Suggestion Set Utility Maximization Using Session Logs

Umut Ozertem
Yahoo! Labs
701 First Ave.
Sunnyvale CA 94089
umut@yahoo-inc.com

Emre Velipasaoglu
Yahoo! Labs
701 First Ave.
Sunnyvale CA 94089
emrev@yahoo-inc.com

Larry Lai
Yahoo! Labs
701 First Ave.
Sunnyvale CA 94089
larrylai@yahoo-inc.com

ABSTRACT

Assistance technology is undoubtedly one of the important elements in the commercial search engines, and routing the user towards the right direction throughout the search sessions is of great importance for providing a good search experience. Most search assistance methods in the literature that involve query generation, query expansion and other techniques consider each suggestion candidate individually, which implies an independence assumption. We challenge this independence assumption and give a method to maximize the utility of a given set of suggestions. For this, we will define a measure of conditional utility for query pairs using query-URL bipartite graphs based on the session logs (clicked and viewed URLs). Afterwards, we remove the redundant queries from the suggestion set using a greedy algorithm to be able to replace them with more useful ones. Both offline (based on user studies and session log analysis) and online (based on millions of user interactions) evaluations show that modeling the conditional utility and maximizing the utility of the set of queries (by eliminating redundant ones) significantly increases the effectiveness of the search assistance both for the presubmit and postsubmit modes.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval

General Terms

Experimentation, Algorithms

Keywords

Search Assistance, Diversification, User Browsing Models

1. INTRODUCTION

Search assistance modules are significant elements in commercial search engines. Two main flavors of the search assistance are the *presubmit* and the *postsubmit* modes, namely the assistance before and after the users submit their queries. Presubmit technology is based on target auto-completion, whereas in the postsubmit, suggestions might be specializations, generalizations or lateral moves. Specializations are suggestions that add terms to the original query such as "python tutorial","monthy python" for the query "python" and generalizations are queries that drop terms from the original query such as suggesting "sfo" for "sfo flight status". Lateral moves are suggestions to related entities or concepts that do not impose any lexical overlap in either direction, such as "canon 50d" to "nikon d90" or "walmart" to "target".

The dominant presubmit search assistance technology in the search industry is based on completing user's partially typed query (will be referred as the prefix hereafter) to help the users save time by not requiring them to manually type the whole query into the search box, and also help them phrase their query when they start typing but do not exactly know which query terms to use or how to spell them. The common solution to this problem is to suggest the most popular (hence, most frequent) queries that match the user's prefix. Presubmit search assistance is a very clear case where the intent of the user is not defined. Let alone the ambiguities in the search queries, in presubmit, the input from the user is not even a full query yet. In this application, the aim is to minimize the number of characters that the user types before she finds at least one useful suggestion. This understanding assumes that as long as the user does not find *at least one useful suggestion* in the suggestion set she will keep typing the query manually. Therefore, if the aim is to minimize the time/effort by the user before she finds a useful suggestion, one needs to maximize the utility of the suggestion set, by trading some popularity (analogous to the relevance in web search ranking) for some diversity.

In the postsubmit mode, after the user issues the query, the intent is much more clear. Therefore, postsubmit assistance technology includes more sophisticated techniques, but still the core techniques are some derivatives of the marginal query frequency that lexically matches the query and popular query reformulations -which handle each suggestion independently. Query reformulations use the probability distribution of possible next queries that the current query is formulated to; more formally $p(q_{next}|q_{current})$. Furthermore, to ensure that the next query is related to the current query, rather than having a high reformulation probability just because it has a high marginal probability, the pointwise mutual information $p(q_{next}, q_{current})/(p(q_{current}) \times p(q_{next}))$ can be used. Here $p(q_{next}, q_{current})$ denotes the probability that these two queries were issued by the same user within a short time frame. In addition to the above, there are also more sophisticated methods that use the geolocation and the search history of the user [18, 6] to provide a more contextualized and personally relevant information. Regardless of the underlying suggestion technology, if the approach is handling each suggestion candidate independently and sorting by some suggestion level score, there might be redundant suggestions in the suggestion sets in postsubmit.

Understanding the user intent is the key for designing an effective search assistance system and when such knowledge is lacking,

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

search engines need to diversify the query suggestions in order to improve the quality of search assistance. The need for diversifying the presented query suggestions to maximize the total utility of a given suggestion set finds a good analogy to the web search ranking problem. In web search ranking, since a particular query might have multiple intents, it is well-known that presenting the most relevant results on the top positions might be suboptimal, and one should trade-off some relevance in return for some diversity in the result set [7]. In this view, the goal is to maximize the probability that the user will find *at least one relevant document* in the result set, hence minimizing query abandonment. The need to do some trade-off between relevance and diversity can be regarded as finding a relevant but also diverse set of documents. It is not well-defined how to adjust the trade-off between these two somewhat conflicting criteria, and the ill-defined nature of the problem yields many different techniques and formulations [1, 8, 9, 26, 2, 20]. A good summary and comparison of the approaches and a unifying view is presented in [13]. Although not nearly as much in web ranking, diversity has also been studied in the context of query suggestions. There are two recent publications on this subject based on random walks over query-URL bipartite graph [17] and manifold ranking [27]. In the next section, we will describe these two recently published methods in more detail, and we will also provide experimental comparisons.

We give a model to maximize the utility of a given suggestion set based on a diversity definition that we derive from a intuitive user browsing model. Since both the hitting time method and the manifold ranking method require the input query, to our knowledge, our approach is the only query suggestion diversification method that is applicable to presubmit, where only the partially-typed query is available. We show that in postsubmit the proposed examination model based utility maximization works better than existing methods, especially for less frequent queries. Another contribution is a path-based evaluation metric for measuring the utility of postsubmit suggestions that we call path gain/loss (PGL). PGL aims to measure the quality of the postsubmit suggestions by the average time it takes the user to reach a destination URL.

This paper is structured as follows. In the next section, we present several related query expansion/generation/rewriting methods in the recent literature, and our suggestion utility model is given in Section 3. In Section 4, we provide the details of our experimental setup. We present results on both offline (based on user studies and analysis of past session logs) and online (based on user interactions in session logs of a major search engine) evaluations of the model in Section 5, and conclude the paper with a discussion and proposed future work in Section 6.

2. RELATED WORK

Our suggestion set utility model is based on defining the conditional utility between pairs of queries, which has similar characteristics to defining a query similarity. There are many methods in the literature that define similarities between queries, and also utilize these for various problems, such as query rewriting, query generation or query suggestions.

Cui et al. look at the co-occurrences of query terms and document terms of the clicked documents and some term clustering to find similar queries [11]. Jones et al. use independence hypothesis log-likelihood ratio to find popular reformulations and use these as candidate suggestions [16], and build a supervised model to build a scoring scheme for suggestions. Probably Baeza-Yates & Tiberi and Craswell & Szummer are the first to use a query-url biparite graph for finding semantic relations [4, 10]. In a more recent work, Mei et al. build a bipartite graph of query-URL pairs, and use the click frequencies as the edges [18]. Using this graph, they generate query suggestions by looking at the pairs of queries that lead to clicks on same set of URL's. They also suggest using the same conditioning on a particular user's -or a user group's- click data only to generate personalized query suggestions. Cao et. al. use the same query-URL clickthrough bipartite graph idea, but they compliment it with a concept-tree [6]. Sahami and Heilman take another approach by investigating the common terms in the top n retrieved documents, and they build a similarity function using tf-idf vectors of these terms [21]. A similar idea is presented by Baeza-Yates et al., where they do clustering based on the aggregation of the term-weight vectors of the clicked URL's [3].

Overall, similarity measures defined over a query-URL bipartite graph that has the clickthrough values on the edges, and cosine similarity or Jensen-Shannon (or Kullback-Leibler) divergence over a set of most common terms in the top retrieved URL's are the most commonly used methods to define query similarity. In our view, the utility of each query is conditioned on the results of the earlier (higher-ranked) suggestions -also the results of the original query in postsubmit mode. Note that all above work focuses on defining a similarity measure between queries and our work is different mostly because we do not model the similarity of two queries; our aim is to model the conditional utility of one query given the other one is already presented.

Recently, studying diversity in the context of query suggestions has also attained some interest [27, 17, 5]. In [5], although the authors do not specifically aim for diversifying the query suggestions, they define an intuitive measure -distinct URLs in the multiset- for measuring the diversity of the suggestion sets that produce. In [27], authors build a query nearest neighbor graph to obtain a query manifold, and use a manifold ranking approach to generate a diverse set of suggestions. Although this approach is theoretically appealing and works effectively for frequent queries, practical usefulness of this approach is rather limited since the relevance of the suggestions drop very significantly for infrequent queries, since the query manifold becomes very sparse. In [17], Ma et al. use random walks over query-URL bipartite graph and hitting time to get a diversified set of query suggestions, and this is similar to the hitting time based method presented earlier by Mei et al. [18].

To our knowledge, the manifold ranking idea by Zhu et al. [27] and the hitting time approach by Ma et al. [17] are the only papers in the literature that are aiming to introduce diversity into query suggestions. One common shortcoming of both these two methods is they define the objective as "*given a query, generate a diverse and relevant suggestion set*", hence neither one of them is -neither directly nor with trivial modifications- suitable for presubmit, where the query is yet to be defined. We define the problem as "*given a set of suggestions of size N, rerank the suggestions such that the overall utility in top K rank is maximized for all* $K \leq N$", which is suitable for presubmit as well as postsubmit.

3. SUGGESTION UTILITY MODEL

Our model is based on estimating the pairwise conditional utility on the returned and clicked URLs[1] for each query pair. We argue that a query suggestion should be presented only if it can lead the user to the URLs that are not reachable more easily via the already presented suggestions -as well as the URLs presented for the original query for the postsubmit mode. We define a measure of utility that a query suggestion provides, conditioned on the original query and higher ranked suggestions. Using this, we define a submodular objective function over the set of queries, and develop a greedy algorithm to maximize the utility of a given suggestion set.

[1] A presentation group of title-abstract-URL tuple is referred as the URL through-out.

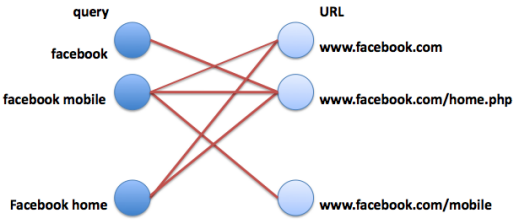

Figure 1: Query-URL bipartite graph

3.1 Query-URL Bipartite Graphs

We build the click bipartite graph of queries and clicked URLs, and the edges are the clickthrough values. Namely the weight on the edge that connects a particular URL u and a query q we have,

$$p(c(u) = 1|q) = N^u_{click}/N^u_{disp} \quad (1)$$

where $c(u)$ is a binary random variable, denoting whether the URL u is clicked or not, and N^u_{disp} and N^u_{click} are the number of times u is displayed and clicked for query q, respectively.

Similarly, we also build a view bipartite graph for queries and displayed URLs, where the edges are the average rank discount of the URL. In search, it is reasonable to assume that the importance of a document depends on its probability of being examined by the user. Therefore, Discounted Cumulative Gain (DCG) is among the most commonly used information retrieval metrics [14], which is given by

$$DCG_n = \sum_{r=1}^{N} \frac{rel_r}{\log_2(r+1)} \quad (2)$$

where r represents the rank of the results, and the typical values for N are 1,3,5. Note that the DCG formula can be considered as the inner product of two vectors, relevance vector $\mathbf{rel} = [rel_1, \ldots, rel_r]$ and discount vector $\mathbf{d} = [1, \ldots, 1/\log_2(r+1)]$, where the relevance vector represents the gain, and the discount vector represents the examination probabilities. In our view bipartite graph, the edges are discount values that DCG uses; however, since the results that the search engine change in time, if one looks into the query logs for a long period of time, one can see that a particular URL might have been returned at different ranks for a given query. Therefore here we use the average rank discount given by

$$E\{d(q,u)\} = \frac{1}{K} \sum_{i=1}^{K} \frac{1}{\log_2(r_i+1)} \quad (3)$$

where K is the number of times the URL u is returned for the query q in the session logs.

3.2 Pairwise Conditional Suggestion Utility

We assert that a query suggestion should be presented only if it leads to a sufficiently different result set as compared to those of the already presented suggestions and the original query. Therefore, we define $U(q_s|q_p)$, the utility of the suggestion q_s conditioned on q_p, a query that is already presented to the user. Simply, q_p is either the original query in the postsubmit mode or the partial query at the word boundaries in the presubmit mode or an already presented suggestion.

Intuitively, given two queries q_s and q_p, for each URL in the result set of q_s, we would like to check whether the same URL is in the result set of q_p; and if so, whether the URL is ranked as high as in q_s or not (so that the user is at least as likely to examine it). The simplest case is that the two queries q_s and q_p having exactly the same result sets. There are many such pairs of queries, mainly due to query rewriting widely used by search engines [24, 15, 22]. For example, for the prefix "bed", the two most frequent suggestions are "bed bath and beyond" and "bed bath & beyond", leading to *exactly* the same result set. If queries are sorted by popularity (for a given prefix), there are examples of such duplicate suggestions for even the simplest case of exactly same result sets.

Note that, in general using the query similarity of some type is not a good choice to model the utility of the suggestions, since it assumes $U(q_s|q_p) \doteq U(q_p|q_s)$ by definition, which is fundamentally incorrect. The relevant URLs in the result set of q_p might be a superset of those in q_s. A symmetric query similarity measure would not capture the required difference between a missing good result in the result set of q_s versus an additional good result in q_p, and the similarity drops in both cases.

Given q_s and q_p, let us define the result sets of these two queries as $URL_s = [u_{s1}, \ldots, u_{sN}]$ and $URL_p = [u_{p1}, \ldots, u_{pN}]$. For q_s, the click probabilities on the URL's are given as in (1). Here we don't consider any constraints in the rank of the URL's.

$$p(c(u_{si}) = 1|q_s) = N^{u_{si}}_{click}/N^{u_{si}}_{disp} \quad (4)$$

To make the model more stable for queries with very few clicks and pageviews, we introduce a prior on the click probabilities based on the average rank, which is available in the view bipartite graph.

$$p(c(u_{si}) = 1|q_s) = N^{u_{si}}_{click}/N^{u_{si}}_{disp} + \alpha\, E\{d(q_s, u_{si})\} \quad (5)$$

where α adjusts the strength of the prior distribution. Low values of α leads to a low precision model for queries with very few clicks, whereas a high α value leads to a too conservative model; hence a significant decrease in recall. We experimented with different values of α, and empirically found $\alpha = 1$ works best.

For each $u_{si} \in URL_s$, we define the examination probability that the user will examine the URL in the result page of q_p as follows,

$$p(e(u_{si}) = 1|q_p) = \begin{cases} 0 & : u_{si} \notin URL_p \\ 1 & : \begin{array}{c} u_{si} \in URL_p, \\ E\{d(q_p, u_{si})\} \geq E\{d(q_s, u_{si})\} \end{array} \\ \frac{E\{d(q_p,u_{si})\}}{E\{d(q_s,u_{si})\}} & : \begin{array}{c} u_{si} \in URL_p, \\ E\{d(q_p, u_{si})\} < E\{d(q_s, u_{si})\} \end{array} \end{cases} \quad (6)$$

where $e(u)$ is a binary random variable denoting whether u is examined or not. In words,

- u_{si} cannot be observed via q_p, if it is not in the result set of q_p, hence the examination probability is zero.
- If q_p returns u_{si} at least as high as q_s does, the examination probability is 1. The underlying assumption here is that a user that would examine the results of q_p at least as deep as she would go in the result set of q_s.
- If q_p returns u_{si} at a lower rank than q_s does, the examination probability of this URL is the ratio of the rank discounts of the two corresponding ranks. This ratio decreases when the rank difference increases, and for a given rank difference, it increases and gets close to 1 when both of the ranks are getting lower. With this, we want to capture two main effects presented in a comprehensive study on how people examine, recall, and reuse search results [23]:
  - The probability that people will recall a result decreases monotonically with the rank.
  - The probability that people will realize that a pair of documents were swapped increases with the difference in the rank, and decreases as the maximum of these two ranks are getting lower.

Combining (5) and (6), we define the conditional utility $U(q_s|q_p)$:

$$U(q_s|q_p) = 1 - \sum_{u \in URL_s} p(c(u) = 1|q_s)\, p(e(u) = 1|q_p) \quad (7)$$

where both terms in the summation can be obtained from the click and view bipartite graphs directly. Intuitively, the first term in the summation gives how important this particular URL is for the query q_s, and the second term gives how likely it is that the same user would examine this URL in q_p, with the assumption that the user would go as deep into the result set in q_p as well. Hence, $U(q_s|q_p)$ is, by definition 0 if the results of the two queries are exactly the same (since $p(e(u) = 1|q_p) = 1$ for all URLs, and $p(c(u) = 1|q_s)$ sums up to 1). Also, $U(q_s|q_p)$ would be close to 0 for queries that share many URLs and rank them similarly, and vice versa.

One missing piece in the utility definition is that it measures the amount of novelty that the suggestion may bring, however it does not model the relevance of the suggestion in any way. For example for two queries that are totally irrelevant, we have $U(q_s|q_p)$ values very close to 1. Same can be said about the diversity measures in web ranking; they are not useful by themselves, and have to be used along with some relevance criteria. Next section presents how we use this pairwise measure for a set of suggestions also taking their relevance into account.

3.3 Suggestion Set Utility Maximization

People with the same/similar intent do not phrase their queries in exactly the same way. Therefore, a search assistance system that is handling each suggestion candidate independently may perform poorly. For example, in the presubmit, the most popular queries that are matching the prefix "wal" include "wal mart", "wal mart stores", and "wal mart online", leading to almost identical result pages. Similarly, in the postsubmit, two most popular reformulations of the query "wal mart" includes "target" and "target stores". The missing piece is to model the additional utility of a suggestion, given the other ones that are already presented.

The pairwise conditional utility definition given in (7) can be regarded as the probability that q_s might satisfy the user given that she is not satisfied by q_p. We now state the problem of suggestion set utility maximization, as given an input (prefix or full query) maximizing the probability that an average user finds at least one useful suggestion within the suggestion set of size K.

Let sat be a binary random variable that denotes the probability that the user is satisfied, and S denote the suggestion set. For a particular user input q_{in}, given the satisfaction probabilities $p(sat = 1|s)$ for each suggestion $s \in S$, we would like to find the subset of suggestions $S' \subseteq S$ with $|S'| = K$ that maximizes the probability of satisfaction

$$p(sat = 1|S') = \sum_{s_i \in S'} p(sat = 1|s_i) \prod_{\substack{s_j \in S' \\ s_j \neq s_i}} U(s_i|s_j) \quad (8)$$

The satisfaction probability $p(sat = 1|s)$ depends on the application and might be popularity for presubmit, and reformulation probability for postsubmit. For brevity, we drop the conditioning on q_{in} since the formulation and all below derivation is given for a particular user input.

Intuitively, we would like to find a set of queries, with highest individual probability of satisfaction and all of which are sufficiently different from each other. One important observation is that the objective $p(sat = 1|S')$ is submodular, and a greedy algorithm that picks the query with highest marginal utility at each step can approximately solve the problem [19]. Submodularity is defined as follows,

Definition 1: Given a finite set N, a set function $f : 2^N \rightarrow \mathbb{R}$ is submodular iff for all sets S_1 and $S_2 \subseteq N$ such that $S_1 \subseteq S_2$, and $e \in N$ S_2, $f(S_1+e)-f(S_1) \geq f(S_2+e)-f(S_2)$.

In words, a function is submodular if it satisfies *the principle of diminishing returns*. In our case the utility of adding a suggestion into a larger set is no more than the utility of adding the same suggestion into a smaller set, because a larger set of suggestions has higher probability that more of the users have already been satisfied, and the added value of the same additional suggestion gets smaller as the size of the set increases. More formally, let $T_1 = S_1 \cup e$, and $T_2 = S_2 \cup e$. Following the notation in Definition 1, one can write

$$\begin{aligned} p(sat = 1|T_1) = & \sum_{s_i \in T_1} p(sat = 1|s_i) \prod_{\substack{s_j \in T_1 \\ s_j \neq s_i}} U(s_i|s_j) \\ = & \sum_{s_i \in S_1} p(sat = 1|s_i) \prod_{\substack{s_j \in T_1 \\ s_j \neq s_i}} U(s_i|s_j) \\ & + p(sat = 1|e) \prod_{\substack{s_j \in T_1 \\ s_j \neq e}} U(e|s_j)) \end{aligned} \quad (9)$$

Then, the additional utility becomes

$$p(sat = 1|T_1) - p(sat = 1|S_1) = p(sat = 1|e) \prod_{\substack{s_j \in T_1 \\ s_j \neq e}} U(e|s_j)) \quad (10)$$

Very similarly, one can write the same for the other sets, S_2 and T_2

$$p(sat = 1|T_2) - p(sat = 1|S_2) = p(sat = 1|e) \prod_{\substack{s_j \in T_2 \\ s_j \neq e}} U(e|s_j)) \quad (11)$$

Since we have $0 \geq U(e|s_j) \geq 1$, and $T_1 \subseteq T_2$, we can conclude

$$p(sat = 1|T_1) - p(sat = 1|S_1) \geq p(sat = 1|T_2) - p(sat = 1|S_2) \quad (12)$$

Hence, the objective function given in (8) is submodular. For a submodular function, Nemhauser et al. show that the error of a greedy algorithm that selects the maximum marginal utility at each step is bounded [19], and the constructed set satisfies

$$f(S') \geq (1 - 1/e) f(S^*) \quad (13)$$

where S^* is the optimal set and S' is the set selected by the greedy algorithm. We will design a such greedy algorithm to select S', however one thing to consider is that this submodular objective function is defined over the set, hence it does not take the ordering into account. This is due to the assumption that the users will examine all K suggestions, which in practice is not true; users have higher probability of examining the suggestions that are ranked higher. Therefore, our algorithm is also designed to give an ordered list rather than a set, which is of the following form;

- Given the user input (either a prefix or full query), get the set of suggestions S, ordered with descending $p(sat = 1|s)$.
- Place the query that has the highest $p(sat = 1|s)$ into the output set S'.
- For all remaining queries in the input set S, get the query with highest $p(sat = 1|s) \prod_{\substack{s_j \in S' \\ s_j \neq s_i}} U(s|s_j)$.

One constraint we have here is that the algorithm should not increase the latency significantly, and the bottleneck here is the utility function has to be stored for all query pairs. To increase the time efficiency of the algorithm we binarize the conditional utility, and use $U(s|s_j) < \gamma$ at the last step. For the optimal γ value that

we found with editorial tests (will be presented shortly), the utility function does fit into the memory, even for the web scale.

Another observation is that the above approach captures the utility optimization within the presented suggestions and rank them with optimal order, but one piece that is missing is to compare the suggestion candidates against the original query, to ensure that all presented suggestions satisfy $U(q_s|q_{in})$ as well. However, one thing that one should consider in the prefix phase is that we don't know q_{in} is a full query or not. For example, "faceb" is a very common misspelling of "facebook" and since the search engine corrects this misspelling, the result sets of these two queries are identical, therefore $U(\text{“}facebook''|\text{“}faceb'') = 0$. This suggests that "facebook" should not be displayed for the prefix "faceb", which is quite counter-intuitive. Therefore we examine the conditional utility with the user input (and remove the suggestion q_s, when $U(q_s|q_{in}) < \gamma$), only if the marginal query frequency of q_{in} is greater than q_s.

One final remark is, to preserve the optimal ordering of the output set S', one should account for the probability of satisfaction of the queries that are not included into the set. This is in fact one of the main differences of our greedy algorithm compared to the existing greedy algorithms proposed for result set diversification [1]. For a query, $p(sat = 1|s)$ should be its satisfaction probability plus the satisfaction probabilities of all of its near-duplicates (all suggestions that were not added to the output set S' since they failed to pass the conditional utility thresholding due to this particular query). For example, assume for the prefix "awk" the suggestions in the input set S are, "awkward", "awkward tv moments", "awk example", "awk examples" and "unix awk example", with decreasing $p(sat = 1|s)$. If "awk examples" and "unix awk example" are removed from the list since they are a duplicate of an earlier presented suggestion "awk example", then one should adjust $p(sat = 1|\text{“}awk\ example'')$ as $p(sat = 1|\text{“}awk\ example'') += p(sat = 1|\text{“}awk\ examples'') + p(sat = 1|\text{“}unix\ awk\ example'')$, so that after the duplicate removal the total satisfaction probability of this intent is properly represented, and this query has the chance to move upwards in the suggestion list.

Let us give a few definitions for formalizing the algorithm.

$f(q)$: The marginal query frequency

$S_{list}(q)$: A function which given an input query q, returns the set of query suggestions in descending probability of satisfaction.

$S_{dup}(q, \gamma)$: A function which given an input query and a threshold γ, returns the set of query suggestions with utility lower than the given threshold, $U(q_s|q_p) < \gamma$. We implement this as a look up table for a predetermined γ value, which yields a very fast computation time. Even for the web scale, the database required for this step fits into the memory for the optimal γ value.

S_{pre}: A function which given an input query q and the output set S', returns the set of queries within S', that q is a duplicate of. Simply, $S_{pre}(q)$ is the query (or queries) that caused q to be removed. In almost all cases this only returns one query, but in general, a new query might be a near-duplicate of more than one query in S'.

The pseudo-code of the set utility maximization algorithm is presented in Algorithm 1. After the initializations (lines 1-4), the first for-loop eliminates the duplicates of the user input (prefix or full query) by also considering the marginal frequency of the input (lines 5-9). Afterwards, the second for-loop does the greedy utility optimization within the suggestion set (lines 10-21). Every time a suggestion is decided to be added to the output set S', this query and all its duplicates are added to the set Dup, the set of queries that we remove the duplicates of which keeps growing as the for-loop proceeds (line 14). Every time a query is decided not to be added to the output set S', we move its satisfaction probability to the corresponding query (or queries) in the output set S' (lines 16-19), and once we reach the desired set size K, we sort results with decreasing probability of satisfaction to present them in the optimal order (line 22). At the end, all queries in S' are sufficiently different from each other (and sufficiently different from the user input q_{in} if q_{in} is a full query), and the queries in S' are sorted by descending satisfaction probability.

Algorithm 1 Greedy Suggestion Set Utility Maximization

```
1: I = S_list(q_in)
2: S' = {}
3: Dup = {}
4: Dup_q = S_dup(q_in, γ)
5: for q ∈ Dup_q do
6:    if f(q) < f(q_in) then
7:       Dup+ = q
8:    end if
9: end for
10: for q ∈ I do
11:    if q ∉ Dup then
12:       S' += q
13:       Dup_q = S_dup(q, γ)
14:       Dup+ = Dup_q
15:    else
16:       P = S_pre(q, S')
17:       for p ∈ P do
18:          p(sat = 1|p)+ = p(sat = 1|q)/length(P)
19:       end for
20:    end if
21: end for
22: Sort S' by descending p(sat = 1|s)
```

4. EXPERIMENTAL SETUP

Before going into the results that we obtained with our model, in this section we describe the session logs data that we used to build the bipartite graphs presented in Section 3.1, and the details of our offline and online test setup.

4.1 Model Training Data

We use 6-month of anonymized session logs of a major commercial search engine to build the view and click bipartite graphs. To make the data size tractable we ignored queries that appeared less than 3 times in this 6-month period, and bipartite graphs consist of roughly 119M unique queries and 658M unique URLs in total. The view graph is much denser than the click graph, since by definition the edges in the view graph are a superset of the edges in the click graph. After building the graphs we construct the non-symmetric conditional utility matrix, whose entries are $U(q_s|q_p)$, for all query pairs. Given a γ value, constructing the look-up table needed for S_{dup} is rather straightforward from this conditional utility matrix.

4.2 User Studies

In the user study, we provide a pair of queries to the judges and ask them to assess the utility of the second query suggestion, given that the first one is already presented. Note that small differences in queries that seem insignificant to people just by looking at the query strings can have very different intents and search results (for example, "office" vs "the office"). Therefore, in this task, we specifically ask the judges to examine the search results of each query before assessing the utility/redundancy. We also do a side-by-side test, where we present two sets of queries to the judges and ask them to assess which of the suggestion sets is better. All judges are professionals trained to evaluate search engines, and this test includes 12 individuals. The judges were asked to check the search result pages of each suggestion before making a decision according to the following guidelines in a 4-level grade scale.

3 - Useful: This suggestion introduces distinct, useful, unique information that is different from that of the presented suggestion.

2 - Somewhat useful: There is some content overlap with the presented suggestion, but there is enough of a difference that you do not consider the suggestions wholly redundant.

1 - Somewhat redundant: There is a minor difference in intent between the first suggestion and second, but they are mostly the same. Showing this suggestion might offer a somewhat different experience for the user, but the overlap between first and second suggestion is great and the second is not likely to be useful.

0 - Redundant: This suggestion definitely does not add any new information to presented suggestion. It would not be useful to show to any user because its intent/content is wholly redundant with the presented suggestion.

4.3 Session Log Analysis

In the session log analysis, we perform comparisons of suggestion relevance and diversity, with respect to the following evaluation metrics.

Relevance metric: In presubmit, the query is yet to be defined and relevance cannot be defined. Therefore, we use average query popularity -average marginal query frequency- instead of relevance. In postsubmit evaluation, we will use the reformulation probability $p(q_{next}|q_{current})$ estimated from the query logs and averaged over the suggestions in the set, as the relevance measurement. Reformulation probability is defined as follows, given the user issued $q_{current}$ the probability that the same user issues q_{next} within 10 minutes. Not to introduce a presentation bias, here we only consider manual reformulations (reformulations due to query suggestion clicks are not counted).

Diversity metric: To measure diversity, we adopt a similar measure as in [5], the total number of URLs in the multiset per query: the total number of unique URLs at top 5 ranks for all queries in the suggestion set, divided by the number of queries in the set.

The set of algorithms to be compared in the session log analysis are given below.

`REL:` Optimize for relevance; at any point add the query that maximizes the relevance metric into the suggestion set. Optimizing for relevance is in fact the dominant approach in the commercial search engines.

`DIV:` Optimize for diversity and use relevance to break ties; start with the most popular query, and at any point among the queries that maximize the diversity metric, add the query with maximum query frequency into the suggestion set.

`DQS:` The random walk hitting time based method in [17].

`UM:` Our suggestion utility maximization algorithm.

Note that we decided not to include manifold ranking method in [27] into the comparisons. Although this method is theoretically appealing, it seems applicable to only frequent queries, where there exist a sufficient number of related queries in the query neighborhood to construct the query manifold reliably. In their paper, the authors provide results over queries that have appeared at least 700 times in a month of query logs, which roughly corresponds to 0.002% of the overall unique query traffic. For infrequent queries, the query manifold becomes very unreliable and the relevance of the query suggestions drop very significantly, making this method impractical for real applications.

4.4 Online Test Setup

The online test is a classical A/B test setting, where the search traffic is split into two exclusive groups, the baseline (control) and the test buckets. In the A/B test the method and the baseline is presented to millions of users, and their interactions are recorded for the analysis. The user populations in the baseline and the test buckets are uniformly sampled, hence they are not biased towards heavy or light users. The query volume, and query distributions (for example the proportion of navigational and informational queries or other similar statistics etc.) are very similar in both buckets. The test bucket implements our suggestion set utility maximization algorithm, both in the presubmit and postsubmit. For both presubmit and postsubmit, we log when our algorithm modifies the suggestion set in order to be able to find the subset of queries which we will refer to as *affected queries*. In the following, we describe the configuration in the baseline bucket as well as the search assist modules within which the test and baseline configurations are applied. Except these search assist modules mentioned below, everything on the search result page is identical for both the test and baseline buckets, including the search results, sponsored search results, page layout and user interface. The test is kept alive for a sufficient period of time such that all metrics that we are interested can reach statistically significant values.

Baseline Configuration: The baseline configuration is a simple modification of `REL`. It disregards diversity and does not consider the conditional utility of each suggestion and it is optimized for relevance (popularity in presubmit), except that it implements two simple query string based deduping operations:

- If two query suggestions in the list are identical except for one word is plural (for example, "hp laptop" and "hp laptops"), remove the less frequent of these two from the suggestions, and add the popularity mass of the less popular suggestion to the more popular one.
- If one of the query suggestions is identified as a misspelling of one of the others, remove the misspelled query from the suggestions and add the popularity mass of the misspelled query to the correctly spelled one.

Test Configuration: `UM`, our suggestion utility maximization method.

Search Assistance Modules: There are three search assistance modules that we will apply the changes in this A/B test. *Suggest-as-you-type* (SAYT) is the presubmit search assistance module. Given a prefix, the top 5 queries with the highest score that match the prefix are presented under the query box like a dropdown menu. *North Also Try* (NAT) is the postsubmit search assistance module that is displayed above the search results. The user is likely to see these suggestions -probably even before examining the search results. *South Also Try* (SAT) is the postsubmit search assistance module that is displayed below the 10 search results, right above the next page button. For a given query, the suggestions presented in NAT and SAT modules are exactly the same (same suggestion set in the same order), and depending on the number of characters in each suggestions, up to 5 suggestions can be displayed.

5. EXPERIMENTAL RESULTS

Our experiments consist of three parts, (i) user studies that we did for selecting the γ value and to measure the accuracy at this operating point, (ii) session log analysis to compare our method (at this γ value) to relevance and diversity baselines as well as an earlier proposed method for both presubmit and postsubmit, (iii) a thorough analysis of the results from the A/B test, again independently for presubmit and postsubmit.

5.1 User Studies

We present two user studies. First one is a pairwise test for tuning the γ value presented in the greedy algorithm, which is basically the threshold of $U(q_s|q_p)$ that we will consider q_s redundant if q_p is already presented. The second test is a side-by-side test at the particular γ value selected.

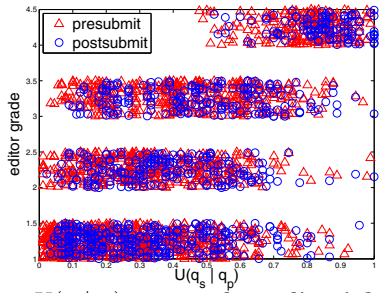

Figure 2: $U(q_s|q_p)$ vs. the editorial grades (3-useful, 2-somewhat useful, 1-somewhat redundant, 0-redundant) Query pairs sampled from presubmit (red) and postsubmit (blue) are presented.

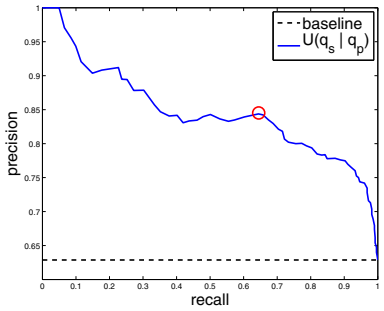

Figure 3: precision-recall curve of the $U(q_s|q_p)$ and the precision-recall curve of the majority classifier. The marker shows the selected operating point where $\gamma = 0.24$.

5.1.1 *Pairwise Comparison*

To find the optimal γ value to execute the greedy algorithm, we sampled 2400 unique (q_s, q_p) pairs. 1600 of these samples are coming from query pairs that were displayed together in Suggest-as-you-type. While collecting the supervised learning data for training/testing a query classifier or some retrieval model, it is a common practice to generate sample set weighted with the traffic, hence the frequent queries are respected more and added into the training set with greater likelihood. With the same motivation, we collect this sample in a way that the sampling probability of a particular query pair is directly proportional to the number of times that they are displayed together (so pairs like "facebook" and "facebook login" displayed millions of times in the suggest-as-you-type together for the same prefix have much higher likelihood of getting into this sample set). Similarly, we sampled 800 query pairs from the postsubmit suggestion pairs that are popular reformulations of the same query, where the sampling probability is directly proportional with the reformulation probability. Hence, we again sample more from more frequent query pairs.

The editorial test evaluates how useful the suggestion q_s is given that q_p is already presented to the user in a four grade scale (3-useful, 2-somewhat useful, 1-somewhat redundant, 0-redundant). Figure 2 shows the distribution of the data samples, $U(q_s|q_p)$ vs. the editorial grade. We binarize this 4-grade by mapping useful and somewhat useful to 1, and redundant and somewhat redundant to 0. We investigate the precision-recall curve of this dataset (shown in Figure 3) to select γ, and decided to use $\gamma = 0.24$. This corresponds to 0.85 precision and 0.65 recall on this dataset.

5.1.2 *Side-by-side Evaluation*

To measure the performance of UM at the particular γ value selected, we designed a side-by-side test. For a given prefix or full query, we provide the judges the top 5 suggestions of the baseline configuration and the test configurations defined in Section 4.4 (REL with some tweaks and UM), and ask them which of the suggestion sets is better.

Table 1: A side-by-side sample of REL (left) and UM (right)

netf	netf
netflix	**netf**lix
netflix rentals	**netf**irms
netflix movies	blockbuster vs **netf**lix
netflix queue	**netf**ile
netfirms	**netf**ront
wal m	wal m
wal mart	**wal m**art
wal mart stores	**wal m**art canada
wal mart supercenter	**wal m**art black friday 2008
wal mart online	**wal m**art benefits
wal mart canada	**wal m**art black friday 2009
faceb	faceb
facebook	**faceb**ook
facebook login	farmville **faceb**ook
facebook home	**faceb**ook farm town
faceboklogin	**faceb**ook layouts
facebook friends	**faceb**ook ipo
well	well
wells fargo	**well**s fargo
wellsbutrin	**well**sbutrin
wells fargo bank	tom **well**ing
wells fargo online	**well**point
tom **well**ing	dawn **well**s

To construct the sample set first we get the 500 most frequent queries in our session logs. From these, for each query we grab 3 random prefixes between 2-6 characters, and we end up with 758 unique prefixes. For these 758 prefixes, the side-by-side test significantly favors the UM against REL. The judgments are 42 (5.5%) REL is better, 191 (25.2%) UM is better, and 525 (69.3%) no noticeable difference. Table 1 shows some samples from this side-by-side test.

To follow up with the cases that the editor thinks UM is worse, for these 42 cases we provided the editors the individual (q_s, q_p) pairs, and asked them to do a pairwise comparison (exactly as in the previous subsection), to identify the useful suggestions that we incorrectly removed. For 28 of these 42 cases, the editors did not report any useful suggestion that is removed, and mentioned that the problem is not the useful suggestions that are incorrectly removed, but the relevance of the suggestions that rank higher after removing these. For example, see the "wal m" example in Table 1 where the algorithm removes "wal mart stores", "wal mart supercenter", "wal mart online" as reduntant since "wal mart" is presented, and the editor agrees that these are all redundant suggestions. However, the editor notes that the relevance of the suggestions (originally ranked 7^{th}, 8^{th} and 9^{th}) that bump up after removing the redundant ones are really low, and despite the redundancy, the original suggestion set *looks* better.

The small follow-up study over the negative side-by-side examples suggests that tuning γ in a query-dependent manner can further improve the accuracy, and will be the focus of our future work. Overall, the side-by-side test shows that the judges think that UM leads to better results as compared to the baseline REL more than 82% of the time, when it introduces noticeable changes.

5.2 Session Logs Analysis

Having set the γ value, before designing an online test we evaluate UM against the baseline algorithms (REL and DIV) as well as a recently proposed algorithm DQS by analyzing our past session

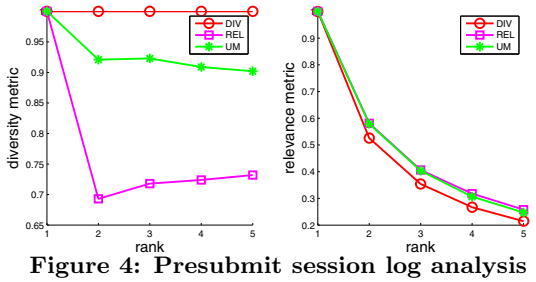

Figure 4: Presubmit session log analysis

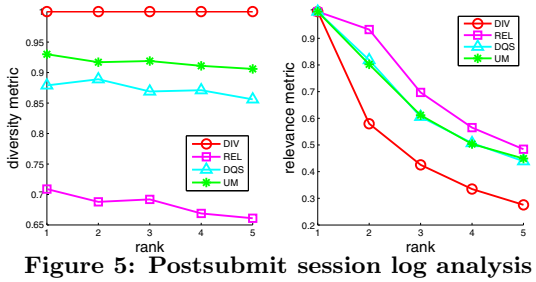

Figure 5: Postsubmit session log analysis

logs. We present two comparisons, one for presubmit and one for postsubmit using the metrics defined in Section 4.3.

For presubmit evaluation, we randomly sample 25K prefixes from query logs and evaluate the diversity and relevance metrics (relevance metric at prefix is marginal query frequency) at each rank up to 5. Note that DQS method is not included in the presubmit evaluation, since by definition it requires the input query, which is yet to be defined in presubmit. Figure 4 shows the presubmit evaluation from session logs, and we normalize the relevance and diversity metrics to [0,1]. Similarly for postsubmit evaluation, we randomly sample 25K queries and repeat the same experiment with DQS also included, results are shown in 5.

In both presubmit and postsubmit, the diversity baseline DIV seems that it is always -at least up to rank 5- able to find some query suggestion that keeps the diversity metric maximal at 1.0, however it has significantly worse relevance at all ranks. Also, the relevance baseline REL with the optimal relevance metric has the worst diversity metrics at all ranks for both presubmit and postsubmit. Note that unlike the diversity metric, the optimal relevance metric, hence average marginal query frequency (presubmit) and average reformulation probability (postsubmit) decreases with rank.

In presubmit, for the relevance baseline REL, the average diversity at rank 2 is significantly lower than the average over top 3, 4 or 5 ranks. This is presumably due to the fact that popular queries of generally navigational nature and their redundant variants appear together at top two ranks more frequently. Overall, in presubmit, UM improves the diversity metric over the relevance baseline REL significantly, without inducing any noticeable loss in relevance.

In postsubmit, relevance baseline REL again has the worst diversity metric, and diversity baseline DIV has the worst relevance metric. In terms of diversity, UM and DQS both improve significantly over the baseline with similar amount of compromise on the relevance metric. In terms of diversity, UM performs better than DQS and similar to the postsubmit, it brings roughly a 30% improvement in the diversity metric over the relevance baseline REL.

5.3 Online Evaluation

In this section we will provide the evaluation of the session logs obtained by the A/B test on a major search engine, and unless stated otherwise, all presented results are statistically significant. The key for evaluating the online tests effectively is to interpret the user interactions correctly. Therefore, throughout this section we will

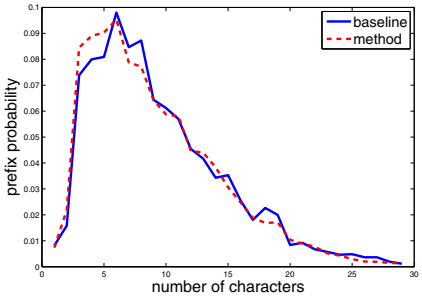

Figure 6: The probability of prefix length (as a function of number of characters), for queries that the algorithm modified the results before a query suggestion is clicked.

also briefly provide the reason why we think the metrics mentioned here are useful ones.

5.3.1 *Presubmit*

The overall aim of Suggest-as-you-type is to auto-complete the user's prefix, hence to minimize the time and effort it takes to the users while they enter their search queries. Therefore, a natural metric for comparing the test and baseline configurations in the presubmit phase is measuring the change in the time and effort.

Let us define the affected queries as follows: the algorithm affected the query if the Suggest-as-you-type suggestion set was modified by the algorithm *at anytime before the query is submitted*. We found that, for this definition, on the affected queries the utility maximization model decreases the average number of characters typed by the user before submitting the query by 1.2% (p-value<0.0001) against the baseline. Although in the right direction, this does not seem to be a big difference. The reason why becomes clearer after observing Figure 6, where we show the probability that the user stops (either clicks a suggestion, or finishes typing the whole query) at a given prefix length, as a function of number of characters typed. Both distributions show very similar characteristics except the 3 to 8 characters region, and that is presumably because this is not a good definition of affected queries. The reason is, for example, if what the user has in mind was "faucet handle opens wrong way" she probably did not care too much about that our algorithm modified results along the way, for example the fact that the "facebook" variants were removed for the prefix "fa".

Let us now define the affected queries as: Suggest-as-you-type suggestion set was modified by the algorithm *at the prefix that a query suggestion is clicked*. For this definition of affected queries, we see a much more significant difference as shown in Figure 7. The difference in average prefix length is -7.4% (p<0.0001), the probability that the user finished the search before 5 characters given that the query is longer than 5 characters increases by 31.3% (p=0.0004). Also, the time it takes the user to get to a long-dwell-time URL decreases by roughly one second on average (p=0.0021). One second might not sound that significant in the beginning, but considering the overall query volume, it adds up to a lot.

For these two definitions of affected queries, the overall suggest-as-you-type click through rate change is 1.14% increase, and 0.75% decrease against baseline, both not statistically significant. Hence, the overall clickthrough rate of the module is flat. Note that unlike the web search scenario, multiple clicks per query (here prefix) is not possible. One can either type the whole query manually (which can be regarded as abandonment of the suggestion module) or select a single suggestion, and a flat click through difference between the baseline and the bucket can be interpreted as UM does not remove many useful queries while introducing diversity into the sug-

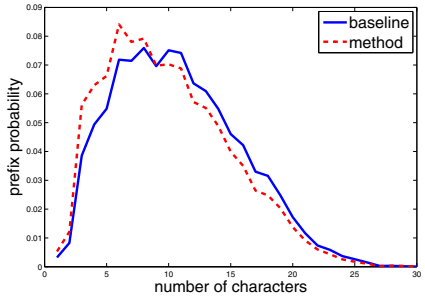

Figure 7: The probability of prefix length (as a function of number of characters), for queries that the algorithm modified the results at the prefix that a query suggestion is clicked.

gestion set, and the suggest-as-you-type does not get abandoned more. We don't directly aim to increase the number of clicks in this module, we just want the users to get what they want sooner.

5.3.2 *Postsubmit*

In the postsubmit, we tested both algorithms for two different suggestion placements, above the search results (NAT) and beneath the search results (SAT). For both the baseline and the bucket, the NAT and SAT suggestions are the same, hence we have four configurations to compare: 1) `REL` at NAT, 2) g`UM` at NAT, 3) `REL` at SAT, and 4) `UM` at SAT.

An argument similar to that in the prefix phase also applies here, we would like to present the suggestion set that minimizes the time it takes to the user to get to what they want. Note that suggestion click trough rate is not a good metric for this, since what the user want is not a suggestion anymore, it is a search result. In fact, there are many cases that decreasing the suggestion click through rate by removing low utility suggestions can save the user a lot of time. Suppose for the query "bank of america", the suggestion is "bank of america online". Clicking on this suggestion very rarely takes the user to a destination URL that was not already on the first search result page, and this suggestion just distracts and causes the user to lose time. We want the users to spend less time for reaching the destination URLs, and this is exactly the sort of suggestion click that we would like to get rid of. On the other hand, if a useful suggestion is removed that could take the user to a destination URL that is not on the original page, the user would have to reformulate the query manually, which would take longer time.

One thing that we have observed is when not distracted with low utility suggestions, users tend to click more on the search results. All result set click trough rate on the bucket is 1.296% (p=0.0007) higher than the baseline, where the biggest gains were in position 2 to 5 with 4.0%, 8.3%, 9.8% and 11.0% increase in clickthrough, respectively (all with p<0.0001). Also, while they show exactly the same suggestions and NAT click trough rate decreases very sharply -7.9% (p<0.0001) against the baseline, there is no statistically significant change in the SAT results -0.4% (p=0.11). Given that the same suggestions were displayed at NAT and SAT, the difference can be interpreted as `UM` is not removing suggestions that would be useful to the user after examining the first 10 results.

In addition to the click trough rates above, we also use a metric that based on user's search paths to evaluate the effectiveness of our utility maximization algorithm. A search path is the sequence of search events conducted by a search user that started from an initial query, possibly went through a few query reformulations and result clicks, finally reaching a destination page. Intuitively, if either the retrieval results and/or the suggestions are better, the average time of the search paths to final destination URL should get smaller.

Table 2: PGL results for four test configurations

Configuration	Total	Loss	Gain	PGL
`REL`, NAT	185,910	120,017	65,893	35%
`UM`, NAT	144,654	89,961	54,693	38%
`REL`, SAT	7,429	3,751	3,678	50%
`UM`, SAT	6,094	2,681	3,413	56%

We introduce a path-based metric which measures the performance of a suggestion algorithm in terms of additional gain/loss of suggestions observed by users. We call this metric as "path gain/loss" (PGL) metric. To calculate the PGL metric, we first mine the session logs of the A/B test and pulled out the competitive search paths that start from the same query and end at the same destination URL. There are several identification methods for destination URL in the literature [25, 12]. Without loss of generality, we used a simple destination URL classifier based on page dwell time. We consider a clicked page is a destination URL if its dwell time is greater than 100 seconds. Among those competitive search paths (i.e., paths having the same start and end points), we further divided them into two classes.

The first class of paths all involve some query suggestion clicks. We call this class of paths as *assisted paths*. For the other class of paths, they do not have any query suggestion click. This class of paths is called *algo paths*. Intuitively, if we see the time distribution for the assisted paths is shorter than that for algo paths, we can conclude the query suggestions are good at saving user's time.

To compare the time distribution between these two classes of paths, we sorted the algo paths by their path times and chose the path time at 20th percentile as the pivotal path time. We then compared the assisted path one by one against the pivotal path time. If the path time is shorter than the pivotal path time, we count it as a gain for the clicked query suggestion. If the path time is longer than the pivotal path time, we count it as a loss for the clicked query suggestion. (Note that the reason we chose 20th percentile point is our selected system requirement, and this selection is a design choice - we want our assisted paths to be significantly faster than algo paths on average.) In this process, we updated the gain/loss counters associated to each suggestion. We repeated the above gain/loss analysis for all combinations of starting query and destination page observed in the session logs. At the end, we aggregate the gain/loss counts across all clicked suggestions for the total gain/loss counts. The total gain/loss counts reflect the overall query suggestion performance observed by search users. We simply define PGL as the following equation:

$$PGL = \frac{gaincount}{gaincount + losscount} \tag{14}$$

Note that PGL value is in the range of 0 to 1, and higher PGL value means the query suggestion algorithm that is being evaluated is on average reducing the time that it takes the user to reach a destination URL. Table 2 shows the PGL results for four test configurations. For both NAT and SAT, `UM` performs better than the baseline algorithm `REL`, with 8.5% and 12% relative improvements in PGL ratio, respectively.

Note that for both NAT and SAT, we see the utility maximization algorithm has fewer assisted paths than the baseline algorithm. This is expected since our goal is to remove useless suggestions that the user is likely end up at a destination URL that she could have found in the first result page anyway. Therefore, the algorithm tends to offer suggestions for the minor intents, saving time of the people that are interested in these minor intents to help them get where they want without typing the query manually, and saving time of the people that are interested in the major intent by not distracting them. On the affected queries, we found that the average search path in the test bucket is 0.87 seconds shorter than the baseline on

Table 3: PGL results and path time for example queries

Query	Config.	Total	Loss	Gain	PGL	Avg. Path Time	Suggestions
usps	REL	2212	116	34	22%	6.29s	usps tracking, usps rates, usps zip codes
usps	UM	2470	2	8	80%	5.96s	usps jobs, usps priority mail, usps liteblue
hulu	REL	4502	90	11	10%	6.67s	hulu tv, hulu movies, hulu saturday nitght live
hulu	UM	4977	9	9	50%	6.37s	hulu saturday night live, hulu family guy, hulu desktop
food network	REL	2266	43	6	12%	7.64s	food network recipes, tv food network, food network channel
food network	UM	2446	6	6	50%	7.19s	food network recipes, next food network star, food network canada
target	REL	6209	67	22	0.29	7.57s	target online stores, target stores, target department store
target	UM	7164	68	45	0.41	7.29s	target baby registry, target coupon, target jobs
wal mart	REL	2532	70	21	23%	10.77	wal mart stores, wal mart supercenter, wal mart online
wal mart	UM	2748	11	12	52%	10.35	wal mart canada, wal mart jobs, wal mart benefits

average. In Table 5.3.2, we show the NAT results of some example queries in our A/B test. The table columns shows the query itself, query suggestion algorithm, total number of queries received, total loss count for the query, total gain count, PGL value, and average path time, and the corresponding suggestions.

6. CONCLUSIONS

Most query expansion and query generation methods in the literature that are being used for search assistance handle the suggestion candidates individually, and diversity of the query suggestions is generally neglected. We present a utility maximization algorithm to introduce diversity into the suggestion sets by modeling the conditional utility for pairs of queries, and we use this pairwise measure to build an algorithm to maximize the utility of a suggestion set. Overall, the problem that we tackle and the method we provide is analogous to the result set diversification problem, while the formalization of added utility of a suggestion is novel, and the greedy algorithm we develop is customized for the search assistance.

Our problem formulation is different from existing suggestion diversification models in the literature that define the problem as "*given a query*, generate a diverse and relevant suggestion set" [27, 17]. Our approach is to frame the problem as "*given a set of suggestions of size N, rerank the suggestions such that the overall utility in top K rank is maximized for all* $K \leq N$", which not require the query to be given, hence the model is suitable for both presubmit and postsubmit suggestions. Our model is based on a user examination model, and its performance does not drop for infrequent queries -presumably thanks to the smoothing in (5) and the fact that the examination probabilities in (6), which are derived from the rank discount factors in DCG, can reliably be estimated even with a few observations. In the session logs analysis, our model performed better than the random walk based diversification approach presented earlier by Ma et al. [17]. In the online evaluations, we show that our a model gets the users to what they want -a query completion suggestion that they find useful in the presubmit mode, and a destination URL in the postsubmit mode- in shorter time.

7. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM '09*, pages 5–14, New York, NY, USA, 2009. ACM.

[2] A. Anagnostopoulos, A. Z. Broder, and D. Carmel. Sampling search-engine results. In *WWW '05*, pages 245–256, New York, NY, USA, 2005. ACM.

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

[4] R. A. Baeza-Yates and A. Tiberi. Extracting semantic relations from query logs. In *KDD*, pages 76–85, 2007.

[5] P. Boldi, F. Bonchi, C. Castillo, D. Donato, and S. Vigna. Query suggestions using query-flow graphs. In *Proceedings of the 2009 workshop on Web Search Click Data*, WSCD '09, pages 56–63, New York, NY, USA, 2009. ACM.

[6] H. Cao, D. Jiang, J. Pei, Q. He, Z. Liao, E. Chen, and H. Li. Context-aware query suggestion by mining click-through and session data. In *KDD '08*, pages 875–883, New York, NY, USA, 2008. ACM.

[7] J. Carbonell and J. Goldstein. The use of mmr, diversity-based reranking for reordering documents and producing summaries. In *SIGIR '98*, pages 335–336, New York, NY, USA, 1998. ACM.

[8] H. Chen and D. R. Karger. Less is more: probabilistic models for retrieving fewer relevant documents. In *SIGIR '06*, pages 429–436, New York, NY, USA, 2006. ACM.

[9] C. L. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR '08*, pages 659–666, New York, NY, USA, 2008. ACM.

[10] N. Craswell and M. Szummer. Random walks on the click graph. In *SIGIR*, pages 239–246, Amsterdam, The Netherlands, 2007.

[11] H. Cui, J.-R. Wen, J.-Y. Nie, and W.-Y. Ma. Probabilistic query expansion using query logs. In *WWW '02*, pages 325–332, 2002.

[12] D. Downey, S. Dumais, D. Liebling, and E. Horvitz. Understanding the relationship between searchers' queries and information goals. In *CIKM '08*, pages 449–458, New York, NY, USA, 2008. ACM.

[13] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW '09*, pages 381–390, New York, NY, USA, 2009. ACM.

[14] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[15] R. Jones and F. Peng. Web search query rewriting. In *Encyclopedia of Database Systems*, pages 3491–3493. 2009.

[16] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *WWW '06*, pages 387–396, 2006.

[17] H. Ma, M. R. Lyu, and I. King. Diversifying query suggestion results. In *AAAI '10*, 2010.

[18] Q. Mei, D. Zhou, and K. Church. Query suggestion using hitting time. In *CIKM '08*, 2008.

[19] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functions. *Math. Programming*, 14(3):265–294, 1978.

[20] F. Radlinski and S. Dumais. Improving personalized web search using result diversification. In *SIGIR '06*, pages 691–692, New York, NY, USA, 2006. ACM.

[21] M. Sahami and T. D. Heilman. A web-based kernel function for measuring the similarity of short text snippets. In *WWW '06*, pages 377–386, New York, NY, USA, 2006. ACM.

[22] B. Tan and F. Peng. Unsupervised query segmentation using generative language models and wikipedia. In *WWW '08*, pages 347–356, New York, NY, USA, 2008. ACM.

[23] J. Teevan. How people recall, recognize, and reuse search results. *ACM Trans. Inf. Syst.*, 26(4):1–27, 2008.

[24] X. Wei, F. Peng, H. Tseng, Y. Lu, and B. Dumoulin. Context sensitive synonym discovery for web search queries. In D. W.-L. Cheung, I.-Y. Song, W. W. Chu, X. Hu, and J. J. Lin, editors, *CIKM '09*, pages 1585–1588. ACM, 2009.

[25] R. W. White, M. Bilenko, and S. Cucerzan. Studying the use of popular destinations to enhance web search interaction. In *SIGIR '07*, pages 159–166, 2007.

[26] C. Zhai and J. Lafferty. A risk minimization framework for information retrieval. *Inf. Process. Manage.*, 42(1):31–55, 2006.

[27] X. Zhu, J. Guo, and X. Cheng. Recommending diverse and relevant queries with a manifold ranking based approach. In *SIGIR '10 Workshop on Query Representation and Understanding*, 2010.