicon Valley
sreenig@microsoft.com

ABSTRACT

Commerce search engines have become popular in recent years, as users increasingly search for (and buy) products on the web. In response to an user query, they surface links to products in their catalog (or index) that match the requirements specified in the query. Often, few or no product in the catalog matches the user query exactly, and the search engine is forced to return a set of products that *partially* match the query. This paper considers the problem of choosing a set of products in response to an user query, so as to ensure maximum user satisfaction. We call this the *result enrichment* problem in commerce search.

The challenge in result enrichment is two-fold: the search engine needs to estimate the extent to which a user genuinely cares about an attribute that she has specified in a query; then, it must display products in the catalog that match the user requirement on the *important* attributes, but have a *similar* but possibly non-identical value on the less important ones. To this end, we propose a technique for measuring the importance of individual attribute values and the similarity between different values of an attribute. A novelty of our approach is that we use entire browse trails, rather than just clickthrough rates, in this estimation algorithm. We develop a model for this problem, propose an algorithm to solve it, and support our theoretical findings via experiments conducted on actual user data.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]

General Terms

Algorithm, Performance

Keywords

Streaming Algorithms, Structured Search

*Work done while the author was an intern at Microsoft Search Labs.

WSDM'11, February 9–12, 2011, Hong Kong, China.

1. INTRODUCTION

In recent years, commerce search has become an integral aspect of search engines, as users spend increasing amounts of time searching for (and buying) products online. The quality of the search results surfaced by a search engine in response to a product query is critical to improving user satisfaction and enriching the search experience. In fact, the importance of effectively handling commercial queries can be inferred from the plethora of web portals that are dedicated to commerce search. Typically, these portals maintain a catalog (or index) of products, and surface a set of products from the catalog that best match the user query. In many ways, commerce search is different from traditional web search. Unlike in web search, the search results are a set of products, and not a set of web links. These products are stored in a well-structured catalog maintained by the commerce search engine, and boast rich metadata. For example, each product has a `category` (e.g., *laptop*, *digital camera*, etc) and within a category, product data is organized into multiple fields (e.g., `manufacturer`, `model`). The queries themselves are partially structured as well, in that a set of attribute values relevant to the product being searched for can typically be extracted from the query string. For example, if a user is looking for digital cameras, it is likely that her query will include (a subset of) values for attributes like manufacturer, model, color, resolution, zoom, etc. Further, the goal of commerce search is to help users make a transaction, which is different from the goals of web search. This necessitates the development of techniques dedicated to solving problems in commerce search. In this paper, we design such a technique for solving the problem of result *enrichment* (that we describe below) in commerce search.

Let us introduce our problem through an example. Suppose a user is searching for a `Pink Pentax Optio Camera`. It is quite possible that such a product is not available in the catalog. Suppose, instead the catalog contains a `Silver Pentax Optio Camera`, a `Pink Canon Powershot Camera`, a `Purple Pentax Optio Camera`, and a `Pink Vivitar ViviCam Camera`. In the absence of the exact product in the catalog (or if the number of products in the catalog that exactly match the query specifications is small), the search engine needs to surface a set of related products. This is called *result enrichment*. Typically, user data is used to solve this problem. Suppose that, in the example above, users who searched for pink Pentax cameras earlier often settled for a Pentax camera of a different color, but never for a camera of a different manufacturer. If the search engine has only two result slots, prior user behavior indicates that the two Pentax cameras in the catalog should be surfaced, rather than the Canon or Vivitar cameras. What if only a single result slot is available? Then, we might need to use additional information—say, users who searched for pink cameras in the past often bought purple cameras but rarely silver ones. In that

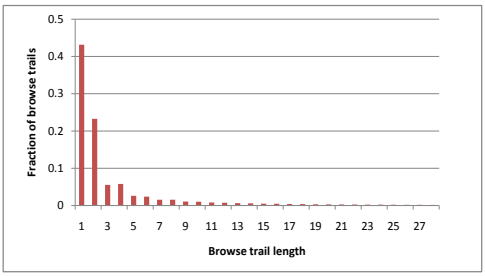

Figure 1: A histogram of length of browsetrails

case, the purple Pentax camera should be surfaced rather than the silver one.

To substantiate our point, we introduce the notions of *importance* of attribute values and *similarity* between them.

- The *importance* of an attribute value v is a measure of the likelihood of users who queried for products with attribute value v to actually end up buying products having attribute value v. In the previous example, Pentax has high importance.
- The *similarity* between attribute values v and v' is a measure of the likelihood of users who queried for products with attribute value v to actually end up buying a product having attribute value v'. In the previous example, purple is more similar to pink than silver.

These notions are central to what a commerce search engine does in response to a query; clearly, it will try to surface products that retain high importance attribute values in the query, while possibly replacing the low importance ones with other similar attribute values. In this paper, we define the importance and similarity parameters of an attribute value, and then present a technique for estimating these parameters that can be used by commerce search engines for the purposes of result enrichment.

Some of the above aspects have been studied in the context of general web search. Query and document classification has been studied to taxonomize information in web search [14, 25, 4]. Studies have also looked at entity extraction from queries in order to improve classification [17], query alterations [10], and extracting structured information from a query [21]. The problems of computing proximity in terms of related queries and similarity between documents has also been well-studied in the literature [16, 3, 26, 8, 2]. Further to these studies, there has been a considerable amount of research directed toward incorporating user feedback into related problems [7, 1]. An important outcome of this area of research is that clicks offer one of the most important signals of user feedback. However, often the first click is not representative of user behavior in our context. For example, a user looking for a certain product on the web might choose to click on a product page and then decide to look for another product, often following links on the product page itself, e.g. recommendations. In fact, the toolbar browse logs of a commercial search engine confirm our hypothesis that the user spends a considerable amount of time beyond the first click browsing the web. Figure 1 shows the distribution of the length of *browse trails* in a browsing session of a user over a six-month period. We define a browse trail to begin with a user query to a search engine followed by a sequence of web pages that are visited by the user. Query reformulation or a new query to the search engine marks the beginning of a new browse trail.

Browse trails have many advantages over clicks. One of the major disadvantages of using first click data is that it is extremely sparse since most web domains do not receive a statistically significant number of first clicks. For example, web domains that are important for specific attribute values that are infrequent in queries often have comparatively low page rank values, and therefore rarely appear in the list of urls displayed by a search engine. As a result, these domains do not have significant presence in first click data. However, users often navigate to such domains in the course of a long browsing session if they are interested in the attribute values for which these domains are relevant. Therefore, such domains appear more frequently in browse trails data. Another major problem with clickthrough data is that it usually has a large fraction of noisy clicks, i.e. urls that the user clicked on simply because they showed up in the list returned by a search engine in response to her query, though the domains are not actually relevant to her query. To alleviate this problem of noisy clicks, researchers have proposed various techniques, e.g. use the *dwell time* of users on these pages to separate between spurious and genuine clicks. Using browse trails instead of first clicks automatically solves this problem since genuine clicks usually lead to longer browse trails and therefore contribute significantly more to the browse trail data corpus than short spurious browse trails.

Our Contributions. In this study, we propose to use the browse trails of users performing web search for products to estimate similarity and importance parameters of attribute values, and thereby aid in enriching the results returned for a query submitted to a commerce search engine. It is important to note that while the browsing data we use comes from commercial search queries in general web search engines, we aim to apply the user behavior knowledge gained from this data in commerce search engines. We assume that queries are annotated with `<attribute, attribute value>` pairs (e.g., by [24]). Our main contribution is *a technique that uses these annotated queries and their corresponding browse trails to estimate the importance and similarity of attribute values*. As a key step in these computations, we propose *an efficient streaming algorithm that computes a list of important attribute values for every domain visited in the browse trails.* Given that it can be implemented in the streaming model, our algorithm very easily scales to large data sets such as browse trails from toolbar logs. Another key feature of our algorithm is that it does not work on the content of a page, thereby avoiding the common pitfalls of data extraction on the web (crawling issues, handling multiple languages) and performance bottlenecks that come with it. Finally, *we perform extensive experiments on a large corpus of browse data, and the experimental results confirm that our technique and algorithms work well in practice.*

2. MODEL AND PROBLEM DEFINITION

We will now set up the model for interpreting the browse trail data that we obtain from web search engines. This model will lead us to the definition of similarity and importance parameters of attribute values; our concrete problem will then be to estimate these parameters for the various attribute values. We will show that the key step in this estimation involves annotating each web domain with a list of attribute values that are related to it. For example, a website that only sells `Sony` cameras should be annotated only with the keyword `Sony`, while one that contains reviews of various cameras of different brands should be annotated with a list of multiple manufacturers. Our next step is to give an algorithm for such annotation; however, along the way, we will need to strengthen our model with a set of well-motivated and practical assumptions. We will return to these assumptions in section 6 when we validate them via experimental data. Finally, we will show that we can use the annotated list of web domains produced by the above algorithm and

Query	black dell laptop
Urls	http://www.bizrate.com/laptop-computers/dell/
	http://www.bizrate.com/laptop-computers/dell-c610-laptop/
	http://www.amazon.com/s/?ie=UTF8&keywords=d610...

Figure 2: A typical browse trail

our original browse trail data to compute similarity and importance parameters of attribute values.

From this point onward, we will focus on attribute values corresponding to a single attribute. The algorithms that we are going to describe have to be run for each attribute separately to obtain similarity and importance estimates. We will refer to attribute values (for the attribute that we have fixed) that appear in at least one query in our browse trail data as *keywords*. Let P denote the set of web domains and Q the set of keywords. Let m and n respectively denote the size of the sets P and Q. Typically, each domain contains relevant information for a set of keywords. However, for simplicity of presentation, we will assume, for the time being, that there is a single keyword associated with each domain. We will show later that our techniques extend to the more realistic scenario of domains being relevant for multiple keywords. The mapping from a domain to its relevant keyword is given by an unknown function $I : P \to Q$. Function I produces a partition of P into sets of domains P_q, where $P_q = I^{-1}(q)$ is the set of domains relevant to keyword q. The number of domains in P_q is denoted by m_q.

The input to the algorithm is a set of browse trails, each corresponding to a particular query. Queries not containing any keyword in Q do not contain any information about the attribute we are interested in—hence such queries are discarded. For example, if we are interested in the `manufacturer` attribute of queries for digital cameras, then a query `Red 10 MegaPixel Camera` is useless for our purpose. Further, queries containing multiple keywords in Q are also discarded since they are ambiguous. We are therefore left with a set of browse trails corresponding to queries containing exactly one keyword in Q. A browse trail comprises a sequence of urls, and each url can be mapped to a particular web domain. Figure 2 depicts a typical browse trail. We convert such a browse trail into a set of (keyword, domain) pairs, where each pair contains the keyword in the query and the domain corresponding to a particular url in the browse trail. The number of pairs, therefore, is equal to the length of the browse trail. However, all domains in a browse trail are not likely to be equally relevant to the original query. For example, web domains visited by a user towards the end of a long browse trail are unlikely to be relevant to her original query. Therefore, each (keyword, domain) pair is associated with a weight, where the weight of a particular pair is a ρ-fraction of the weight of the previous pair on the browse trail. The parameter ρ is a fixed constant less than 1 (typically 0.9 in our experiments). Thus, the pair containing the first domain in the browse trail has weight 1, the second has weight ρ, the third has weight ρ^2, and so on. Note that a very small value of ρ is undesirable since it would imply that we are practically ignoring the entire browse trail other than the first click. As an example, consider the browse trail in Figure 2. For the attribute `Manufacturer`, we get two copies of $(dell, bizrate)$ (with weights 1 and ρ) and one $(dell, amazon)$ (with weight ρ^2) from this browse trail.

Let us now collect all the pairs formed by all the browse trails in the input data. The *frequency* of a particular (keyword, domain) pair is the sum of weights of all its appearances in the input data. We denote this frequency by the function $f : Q \times P \to \mathbb{R}_0^+$.[1] We will often use the shorthand f_{qp} for $f(q,p)$. The marginal frequencies of keywords is denoted by n_q, i.e. $n_q = \sum_{p \in P} f_{qp}$. Also, let

$$F = \sum_{q \in Q} n_q = \sum_{q \in Q} \sum_{p \in P} f_{qp}.$$

Similarity and Importance. Recall that the similarity of two attribute values (q, q') is the likelihood that a user searching for a product with q settles for one with q'. It is important to note that similarity is not symmetric, i.e. the similarity of (q, q') might be very different from that of (q', q). For example, a user searching for a `magenta` camera might settle for a `red` one, but not vice-versa. The importance of an attribute value q is therefore its similarity with itself, i.e. the likelihood that a user searching for a product with q settles for one with q itself and does not opt for one with a different attribute value.

In defining similarity in our model, we assert that if a large fraction of queries containing q visit domains in $P_{q'}$ (i.e. domains relevant to q') early on in their browse trails, then the similarity of the (q, q') pair is high; else, their similarity value is low. Formally, similarity is defined as the *relative frequency* of queries containing keyword q visiting domains relevant to keyword q' (where q' may or may not be q itself), and is denoted by

$$s_{qq'} = \frac{\sum_{p \in P_{q'}} f_{qp}}{n_q}.$$

Then, the importance of keyword q is its similarity with itself, i.e. s_{qq}.

We will now focus on presenting a technique for estimating $s_{qq'}$. The definition above implies that this requires finding n_q and f_{qp} for all $p \in P_{q'}$. Of these, n_q is available from the input. However, though we know the values of f_{qp} for each domain p, we do not know which domains are in $P_{q'}$. This constitutes our primary algorithmic challenge: *annotate each domain p with the keyword it is relevant to, i.e. $I(p)$.*

2.1 Assumptions

It turns out that it is impossible to achieve our algorithmic goal, that of inferring the function I, without additional assumptions (proof omitted due to lack of space).

LEMMA 1. *Without additional assumptions, the function I cannot be inferred completely.*

Our aim, in this section, will be to come up with a set of well-motivated assumptions that make our goal feasible, while retaining practicality of our model.

Positive Bias. In general, it is unlikely that queries containing a particular keyword q visit domains containing information about another keyword $q'(\neq q)$ more frequently than domains containing information about q. Our first assumption formalizes this: *a browse trail for a query containing keyword q is more likely to visit domains in P_q than domains in $P_{q'}$, where $q' \neq q$.* Mathematically, this is expressed as

$$s_{qq} \geq s_{qq'}, \ \forall q, q' \in Q, q \neq q'. \tag{1}$$

We call this the *positive bias* property. In section 6, we will empirically demonstrate that this property is typically true.

Uniformity. For statistically significant data, it is unlikely that different domains in P_q (for some keyword q) will display significantly different distribution of queries visiting them. We formalize this as the (α, δ)-*uniformity* property: *different domains in P_q (where $q \in Q$) have similar distribution of queries visiting them, up to fac-*

[1] $\mathbb{R}_0^+$ is the set of non-negative reals.

tors of α and δ. Mathematically,

$$\alpha\left(\frac{s_{qq'}n_q}{m_{q'}}\right) \leq f_{qp} \leq \delta\left(\frac{s_{qq'}n_q}{m_{q'}}\right),\ \forall p \in P_{q'},$$

where $\alpha \leq 1$ and $\delta \geq 1$. Typically, we will assume that the parameters α and δ are close to 1. As earlier, we will validate this assumption further via experiments in section 6.

Proportionality. In practice, keywords that appear frequently in queries also appear frequently in web domains, and vice-versa. Thus, there is a proportionality between the relative frequency of a keyword in queries, and the fraction of domains for which it is relevant. We formalize this as the (β,γ)-*proportionality* property, which states that *for any keyword q, its relative frequency in queries and domains are similar, up to factors of β and γ*. Mathematically,

$$\beta\left(\frac{m_q}{m}\right) \leq \frac{n_q}{F} \leq \gamma\left(\frac{m_q}{m}\right),$$

where $\beta \leq 1$ and $\gamma \geq 1$. As with the other assumptions, we will provide experimental evidence to support our conjecture that the proportionality property approximately holds in typical situations.

Unfortunately, we show below that in spite of the above assumptions, the problem of learning the mapping I continues to be unsolvable.

THEOREM 1. *For any $\alpha < 1$, $\beta < 1$ and $\gamma > 1$, there exists a $\delta > 1$ such that the function I cannot be inferred completely, even if the positive bias, (α,δ)-uniformity and (β,γ)-proportionality properties hold.*

PROOF. We create two different functions I having identical browse trails. Let $Q = \{q_1, q_2\}$ and $P = \{p_1, p_2, \ldots, p_{2k}\}$; we will determine the value of k later. Instead of describing browse trails, let us define the frequency distribution f; any set of browse trails that realizes this distribution serves our purpose.

- $f_{q_1p_i} = f_{q_2p_j} = xa,\ \forall 1 \leq i \leq k, k+1 \leq j \leq 2k$, and
- $f_{q_2p_i} = f_{q_1p_j} = a,\ \forall 1 \leq i \leq k, k+1 \leq j \leq 2k$,

where a, x and k are determined as follows. First, let us ensure that the following two scenarios both satisfy the (α,δ)-uniformity and positive bias properties:

- $P_{q_1} = \{P_1, P_2, \ldots, P_k\}$; $P_{q_2} = \{P_{k+1}, P_{k+2}, \ldots, P_{2k}\}$, and
- $P_{q_1} = \{P_1, P_2, \ldots, P_{k-1}\}$; $P_{q_2} = \{P_k, P_{k+1}, \ldots, P_{2k}\}$.

In the first situation, both properties are trivially true for any values of α and δ. So, we focus on the second scenario. First, consider the positive bias property. Clearly, it holds for q_2. For q_1, we need

$$\begin{aligned}(k-1)xa &> ka + xa \\ \Rightarrow x &> \frac{k}{k-2}. \end{aligned} \quad (2)$$

Now, consider the (α,δ)-uniformity property. This clearly holds over P_{q_1} for both q_1 and q_2. For this property to hold over the set P_{q_2} for q_1, we need

$$\alpha\left(\frac{ka+xa}{k+1}\right) \leq a < ax \leq \delta\left(\frac{ka+xa}{k+1}\right),$$

and for q_2, we need

$$\alpha\left(\frac{kxa+a}{k+1}\right) \leq a < ax \leq \delta\left(\frac{kxa+a}{k+1}\right).$$

Since

$$(kxa+a) - (ka+xa) = (k-1)(x-1)a > 0,$$

we need to satisfy only the following conditions:

$$\alpha\left(\frac{kxa+a}{k+1}\right) \leq a < ax \leq \delta\left(\frac{ka+xa}{k+1}\right),$$

$$\text{i.e.,}\quad \alpha(kx+1) \leq k+1 < (k+1)x \leq \delta(k+x). \quad (3)$$

Given values of k and x, there always exist values of δ that satisfy

$$(k+1)x \leq \delta(k+x)$$

since δ is not bounded above. So, it is sufficient to ensure that

$$\frac{k}{k-2} < x \leq \frac{k+1-\alpha}{k\alpha}. \quad (4)$$

For Eq. 4 to hold, we need

$$(1-\alpha)k^2 - (1+\alpha)k - 2(1-\alpha) > 0.$$

For $0 < \alpha < 1$ and $k > 0$, this is ensured if

$$k > \frac{(1+\alpha) + \sqrt{(1+\alpha)^2 + 8(1-\alpha)^2}}{2(1-\alpha)}.$$

k can be made large enough to ensure this. Then, we can choose any x satisfying Eqn. 4. Finally, we choose a large enough a to ensure that xa is an integer.

Now, the (β,γ)-proportionality property holds if

$$\beta\left(\frac{k+1}{2k}\right) \leq \frac{1}{2} \leq \gamma\left(\frac{k-1}{2k}\right),$$

$$\text{i.e.,}\quad \beta \leq \frac{k}{k+1} \leq \frac{k}{k-1}\gamma.$$

Since k is not bounded above in satisfying the positive bias and (α,δ)-uniformity properties, we can choose k to be large enough to satisfy the (β,γ)-proportionality property as well. □

2.2 The List Annotation Problem

Recall that we started out to estimate similarity and importance parameters of attribute values, identified that the key intermediate step was to annotate web domains with relevant attribute values and then saw that this problem remains unsolvable even after strengthening our model with three well-motivated assumptions. To make the annotation problem solvable, we now slightly relax our requirement. Instead of annotating each domain with a single keyword, we now allow a domain to be annotated by a *small* list of keywords as long as it satisfies the following two requirements.

Completeness. Ideally, the list L_p for domain p must contain the *correct* annotation $I(p)$. In most situations however, we are interested only in *important* keywords q, i.e. those that have $s_{qq} > \theta$ for some threshold θ. Thus, our *completeness requirement* asserts that if $q = I(p)$ and $s_{qq} > \theta$ for some threshold θ, then q must be in the list L_p that domain p is annotated with.

Soundness. In practice, if $q = I(p)$ and $s_{qq} > \theta$, we would not only want q to be in L_p, but would also like to ensure that all other keywords q' in L_p are also relevant to domain p, in that queries containing q' frequently visit pages in P_q. Mathematically, for each $q' \in L_p$, we need that $s_{q'q} > T$, for some threshold T. We call this the *soundness requirement*. Clearly, $T \leq \theta$; the larger the value of T attained by an algorithm, the better it is.

For practicality, it is also critical that the lists be kept as small as possible. We can show a lower bound of $1/\theta$ on the *average size* of the lists (proof omitted due to lack of space), and therefore we aim for this size bound in our algorithm. In summary, given a parameter

$\theta \leq 1$, we want to construct lists of average size $1/\theta$ for each domain p, which satisfy the completeness requirement with parameter θ, and the soundness requirement with parameter $T < \theta$, where T is as close to θ as possible. (We will assume that the positive bias, (α,δ)-uniformity and (β,γ)-proportionality properties hold, for some parameters $\alpha,\beta \leq 1$ and $\gamma,\delta \geq 1$.)

3. ALGORITHM

We will now present an algorithm for computing similarity and importance parameters of attribute values. As discussed previously, the principal algorithmic challenge lies in solving the list annotation problem described above. Therefore, we first present two algorithms for constructing the lists required by the list annotation problem. The first algorithm produces a list L_p^1 for each domain p that satisfies the completeness requirement. It has the additional property that each list *on an average* contains at most $\frac{\gamma}{\alpha\beta\theta}$ ($\simeq 1/\theta$ since $\alpha,\beta,\gamma \simeq 1$) keywords. The second algorithm produces a list L_p^2 for each domain p that satisfies both the completeness requirement and the soundness requirement with parameter $T = \left(\frac{\alpha\beta\theta}{\gamma\delta}\right)^2$ ($\simeq \theta^2$ since $\alpha,\beta,\gamma,\delta \simeq 1$). The final list L_p for each domain p is computed as the intersection of the two lists, L_p^1 and L_p^2. Clearly, this list satisfies the completeness requirement, the soundness requirement (due to L_p^2) and also the size bound (due to L_p^1). In describing these algorithms, we will assume that the values of α,β,γ and δ are known. Typical values of these parameters can be estimated using training data. We will also assume that the input is presented in the form of the frequency distribution f, which is easily computable from the browse trails data. In some practical situations however, the browse trails data might be very large, and therefore exact computation of f might be too expensive. We will show later how to adapt our algorithm to the streaming data model, where the browse trails appear as elements in a data stream.

Construction of L_p^1. This algorithm is particularly simple; we compute $g_{qp} = f_{qp}\left(\frac{m}{F}\right)$ for each keyword q and domain p, and add q to L_p iff $g_{qp} > \alpha\beta\theta$.

THEOREM 2. *The lists L_p^1 satisfy the completeness requirement.*

PROOF. For any domain p, let $q = I(p)$ and $s_{qq} > \theta$. Then,

$$\begin{aligned} f_{qp} &\geq \alpha\left(\frac{s_{qq}n_q}{m_q}\right) \\ \Rightarrow f_{qp} &\geq \alpha\beta s_{qq}\left(\frac{F}{m}\right) \\ \Rightarrow f_{qp}\left(\frac{m}{F}\right) &\geq \alpha\beta s_{qq} \\ \Rightarrow g_{qp}\left(\frac{m}{F}\right) &> \alpha\beta\theta. \end{aligned}$$

The first equation follows from the (α,δ)-uniformity property, the second step from the (β,γ)-proportionality property and the final step from the fact that $s_{qq} > \theta$. □

As promised, we now prove a bound on the size of the lists L_p^1.

THEOREM 3. *The* average size *of a list L_p^1 is at most $\frac{\gamma}{\alpha\beta\theta}$.*

PROOF. For any keyword q,

$$\sum_{p\in P} g_{qp} = \left(\frac{m}{F}\right)\sum_{p\in P} f_{qp} = \left(\frac{m}{F}\right)n_q \leq \gamma m_q,$$

where the last inequality uses the (β,γ)-proportionality property. Thus, a keyword q can feature in at most $\frac{\gamma m_q}{\alpha\beta\theta}$ lists, and the average length of a list is at most

$$\left(\frac{1}{m}\right)\sum_{q\in Q}\frac{\gamma m_q}{\alpha\beta\theta} = \frac{\gamma}{\alpha\beta\theta}. \quad \square$$

Construction of L_p^2. For each keyword q and domain p, we first compute $h_{qp} = \frac{g_{qp}}{n_q}$. Let $H = \max_{q\in Q} h_{qp}$. For each domain p, we compute $h'_{qp} = \frac{h_{qp}}{H}$ and add q to the list L_p^2 iff $h'_{qp} \geq \frac{\alpha\beta\theta}{\gamma\delta}$.

THEOREM 4. *The lists L_p^2 satisfy the completeness requirement.*

PROOF. For any domain p, let $q = I(p)$ and $s_{qq} > \theta$. Now, for any keyword q',

$$h_{q'p} = \frac{g_{q'p}}{n_{q'}} = \frac{f_{q'p}m}{n_{q'}F}.$$

By the (α,δ)-uniformity property,

$$\alpha\left(\frac{s_{q'q}m}{m_qF}\right) \leq h_{q'p} \leq \delta\left(\frac{s_{q'q}m}{m_qF}\right).$$

By the (β,γ)-proportionality property,

$$\alpha\beta\left(\frac{s_{q'q}}{n_q}\right) \leq h_{q'p} \leq \gamma\delta\left(\frac{s_{q'q}}{n_q}\right). \tag{5}$$

Since $s_{qq} \geq \theta$, we have $h_{qp} \geq \frac{\alpha\beta\theta}{n_q}$; and since $s_{q'q} \leq 1$ for any keyword q', $H = \max_{q'} h_{q'p} \leq \gamma\delta\left(\frac{1}{n_q}\right)$. Hence, $h'_{qp} = \frac{h_{qp}}{H} \geq \frac{\alpha\beta\theta}{\gamma\delta}$. □

We now prove that the lists L_p^2 satisfy the soundness requirement.

THEOREM 5. *If $q' \in L_p^2$ and $I(p) = q$, then $s_{q'q} \geq \left(\frac{\alpha\beta\theta}{\gamma\delta}\right)^2$.*

PROOF. Since $s_{qq} \geq \theta$, we have $H \geq \alpha\beta\left(\frac{\theta}{n_q}\right)$. From Eqn. 5, it follows that $h'_{q'p} \leq \left(\frac{\gamma\delta}{\alpha\beta\theta}\right)s_{q'q}$. Since $q' \in L_p^2$, we have $h'_{q'p} \geq \frac{\alpha\beta\theta}{\gamma\delta}$; hence, $s_{q'q} \geq \left(\frac{\alpha\beta\theta}{\gamma\delta}\right)^2$. □

Algorithm for computing importance and similarity values. Typically, $s_{qq} >> s_{q'q}$ for any keywords q and q' since browse trails for queries with keyword q most often visit domains relevant to q. The soundness and completeness properties of our algorithm then ensure that the lists L_p contain only $I(p)$. In fact, in our experiments, most domains had very short lists, and many of them had a single keyword. In this case, we annotate the domain with the keyword in the list, and compute importance and similarity values using this annotation of domains.

4. MIXED CONTENT OF DOMAINS

In practice, often a web domain has relevant content for multiple keywords pertaining to a single attribute. For example, a website offering reviews of digital cameras will typically not restrict itself to a single model line, or even a single manufacturer. Hence, our assumption that the function I maps each domain to a single keyword is over simplistic. In reality, $I(p)$ (henceforth, we use the shorthand I_p) is a set of weights over the set of keywords, where the weight of a keyword q represents the amount of content in domain p that relates to q. Mathematically, for each $p \in P$, $I_p : Q \to \mathbb{R}_0^+$ is a non-negative real-valued function. The relative amount of content in domain p that is relevant to keyword q is denoted by $r_p(q) = \frac{I_p(q)}{\sum_{q'\in Q} I_p(q')}$. We assume that $r_p(q')$ fraction of the

queries with keyword q that visit a domain p are searching for information on q' in p. Then, the similarity $s_{qq'}$ for a pair of keywords q, q' is redefined as

$$s_{qq'} = \frac{\sum_{p \in P} f_{qp} r_p(q')}{n_q}.$$

To interpret this definition, note that $f_{qp} r_p(q')$ represents the total weight of queries containing keyword q that visit domain p for information relevant to keyword q'.

Our goal now is to create a list L_p for each domain p that satisfies the following strong *correctness* requirement (this combines the completeness and soundness requirements): a keyword $q \in L_p$ iff $s_{qq} \geq \theta$ *and* $r_p(q) \geq \varepsilon$, for some parameter $\varepsilon > 0$. We also give strong guarantees on the size of the lists constructed—we show that each list L_p contains at most $\frac{1}{\varepsilon}$ keywords.

Assumptions. In this model, we assume a slightly stronger version of the positive bias property that a user visits a domain p on a query containing keyword q usually if she is interested in information about the keyword q itself. Mathematically, this implies that

$$f_{qp} \simeq \left(\frac{I_p(q)}{M_q} \right) s_{qq} n_q,$$

where $M_q = \sum_{p \in P} I_p(q)$. The uniformity property needs to be redefined in the new model. It now states that the relative frequency of any domain p is proportional to the amount of information it contains, i.e. $\frac{M_p}{M} = \frac{m_p}{F}$, where $M_p = \sum_{q \in Q} I_p(q)$ and $M = \sum_{p \in P} M_p (= \sum_{q \in Q} M_q)$. We also assume the proportionality property which states that $\frac{n_q}{F} = \frac{M_q}{M}$ for any keyword q.

Algorithm for computing L_p. We compute $g_{qp} = \frac{f_{qp}}{m_p}$ for each keyword q and domain p. We will show that we can compute s_{qq} for each keyword q using the values of g_{qp}. We now compute $h_{qp} = \frac{g_{qp}}{s_{qq}}$, and add a keyword q to list L_p iff $s_{qq} \geq \theta$ *and* $h_{qp} \geq \varepsilon$.

THEOREM 6. *The algorithm satisfies the correctness requirement.*

PROOF. Using proportionality and the stronger positive bias property, we have

$$h_{qp} \simeq \frac{n_q I_p(q)}{M_q m_p} = r_p(q) \left(\frac{n_q M}{M_q F} \right) \left(\frac{M_p F}{M m_p} \right) = r_p(q). \quad \square$$

THEOREM 7. *The size of a list L_p produced by the above algorithm is at most $\frac{1}{\varepsilon}$.*

PROOF. For any domain p, $\sum_{q \in Q} h_{qp} = \sum_{q \in Q} r_{qp} = 1$. Thus, each list contains at most $\frac{1}{\varepsilon}$ keywords. $\square$

Algorithm for computing importance and similarity values. We first compute s_{qq} for all keywords q.

$$s_{qq} = \frac{\sum_{p \in P} f_{qp} r_p(q)}{n_q} = \frac{\sum_{p \in P} f_{qp} g_p(q)}{s_{qq} n_q} = \sqrt{\frac{\sum_{p \in P} g_{qp}^2 m_p}{n_q}}.$$

We will show that we can compute g_{qp} if it is large enough, even in a streaming model. It follows that we can compute s_{qq} for all keywords q, provided it is large enough. Now,

$$s_{qq'} = \frac{\sum_{p \in P} f_{qp} r_p(q')}{n_q} = \frac{\sum_{p \in P} g_{q'p} g_{qp} m_p}{s_{q'q'} n_q}.$$

Again, we will show that we can compute g_{qp} if it is large enough, and also that we can compute s_{qq}. It follows that we can compute $s_{qq'}$ for keyword pairs (q, q').

5. EXTENSION: STREAMING MODEL

In reality, often the size of the data is too large to allow computation of function f for each (keyword, domain) pair. Thus, we need to modify the list annotation algorithm so that it works in the data streaming model, where browse trails appear sequentially in a stream, and the overall space used by the algorithm is significantly less than the total space required to store function f (which is $\Theta(nm)$ ignoring numerical factors). We will describe the streaming version of the list annotation algorithm for domains with a single relevant keyword; similar extensions can be done if domains that mixed content as well.

The key observation is that the algorithm only uses (possibly scaled) frequency counts. Let us first consider the construction of the lists L_p^1. Recall that the algorithm seeks to populate list L_p^1 for a domain p with keywords q such that $f_{qp} \geq \frac{\alpha\beta\theta F}{m}$. For this purpose, we use a standard heavy hitters algorithm [23, 11, 20]. We maintain a list $\mathscr{L}$ of $\frac{m}{S\alpha\beta\theta}$ (keyword, domain) pairs (q, p) and their frequencies $f'(q, p)$, for some parameter $S < 1$. When a new (keyword, domain) pair (q, p) appears on the data stream, we update $\mathscr{L}$ as follows:

- if $(q, p) \in \mathscr{L}$, increase $f'(q, p)$ by 1,
- otherwise, if $\mathscr{L}$ is not full (i.e. contains $< \frac{m}{S\alpha\beta\theta}$ pairs), insert (q, p) into $\mathscr{L}$ and set $f'(q, p) = 1$,
- otherwise, decrease $f(q', p')$ for each pair $(q', p') \in \mathscr{L}$ by 1, and remove all pairs whose frequency becomes 0.

After all the browse trails have been considered, we remove all the (query, keyword) pairs (q, p) with $f'(q, p) < (1 - S)\left(\frac{\alpha\beta\theta F}{m}\right)$ from $\mathscr{L}$. L_p^1 for any particular domain p is now precisely the set of keywords q such that $(q, p) \in \mathscr{L}$.

The following theorem, which is standard in the streaming literature (for e.g., in [23, 11, 20]), asserts that the algorithm satisfies the completeness requirement.

THEOREM 8. *At the termination of the above algorithm, all (q, p) pairs with $f_{qp} \geq \frac{\alpha\beta\theta F}{m}$ are retained in $\mathscr{L}$, and have*

$$f'(q, p) \geq (1 - S)\left(\frac{\alpha\beta\theta F}{m}\right).$$

Extending the proof of Theorem 3, we also get a bound on the average size of the lists produced by this algorithm .

THEOREM 9. *The average size of a list L_p^1 produced by the above algorithm is at most* $\min\left(\frac{1}{S\alpha\beta\theta}, \frac{\gamma}{(1-S)\alpha\beta\theta}\right)$.

We now need to construct the lists L_p^2. We use the algorithm described in section 3, substituting $f'(q, p)$ for f_{qp}. We include a keyword q in L_p^2 iff

$$h'_{qp} \geq \frac{(1 - S)\alpha\beta\theta}{\gamma\delta}.$$

This algorithm continues to satisfy the completeness requirement (proof omitted due to lack of space). However, the parameter in the soundness requirement has to be modified to give the following theorem (proof omitted due to lack of space).

THEOREM 10. *If $q' \in L_p^2$ and $I(p) = q$, then*

$$s_{q'q} \geq \left(\frac{(1 - S)\alpha\beta\theta}{\gamma\delta}\right)^2.$$

6. EXPERIMENTS

In this section, we do an empirical evaluation of our model and algorithms described in the earlier sections. All our experiments are conducted on data extracted from Windows Live toolbar logs. So, we first describe the methodology for extracting browse trails from toolbar logs that we used in this work.

Data source and methodology. With functionality such as directly searching the web without accessing the web page of any search engine, web browser toolbars are finding increasing use in initiating searches. Thus, they are a rich source of useful signals of user behavior. We generate the user browse trails using the methodology described in White and Drucker [27], and Bilenko and White [5]. We summarize the method here for completeness. Typically, Windows Live toolbar logs comprise a sequence of records, each containing an anonymous session identifier, a timestamp, and the URL of the visited webpage. For each user, her interaction was extracted in the form of a browse trail (i.e. a sequence of visited URLs) with an associated session identifier. This identifier helps in dealing with concurrent writes to the backend toolbar log. Each trail originates with a query submission to a search engine, and contains the sequence of webpages visited by the user after receiving the results of her search query until a point of termination where it is assumed that the user's information need has been satisfied. We note that a session started from a new browser or a new browser tab has its own browse trail. Further, note that toolbars log the history of browsing behavior only for those users who consented to such logging.

Next, we describe our processing of user queries. Using a Naïve Bayes classifier, we first classify a user query into a product category. For example, this creates a set of queries all pertaining to digital cameras, or to laptops, and so on. The classifier was trained on around 2.0 million product and offer titles in 26 top-level categories (with 609 leaf-level categories). Our dataset provided us around 420 million browse trails for queries in these categories with an average length of 6.84 hops. Each leaf category is characterized by a set of attributes—e.g. `Manufacturer`, `Product Line`, `Resolution`, etc. for digital cameras—and each query contains particular values for a subset of these attributes. To extract these attribute values from a given query, we used the attribute extraction technique described in [24].

We perform two categories of experiments. The first category aims at verifying the correctness of the assumptions used in our model. The second category computes similarity and importance measures of attribute values using our algorithm, and evaluates the quality of the solution produced by comparing with a) *ground truth* similarity and importance values obtained from restricted pre-classified web domains; and b) corresponding values obtained from a user study.

6.1 Validating our assumptions

The Ground Truth Dataset. We need browsetrails involving already annotated pages to carry out our validation. To this end, we generated a set of browsetrails restricted to the domain `amazon.com`. The Amazon browsetrails begin with a search on `amazon.com` and contain all product pages visited by the user after issuing the query. As before, a new query marks the beginning of a new trail. The Amazon test data consists of 145,680 trails across the three top-level categories—`computing`, `digital cameras`, and `electronics`. We annotated each page with the product information from the page. For example, a page describing the `Nikon Coolpix L20 10MP Digital Camera (Red)` is annotated with the attribute values `Manufacturer = Nikon`, `Model line = Coolpix`, `Model = L20`, `Color = Red`, and `Resolution = 10`. We similarly annotated the query too. For every attribute value q, we used the query and product page annotations to compute the frequencies of the different attribute values q' on webpages visited by queries containing q. For example, if over all queries containing the attribute value `Canon`, the users visited *amazon* product pages containing some `Nikon` product 10,000 times, then the above frequency for the `(Canon, Nikon)` pair is 10,000. We will refer to this frequency data as TESTSET-I. We use the same `amazon` dataset as ground truth in all our experiments.

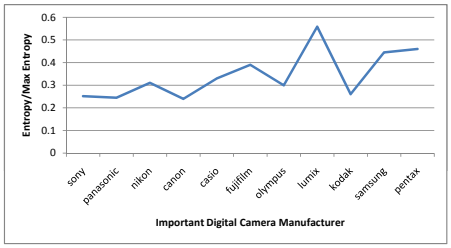

Figure 3: The entropy of the important attribute values in `digital cameras`. A small value of entropy validates the positive bias assumption.

Positive Bias. The *positive bias* assumption states that queries containing a particular attribute value visit pages annotated with that attribute value more frequently than pages annotated with other attribute values. From TESTSET-I, we computed the normalized frequency distribution of product attribute values for a given query attribute value. The entropy of this frequency distribution[2] is an indicator of its dispersion; therefore, a low entropy value would imply that these queries tend to mostly end up on pages having the same attribute value, thus validating our assumption. Figure 3 reports the entropy of this frequency distribution (scaled by the maximum possible entropy $\log n$, where n is the number of different attribute values) for different values of the attribute `Manufacturer` in the category `digital cameras`. This ratio is small for almost all the attribute values, i.e. most of the queries ended up on product pages relevant to their corresponding attribute values. We repeated this experiment for other categories as well and the results were similar. We will report the entire set of results in a full version of the paper.

Uniformity. The *uniformity* assumption states that queries containing a particular attribute value q visit different pages annotated with attribute value q' (q may or may not be equal to q') with roughly equal frequency. We compute the entropy of the frequency distribution of all visited product pages that contain keyword q'. The higher this value, the more uniform is the distribution. Table 1 reports the entropy of this frequency distribution (scaled by the maximum entropy $\log n$, as earlier) for several values of the attribute `Manufacturer` in the category `digital cameras`. We repeated the experiment for different values of the attribute `Manufacturer` in the other product categories and saw very similar results. Clearly, all the values are close to 1 suggesting that the (α, δ)-uniformity assumption holds with α and δ close to 1.

Proportionality. The *proportionality* assumption states that the relative frequency of an attribute value in queries is approximately equal to the relative frequency of that attribute value in product page annotations. Table 2 shows the frequency distribution of the different values of the `Manufacturer` attribute in the category `digital cameras` in queries and in product page annotations. As the table shows, the relative frequency of the attribute values in the

[2]Entropy of a discrete random variable X is $H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i)$.

manufacturer	sony	panasonic	nikon	casio	samsung	canon	fujifilm	olympus	kodak	pentax
sony	0.862	0.952	0.975	0.968	0.949	0.974	0.775	0.949	0.919	1.000
panasonic	0.970	0.885	0.922	1.000	0.970	0.947	0.920	0.931	1.000	1.000
nikon	0.911	0.946	0.921	0.775	0.961	0.985	0.932	0.963	1.000	1.000
casio	0.960	0.971	1.000	0.851	0.909	0.915	1.000	1.000	1.000	1.000
samsung	1.000	1.000	1.000	1.000	0.956	1.000	0.963	1.000	1.000	1.000
canon	0.885	0.953	0.960	0.921	0.885	0.855	0.929	0.883	1.000	1.000
fujifilm	0.833	1.000	0.811	1.000	1.000	0.975	0.897	1.000	1.000	1.000
olympus	1.000	0.570	0.960	1.000	1.000	0.915	1.000	0.922	0.845	1.000
kodak	1.000	1.000	1.000	1.000	1.000	0.960	1.000	0.919	0.882	1.000
pentax	1.000	0.919	1.000	1.000	1.000	0.970	1.000	1.000	1.000	0.883

Table 1: The ratio of the entropy to the maximum entropy for the important attribute values in `digital cameras`. A value close to 1.0 validates the uniformity assumption.

manufacturer	relative frequency in queries	relative frequency in products
olympus	0.076	0.062
casio	0.107	0.089
canon	0.242	0.224
fujifilm	0.031	0.039
samsung	0.024	0.037
pentax	0.020	0.013
nikon	0.069	0.064
kodak	0.049	0.038
panasonic	0.185	0.253
sony	0.197	0.179

Table 2: The relative frequency of manufacturer values (for `digital cameras`) in the queries and product pages.

queries is correlated to the relative frequency in the product pages. This implies that the (β, γ)-proportionality assumption holds with values of β and γ close to 1.

6.2 Similarity and Importance

In this set of experiments, we run our algorithm to compute the similarity and importance parameters using general browsetrails on the web (i.e. we do not restrict ourselves to `amazon` browsetrails any more). As stated earlier, these browsetrails are also generated from Windows Live toolbar data.

Using our algorithm for similarity computation, we computed a list of top-k ($k = 20$) *most similar* attribute values for each attribute value q. We also ranked these similar attribute values according to their similarity ($s_{qq'}$) to q. To verify the accuracy of the ordering, we compare it with the corresponding orderings obtained from two different sources—the ground truth set TESTSET-I and a user study. In both cases, we compute an ordered list of similar values for each attribute value q using techniques we describe later. Thus, we have two ranked lists (in each case)—one desired and the other computed by our algorithm. We measure the accuracy of the similarity computation as the distance between these two ranked lists. Given the ranked list of similar attribute values S_q computed by our algorithm, and the ideal ranked list of similar attribute values S'_q, we compute the distance as

$$R(q) = \sum_{s \in S_q \cap S'_q} \left| \frac{1}{r_{s_q}} - \frac{1}{r'_{s_q}} \right|$$

where r_{s_q} is the rank of attribute value s in S_q and r'_{s_q} is the rank of s in S'_q. We use a simple rank function which is the position of an attribute value in each list after removing the entries not common to both lists. We note that the function $R(q)$ penalizes mismatches in the higher ranked attribute values more than the lower ranked attribute values. A small value of $R(q)$ indicates two similar lists and hence a closer agreement of the list produced by our algorithm with the ideal list of similar values for a particular attribute value q. We normalized the distance values by the maximum distance (which comes from a completely inverted list).

category	attribute	avg normalized $R(q)$	
		$\rho = 1$	$\rho = 0$
Laptops	Manufacturer	0.102	0.204
Digital Camera	Manufacturer	0.126	0.167
	Product Line	0.045	0.111
Hard Drives	Manufacturer	0.098	0.227
LCD TVs	Manufacturer	0.072	0.147

Table 3: The frequency weighted normalized distance between the list of nearest neighbors produced by our algorithm and an ideal list of neighbors obtained from TESTSET-I for $rho = 0$ and $rho = 1$.

Using TESTSET-I. For TESTSET-I, we compute the desired ordering from the most frequent attribute values in the product pages visited for queries containing attribute value q. Table 3 shows the average normalized distance computed for selected attributes in each category. The results in Table 3 suggest there is a significant overlap in the list produced by our algorithm and the ground truth.

User study. We carried out a user study using the Amazon Mechanical Turk[3]. We used 120 randomly sampled queries consisting of ⟨`product, attribute, attribute value`⟩ triples along with a set of 10 similar attribute values for the chosen attribute value in the triple. These queries came from a diverse set of 21 leaf categories such as `sinks`, `mattresses`, `cooktops`, and `gardening tools` with each category contributing around 4 queries to the test set. Each query was presented to 11 human judges. Every judge was asked two questions—1) whether she would terminate her search if she did not find products with the attribute value specified in the query; and 2) if she decided not to terminate her search, then select *up to* 5 similar attribute values (from the provided 10) that she would look for in the search results. An overwhelming 98% of the judges answered in the negative to question 1—in other words users preferred to continue with their search and look for alternatives. This highlights the importance of result enrichment in product search since users are happy with substitutes in case the exact product they were looking for is not present in the product catalog/index. Next, we ordered the alternative attribute values provided to the user based on how many users selected the attribute value. We then computed the averaged normalized distance $R(q)$ between this ordering and the ordering produced by our algorithm for all the products in our test set. We averaged the normalized distance over all queries in a given category and report the averaged normalized $R(q)$ for each category in Table 4. The results show that the distance between the two orderings is indeed small and underscore the effectiveness of our algorithm in computing similarity scores.

[3] `https://www.mturk.com/`

category	attribute	avg normalized $R(q)$
desktop computers	manufacturer	0.045
cabinet & drawer hardware	hardware material	0.063
mattresses	manufacturer	0.060
mowers & tractors	manufacturer	0.064
door hardware & locks	hardware material	0.070
cooktops	manufacturer	0.075
rings	stone	0.082
pants	bottom style	0.055
laptop computers	color	0.057
gardening tools	manufacturer	0.070
sweaters	apparel material	0.064
home theater systems	manufacturer	0.068
amplifiers	manufacturer	0.054
action figures	character	0.071
generators	manufacturer	0.051
vehicle playsets	manufacturer	0.065
vacuums	manufacturer	0.063
printers	manufacturer	0.055
radio controlled toys	manufacturer	0.079
cell phones	product line	0.065
cell phones	manufacturer	0.063
shirts	apparel material	0.066

Table 4: The frequency weighted normalized distance between the list of nearest neighbors produced by our algorithm and an ideal list of neighbors obtained from the user study for $\rho = 1$.

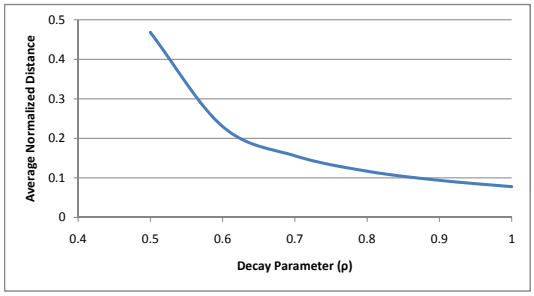

Figure 4: Effect of the parameter ρ on the average normalized distance for the sub-category `televisions`

Effect of the decay parameter ρ. To verify the significance of considering clicks beyond the first click, we compare the normalized distance values computed using the full browse trails ($\rho = 1$) with the first click case ($\rho = 0$). Table 3 shows significant improvement in the similarity values when using browse trails compared to using just the first click. Further, Figure 4 illustrates the decrease in the average normalized distance as the value of ρ increases.

Importance parameter. Though we concentrated on similarity scores so far, we also computed the importance parameter of attribute values as s_{qq}. Table 5 shows the importance of top attribute values for selected attributes.

6.3 Multiple Attributes

Till now, we have computed similarity scores for values of individual attributes. However, a typical product category has multiple attributes; therefore, to make our techniques useful for result enrichment, we need to extend it to computing the similarity scores between two products in the same category, each having a set of values for different attributes. We propose two solutions for this problem. If the number of distinct combinations of attribute values is small, then we can consider the combination of these attribute as a single attribute. For example, in any product category, we can typically consider `(Manufacturer, Product Line)` as a single attribute since each product line is usually exclusive to a particular manufacturer. Thus, the total number of such pairs is the same as the total number of product lines. As an example, for the `digital camera` category, the product line `Powershot` is exclusive to the manufacturer `Canon` and therefore `Canon Powershot` can be considered to be a single attribute value. We ran our algorithm for finding similarity and importance scores for such combinations of attributes. Table 6 shows that the combination of `Manufacturer` and `Product Line` attributes for `digital cameras` and `laptops` does indeed produce a list of related products that overlaps significantly with the corresponding lists obtained from TESTSET-I. Also, Table 7 shows some anecdotal results for the same attribute combination and categories used in the experiment.

category	attributes	avg normalized $R(q)$
Laptops	Manufacturer & Product Line	0.0893
Digital Cameras	Manufacturer & Product Line	0.0975

Table 6: The frequency weighted normalized distance between the list of nearest neighbors produced by our algorithm and an ideal list of neighbors obtained from TESTSET-I for multiple attributes.

However, consider the attribute values `Manufacturer` and `Color` for any product category. These attribute values are not related; in fact, most manufacturers are likely to have products in most colors. In this case, combining these two attributes into one might create a prohibitively large number of attribute values, thus resulting in sparsity of data and also making the algorithm inefficient. In this case, we estimate the similarity of products using a weighted sum of the similarity of individual (or combinations of) attribute values. The weight of an attribute is usually the normalized average of the importance parameter of the different values of the attribute. This gives us a scalable technique for measuring the similarity between pairs of products, thereby offering a solution to the result enrichment problem.

7. RELATED WORK

There is prior work on incorporating browsing along with search to improve the relevance of search results [12, 5, 6]. Bilenko and White [5] show that user's post-search browsing activity is a strong signal for computing relevance of the visited pages. Specifically, they show empirically that features extracted from browse trails (e.g., dwell times and visitation count) improves the ranking of search results compared to alternatives like clickthrough logs. Bilenko et. al. [6] propose computing a user profile using the frequent domains visited by the user. Each domain in turn maintains a list of query terms issued by users to visit the domain. Our idea of computing a list of annotated query terms for each page (or domain) is similar to theirs. However, the model we use to compute the lists admits efficient streaming algorithms that can scale to large data sets. Moreover, the list of annotations are attribute specific and these lists are used to compute the similarity between attribute values and their relative importance.

There is a lot of research on finding related queries [3, 26, 8, 2, 13, 22] and query reformulation [15, 9, 19]. Jones et al. [19] propose a technique to compute query substitutions based on precomputed query and phrase similarity using statistical techniques on query logs. A large section of the work on this problem can be classified into two categories—those that exploit only the structure of the query-click graph [3, 8, 22] and others that consider query terms as well as the content of the clicked URLs [15, 2]. In both categories, the underlying model is the query-click bipartite graph and the relationship between queries is computed indirectly as a function of their shared associations with entities such as urls or some important features thereof (such as terms).

Laptops		Hard Drives	Kitchen Appliances		Beds	Action Figures
Manufacturer	ModelLine	Manufacturer	Manufacturer	Color	Bed Type	Character
sony (0.32)	thinkpad (0.38)	dell (0.23)	ge (0.23)	stainless steel (0.23)	Platform (0.23)	transformers (0.28)
toshiba (0.25)	vaio (0.31)	seagate (0.22)	lg (0.12)	black (0.06)	Bunk (0.19)	wwe (0.17)
hp (0.23)	macbook (0.27)	hitachi (0.19)	samsung (0.12)	white (0.04)	Teen (0.17)	godzilla (0.14)
acer (0.21)	toughbook (0.24)	toshiba (0.17)	sharp (0.12)	steel (0.02)	Toddler (0.11)	rescue heroes (0.11)
lenovo (0.19)	pavilion (0.22)	simpletech (0.16)	maytag (0.10)	silver (0.02)	Loft (0.11)	spawn (0.09)
apple (0.12)	satellite (0.19)	w. digital (0.15)	whirlpool (0.06)	gold (0.02)	Trundle (0.07)	star wars (0.05)
asus (0.10)	eee pc (0.14)	samsung (0.12)	panasonic (0.05)	orange (0.01)	Kids (0.04)	halo (0.04)
dell (0.07)	inspiron (0.11)	lenovo (0.06)	electrolux (0.04)	clean steel (0.01)	Sleigh (0.03)	superman (0.02)
samsung (0.05)	aspire (0.11)	ibm (0.04)	siemens (0.03)	graphite (0.01)	Canopy (0.03)	gundam (0.02)
ibm (0.04)	nc10 (0.09)	maxtor (0.03)	frigidaire (0.03)	blue (0.01)	Adjustable (0.01)	lord of the rings (0.01)

Table 5: Important attribute values along with their s_{qq} values.

Product Query	Related Products				
sony vaio	sony vaio(0.550)	apple macbook (0.050)	hp pavilion (0.034)	acer aspire (0.031)	asus eee pc (0.040)
lenovo thinkpad	lenovo thinkpad (0.157)	ibm thinkpad (0.130)	apple macbook (0.060)	dell inspiron (0.050)	asus eee pc (0.049)
pentax optio	panasonic lumix (0.219)	pentax optio (0.162)	canon powershot (0.151)	nikon coolpix (0.119)	kodak easyshare (0.079)
canon powershot	canon powershot (0.359)	panasonic lumix (0.160)	nikon coolpix (0.115)	kodak easyshare (0.088)	fujifilm finepix (0.044)

Table 7: Anecdotal examples of top-5 related products for queries in `laptops` and `digital cameras` using attribute values in `Manufacturer` and `Product Line`.

A related problem is that of associating important query terms to a web page. Again, some of the graph-theoretic techniques used for finding related queries can be employed here as well [8]. A recent work of Gupta, Bilenko, and Richardson [18] studied the problem of associating keywords to web pages in the context of online advertising. They propose a learner that can effectively suggest the best query terms that are most likely related to the advertiser's terms (and thus increase the likelihood of a click on the ad).

8. CONCLUSION

We proposed a technique for result enrichment in commerce search. In order to quantify the replaceability of products, we introduced the notions of similarity and importance of attribute values. We then designed techniques for estimating these parameters that leverage browse trails of users on the web graph. Finally, we verified the scalability and accuracy of these techniques via extensive experiments on very large real-life data sets.

9. REFERENCES

[1] Eugene Agichtein, Eric Brill, and Susan T. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, pages 19–26, 2006.

[2] Ricardo A. Baeza-Yates, Carlos A. Hurtado, and Marcelo Mendoza. Improving search engines by query clustering. *JASIST*, 58(12):1793–1804, 2007.

[3] Doug Beeferman and Adam Berger. Agglomerative clustering of a search engine query log. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 407–416, 2000.

[4] Paul N. Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *SIGIR*, pages 11–18, 2009.

[5] Mikhail Bilenko and Ryen W. White. Mining the search trails of surfing crowds: identifying relevant websites from user activity. In *WWW*, pages 51–60, 2008.

[6] Mikhail Bilenko, Ryen W. White, Matthew Richardson, and G. Craig Murray. Talking the talk vs. walking the walk: salience of information needs in querying vs. browsing. In *SIGIR*, pages 705–706, 2008.

[7] Chris Buckley, Gerard Salton, and James Allan. The effect of adding relevance information in a relevance feedback environment. In *SIGIR*, pages 292–300, 1994.

[8] Nick Craswell and Martin Szummer. Random walks on the click graph. In *SIGIR*, pages 239–246, 2007.

[9] Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Probabilistic query expansion using query logs. In *Proceedings of International World Wide Web Conferences (WWW)*, pages 325–332, 2002.

[10] Erika F. de Lima and Jan O. Pedersen. Phrase recognition and expansion for short, precision-biased queries based on a query log. In *SIGIR*, pages 145–152, 1999.

[11] Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *ESA*, pages 348–360, 2002.

[12] Doug Downey, Susan T. Dumais, and Eric Horvitz. Models of searching and browsing: Languages, studies, and application. In *IJCAI*, pages 2740–2747, 2007.

[13] Doug Downey, Susan T. Dumais, Daniel J. Liebling, and Eric Horvitz. Understanding the relationship between searchers' queries and information goals. In *CIKM*, pages 449–458, 2008.

[14] Susan T. Dumais, Edward Cutrell, and Hao Chen. Optimizing search by showing results in context. In *CHI*, pages 277–284, 2001.

[15] Efthimis Efthimiadis. Query expansion. *Annual Review of Information Systems and Technology (ARTIST)*, 31:121–187, 1996.

[16] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[17] Jiafeng Guo, Gu Xu, Xueqi Cheng, and Hang Li. Named entity recognition in query. In *SIGIR*, pages 267–274, 2009.

[18] Sonal Gupta, Mikhail Bilenko, and Matthew Richardson. Catching the drift: learning broad matches from clickthrough data. In *KDD*, pages 1165–1174, 2009.

[19] Rosie Jones, Benjamin Rey, Omid Madani, and Wiley Greiner. Generating query substitutions. In *WWW*, pages 387–396, 2006.

[20] Richard M. Karp, Scott Shenker, and Christos H. Papadimitriou. A simple algorithm for finding frequent elements in streams and bags. *TODS*, 28:51–55, 2003.

[21] Xiao Li, Ye-Yi Wang, and Alex Acero. Extracting structured information from user queries with semi-supervised conditional random fields. In *SIGIR*, pages 572–579, 2009.

[22] Qiaozhu Mei, Dengyong Zhou, and Kenneth Ward Church. Query suggestion using hitting time. In *CIKM*, pages 469–478, 2008.

[23] J. Misra and D. Gries. Finding repeated elements. *Science of Computer Programming*, 2:142Ú–152, 1982.

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

[25] Dou Shen, Rong Pan, Jian-Tao Sun, Jeffrey Junfeng Pan, Kangheng Wu, Jie Yin, and Qiang Yang. Query enrichment for web-query classification. *ACM Trans. Inf. Syst.*, 24(3):320–352, 2006.

[26] Ji-Rong Wen, Jian-Yun Nie, and Hong-Jiang Zhang. Clustering user queries of a search engine. In *Proceedings of International World Wide Web Conferences (WWW)*, pages 162–168, 2001.

[27] Ryen W. White and Steven M. Drucker. Investigating behavioral variability in web search. In *WWW*, pages 21–30, 2007.