Query Suggestion Using Hitting Time

Qiaozhu Mei *
University of Illinois at Urbana-Champaign

Dengyong Zhou
Microsoft Research
Redmond,WA 98052

Kenneth Church
Microsoft Research
Redmond,WA 98052

ABSTRACT

Generating alternative queries, also known as query suggestion, has long been proved useful to help a user explore and express his information need. In many scenarios, such suggestions can be generated from a large scale graph of queries and other accessory information, such as the clickthrough. However, how to generate suggestions while ensuring their semantic consistency with the original query remains a challenging problem.

In this work, we propose a novel query suggestion algorithm based on ranking queries with the hitting time on a large scale bipartite graph. Without involvement of twisted heuristics or heavy tuning of parameters, this method clearly captures the semantic consistency between the suggested query and the original query. Empirical experiments on a large scale query log of a commercial search engine and a scientific literature collection show that hitting time is effective to generate semantically consistent query suggestions. The proposed algorithm and its variations can successfully boost long tail queries, accommodating personalized query suggestion, as well as finding related authors in research.

Categories and Subject Descriptors: H.3.3 [Information Search and Retrieval]: Text Mining

General Terms: Algorithms

Keywords: Hitting time, bipartite graph, query suggestion, personalized query suggestion

1. INTRODUCTION

The explosive growth of web information has not only created a crucial challenge for search engine companies to handle large scale data, but also increased the difficulty for a user to manage his information need. It has become increasingly difficult for a user to compose a succinct and precise query to present his search need. Instead of pushing this burden to the users, it is common practice for a search engine to provide some types of query suggestions.

When a user types a query "msg" to the search engines, he will be provided with quite a few alternative potential queries. For example, he will be suggested "msg chinese food," "msg health," and "other names for msg" by Google, and "msg error," "msg network," and "msg seating chart" by Yahoo [1]. There are also other query suggestion mechanisms which could automatically complete a query [7], and automatically correct spelling mistakes [10].

Such query suggestion mechanisms are usually developed based on morphological information of queries, or cooccurrence of one query word with other queries (e.g., in the same query, or in the same working session). Although such query suggestions are proved useful in different ways, there is usually no guarantee that the suggested queries convey close semantic information with the original query. Indeed, it is usually annoying for a researcher who searches for "Chris Burges" but is suggested with "chris burgess [2]" or "Chris Burge Ministries [3]." Similarly, it is not very helpful to suggest "KDD" with "KBB", "kddi," "Ntt [4]," and "Harry Shum" with "Harry Potter [5]." People searching for "larry page" maybe interested in "sergey brin" but not "yellow page." A good query suggestion system should consider a handful of features, but in most cases it is important to ensure that the semantics of the suggested query do not drift too much from the original one.

The problem becomes more challenging when personalization is taking into consideration. Some users will issue the query "msg" to search for the sports center in New York and others use it to search the food additive. "msr" could mean "microsoft research," but also "mountain safety research, " or even "mortgage servicing rights." Without the constraint of semantics, a general suggestion to such ambiguous queries would easily be off the track.

Another big challenge and opportunity for the current query suggestion systems lies in the suggestion of infrequent queries. It has been a well known theory in business that a company could "sell less of more" by boosting the long tail of the power law distribution [2]. Netflix spends millions to look for an effective way to suggest hard-to-find movies. The same question lies in search engine business, especially in advertising where customers bid for query terms. Frequently clicked queries cost more and long tail queries cost less. If a well designed query suggestion system could route the traf-

* This work was done when the first author was on a summer internship at Microsoft Research.

CIKM'08, October 26–30, 2008, Napa Valley, California, USA.

[1] All real examples are collected on Feb. 25th, 2008.
[2] http://search.yahoo.com/search?p=chris+burges
[3] http://search.live.com/results.aspx?q=chris+burges
[4] http://search.live.com/results.aspx?q=KDD
[5] http://search.live.com/results.aspx?q=harry+shum

fic and boost the clickthrough of long tail queries, there is a huge opportunity to maximize the benefits for both a search engine company and customers of its advertising system.

Is there a principled way to suggest semantically similar queries while also boosting long tail queries? Can such a method also provide a natural solution to personalization? It is challenging because "semantics" is hard to define and both long tail queries and personalization usually suffer from data sparsity.

In this paper, we propose a unified approach to query suggestion, by computing the hitting time on a large scale bipartite graph of queries and clickthrough. Despite its simplicity, this novel approach introduces quite a few benefits to query suggestion: 1) the suggestions generated with the proposed algorithm are semantically similar to the original query; 2) the suggestions generated do not have to occur with the original query; 3) this approach boosts the long tail queries as suggestions; and 4) this model provides a natural treatment for personalized query suggestion. Empirical experiments on a large scale query log of a commercial search engine, as well as a public available scientific bibliography dataset show that our proposed algorithm is effective for semantically coherent query suggestion, which provides a potential new framework, or an important and novel feature for building a real query suggestion system. The approach of using hitting time is quite general, which could provide potential solutions to many other search related problems other than query suggestion. We will discuss these possibilities later in Section 6.

The rest of the paper is organized as follows. In Section 2, we formally introduce the concept of hitting time on a bipartite graph. In Section 3, we propose the algorithm of query suggestion using hitting time. We show our experiments and results in Section 4, introduce the related work in Section 5, and conclude in Section 7.

2. BIPARTITE GRAPH AND HITTING TIME

A bipartite graph is a graph $G = (V, E)$ in which there exists an partition $V = V_1 \cup V_2$ such that every edge in E connects a vertex in V_1 and one in V_2; that is, there is no edge between two vertices in the same set. Let $w : V_1 \times V_2 \to \mathbb{R}^+$ denote the weight function. Given $i \in V_1$ and $j \in V_2$, if there is an edge connecting i and j, then $w(i, j)$ is positive; otherwise, $w(i, j) = 0$.

Given a bipartite graph, a random walk can be formed as follows. Assume the current position is at a vertex in V_1. Then an edge connected to this vertex is chosen with the probability proportional to the weight of the edge. By following this edge, the random walk arrives at a vertex in V_2. Then, similarly, an edge connected to V_2 is chosen to follow and the random walk goes back to V_1. Given $i \in V_1$ and $j \in V_2$, the transition probability is defined as

$$p_{ij} = \frac{w(i,j)}{d_i}.$$

where $d_i = \sum_{j \in V_2} w(i, j)$. If one is only interested in the vertices in one side, such as V_1, then a new random walk based on the above one can be introduced by

$$p_{ij} = \sum_{k \in V_2} \frac{w(i,k)}{d_i} \frac{w(k,j)}{d_k}.$$

It is easy to check that the stationary probability π_i is proportional to d_i. In what follows, we discuss hitting time on a graph $G = (V, E)$. All materials are self-contained. For readers who are familiar with this concept, they can skip the discussion.

Let A be a subset of V. Let X_t denote the position of the random walk at discrete time t. The hitting time T^A is the first time that the random walk is at a vertex in A, thus $T^A = \min\{t : X_t \in A, t \geq 0\}$. It is obvious that T^A is a random variable. From the definition of the hitting time, given $i \notin A$, we immediately have

$$\begin{aligned} P[T^A = m | X_0 = i] &= \sum_{j \in V} P[X_1 = j | X_0 = i] \\ &\quad \cdot P[T^A = m - 1 | X_0 = j] \\ &= \sum_{j \in V} p_{ij} P[T_A = m - 1 | X_0 = j]. \end{aligned}$$

The mean hitting time h_i^A is the expectation of T^A under the condition $X_0 = i$, that is, $h_i^A = E[T_A | X_0 = i]$. Thus

$$\begin{aligned} h_i^A &= \sum_{m=1}^{\infty} m P[T^A = m | X_0 = i] \\ &= \sum_{m=1}^{\infty} m \sum_{j \in V} p_{ij} P[T^A = m - 1 | X_0 = j] \\ &= \sum_{j \in V} \sum_{m=1}^{\infty} (m-1) p_{ij} P[T^A = m - 1 | X_0 = j] \\ &\quad + \sum_{j \in V} \sum_{m=1}^{\infty} p_{ij} P[T^A = m - 1 | X_0 = j] \end{aligned}$$

Obviously,

$$\begin{aligned} &\sum_{j \in V} \sum_{m=1}^{\infty} (m-1) p_{ij} P[T^A = m - 1 | X_0 = j] \\ &= \sum_{j \in V} p_{ij} \sum_{m=1}^{\infty} m P[T^A = m | X_0 = j] \\ &= \sum_{j \in V} p_{ij} h_j^A \end{aligned}$$

For computing the second term, it is necessary to notice that

$$\sum_{m=1}^{\infty} P[T^A = m - 1 | X_0 = j] = 1.$$

Thus,

$$\sum_{j \in V} \sum_{m=1}^{\infty} p_{ij} P[T^A = m - 1 | X_0 = j] = \sum_{j \in V} p_{ij} = 1.$$

Consequently,

$$h_i^A = \sum_{j \in V} p_{ij} h_j^A + 1.$$

In addition, it is obvious that

$$h_i^A = 0, \text{ for } i \in A.$$

Combining all pieces together, we obtain the linear system for computing the hitting time:

$$\begin{cases} h_i^A = 0 & \text{for } i \in A \\ h_i^A = \sum_{j \notin A} p_{ij} h_j^A + 1 & \text{for } i \notin A \end{cases}$$

This linear system has a unique solution. This fact can be elegantly verified by using Maximum Principle and Uniqueness Principle in the discrete potential theory.

3. SUGGESTION USING HITTING TIME

Based on the formal definition of hitting time, we now propose our algorithm of query suggestion using hitting time.

3.1 The Algorithm

Let us begin with a query log dataset, from each record of which we can extract a pair $\langle Query, URL \rangle$. By summarizing all such pairs, we can construct a bipartite graph $G = \langle V, E \rangle$, where $V = V_1 \cup V_2$. Clearly, V_1 corresponds to all queries, and V_2 corresponds to all URLs. Each edge $e = (i, j) \in E$ corresponds to a pair $\langle Q_i, U_j \rangle$ with positive frequency. We weight each edge with $w(i, j) = C(Q_i, U_j)$, which is the number of records where this pair appears.

There are also other variations to this setup, e.g., by normalizing the edge weights, constructing a k-Nearest-Neighbor graph, or using a Query-Query graph, Query-IP graph, etc. In this section, we use the undirected Query-URL bipartite graph as a representative case to illustrate our algorithm. A simple example of such a graph is shown in Figure 1.

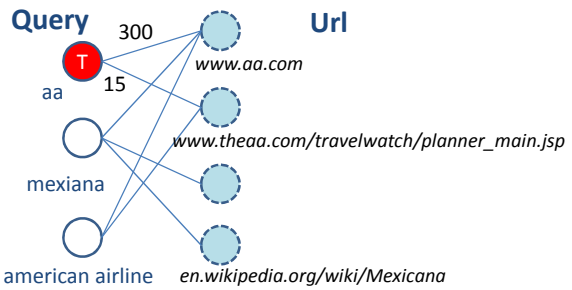

Figure 1: Example of an undirected Query-URL bipartite graph

From Figure 1, we see that every query is connected with a number of URLs, on which the users clicked when submitting this query to the search engine. The weights on the edges present how many times the users used this query to access this URL. Please note that there is no edge connecting two queries, or two URLs.

The labeled query indicates the query for which we want to generate suggestions. Intuitively, if for all URLs that we use a query to access, other people exclusively use another query to access, that query is a good suggestion to the original query, e.g., "american airline" to "aa" in Figure 1.

Let Q_T be the original (target) query. In principle, we can set $A = \{Q_T\}$ and compute the Hitting time $h^A(i)$ for all other queries Q_i based on this graph, use this measure to rank Q_is, and select the top-k queries as suggestions to Q_T. However, there are two concerns for using the straightforward and formal solution we presented in Section 2.

- The graph G can be too large (e.g., 500M queries and URLs). In fact, most vertices are irrelevant to the original query, but they increase the computational cost.
- Solving the linear system can be time consuming. When the number of variables of the linear system is millions, it becomes extremely inefficient to get an exact solution to that linear system.

To overcome these two concerns, we propose the following efficient algorithm for query suggestion using hitting time:

Algorithm 1 Query Suggestion Using Hitting Time

A bipartite graph $G = (V_1 \cup V_2, E)$ consists of query set V_1 and URL set V_2. There is an edge in E from a query i to an URL k if this URL is clicked, and the edge is weighted by the click frequency $w(i, k)$.

1. Given a query s in V_1, a subgraph is constructed by using depth-first search in G. The search stops when the number of queries is larger than a predefined number of n queries.
2. Form a random walk on the subgraph by defining transition probabilities between two queries i and j in V_1 as
$$p_{ij} = \sum_{k \in V_2} \frac{w(i,k)}{d_i} \frac{w(k,j)}{d_k}.$$
3. For all queries except the given one, iterate
$$h_i(t+1) = \sum_{j \neq s} p_{ij} h_j(t) + 1$$
for a predefined number of m iterations started with $h_i(0) = 0$.
4. Let h_i^* be the final value of $h_i(t)$. Output the queries which have the top k largest h_i^* as suggestions.

A good selection of k would control that the ranking of top k queries stays stable in future iterations. In Section 4, we will show that k does not need to be large, which ensures the efficiency of this algorithm. In some scenarios, a different initialization of W can be used. For example, one can use mutual information of a query and a URL instead of the clickthrough frequency to initialize w_{ij}. One can also use a weighting schema such that $w_{ij} \neq w_{ji}$, which naturally generalizes this method to directed graphes.

Please note that since we are interested in query suggestions, we fold the bipartite graph into a general graph in the algorithm above. In general, we can easily unfold the graph in the algorithm, by setting $p_{ij} = \frac{w(i,j)}{d_i}$.

3.2 Personalized Query Suggestion

Personalization is desirable for many scenarios where different user has different information need. People in New York are likely to use "msg" to access the sports center, thus a suggestion like "madison square garden" is quite useful. People in other states, on the other hand, may use "msg" to access the food additive, and a suggestion like "Monosodium glutamate" is desirable.

There has been quite a few work on personalized search [11]. However, how to generate personalized query suggestion is still an unsolved problem. [8] presents automatical query completion with local information, but that method is based on query morphology and cannot be applied to generate personalized semantic suggestions.

We now present that our method using hitting time on bipartite graph can be easily adapted to generate personalized

query suggestions. Intuitively, when we know the identity of the user (e.g., his IP address), we should update our knowledge about the information need of this query.

One may say that a simple method is to construct the bipartite graph solely based on the history of that user. However, that could easily fall into the problem of data sparsity. The simple treatment also loses the opportunity of using common wisdom. If a user already knows what query to use (e.g., learning from his history), it is not clear how much query suggestion could help.

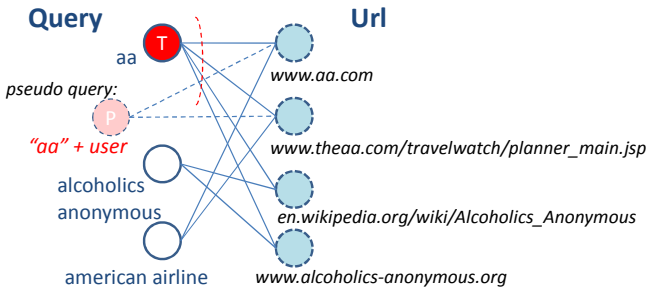

Figure 2: Personalized query suggestion

Figure 2 illustrates an intuitive treatment of personalized query suggestion. Once we know the user, we need to update what we know about the query. This can be viewed as equivalent to replacing the original query (e.g., "aa") with a pseudo query, which is user specified (e.g., "aa" + user).

Once we know the user, we adjust the URLs that he would click with this query based on the history of this user. Now the pseudo query connects to some URLs instead of others. And different suggestions will be generated comparing with the non-personalized suggestions. In Figure 2, "american airline" is now a better suggestion than "alcoholics anonymous" to "aa" given the user.

The remain problem is how to adjust the weights on the edges between the pseudo query and the URLs. In principle, giving the original query Q_T, in computing the hitting time we only cares about p_ij where $v_j = Q_T$ and v_i is a URL, or simply as $p(Q_T|URL)$. $p(Q_T|URL)$ is computed with $\frac{c(Q_T,URL)}{\sum_Q c(Q,URL)}$ in Step 2 in the algorithm in Section 3. Similarly, giving the pseudo query Q_P, we are only interested in $p(Q_P|URL)$, or $p(Q_T|Url, User)$.

A simple computation is

$$p(Q_T|URL, User) = \frac{c(Q_T, URL, User)}{\sum_Q c(Q, URL, User)}. \quad (1)$$

However, it could still fall into the problem of data sparsity. Interestingly, one can notice that many probabilistic personalized search algorithms proposed nowadays are essentially computing $\hat{p}(Url|Q, User)$ [20, 17, 16]. This means that we can easily adopt any such personalized search algorithm, and compute

$$p(Q_T|Url, User) = \frac{\hat{p}(URL|Q_T, User)p(Q_T|User)}{p(URL|User)}. \quad (2)$$

This suggests that without twisting the algorithm structure or the whole graph, we can easily embed personalization into query suggestion, by adjusting w_{ij} at step 2 in the algorithm in Section 3. Specifically, if $v_j = Q_T$ (the original query) and v_i is a URL, we assign new weights for W_{ij} by

$$w_{ij} = \frac{p(Q_T|URL, User)\sum_{j' \neq j} w_{ij'}}{1 - p(Q_T|URL, User)}. \quad (3)$$

The rest of the algorithm remains the same with the non-personalized query suggestion.

As a concrete example, if we use the IP address to identify a user, we can adopt the personalization with backoff model in [16] and embed in Equation 2 with

$$\hat{p}(URL|Q, IP) = \sum_{i=0}^{4} \lambda_i p(URL|Q, IP_i). \quad (4)$$

Here IP_i means the first i bytes of IP.

An alternative treatment for personalized query suggestion is to add some related vertices (queries or URLs) into the target set A based on the user interest. We leave this as a direction in the future.

4. EXPERIMENTS

In Section 3, we introduced the algorithm of query suggestion using hitting time, and its natural adaptation to personalized query suggestion. In this section, we use empirical results to show the effectiveness of the proposed algorithms.

We collect a large scale query log dataset from a commercial search engine of about 1.5 years up to July 2007. This 1.5 years data contains 637 million unique queries, and 585 million unique URLs. We use IP address to identify users in personalized query suggestion. This dataset contains 193 million unique IP addresses.

To make it easier for others to reproduce our results, we also collected a publicly available dataset of authors and titles from DBLP[6], as of Feb. 2008. We extract 110k papers in computer science and around 580k unique authors from that dataset.

The experiments and results from the two datasets are presented in the following sections.

4.1 Experiments on Search Log Data

Semantic Query Suggestion

We use the Query-URL bipartite graph extracted from the 18 month data and generate suggestions for all queries. We selectively show the results in Table 1.

We present the comparison of suggestions generated from our algorithm and those from Google, Yahoo, and Live. Clearly, we can see that the suggestions generated with hitting time focus on different aspects than suggestions currently provided by the three major search engines. Specifically, there are several major differences:

1. The suggestions generated with our method are more semantically consistent to the original query.
2. Our method generates suggestions that are morphologically different, but semantically relevant to the original query.
3. Our method generates useful suggestions even for infrequent queries. For original queries that are infrequent themselves, our method makes reasonable suggestions while the major search engines don't.

[6]http://www.informatik.uni-trier.de/~ley/db/

Table 1: Query suggestions generated using hitting time on Query-URL graph

Query = msg				Query = harry shum
HititngTime	Google	Yahoo	Live	HittingTime
msg facts	msg chinese food	msg error	Madison Square Garden	ce liu
food msg	msg health	msg network	Msg Allergy	managing director of-microsoft
poisoning of america	other names for msg	msg seating chart	MSN	shum
msg in fast food	msg duty	valentine msg	Msg Food	microsoft distinguished-engineers
...	msg symptoms	foods with msg	Monosodium Glutamate	microsoft engineers
msg network	marine security guard	yahoo msg	Ticketmaster	
madison square garden	michael schenker	verizon text msg	Msg Tickets	
Query = friends				**Query = ranknet**
HittingTime	Google	Yahoo	Live	HittingTime
wikipedia friends	friendship	secret friends	Find Friend	learning to rank
friends tv show wikipedia	friends poem	friends reunited	Friendship	ndcg measure ir
friends home page	friendster	hide friends	Friends TV Show	ndcg
friends warner bros	friends episode guide	hi 5 friends	Best Friends	lambdarank
the friends series	friends scripts	find friends	Secret Friends	chris burges
friends official site	how to make friends	poems for friends	Jennifer Aniston	pairwise test
friends(1994)	true friends	friends quotes	Friendster	rank function
Query = aa				**Query = long tail**
HittingTime	Google	Yahoo	Live	HittingTime
alcoholics anonymous		aa route planner	AA Route Finder	wikipedia long tail
automobile association		aa route finder	AA Route Planner	long tail chris anderson
theaa	N/A	aa airlines	AA Airlines	long tail wired
american airlines		aa meetings	American Airlines	chris anderson
american air		aa autoroute	American Airlines	full sentence outline-on outsourcing better-quality at a lower cost
american airline-ticket reservations		aa road map	AA Meetings	
		aa 12 shotgun		

The system generated suggestions for "msg" are comparable with those suggested by the major search engines. Our method captures the food additive in the suggestions, which is the most commonly known semantics of msg. We also see another meaning of msg, madison square garden, ranked lower in top 10 suggestions. When people use "friends" as a query instead of "friend," it usually carries a special semantics (i.e., the Friends TV series). The query suggestions generated using our system well captured this special semantics, while the major search engines mostly return suggestions about the common sense of friend. Indeed, the three search engines generate mostly the same suggestions for "friend" and "friends," while our system generate quite different suggestions for "friend," such as "friend to friend shelter," "friend dictionary," and "web friend."

From the suggestions for "aa," we see that our system not only captures the most common sense, the "american airline, " it also successfully boosts infrequent queries as suggestion ("alcoholics anonymous" and "automobile association"). The major search engines, however, captured the most common meaning but lost the opportunity of visiting the long tails.

On the rightmost column of Table 1, we show three example queries for which our system generates good suggestions while the search engines do not. All such queries are "long tail" queries in search business. We see that using hitting time on the Query-URL graph, we generate meaningful suggestions even if the suggested terms do not co-occur with the original query.

For example, the query "ranknet" is a learning-based ranking algorithm used in web IR based on "pairwise tests." "learning to rank" is a nice generalization of the query, while "ndcg" is the key performance measure used in web search, which such an algorithm tries to optimize. Chris Burges is one of the inventors of RankNet, and "lambdarank" is their following work. Such suggestions are all semantically relevant to the original query.

In another example, "the long tail" is a famous book by Chris Anderson, the theory of which is well applied in outsourcing better quality at a lower cost.

All experiments presented above show that our algorithm effectively generates semantic consistent query suggestions, and provides a way of treating and boosting long tail queries.

Personalized Query Suggestion

As discussed in Section 3.2, the algorithm of query suggestion with hitting time can be easily adapted to personalized query suggestion. As an illustrative example, we use an IP address to identify a user, and embed the personalization with backoff model (Equation 4) in computing Equation 2. The personalized query suggestion compared with non-personalized query suggestion are shown in Table 2.

From Table 2, we see that given the IP address of the user, the system generates quite different query suggestions than if the IP is unknown. If the query is from Microsoft, the system will suggest "microsoft research" related queries for "msr" instead of "mountain safety research," and "kdd conference" related queries for "kdd." "KDDI" is a Japanese corporation on telephone business which is used to be called "Kokusai Denshin Denwa (KDD)," and is more interesting to the common audience. Similar treatments can be found for the query "msg," where "madison square garden" related queries are suggested to people who live near New York.

Another interesting example is the query "football", which means *American football* in the States but usually means *soccer* in Europe. When the query is from the United States, the system suggests American football related concepts, as well as the "fox nfl", where "Fox" is a well-known television network in the States. If the query is from United Kingdom, the systems suggests soccer related concepts, as well as "bbc", a broadcasting corporation located in UK.

4.2 Experiments on DBLP Data

Search log data is associated with privacy concerns as well

Table 2: Personalized Query Suggestions using IP Address

Query	Non-personalized	Personalized	Query	Non-personalized	Personalized
	"mountain safety research"	microsoft research		msg facts	madison square garden
	msrcorp	research		food msg	madison square gardens
msr	mountain safety research	what is research	**msg**	poisoning of america	madison square garden events
	msr outdoor	research website		msg in fast food	madison square garden tickets
131.107.*.*	msr camp stoves	ms research	71.250.*.*	what is msg	madison square
(Microsoft)	msr outdoor equipment	university of pittsburgh-clinical trials	(NJ, US)	msg food additive	madison square garden-new york
	msr snowshoes	microsoft research-and development		monosodium glutamate	madison square garden-box office
	msr racing	yahoo research labs		msg network	madison sq garden
Query	**Non-personalized**	**Personalized**	**Query**	**169.229.*.* (US)**	**212.58.*.* (UK)**
	kidd	kdd 2007		cheerleader of the day	bbc football
	kid (kidd)‡	kdd 2006		terms in football	bbc sport football
kdd	kddi corporation‡		**football**	nfl franchise	football news
		sigkdd		nfl	football fixtures
131.107.*.*		kdd international tel.‡		national football league	bbc football news
(Microsoft)	kddi international-telephone‡	kdd2006		fox nfl	bbc soccer
	kidd group	sigkdd2006		football mercato	football mercato
		kdd conference 2007		football uk	abedi andre dede

* We omit the last two bytes of IP addresses for privacy reason. Corresponding locations of IP are shown in brackets.
‡: Original suggested queries are in Japanese. We present their translations in English.

as Intelligence Property issues, and is usually not accessible to people outside the search engine companies. To generate reproducible results for the common audience, we design similar experiments on a publicly available dataset, DBLP.

Query Suggestion on Coauthor Graph

The most commonly explored graph on the bibliography data is the coauthor graph. In this experiment, we constructed a coauthor graph from the DBLP data, by making a vertex for every author, and an edge between two authors if they coauthored in a paper. The weight on each edge corresponds to how many papers that the two researchers have coauthored.

The method of query suggestion using hitting time is generalized, and can be applied on both bipartite graph, or a general graph (e.g., a query-query graph). Indeed, with the notion introduced in Section 2, we can *fold* a bipartite graph into a graph with only one group of vertices, but with a different weighing function for the folded edges. We use such a coauthor graph to demonstrate that our algorithm can generate interesting suggestions with a general coocurrence graph.

In Table 3, we present the suggestions generated based on computing the hitting time on the coauthor graph. The middle three columns presents the suggestions based on the hitting time computation. By default, for each author name query (shown in column 1), we construct the subgraph by including all authors that are less or equal to distance 6 to the target author. The six degrees of separation [7] is well known in social network analysis that the average distance of a vertex to all others is around 6. Indeed, such a subgraph usually contains 410∼450k authors, which covers more than 70% of the entire graph. We vary the number of iterations and the size of the subgraph to show the robustness of our method.

From Column 3 through 5, we see that our system tends to suggest authors that collaborate intensively with the target query (e.g., students of a professor, and collaborators who works exclusively with the author). It is interesting to see that the system suggests "Lawrence Page" for "Sergey Brin". When we use 10 iterations for our algorithm in Section 3, it already achieves similar ranking to the exact solution, which we get from solving the linear system completely.

Can we use a smaller subgraph? We present the experiments on smaller subgraphs, where all authors are at most 2 steps away from the original author query. From the 4^{th} column of Table 3, we see that a smaller graph captures most of the values of the larger one in terms of query suggestion. This experiment shows that without much loss of performance, the algorithm of query suggestion using hitting time can be made more efficient by using a smaller subgraph (∼1000 vertices) and a few iterations.

Since our work is also based on a random walk on a large scale graph, it is interesting to show how different are our results from other random walk methods. For example, personalized PageRank [11] is a method that is usually used to rank vertices on the graph in a query dependant way. The corresponding linear system of personalized PageRank can be shown as:

$$R_i = (1-s)R_i^{(0)} + s \cdot \sum_j p_{ji} R_j^{(0)}. \quad (5)$$

where $R_i^{(0)}$ is a personalized (or query dependent) initial values for vertex i. We may set $R_i^{(0)} = 1$ if $v_i = Q_T$ and 0 otherwise. It is easy to show that if $s = 1$, R will be the stationary distribution of random walking on the graph, which is proportional to the degree of vertices. We expect that personalized PageRank would still favor authors who published a lot of papers and have a lot of coauthors.

We present the query suggestions ranked by personalized PageRank in the rightmost column in Table 3. We also present the nearest neighbors of Q_T (i.e., v_j's with the largest $w(Q_T, j)$).

We see that personalized PageRank generates quite different suggestions to hitting time. It indeed favors authors with higher degrees, e.g., "Andrew Tomkins" and "Sridhar Rajagopalan" for "Jon M. Kleinberg," and "Monika R. Henzinger" for "Sergey Brin." In fact, we can see that using personalized PageRank does not gain much different top-k

[7] http://en.wikipedia.org/wiki/Six_degrees_of_separation

Table 3: Author suggestions generated using hitting time on coauthor graph

Query	Nearest Neighbors	Iteration = 10	Exact Solution	SmallGraph (d=2)	Personalized PgRank
Jon M. Kleinberg	Prabhakar Raghavan	Aleksandrs Slivkins	Aleksandrs Slivkins	Mark Sandler	Prabhakar Raghavan
	Éva Tardos	Mark Sandler	Mark Sandler	Samer A. Abdallah	Éva Tardos
	Daniel P. Huttenlocher	Tom Wexler	Tom Wexler	Christophe Rhodes	Daniel P. Huttenlocher
	David Kempe	Lars Backstrom	Lars Backstrom	Michael Casey	David Kempe
	Amit Kumar	Elliot Anshelevich	Xiangyang Lan	Aleksandrs Slivkins	Andrew Tomkins
	Andrew Tomkins	Xiangyang Lan	Elliot Anshelevich	Xiangyang Lan	Amit Kumar
	Christos H. Papadimitriou	Daniel P. Huttenlocher	Daniel P. Huttenlocher	Lars Backstrom	Sridhar Rajagopalan
Query	Nearest Neighbors	Iteration = 10	Exact Solution	SmallGraph (d=2)	Personalized PgRank
Sergey Brin	Rajeev Motwani	Lawrence Page	Lawrence Page	Lawrence Page	Rajeev Motwani
	Craig Silverstein	Craig Silverstein	Craig Silverstein	Craig Silverstein	Craig Silverstein
	Jeffrey D. Ullman	Brian Milch	Brian Milch	Brian Milch	Jeffrey D. Ullman
	Bay-Wei Chang	Bay-Wei Chang	Bay-Wei Chang	Bay-Wei Chang	Monika R. Henzinger
	Brian Milch	Hannes Marais	Hannes Marais	Andrey Kolobov	Rajeev Rastogi
	Kyuseok Shim	Andrey Kolobov	Andrey Kolobov	Daniel L. Ong	Brian Milch
	Lawrence Page	Daniel L. Ong	Daniel L. Ong	David Sontag	Bay-Wei Chang
	Monika R. Henzinger	Monika R. Henzinger	David Sontag	Polle Zellweger	Kyuseok Shim

We only include authors with a degree larger than 5. Dumping parameter used in Personalized PageRank: 0.5.

suggestions compared with using just the k nearest neighbors. Although "big" authors are more visible, putting them in the query suggestions blocks the "smaller" authors to be seen, and also causes a topic drift. One could imagine that when a user want to find a Ph.D student whose name he couldn't remember, he is likely to begin with searching his advisor (and a query suggestion of his students would be very helpful). On the other hand, there are way many better directions to find a "big" name than beginning with another "big" name. Please note that our method does not have a model parameter to tune.

The experiments above suggest that our proposed method applies well on query-query graphs, generates better suggestions than other random walk method, and can be made quite efficient.

Query Suggestion on Author-Keyword Bipartite Graph

We then constructed a bipartite graph from the bibliography data by segmenting the titles of every paper into all unigram and bigram words it contains. We made a vertex for every author and every keyword. We then connected an author and a keyword with an edge if the author used that keyword. The weight on each edge corresponds to the frequency that the user used that keyword. We removed the stop words from the title, and no domain knowledge has been applied.

In this way, we get a bipartite graph $G = (V, E)$. $V = V_1 \cup V_2$ where V_1 are the set of all authors and V_2 is the set of all unigram and bigram title terms. As a result, this provides us a bipartite graph of 1.6M vertices, in which around 1M vertices are unigram and bigram keywords. We then simulate that a user would type a keyword (either a unigram or a bigram) as a query, and use our proposed algorithm to generate suggestions to the keyword query. The sample results are presented in Table 4.

It can be easily discovered from Table 4 that our system generates very reasonable suggestions to those keyword queries. All suggestions for the six given queries are semantically close the original query.

Another interesting question on the DBLP data is whether we can suggest keywords for a query of author. Indeed, we can apply our algorithm by computing the hitting time from every keywords to the original query (an author). We then select the top ranked keywords as keyword suggestions for an author query. The results are selectively shown in Table 5.

Presumably, the suggestion keywords should well capture the semantics of the author, or the research topics that the author mostly works on. We first present the k nearest neighbor keywords of each author in the Author-Keyword bipartite graph. For both queries, we see that all such nearest neighbor terms are too broad. They tend to be too general to capture the author's specific research topics. We also used personalized PageRank to generate suggestions (as in column 4), but unfortunately it still improves tiny over the k nearest neighbors. From the column 3 of Table 5, however, we clearly see that we get much better suggestions using hitting time. Indeed, the suggestions generated are general enough to convey coherence meanings, and also tight enough to represent the special interest of the author query. Interestingly, because we segment unigrams and bigrams in a totally unsupervised way, a huge number of bigrams are non-meaningful segments, such as "based approach," "guided mining," and "clusters among," the suggested bigrams using hitting time are all meaningful phrases.

In this Section, we use experiments on two different datasets to show that query suggestion using hitting time is effective to generate semantically consistent query suggestions, long tail suggestions, as well as personalized suggestions.

5. RELATED WORK

Query suggestion has been a well-accepted utility used by many search engines to help user explore and express their information need. While there are quite a few work on generating different types of query suggestions, such as query auto completion [7, 8], query spelling correction [10, 15], query expansion [21, 22, 3], and query rewriting [1, 13]. While most early query suggestion methods explore document information, query log data has been widely used recently.

Query frequency [15, 7, 8], term coocurrence [4, 13, 19], query clickthrough [12, 23, 9, 18], and query chains [17] are among the most used types of information in query log. In this paper, we adopt the query clickthrough information, but the proposed method of query suggestion using hitting time does not rely on such information. Indeed, our algorithm can be applied with all such types of information, as long as an undirected graph of queries, or a bipartite graph of queries and other types of entities, can be constructed.

There are different ranking methods proposed using ran-

Table 4: Keyword suggestions generated from Keyword-Author graph using hitting time

Query	Suggestions	Query	Suggestions	Query	Suggestions
olap	dimension updates olap data olap cubes olap queries view size range top hierarchical clustered	pagerank	pagerank computation ranking systems pagerank approximation peer web incremental computations web spam iterative computation	clickthrough	clickthrough data page classification query classification implicit feedback recommending optimizing web based smoothing
Query	Suggestions	Query	Suggestions	Query	Suggestions
collaborative filtering	amazon dynamic collaborative recommendation algorithms re ranking item based design recommender demographics	random walk	walk timing recovery rao bound hyperlink analysis stabilizing group based scoring between nodes	social networks	knowledge collaboration community structure resource organization information kiosks efficient searching exploit social network extraction

* We omit keywords that has a degree less than 10.

Table 5: Generating keyword suggestions to author queries in DBLP

Query	kNN-keywords	Hitting Time Suggestions	Personalized PgRank
Jiawei Han	mining data frequent based efficient pattern data mining	large databases frequent pattern sequential pattern frequent patterns pattern mining frequent multi dimensional	mining data based efficient frequent pattern data mining
Query	kNN-keywords	Hitting Time Suggestions	Personalized PgRank
Michael I. Jordan	learning statistical kernel markov inference model bayesian	dirichlet process approximate inference dirichlet mean field supervised learning graphic models mixture	learning based statistical model kernel markov bayesian

* We omit keywords that has a degree less than 10.

dom walk on a Query-URL graph. PageRank [5] is basically computing the stationary distribution of a smoothed Markov chain. Personalized PageRank generalizes PageRank by smoothing the Markov chain with a user (or query) specific jumping probability vector instead of a uniform vector, thus is often used for query-dependent ranking [11]. HITS [14] is an alternative query-dependent ranking algorithm which computes two different scores (hub and authority) in an alternating way. [9] proposed a ranking function which is basically computing the n-step transition probability from the original vertex to the target.

However, all such methods are essentially computing "how much weights can be *distributed* to a vertex from its neighbors". This ends up with favoring vertices with large degree and usually results in topic drift. Indeed, topic drift has been a well discussed problem of HITS [6]. Instead, our algorithm computes "how soon can I reach the original query if I begin at a suggestion, with an average of all possible paths". This guarantees that the semantics of the top ranked suggestion will be coherent with the original query. Unlike other random walk methods, it also boosts infrequent queries. Another advantage of the hitting time is that it does not have a parameter to tune, while all the self-jump based methods (e.g., PageRank, personalized PageRank, and n-step transition) all has one or more critical parameters to tune.

In terms of updating the original query, our work is also relevant to feedback [24, 20, 17] in information retrieval. However, both pseudo-feedback and implicit-feedback could easily add in irrelevant terms into the query, especially when the feedback documents has a rich content. Our method utilizes the common wisdom and control the relevance of suggested queries.

6. DISCUSSION

As a ranking function on a graph, hitting time is general and does not rely on the specific type of graphs. We illustrated its power by generating query suggestions from a Query-URL bipartite graph, but there are many other possibilities. On the other hand, the Query-URL relation (i.e., clickthrough) is not the only information conveyed in a large scale search log. Indeed, one can extract Query-IP graphs, Query-Query relations considering session information, etc.

Ranking search-related entities on a graph using hitting time can be regarded as a general treatment of a lot of interesting problem. For example, ranking URLs given a query (by computing $h^A(URL \to Q)$) suggests a method of ranking web pages without looking at their content. $h^A(URL \to URL)$ leads to finding similar pages, $h^A(Q \to URL)$ suggests search terms for a webpage, and $h^A(IP \to IP)$ provides a way to find people who have similar interests like you. All these are interesting directions to apply the method proposed in this paper.

There are many interesting future directions to this work. A real query suggestion system should balance many different features. It is interesting to embed our method as a new feature into a real query suggestion system, and quantita-

tively evaluate how much our method can benefit the current system. Another possible future work is to apply the general algorithm on other types of graphs, for example graphs built from query-session data, query-user graphs, as well as directed graphs. It will then be interesting to generate query suggestions using multiple types of graphs.

7. CONCLUSION

In this paper, we proposed a novel query suggestion approach based on the computation of hitting time on large scale bipartite graphs. Unlike existing query suggestion methods, our proposed method controls the semantic consistency of the suggestions to the original query. The proposed method has several advantages over existing methods: 1) the generated suggestions are semantically consistent to the original query; 2) the method boosts long tail queries as suggestion, and also generates suggestions for long tail queries despite of sparsity of data; 3) the method extracts suggestions that did not cooccur with original query; and 4) our method can be generalized to personalized query suggestion by simply embedding in any probabilistic personalized search methods. Experiments show that our method effectively generates semantic query suggestions as well as personalized query suggestions. The hitting time based method does not have a model parameter to tune, and can be easily transformed as a feature in existing query suggestion systems.

8. REFERENCES

[1] E. Agichtein, S. Lawrence, and L. Gravano. Learning search engine specific query transformations for question answering. In *Proceedings of the 10th international conference on World Wide Web*, pages 169–178, 2001.

[2] C. Anderson. *The Long Tail: Why the Future of Business is Selling Less of More*. Hyperion, 2006.

[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] D. Beeferman and A. Berger. Agglomerative clustering of a search engine query log. In *Proceedings of KDD '00*, pages 407–416, 2000.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.

[6] S. Chakrabarti, M. Joshi, and V. Tawde. Enhanced topic distillation using text, markup tags, and hyperlinks. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 208–216, 2001.

[7] K. Church and B. Thiesson. The wild thing! In *Proceedings of the ACL 2005 on Interactive poster and demonstration sessions*, pages 93–96, 2005.

[8] K. W. Church and B. Thiesson. The wild thing goes local. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 901–901, 2007.

[9] N. Craswell and M. Szummer. Random walks on the click graph. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 239–246, 2007.

[10] S. Cucerzan and E. Brill. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of EMNLP 2004*, pages 293–300, 2004.

[11] T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing pagerank, 2003.

[12] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 154–161, 2005.

[13] R. Jones, B. Rey, O. Madani, and W. Greiner. Generating query substitutions. In *Proceedings of the 15th international conference on World Wide Web*, pages 387–396, 2006.

[14] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632.

[15] M. Li, Y. Zhang, M. Zhu, and M. Zhou. Exploring distributional similarity based models for query spelling correction. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 1025–1032, 2006.

[16] Q. Mei and K. Church. Entropy of search logs: how hard is search? with personalization? with backoff? In *Proceedings of the international conference on Web search and web data mining*, pages 45–54, 2008.

[17] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 239–248, 2005.

[18] F. Radlinski and T. Joachims. Active exploration for learning rankings from clickthrough data. In *Proceedings of KDD' 07*, pages 570–579, 2007.

[19] D. Shen, T. Walkery, Z. Zhengy, Q. Yangz, and Y. Li. Personal name classification in web queries. In *Proceedings of the international conference on Web search and web data mining*, pages 149–158, 2008.

[20] X. Shen, B. Tan, and C. Zhai. Implicit user modeling for personalized search. In *Proceedings of CIKM' 05*, pages 824–831, 2005.

[21] J.-R. Wen, J.-Y. Nie, and H.-J. Zhang. Clustering user queries of a search engine. In *Proceedings of WWW '01*, pages 162–168, 2001.

[22] R. W. White and G. Marchionini. Examining the effectiveness of real-time query expansion. *Inf. Process. Manage.*, 43(3):685–704, 2007.

[23] G.-R. Xue, H.-J. Zeng, Z. Chen, Y. Yu, W.-Y. Ma, W. Xi, and W. Fan. Optimizing web search using web click-through data. In *Proceedings of CIKM '04*, pages 118–126, 2004.

[24] C. Zhai and J. Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of CIKM' 01*, pages 403–410, 2001.