A vlHMM Approach to Context-Aware Search

ZHEN LIAO, Nankai University
DAXIN JIANG, Microsoft Research Asia
JIAN PEI, Simon Fraser University
YALOU HUANG, Nankai University
ENHONG CHEN and HUANHUAN CAO, University of Science and Technology of China
HANG LI, Huawei Noah's Ark Lab

Capturing the context of a user's query from the previous queries and clicks in the same session leads to a better understanding of the user's information need. A context-aware approach to document reranking, URL recommendation, and query suggestion may substantially improve users' search experience. In this article, we propose a general approach to context-aware search by learning a *variable length hidden Markov model* (*vlHMM*) from search sessions extracted from log data. While the mathematical model is powerful, the huge amounts of log data present great challenges. We develop several distributed learning techniques to learn a very large vlHMM under the *map-reduce* framework. Moreover, we construct feature vectors for each state of the vlHMM model to handle users' novel queries not covered by the training data. We test our approach on a raw dataset consisting of 1.9 billion queries, 2.9 billion clicks, and 1.2 billion search sessions before filtering, and evaluate the effectiveness of the vlHMM learned from the real data on three search applications: document reranking, query suggestion, and URL recommendation. The experiment results validate the effectiveness of vlHMM in the applications of document reranking, URL recommendation, and query suggestion.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Search process*

General Terms: Algorithms, Experimentation

Additional Key Words and Phrases: Context-aware search, variable length hidden Markov model

ACM Reference Format:
Liao, Z., Jiang, D., Pei, J., Huang, Y., Chen, E., Cao, H., and Li, H. 2013. A vlHMM approach to context-aware search. ACM Trans. Web 7, 4, Article 22 (October 2013), 38 pages.
DOI: http://dx.doi.org/10.1145/2490255

1. INTRODUCTION

Capturing the context of a user's query from the previous queries and clicks in the same session leads to a better understanding of the user's information need. A context-aware approach to document reranking, query suggestion, and URL recommendation

A preliminary version of this article [Cao et al. 2009] appeared in *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*.
Z. Liao's research is supported in part by the Microsoft Research Asia Fellowship. J. Pei's research is supported in part by an NSERC Discovery Grant, a BCFRST NRAS Endowment Research Team Program project, and a GRAND NCE project. Y. Huang's research is supported in part by the National Natural Science Foundation of China under Grant No. 61105049. All opinions, findings, conclusions and recommendations in this article are those of the authors and do not necessarily reflect the views of the funding agencies.
Corresponding author's email: djiang@microsoft.com.

DOI: http://dx.doi.org/10.1145/2490255

Table I. Four Search Sessions from Real Search Logs

UID	SID	Search session
U_1	S_1	yahoo ↓ www.yahoo.com
U_2	S_2	gmail → hotmail → yahoo ↓ my.yahoo.com
U_3	S_3	dictionary → webster → online dictionary ↓ www.merriam-webster.com
U_4	S_4	citi bank → webster → chase bank ↓ www.websteronline.com

may substantially improve users' search experience. In this article, *context* is defined as the previous queries and clicks within the same session before a user's current query, and *session* is extracted by a widely used method [White et al. 2007]: two consecutive query events are segmented into two sessions if the time interval between them exceeds 30 minutes.

Consider the search sessions in Table I, which are several sessions of different users from Bing (`http://www.bing.com`). From the table, we can observe that users' searching and clicking behaviors are different in different context settings. First, users' behaviors can be different when searching with or without context. For example, when searching for "yahoo" alone, users are more likely to click the homepage of Yahoo!. When searching in a sequence "gmail → hotmail → yahoo", users may want to check personal emails. Second, users' behaviors can be different when searching in different contexts. For example, the next possible queries after searching for "dictionary → webster" are different from that of "citi bank → webster". Therefore, in the application of context-aware search, it is necessary to systematically capture the contexts.

Recently, several studies focused on mining users' search or browsing logs to improve users' search experience. For example, Joachims [2002], Radlinski and Joachims [2005], and Zhao et al. [2006] used click-through information to improve document ranking. White et al. [2007] provided URL recommendations to a user by finding the websites frequently visited by other users with similar information needs. Fonseca et al. [2005], Huang et al. [2003], and Jones et al. [2006] mined query pairs which were adjacent to each other or co-occurring in the same sessions, and used those query pairs to derive candidates for query expansion, suggestion, or substitution. To some extent, those studies are initiatives towards context-aware search.

Modeling query contexts by mining search sessions is a fundamental and challenging problem. Although some progress has been made by the previous studies, the state-of-the-art methods largely consider correlations within query pairs. Such a method cannot capture well the contexts indicated in the preceding example, which are carried by a series of queries and clicks. Moreover, each of the previous methods builds a model only for a specific search application. To achieve general context-aware search, we need a general model which can be used simultaneously for multiple applications, such as document reranking, query suggestion, and URL recommendation.

In this article, we propose modeling query contexts by a *variable length hidden Markov model* (*vlHMM*) for context-aware search. Using vlHMM, we can model each search intention as a state and consider the query and clicked URLs as observations generated by the state. Then, the whole search process can be modeled as a sequence of

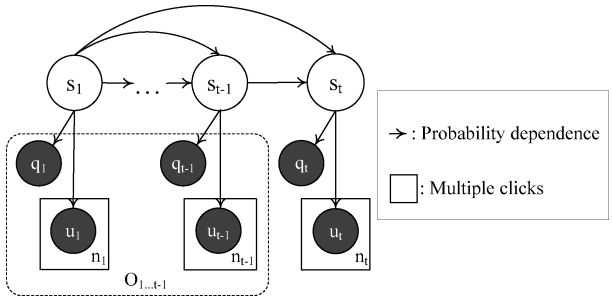


Fig. 1. Graphical representation of vlHMM.

transitions between states, as illustrated in Figure 1. Simply speaking, a search state contains different queries and URLs for the same search intention. It is possible that a state transits to itself.

In Figure 1, let q_t be the current query submitted by a user and $u_{t,i}$ $(1 \leq i \leq N_i)$ be the clicked URLs in the search result pages of q_t. The vlHMM infers conditional probability $P(s_t|q_t, O_{1\cdots t-1})$, where s_t is the current search intent and $O_{1\cdots t-1}$ is the context of q_t, which is captured by the past queries $q_1 \ldots q_{t-1}$, as well as the clicks $u_{1,*}, \ldots u_{t-1,*}$ for those queries. Based on the inferred search intent s_t, the vlHMM can rerank the search results according to the likelihood they would be clicked by users under the intent s_t. Moreover, the vlHMM can even predict the user's next search intent s_{t+1} by $P(s_{t+1}|q_t, O_{1\cdots t-1})$ and generate query suggestions and URL recommendations accordingly.

The probability distributions of s_t and s_{t+1} are inferred from not only the current query but also the whole context observed so far. As shown in the preceding example, given the current query *"webster"* alone, the probability of searching for the homepage of "Webster dictionary" is likely to be higher than that of searching for "Webster Bank". However, given the context that the users have searched "citi bank" before, the probability of searching for "Webster Bank" may increase, while the probability of searching for "Webster dictionary" may decrease. Consequently, the vlHMM can boost the "Webster bank" page and provide suggestions about online banking.

Learning hidden Markov models (HMM) and its variants have been well studied in many applications. However, applying HMM in context-aware search is far from trivial. Due to the huge amounts of search logs, it is costly to apply the existing algorithms. In commercial search engines, search logs may contain hundreds of millions of training examples. For instance, in the raw dataset of our experiments, there are 1.2 billion search sessions. Moreover, search logs may contain millions of unique queries and URLs, for example, 12.8 million unique queries and 48.4 million unique URLs in our experimental data, which make the number of model parameters extremely large. To tackle the challenge, we propose methods for grouping queries and URLs into states and initialize model parameters. In our experiments, the number of the states is 7,242,693 and the size of parameter space can be 10^{34}. We point out that most parameters in the space do not need to be calculated based on the observation that (1) many queries and URLs are not related and should not be emitted from the same state, and (2) the state transition matrix is sparse since many states do not co-occur in the training sessions. Therefore, the size of actual parameters can be reduced to 10^8. Then we adapt a distributed training process to learn a very large vlHMM model under the map-reduce mode. We further develop a series of techniques for improving the efficiency and scalability of the training process, including a method for identifying deterministic sessions, a method for detecting deterministic parameters, and a heuristic approach for data partition.

Another challenge for the vlHMM model is that users of a commercial search engine raise novel queries every day. For novel queries not observed in the training data, it is usually hard to estimate the probabilities they are generated by the hidden states. Therefore, it is hard to apply the trained model to those novel queries. To handle this challenge, we construct two types of feature vectors for each state in the vlHMM model. A machine learning approach is adapted to map novel queries to existing states.

After addressing the preceding challenges in training vlHMM and handling novel queries, vlHMM can be used online to support context-aware search in various applications, such as document reranking, query suggestion, and URL recommendation.

We make the following contributions in this article. First, we propose a novel model for supporting context-aware search. To the best of our knowledge, this is a first attempt to use the variable length hidden Markov model towards comprehensive modeling of context-aware search in the applications of query and document recommendation. Second, we develop efficient algorithms and strategies for learning a very large vlHMM from huge log data. Third, we develop effective feature vectors to handle novel queries. Finally, we test our approach on a real dataset of 1.9 billion queries, 2.9 billion clicks, and 1.2 billion search sessions before filtering. We evaluate the efficiency and scalability of our training algorithms and examine the effectiveness of the trained vlHMM on three search applications: document reranking, query suggestion, and URL recommendation. The experiment results validate the effectiveness of vlHMM in capturing contexts in the applications of document reranking, URL recommendation, and query suggestion.

The rest of the article is organized as follows. We discuss the related work in Section 2. We then introduce how to model search sessions through the vlHMM model in Section 3. The training method of a very large vlHMM and the applications in context-aware search are presented in Sections 4 and 5, respectively. In Section 6, we report the experimental results. Finally, we conclude this article and point out some future research directions in Section 7.

2. RELATED WORK

We summarize related works into following categories: (1) modeling user behaviors in search sessions; (2) query suggestion using click-through or session logs; (3) user interests and satisfaction prediction using session logs; (4) HMM and related models.

2.1. Modeling User Search Behaviors

Joachims et al. [2005] conducted studies on users' browse and click behavior and learned pairwise user preference on URLs from click-through data. Radlinski and Joachims [2005] extended the study to query chains where each query chain consists of a pair of adjacent queries in a session. The learned user preference was then used in the training of search result rankers [Joachims 2002]. Other than learning pairwise preference, several studies focused on modeling users' sequential clicks in query impressions [Craswell et al. 2008; Chapelle and Zhang 2009; Dupret and Piwowarski 2008]. For example, Craswell et al. [2008] proposed the *Cascade Model* to describe user clicks. Dupret and Piwowarski [2008] proposed the *User Browsing Model*, and Guo et al. [2009] proposed the *Dependent Click Model*. With the trained click models, the search results can be ranked according to their estimated relevance to the query. While the previous studies are effective for deriving user preference from users' click and browse behavior, most of them focused on user behavior with respect to individual queries. Although the study on query chains by Radlinski et al. [2005] involved pairs of adjacent queries, they did not systematically model the query context. Shen et al. [2005] proposed a language-model-based approach for context-sensitive retrieval using historical queries and clicked documents. Later, Xiang et al. [2010] studied different types of context information and developed four context-aware ranking principles. These studies [Shen

et al. 2005; Xiang et al. 2010] leveraged document content beyond the queries during the modeling of context. While these previous works focused on the single task of Web results ranking, our vlHMM model provides a unified framework for three tasks, that is, Web results ranking, URL recommendation and query suggestion.

Considering the multitasking behaviors in sessions, Jones and Klinkler [2008] proposed classifying query pairs into the same task via features based on time, word, Web search results, etc. Their approach achieved more than 90% accuracy in task boundary detection and same task identification. Lucchese et al. [2011] proposed identifying task-based sessions by combining content (query word, edit distance) and semantic (Wikipedia) features. Donato et al. [2010] proposed identifying those complex tasks as research missions which need users to explore multiple pages. Kotov et al. [2005] proposed modeling and analyzing cross-session search tasks, and they applied a classification approach to predict the re-visiting likelihood of different tasks. Liao et al. [2012] validated the effectiveness of task trails over sessions.

2.2. Query Suggestions

The methods using log data for query suggestion generally can be divided into two categories, namely, *document-click-based* approaches and *session-based* approaches. The document-click-based approaches focus on mining similar queries from a click-through bipartite graph constructed from search logs. The basic assumption is that two queries are similar to each other if they share a large number of clicked URLs. For example, Mei et al. [2008] performed a random walk starting from a given query q on the click-through bipartite graph to find queries similar to q. Other methods applied various clustering algorithms to the click-through bipartite graph. After the clustering process, the queries within the same cluster are used as suggestions for each other. For example, Beeferman and Berger [2000] applied a hierarchical agglomerative method to obtain similar queries in an iterative way. Wen et al. [2001] combined query content information and click-through information and applied a density-based method, DBSCAN [Ester et al. 1996], to cluster queries. Baeza-Yates et al. [2004] used the k-means to derive similar clusters. These approaches are effective for grouping similar queries but may have high computational costs and may not scale up to large data.

In the session-based approaches, query pairs that are often adjacent or co-occurring in the same sessions are mined as candidates for query suggestions. For example, Huang et al. [2003] mined co-occurring query pairs from session data and ranked the candidates based on their frequencies of co-occurrence with the user input queries. Jensen et al. [2006] considered not only the co-occurrence frequencies of the candidates, but also their mutual information with the user input queries. Boldi et al. [2008] built a *query-flow graph* where each node represents a distinct query and a directed edge from query q_i to query q_j represents that at least one user submitted query q_j immediately after submitting q_i in the same session. An edge (q_i, q_j) is also associated with a weight to indicate how likely a user moves from q_i to q_j. To generate query suggestions, the edge weights were estimated by the frequencies of observed transitions in search logs, and a straightforward method is to return the queries q_j which have the largest weights of edges (q_i, q_j). Alternative methods proposed [Boldi et al. 2008] conduct random walks starting from either the given query q_i or the last k queries visited before q_i in the same session. Several studies extended the work in Boldi et al. [2008] along various directions. For example, Boldi et al. [2009] suggested labeling the edges in a query-flow graph into four categories, namely, generalization, specialization, error correction, and parallel move, and only using the edges labeled as specialization for query suggestion. Anagnostopoulos et al. [2010] argued that providing query suggestions to users may change user behavior. They thus modeled query suggestions as shortcut links on a query-flow graph and considered the resulted graph as a perturbed version of the

original one. Then the problem of query suggestion was formalized as maximizing the utility function of the paths on the perturbed query-flow graph. Sadikov et al. [2010] extended the query-flow graph by introducing the clicked documents for each query. The queries q_j following a given query q_i are clustered together if they share many clicked documents.

Unlike these preceding session-based methods which only focus on query pairs, we model a variable-length context of the current query. Although Boldi et al. [2008] and Huang et al. [2003] used the adjacent queries to select candidates for query suggestions, they did not model the sequential relationship between the queries of a session in a systematic way. Those studies could be regarded as initiatives to model partial contexts of queries using simple mechanisms.

2.3. User Interests and Satisfaction Prediction

User interests can be predicted using similar behavior of other users or context information. White et al. [2007] proposed mining frequently visited websites from browse logs and recommending to a user the URLs frequently visited by other users with similar information needs. White et al. [2010] assigned the topics from the taxonomy created by the Open Directory Project (`http://www.dmoz.org/`) to three types of context. The first type considered the preceding queries only, while the second and third types added clicked and browsed documents by the user. The authors confirmed that user interests are largely consistent within a session, and thus context information has good potential to predict the users' short-term interests. White et al. [2009] explored various sources of contexts in browsing logs and evaluated their effectiveness for the prediction of user interests. For example, besides the preceding pages browsed within the current session, the authors also considered the pages browsed in a long history, the pages browsed by other users with the same interests, and so on. They found a combination of multiple sources performed better than a single source. Mei et al. [2009] proposed using query sequences in sessions for four types of tasks, including sequence classification, sequence labeling, sequence prediction, and sequence similarity. They found that many tasks, such as segmenting queries in sessions according to use interests, could benefit from the use of context information.

User behaviors in session trails can also be used to determine user satisfaction. Fox et al. [2005] studied relationships between implicit feedback signals and explicit user satisfaction ratings. They collected sessions with labeled information from 146 participants by IE add-on. According to gene analysis on patterns of user behaviors, they found that dwell time on search result pages is a good indicator for user satisfaction. Hassan et al. [2010] further formulated user search processes by Markov models and learned successful and unsuccessful user search behavior models.

2.4. Hidden Markov Model

HMMs have been widely used in various domains, such as speech recognition [Rabiner 1989] and bioinformatics [Durbin et al. 1999]. Wang et al. [2006] developed the notion of vlHMM and applied a vlHMM to mine four kinds of interesting patterns from 3D motion capture data. We use a vlHMM to model query contexts in this article. Our approach is critically different from that of Wang et al. [2006] because our vlHMM automatically adapts to users' search sessions instead of learning an optimized set of contexts. The training of HMMs has also been well studied in the literature. The classical learning algorithm is the Baum-Welch algorithm [Baum et al. 1970], which is essentially an EM algorithm [Dempster et al. 1977]. However, the existing methods cannot be scaled up to huge log data because of their high computational complexity. Recently, parallel and distributed training of very large machine learning models has attracted much interest. For example, Chu et al. [2006] applied the *map-reduce* programming model

[Dean and Ghemawat 2004] to a variety of learning algorithms. However, how to train HMMs from huge logs remains a challenging open problem. This article introduces a series of techniques for improving the scalability and efficiency of the training process of vlHMM. First, to handle very large-scale search log data, we develop a series of techniques and substantially improve the scalability and the efficiency of the training process. Second, we present new techniques for handling novel queries that are not contained in the training data. We also improve the parameter initialization method and report the empirical evaluation results on a much larger real dataset.

3. VLHMM MODEL

In this section, we describe the variable length hidden Markov model for modeling user search intents in a session. We consider search intent as atomic user information need which can be satisfied by one or several webpages. Here *atomic user information need* represents the low level of user information need. For instances, the search intent may be finding the homepage of an organization, finding the lyrics of a song, or finding the entry of E-mail login, but not as a high level as planning for a trip which may contain booking tickets, choosing a hotel, and renting a car. We assume that queries and URLs are conditionally independent given an intent. This is a natural assumption in search. For example, given that the intent is to find the homepage of "*Microsoft Research Asia*", the conditional probabilities of raising the queries of "*Microsoft Research Asia*", "*MSRA*", and "*MS Research Beijing*" should be the same with and without considering the URL. The converse is true as well. Given that the intent is to find the homepage, the conditional probabilities of clicking the URL of "`www.msra.cn`" should be the same with and without considering the queries.

We choose a hidden Markov model rather than a Markov chain to model query intents, since search intents are not directly observable. If we model individual queries and URLs directly as states in a Markov chain, we have to use a large number of states and thus the complexity of the model is high, and we also lose the semantic relationship among the queries and the clicked URLs under the same search intent. Therefore, to achieve better modeling, we assume that queries and clicks are generated by some hidden states where each hidden state corresponds to one search intent.

There are different types of HMMs. The first order HMM (1-HMM) has been widely used in various applications, such as speech recognition [Rabiner 1989] and bioinformatics [Durbin et al. 1999]. For context-aware search, we choose higher-order HMMs. This is because 1-HMM assumes that the probability distribution of the state s_t is independent of the previous states $s_1, \ldots, s_{t-2}$, given the immediately previous state s_{t-1}. In search processes, this assumption usually does not hold. For example, given that a user searched for Hotmail at time point $t-1$, the probability $P_{Webster}$ that the user searches for "Webster Dictionary" at the current time point t may still depend on some of the previous states, such as s_{t-2}, s_{t-3}, etc. For instance, the probability of searching "Webster Dictionary" may be smaller if the user searched for "citi bank" at a time point before $t-1$. Therefore we consider employing higher-order HMMs rather than 1-HMM, since vlHMM is a natural and flexible combination of all-order HMMs.

Given a set of hidden states $\{s_1, \ldots, s_{N_s}\}$, a set of queries $\{q_1, \ldots, q_{N_q}\}$, a set of URLs $\{u_1, \ldots, u_{N_u}\}$, and the maximal length T_{max} of state sequences, a variable length hidden Markov model is defined as follows.

—The transition probability distribution $\Delta = \{P(s_i|S_j)\}$, where S_j is a state sequence with length $T_j < T_{max}$, $P(s_i|S_j)$ is the probability that a user transits to state s_i given the previous states $s_{j,1}, s_{j,2}, \ldots, s_{j,T_j}$, and $s_{j,t}(1 \le t \le T_j)$ is the tth state in sequence S_j.

—The initial state distribution $\Psi = \{P(s_i)\}$, where $P(s_i)$ is the probability that state s_i occurs as the first element of a state sequence.
—The emission probability distribution for each state sequence $\Lambda = \{P(q, U|S_j)\}$, where q is a query, U is a set of URLs, S_j is a state sequence of length $T_j \leq T_{max}$, and $P(q, U|S_j)$ is the joint probability that a user raises the query q and clicks the URLs U from state s_{j,T_j} after the user's $(T_j - 1)$ steps of transitions from state $s_{j,1}$ to s_{j,T_j}.

We assume that query q and URLs U are conditionally independent given the state s_{j,T_j}, that is, $P(q, U|s_{j,T_j}) \equiv P(q|s_{j,T_j}) \prod_{u \in U} P(u|s_{j,T_j})$. Under these assumptions, the emission probability distribution Λ becomes $(\Lambda_q, \Lambda_u) \equiv (\{P(q|s_i)\}, \{P(u|s_i)\})$. First, as described before in Section 3, when the search intent is given, it can be assumed that the queries and URLs are conditionally independent. Second, the assumption can reduce the computation cost. Since the number of parameters in the distribution $P(q, U|s_{j,T_j})$ is larger than that of the distributions $P(q|s_{j,T_j})$ and $P(u|s_{j,T_j})$.

Our task of training a vlHMM model is to learn the parameters $\Theta = (\Psi, \Delta, \Lambda_q, \Lambda_u)$ from search logs. A search log is basically a sequence of query events and click events. As mentioned previously, we separate consecutive query events into different sessions based on widely used 30-minute time cut-off threshold. Note that the click events for a query are always assigned to the same session with the corresponding query event.

Since the parameters Θ contain un-observed hidden states, we choose the expectation maximization algorithm (EM) for parameter inference. The parameter inference process is described as follows.

Let $\mathcal{X} = \{O_1, \ldots, O_N\}$ be the set of training sessions, where a session O_n $(1 \leq n \leq N)$ of length T_n is a sequence of pairs $\langle (q_{n,1}, U_{n,1}) \ldots (q_{n,T_n}, U_{n,T_n}) \rangle$, and $q_{n,t}$ and $U_{n,t}$ $(1 \leq t \leq T_n)$ are the tth query and the set of clicked URLs among the query results, respectively. Moreover, we use $u_{n,t,k}$ to denote the kth URL $(1 \leq k \leq |U_{n,t}|)$ in $U_{n,t}$. Let $\mathcal{Y} = \{S_1 \ldots, S_M\}$ be the set of all possible state sequences, $s_{m,t}$ be the tth state in $S_m \in \mathcal{Y}$ $(1 \leq m \leq M)$, and S_m^{t-1} be the subsequence $s_{m,1}, \ldots, s_{m,t-1}$ of S_m.

Basically, we want to find Θ to maximize the likelihood function L as follows.

$$L = \ln P(\mathcal{X}|\Theta) = \sum_n \ln P(O_n|\Theta) = \sum_n \ln \sum_m P(O_n, S_m|\Theta). \tag{1}$$

Then we construct an auxiliary function $Q(\Theta^{r+1}, \Theta^r)$ to iteratively update Θ for maximizing L [Bilmes 1998].

$$Q(\Theta^{r+1}, \Theta^r) = \mathbf{E}[\ln P(\mathcal{X}, \mathcal{Y}|\Theta^{r+1})|\mathcal{X}, \Theta^r] = \sum_{n,m} P(S_m|O_n, \Theta^r) \ln P(O_n, S_m|\Theta^{r+1}), \tag{2}$$

where Θ^r is the set of parameter values obtained in the last round of iteration.

The E-step estimates the expectation of hidden states based on obtained Θ^r as follows.

$$P(S_m|O_n, \Theta^r) = \frac{P(O_n, S_m|\Theta^r)}{P(O_n|\Theta^r)}, \tag{3}$$

where $P(O_n|\Theta^r) = \sum_{S_m} P(S_m, O_n|\Theta^r)$ and $P(O_n, S_m|\Theta^r) = P(O_n|S_m, \Theta^r) \cdot P(S_m|\Theta^r) = (\prod_{t=1}^{T_n} P^r(q_{n,t}|s_{m,t}) \prod_k P^r(u_{n,t,k}|s_{m,t})) \cdot (P^r(s_{m,1}) \prod_{t=2}^{T_n} P^r(s_{m,t}|S_m^{t-1}))$.

The M-step maximizes $Q(\Theta^{r+1}, \Theta^r)$ by the following formulas.

$$P^{r+1}(s_i) = \frac{\sum_{n,m} P\big(S_m|O_n, \Theta^r\big)\delta(s_{m,1} = s_i)}{\sum_{n,m} P\big(S_m|O_n, \Theta^r\big)}. \tag{4}$$

Table II. An Example of a Hidden State and the Initial Emission Probabilities

Queries	$P^0(q\|s)$
city of bothell	0.52
bothell wa	0.20
bothell washington	0.14
city of bothell washington	0.07
city bothell washington	0.01
URLs	$P^0(u\|s)$
ci.bothell.wa.us	0.21
bothellwashington.com	0.17
ci.bothell.wa.us/dept/pd/pdindex.html	0.15
explorebothell.com	0.14
nwmaps.net/bothell	0.05
ihsadvantage.com/h/hotels/bothell/wa/us	0.02
dianasflowers.com	0.01

$$P^{r+1}(q|s_i) = \frac{\sum_{n,m} P\left(S_m|O_n, \Theta^r\right) \sum_t \delta(s_{m,t} = s_i \wedge q = q_{n,t})}{\sum_{n,m} P\left(S_m|O_n, \Theta^r\right) \sum_t \delta(s_{m,t} = s_i)}. \tag{5}$$

$$P^{r+1}(u|s_i) = \frac{\sum_{n,m} P\left(S_m|O_n, \Theta^r\right) \sum_t \delta(s_{m,t} = s_i \wedge u \in U_{n,t})}{\sum_{n,m} P\left(S_m|O_n, \Theta^r\right) \sum_t \delta(s_{m,t} = s_i)}. \tag{6}$$

$$P^{r+1}(s_i|S_j) = \frac{\sum_{n,m} P\left(S_m|O_n, \Theta^r\right) \delta\left(\exists t\ S_m^{t-1} = S_j \wedge s_{m,t} = s_i\right)}{\sum_{n,m} P\left(S_m|O_n, \Theta^r\right) \delta\left(\exists t\ S_m^{t-1} = S_j\right)}. \tag{7}$$

In these equations, $\delta(p)$ is a boolean function which equals to one if the predicate p is true and zero otherwise.

4. TRAINING A VERY LARGE VLHMM

Although the EM algorithm has been widely used to train HMMs, there are still several challenges to applying it on huge search log data.

First, the EM algorithm needs a user-specified number of hidden states. However, in our problem, the hidden states correspond to user search intents whose number is unknown. Based on the explanation of search intent in Section 3, it is natural to group similar queries which have the same clicked webpages as search intent. To address this challenge, we apply the mining techniques developed in Liao et al. [2011] as a prior process to the parameter learning process. To be specific, we construct a click-through bipartite graph and derive a collection of query clusters [Liao et al. 2011]. For each cluster Q of queries, we find all URLs U such that each URL $u \in U$ is connected to at least one query $q \in Q$ in the click-through bipartite graph. A unit of query and URL cluster (Q, U) will be extended to correspond to a hidden state. The total number of hidden states is then determined by the total number of query clusters. For example, Table II shows a state which is mined from a real dataset. In our experiments, there are more than 7 million states.

Second, a search log often contains a huge amount of training sessions. For example, in our experiments, the training data set contains 1.2 billion sessions. Even after pruning, the distinct number of sessions is more than 100 million. It is impractical to learn a vlHMM from such a huge training dataset using a single machine. To address this challenge, we deploy the learning task on a distributed system under the

map-reduce programming model [Dean and Ghemawat 2004]. We will describe the *map* stage and the *reduce* stage in Section 4.1.

Third, it is well known that the EM algorithm converges at a local maximum of the data likelihood [Dempster et al. 1977]. Therefore, it is critical to set appropriate initial values for model parameters. Moreover, since the log data usually contains millions of unique queries and URLs, the space of parameters is extremely large. For example, the real dataset used in our experiments leads to more than 10^{34} parameters. To handle those challenges, we develop a special strategy for model parameter initialization based on the clusters mined from the click-through bipartite graph. As shown in Table II, the queries and URLs with nonzero emission probabilities from the same state are semantically related. Moreover, we will show in Section 4.2 that our initialization strategy can greatly reduce the number of parameters to be reestimated. Theoretically, the number has an upper bound. Based on our proposed initialization method, the magnitude of parameters can be reduced to 10^8.

Fourth, although the map-reduce programming model and our initialization strategy make it feasible to train a very large vlHMM, the huge volume of the session data and the extremely large number of model parameters still present great challenges for the performance of the training process. To further improve the efficiency and scalability of our training process, we develop several techniques in Section 4.3, including the methods to identify deterministic sessions and deterministic parameters, as well as a heuristic for data partition.

Finally, a search log only covers the historical queries and clicks. In the online stage, users raise novel queries every day. Therefore, the vlHMM model learned offline from search log data cannot handle the novel queries received online. To address this problem, we construct two types of feature vector for each state in Section 4.4. Those feature vectors will be used to capture users' novel queries.

ALGORITHM 1: The EM Training Process under the Map-Reduce Programming Model

Input: training data $\mathcal{X}$; initial model parameters Θ^0

1: partition $\mathcal{X}$ into subsets $\{\mathcal{X}_k\}$;
2: **while not** converged **do**
3: call **map** and **reduce**

Function: map(a subset of training data $\mathcal{X}_k$, model parameters Θ^r)

1: **load** model parameters Θ^{r-1};
2: **for each** session $O_n \in \mathcal{X}_k$ **do**
3: **for each** possible state sequence S_m **do**
4: **let** $p_{n,m} = P(S_m|O_n, \Theta^r)$ by Equation 3;
5: **emit** key/value pairs as in Table III;

Function: reduce(intermediate key k, value list $valList$)
// each element in $valList$ is a tuple (val_1, val_2)

1: **let** $sum_1 = 0$, sum_2=0;
2: **for each** element (val_1, val_2) in $valList$ **do**
3: $sum_1 += val_1$; $sum_2 += val_2$;
4: **output**(k, sum_1/sum_2);

4.1. Distributed Learning of Parameters

Map-reduce is a programming model for distributed processing of large datasets [Dean and Ghemawat 2004]. In this article, the *map* and *reduce* steps correspond to "expectation" (E-step) and "maximization" (M-step), respectively. In the map stage, each machine (called a *process node*) receives a subset of data as input and produces a set of intermediate key/value pairs. In the reduce stage, each process node merges

Table III. Key/Value Pairs Emitted at the Map Stage

Key	Value
s_i	$Value_{n,1} = \sum_m P_{n,m}\delta(s_{m,1} = s_i)$ $Value_{n,2} = \sum_m P_{n,m}$
(s_i, q_j)	$Value_{n,1} = \sum_m P_{n,m} \sum_t \delta(s_{m,t} = s_i \wedge q_j = q_{n,t})$ $Value_{n,2} = \sum_m P_{n,m} \sum_t \delta(s_{m,t} = s_i)$
(s_i, u_j)	$Value_{n,1} = \sum_m P_{n,m} \sum_t \delta(s_{m,t} = s_i \wedge u_j \in U_{n,t})$ $Value_{n,2} = \sum_m P_{n,m} \sum_t \delta(s_{m,t} = s_i)$
(s_i, S_j)	$Value_{n,1} = \sum_m P_{n,m}\delta(\exists t\ S_m^{t-1} = S_j \wedge s_{m,t} = s_i)$ $Value_{n,2} = \sum_m P_{n,m}\delta(\exists t\ S_m^{t-1} = S_j)$

all intermediate values associated with the same intermediate keys and outputs the final computation results.

In our training process (see Algorithm 1), we first partition the training data into subsets and distribute each subset to a process node. In the map stage, each process node scans the assigned subset of training data once. For each training session O_n, the process node infers the posterior probability $P_{n,m} = P(S_m|O_n, \Theta^r)$ by Equation (3) for each possible state sequence S_m and emits the key/value pairs, as shown in Table III.

In the reduce stage, each process node collects all values for an intermediate key. For example, suppose a key (s_i, q_j) is assigned to process node pn_k. Then pn_k receives a list of values $\{(Value_{n,1}, Value_{n,2})\}$ $(1 \le n \le N)$ and derives $P(q_j|s_i)$ by $\frac{\sum_n Value_{n,1}}{\sum_n Value_{n,2}}$. The other parameters, $P(q|s_i)$, $P(u|s_i)$, and $P(s_i|S_j)$ are computed in a similar way.

4.2. Assigning Initial Values

In the vlHMM model, we have four sets of parameters, the initial state probabilities $\{P(s_i)\}$, the query emission probabilities $\{P(q|s_i)\}$, the URL emission probabilities $\{P(u|s_i)\}$, and the transition probabilities $\{P(s_i|S_j)\}$. Let the number of states be N_s, the number of unique queries be N_q, the number of unique URLs be N_u, and the maximal length of a training session be T_{max}. We can derive that $|\{P(s_i)\}| = N_s$, $|\{P(q|s_i)\}| = N_s \cdot N_q$, $|\{P(u|s_i)\}| = N_s \cdot N_u$, $|\{P(s_i|S_j)\}| = \sum_{t=2}^{T_{max}} (N_s)^t$, and the total number of parameters is $\mathcal{N} = N_s \cdot (N_q + N_u + \sum_{t=2}^{T_{max}} (N_s)^{t-1})$. How to assign the initial values for those model parameters presents great challenges. First, since the EM algorithm converges at a local maximum, the initial values greatly influence the training results. Therefore, the first challenge is how to assign appropriate initial parameter values. Second, since a search log may contain millions of unique queries and URLs, and there may be millions of states derived from the click graph, the number of parameters can be extremely large. Thus, the second challenge is how to reduce the number of parameters that need to be reestimated in the EM iterations.

Our idea is to take the advantage on the semantic correlation among queries, URLs, and search intents. For example, a user is unlikely to raise the query *"Harry Potter"* to search for the official website of the Beijing Olympic 2008. Similarly, a user who raises query *"Beijing Olympic 2008"* is unlikely to click on the URL `harrypotter.warnerbros.com`. This observation suggests that although we have a huge space of possible parameters, the optimal solution is sparse—the values of most emission and transition probabilities are zero.

To reflect the inherent relationship among queries, URLs, and search intents, we can assign the initial parameter values based on the correspondence between a cluster $C_i = (Q_i, U_i)$ and a state s_i, where Q_i is a query cluster derived from a click-through bipartite graph by the method in Cao et al. [2008], and U_i is the set of URLs such that each URL $u \in U_i$ is connected to at least one query $q \in Q_i$ in the click-through bipartite

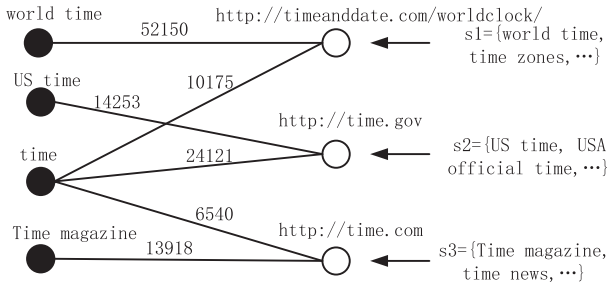


Fig. 2. URLs and search intents of query *"time"*.

Table IV. Initial Query and URL Emission Probability via Random Walks [Cao et al. 2009] and the New Method

	Random Walk Initialization			New Initialization		
Queries	$P^0(q\|s_1)$	$P^0(q\|s_2)$	$P^0(q\|s_3)$	$P^0(q\|s_1)$	$P^0(q\|s_2)$	$P^0(q\|s_3)$
time	0.3624	0.3163	0.7135	0.1684	0.5612	0.2662
world time	0.4628	0.404	0	0.7199	0.0397	0
US time	0.1264	0.1104	0	0.0165	0.3215	0
Time magazine	0	0.1078	0.2432	0	0.0201	0.5663
URLs	$P^0(u\|s_1)$	$P^0(u\|s_2)$	$P^0(u\|s_3)$	$P^0(u\|s_1)$	$P^0(u\|s_2)$	$P^0(u\|s_3)$
u_1 = timea...clock	0.5807	0.5069	0.1778	0.9032	0.0498	0
u_2 = time.gov	0.3612	0.3153	0.4215	0.0471	0.9181	0
u_3 = time.com	0.058	0.1721	0.3883	0	0.032	0.9044

Note: We omit some queries and URLs with nonzero emission probabilities for conciseness of presentation.

graph. As illustrated in Table II, the queries Q_i and the URLs U_i of a cluster C_i are semantically correlated and jointly reflect the search intent represented by state s_i. One possible method is to assign a nonzero probability to $P(q|s_i)$ and $P(u|s_i)$, respectively, if $q \in C_i$ and $u \in C_i$. However, such assignments make the model deterministic since each query can belong to only one cluster. In practice, many queries are ambiguous and should belong to multiple clusters.

Consider the example of query *"time"*, as shown in Figure 2. From the log data, we can observe that users who issued query "*time*" clicked on three URLs u_1 = `http://timeanddate.com/worldclock`, u_2 = `http://time.gov`, and u_3 = `http://time.com` for 10,175, 24,121, and 6,540 times, respectively. We further examine the content of these three webpages and find the pages are about "world time", "the U.S. time", and the "Time magazine", respectively. Therefore, *"time"* is an ambiguous query and ideally it should be grouped together with not only query *"world time"* in cluster C_1, but also query *"U.S. time"* in cluster C_2, and further query *"time magazine"* in cluster C_3. However, the clustering algorithm in Cao et al. [2008] will assign *"time"* only to cluster C_2, since u_2 = `http://time.gov` receives the largest number of clicks.

To allow an ambiguous query to belong to multiple concepts, one approach is to conduct random walks on click-through graphs. As shown in our previous study [Cao et al. 2009], we can limit random walks within two steps. First, we expand each cluster $C_i = (Q_i, U_i)$ into $C_i' = (Q_i', U_i')$, where Q_i' is a set of queries such that each query $q' \in Q_i'$ is connected to at least one URL $u \in U_i$ in the click-through bipartite graph, and U_i' is a set of URLs such that each URL $u' \in U_i'$ is connected to at least one query $q' \in Q_i'$. Then, we assign $P^0(q|s_i) = \frac{\sum_{u' \in U_i'} Count(q,u')}{\sum_{q' \in Q_i'} \sum_{u' \in U_i'} Count(q',u')}$ and $P^0(u|s_i) = \frac{\sum_{q' \in Q_i'} Count(q',u)}{\sum_{q' \in Q_i'} \sum_{u' \in U_i'} Count(q',u')}$,

where $Count(\cdot, \cdot)$ is the number of times that a URL is clicked as an answer to a query in the search log. Table IV shows the initialization results for the example query *"time"* in Figure 2. As we can see, query *"time"* is assigned to all the related states with nonzero initial parameters. However, the random walk approach also brings in noises. For example, state s_1 has nontrivial emission probabilities for two URLs: `http://timeanddate.com/worldclock` and `http://time.gov`. Intuitively, those URLs are about different topics and should not be generated by the same state.

To address these challenges, we revise the method in Cao et al. [2009]. Instead of expanding both Q_i and U_i, our new method only expands the query set Q_i but fixes the URL set U_i. The rationale is that queries are often ambiguous and bearing multiple intents, while URLs are relatively focused and usually about a single topic. In the following, we use the example of query *"time"* in Figure 2 to explain the details of our new initialization process. For each cluster $C_i = (Q_i, U_i)$, we first initialize the $P^0(u|s_i) \propto \frac{\sum_{q \in Q_i} P(u|q)}{|Q_i|}$. This method lets every query in the state vote for the likelihood of $P^0(u|s_i)$ and avoid the probability being biased by popular queries. Next, we initialize the query emission probabilities by $P^0(q|s_i) \propto \sum_{u \in U_i} P(q|u) \cdot P^0(u|s_i)$, where $P(q|u)$ estimates the likelihood that u transits to q by $\frac{Count(q,u)}{\sum_{q' \in Q_i} Count(q',u)}$. The idea is that if all URLs in state s_i are more often clicked as answers to query q, then q is more likely to be generated by the state. Consequently, the new initialization results of *"time"* example are also shown in Table IV. As a result, about 5% (658,977 out of 12,769,284) unique queries have nonzero emission probabilities from multiple states.

The initial emission probabilities are not only semantically meaningful. Moreover, they have the following nice properties.

LEMMA 4.1. *The query emission probability at the rth round of iteration $P^r(q|s_i) = 0$, if the initial value $P^0(q|s_i) = 0$.*

PROOF. We focus on the numerator of Equation (5). For any pair of O_n and S_m, if O_n does not contain query q, the numerator is zero, since $\sum_t \delta(s_{m,t} = s_i \wedge q_{n,t} = q) = 0$.

Suppose O_n contains query q. Without loss of generality, suppose q appears in O_n only at step t_1, that is, $q_{n,t_1} = q$. If $s_{m,t_1} \neq s_i$, then the numerator is zero, since $\sum_t \delta(s_{m,t} = s_i \wedge q_{n,t} = q) = \delta(s_{m,t_1} = s_i \wedge q_{n,t_1} = q) = 0$. Otherwise, if $s_{m,t_1} = s_i$ and $q_{n,t_1} = q$, then $P(O_n|S_m, \Theta^{r-1}) = P^{r-1}(q|s_i) \cdot (\prod_{t \neq t_1} P^{r-1}(q_{n,t}|s_{m,t})) \cdot (\prod_t P^{r-1}(U_{n,t}|s_{m,t}))$. Therefore, if $P^{r-1}(q|s_i) = 0$, then $P(O_n|S_m, \Theta^r) = 0$, and thus $P(S_m|O_n, \Theta^r) = 0$ (Equation (3)).

In summary, for any O_n and S_m, if $P^{r-1}(q|s_i) = 0$, then $P(S_m|O_n, \Theta^{r-1}) \cdot \sum_t \delta(s_{m,t} = s_i \wedge q_{n,t} = q) = 0$, and thus $P^r(q|s_i) = 0$. Therefore, $P^r(q|s_i) = 0$ if $P^0(q|s_i) = 0$. □

Similarly, we can prove the following lemmas.

LEMMA 4.2. *The URL emission probability at the rth round of iteration $P^r(u|s_i) = 0$, if the initial value $P^0(u|s_i) = 0$.*

LEMMA 4.3. *The state initial probability at the rth round of iteration $P^r(s_i) = 0$, if the initial value $P^0(s_i) = 0$.*

LEMMA 4.4. *The transition probability at the rth round of iteration $P^r(s_i|S_j) = 0$, if the initial value $P^0(s_i|S_j) = 0$.*

Based on these lemmas, for each training session O_n, we can construct a set of *candidate state sequences* Γ_n which are likely to generate O_n.

Definition 4.5 (*Candidate States*). Let O_n be the nth training example, T_n be the length of O_n, and $O_{n,t} = (q_{n,t}, U_{n,t})$ be the tth ($1 \leq t \leq T_n$) observation of O_n, where

$q_{n,t}$ and $U_{n,t}$ be the tth query and the tth set of clicked URLs in O_n, respectively. The set of *candidate states* $Cand_{n,t}$ for $O_{n,t}$ includes all the states s such that $(P^0(q_{n,t}|s) \neq 0) \wedge (\forall_{u_{n,t,k} \in U_{n,t}} P^0(u_{n,t,k}|s) \neq 0)$.

In the following, we extend the definition of candidate states for the tth observation $O_{n,t}$ to the definition of candidate state sequences for the whole session O_n.

Definition 4.6 (*Candidate State Sequences*). Let O_n^t be the sequence of the first t observations of the nth training example O_n and T_n be the length of O_n. The set of *candidate state sequences* Γ_n^t for O_n^t is the Cartesian product of the candidate state sets $Cand_{n,1}, \ldots, Cand_{n,t}$, that is, $\Gamma_n^t = \{s_{n,1} \ldots s_{n,t} | s_{n,i} \in Cand_{n,i}, 1 \leq i \leq t \leq T_n\}$, where $Cand_{n,i}$ is the set of candidate states for the ith observation of O_n. In particular, when $t = T_n$, $\Gamma_n^{T_n}$ is simply denoted by Γ_n.

LEMMA 4.7. *Let O_n be the nth training example and Γ_n be the set of candidate state sequences of O_n. $\forall S_m \notin \Gamma_n$, $P(S_m|O_n, \Theta^{(r-1)}) = 0$.*

PROOF. According to the definition of candidate state sequences Γ_n (Definition 4.6), for any $S_m \notin \Gamma_n$, there must exist t $(1 \leq t \leq T_n)$ such that $s_{m,t} \notin Cand_{n,t}$, where T_n is the length of O_n, $s_{m,t}$ is the tth state in S_m, and $Cand_{n,t}$ is the set of candidate states for the tth observation of O_n. From the definition of candidate states $Cand_{n,t}$ (Definition 4.5), if $s_{m,t} \notin Cand_{n,t}$, then either $P^0(q_{n,t}|s_{m,t}) = 0$ or $\exists u_{n,t,k} \in U_{n,t}$ such that $P^0(u_{n,t,k}|s_{m,t}) = 0$). In other words, if $s_{m,t} \notin Cand_{n,t}$, then $P^0(q_{n,t}|s_{m,t}) \prod_k P^0(u_{n,t,k}|s_{m,t}) = 0$. Therefore, we have

$$
\begin{aligned}
&\forall S_m \notin \Gamma_n, \prod_{t=1}^{T_n} P^0(q_{n,t}|s_{m,t}) \prod_k P^0(u_{n,t,k}|s_{m,t}) = 0 \\
\Rightarrow &\forall S_m \notin \Gamma_n, \prod_{t=1}^{T_n} P^{r-1}(q_{n,t}|s_{m,t}) \prod_k P^{r-1}(u_{n,t,k}|s_{m,t}) = 0 \qquad \text{(Lemmas 4.1 and 4.2)} \\
\Rightarrow &\forall S_m \notin \Gamma_n, P(O_n, S_m|\Theta^{r-1}) = 0 \\
\Rightarrow &\forall S_m \notin \Gamma_n, P(S_m|O_n, \Theta^{r-1}) = 0. \qquad \text{(Equation (3)) } \square
\end{aligned}
$$

Lemma 4.7 suggests that for each training session O_n, only the state sequences in Γ_n are possible to contribute to the update of parameters in Equations (4)–(7).

After constructing candidate state sequences, we assign the values to $P^0(s_i)$ and $P^0(s_i|S_j)$ as follows. First, we compute the whole bag of candidate state sequences $\Gamma^+ = \Gamma_1 + \cdots + \Gamma_N$, where '+' denotes the *bag union* operation, and N is the total number of training sessions. We then assign $P^0(s_i) = \frac{Count(s_i)}{|\Gamma^+|}$ and $P^0(s_i|S_j) = \frac{Count(S_j \circ s_i)}{Count(S_j)}$, where $Count(s_i)$, $Count(S_j)$, $Count(S_j \circ s_i)$ are the numbers of the sequences in Γ^+ that start with state s_i, subsequence S_j, and the concatenations of S_j and s_i, respectively.

This initialization limits the number of *active parameters* (i.e., the nonzero parameters in the EM iterations of the training process) to an upper bound $\mathcal{C}$, as indicated in the following theorem.

THEOREM 4.8. *Given training sessions $\mathcal{X} = \{O_1 \ldots O_N\}$ and the initial values assigned to parameters, as described in this section, the number of parameters updated in one iteration of the training of a vlHMM is at most $\mathcal{C} = N_s \cdot (1 + \overline{N_{sq}} + \overline{N_{su}}) + |\Gamma| \cdot (\overline{T} - 1)$, where N_s is the number of states, $\overline{N_{sq}}$ and $\overline{N_{su}}$ are the average sizes of $\{P^0(q|s_i) | P^0(q|s_i) \neq 0\}$ and $\{P^0(u|s_i) | P^0(u|s_i) \neq 0\}$ over all states s_i, respectively, Γ is the set of unique state sequences in Γ^+, and $\overline{T}$ is the average length of the state sequences in Γ.*

PROOF. Let $\widetilde{\Psi^i}$, $\widetilde{\Delta^i}$, $\widetilde{\Lambda_q^i}$ and $\widetilde{\Lambda_u^i}$ be the sets of active initial state probabilities, transition probabilities, query and URL emission probabilities in the ith iteration, respectively. Using Lemmas 4.1 and 4.2, we immediately have $|\widetilde{\Lambda_q^i}| \leq N_s \cdot \overline{N_{sq}}$ and $|\widetilde{\Lambda_u^i}| \leq N_s \cdot \overline{N_{su}}$. Moreover, from the construction of Γ, we can see that in any iteration of the training process, any state sequences $S_m \notin \Gamma$ cannot contribute to the update of $P(s_i)$ and $P(s_i|S_j)$. Therefore, $|\widetilde{\Psi^i}| \leq |\{P^0(s_i)|\ P^0(s_i) \neq 0\}| \leq N_s$ and $|\widetilde{\Delta^i}| \leq |\Gamma| \cdot (\overline{T} - 1)$. □

In practice, the upper bound $\mathcal{C}$ given by Theorem 4.8 is often much smaller than the size of the whole parameter space $\mathcal{N} = N_s \cdot (N_q + N_u + \sum_{t=2}^{T_{max}} N_s^{t-1})$. For example, in our experimental data, $\overline{N_{sq}} = 2.91 \ll N_q = 1.2 \times 10^7$, $\overline{N_{su}} = 14.1 \ll N_u = 4.8 \times 10^7$, and $|\Gamma| \cdot (\overline{T} - 1) = 4.7 \times 10^6 \ll N_s \cdot \sum_{t=2}^{T_{max}} N_s^{t-1} \sim |N_s|^5 \sim 1.99 \times 10^{34}$, since N_s is 7,242,693 in our experiments.

Our initialization strategy also enables an efficient training process. According to Equations (4)–(7), the complexity of the training algorithm is $O(k \cdot N \cdot \overline{|\Gamma_n|})$, where k is the number of iterations, N is the number of training sessions, and $\overline{\Gamma_n}$ is the average number of candidate state sequences for a training session. In practice, $\overline{\Gamma_n}$ is usually small, for example, 1.45 in our experiments. Although N is a very large number (1.2 billion in our experiments), we can distribute the training sessions on multiple machines, as discussed in Section 4.1. Our empirical study shows that the training process converges fast. In our experiments, k is around 10.

4.3. Further Scaling-up Training Process

In the previous sections, we developed a distributed training method under the map-reduce programming model as well as a special initialization strategy to make it feasible to train a very large vlHMM. However, the huge amounts of session data and the extremely large number of model parameters still present great challenges for the performance of the training process. To further improve the efficiency and scalability of our training process, we develop several techniques in this section, including (1) a method for identifying deterministic sessions, (2) a method for detecting deterministic parameters, and (3) a heuristic for data partition. The experimental results in Section 6.4 show that those techniques can reduce the training time substantially. For example, when we used five process nodes and the full training data, the three techniques reduce the overall training time by 65%, 34%, and 18%, respectively. Moreover, the combination of all three techniques reduces the overall training time by 70%.

4.3.1. Identifying Deterministic Sessions. In the EM training process, the E-step infers the probability distribution over the state sequences for the training sessions, while the M-step updates the model parameters. Intuitively, if the probability distribution over the state sequences for some training session O_n never changes during the iteration, then it is not necessary to infer O_n repeatedly in the E-step during the iteration. Moreover, the contribution of O_n to the update of model parameters in the M-step can be fixed. Based on this idea, we develop the concept of *deterministic sessions*.

Definition 4.9 (*Deterministic Sessions*). Let O_n be the nth training example and Γ_n be the set of candidate state sequences, as defined in Definition 4.6. O_n is a *deterministic session* if $|\Gamma_n| = 1$. □

According to Lemma 4.7 and Definition 4.9, for any deterministic session $O_{n'}$, there exists only one $S_{m'}$ such that $P(S_{m'}|O_{n'}, \Theta^{(i-1)}) \neq 0$. It is easy to show that for each deterministic session $O_{n'}$, its contribution to either the denominator or the numerator of any update formula for model parameters (Equations (4)–(7)) in each round of iteration is a constant. Without loss of generality, let us consider the numerator of the update

Table V. Constant Values for Model Parameter Updates from Deterministic Sessions $\overline{\mathcal{X}}$

Key	Constant Values from $\overline{\mathcal{X}}$
s_i	$Const_{n',1} = \sum_{O_{n'} \in \overline{\mathcal{X}}} \delta(s_{m',1} = s_i)$ $Const_{n',2} = \lvert\overline{\mathcal{X}}\rvert$
(s_i, q_j)	$Const_{n',1} = \sum_{O_{n'} \in \overline{\mathcal{X}}} \sum_t \delta(s_{m',t} = s_i \wedge q_j = q_{n',t})$ $Const_{n',2} = \sum_{O_{n'} \in \overline{\mathcal{X}}} \sum_t \delta(s_{m',t} = s_i)$
(s_i, u_j)	$Const_{n',1} = \sum_{O_{n'} \in \overline{\mathcal{X}}} \sum_t \delta(s_{m',t} = s_i \wedge u_j \in U_{n',t})$ $Const_{n',2} = \sum_{O_{n'} \in \overline{\mathcal{X}}} \sum_t \delta(s_{m',t} = s_i)$
(s_i, S_j)	$Const_{n',1} = \sum_{O_{n'} \in \overline{\mathcal{X}}} \delta(\exists t\ S_{m'}^{t-1} = S_j \wedge s_{m',t} = s_i)$ $Const_{n',2} = \sum_{O_{n'} \in \overline{\mathcal{X}}} \delta(\exists t\ S_{m'}^{t-1} = S_j)$

Note: For each session $O_{n'} \in \overline{\mathcal{X}}$, $S_{m'}$ is the only candidate state sequence.

formula for $P(s_i)$ (Equation (4)) as an example. The contribution of a deterministic session $O_{n'}$ to the numerator of Equation (4) is $\sum_m P(S_m|O_{n'}, \Theta^{(i-1)})\delta(s_{m,1} = s_i)$. Since $O_{n'}$ is a deterministic session, let $S_{m'}$ be the only candidate state sequence corresponding to $O_{n'}$. Then, for any state sequence S_m, we have $P(S_m|O_{n'}, \Theta^{(i-1)}) = 0$ if $m \neq m'$, and $P(S_m|O_{n'}, \Theta^{(i-1)}) = 1$, otherwise. Therefore, the contribution of $O_{n'}$ to the numerator of Equation (4) can be reduced to $\delta(s_{m',1} = s_i)$. In other words, if the first state $s_{m',1}$ in $S_{m'}$ is s_i, the contribution is one; otherwise, it is zero. Since $S_{m'}$ is determined after the initialization process (Section 4.2), the contribution of $O_{n'}$ to the numerator of Equation (4) is a constant. Similarly, we can prove the contribution of $O_{n'}$ to the denominator or numerator of any update formula from Equation (4) to Equation (7) is a constant.

From this discussion, we can divide the training examples $\mathcal{X}$ into two subsets, where $\overline{\mathcal{X}} \subseteq \mathcal{X}$ contains all the deterministic sessions, while $\tilde{\mathcal{X}} = \mathcal{X} \setminus \overline{\mathcal{X}}$ contains the remaining sessions. Then those deterministic sessions only need to be inferred once in the first round of iterations. In the following rounds of iteration, we can simply reuse the values in Table V without inferring with the deterministic sessions. For example, the value of $P(s_i)$ can be estimated by $\frac{Const_{n',1}+Value_{n,1}}{Const_{n',2}+Value_{n,2}}$, where $Value_{n,1}$ and $Value_{n,2}$ are calculated according to Table III only on the sessions in $\tilde{\mathcal{X}}$.

In practice, since the percentage of ambiguous queries is low, most observations in sessions map to unique states. Consequently, deterministic sessions occupy a large percentage in the training data, for example, about 80% in our experimental data. Such a large percentage of deterministic sessions can greatly reduce the runtime for each iteration of the EM algorithm. In our empirical study, the runtime for the E-step and M-step on the full data reduced by 75% and 35%, respectively, when five process nodes were used (see experiments in Section 6.4). Although we need a pre-processing step to identify deterministic sessions and calculate the values in Table V, the cost for that step is very small compared to the reduction of runtime for the E-step and M-step. Consequently, the overall training time on the full data, including that for the pre-processing step, E-step, and M-step, reduces by 70% when five process nodes were used.

4.3.2. Detecting Deterministic Parameters. In the preceding discussion, we focused on the training sessions whose contributions to model parameters remain fixed during the EM iteration process. We further consider whether there are model parameters whose values never change after initialization. Clearly, for those parameters, we can

save the computation cost to update them. As a special case, we discussed in Section 4.2 that a model parameter will remain zero if its initial value is zero. Here we explore more general cases.

Definition 4.10 (*Sibling states*). A state s_j is a *sibling state* of state s_i if $\exists n, t$ such that the set of candidate states $Cand_{n,t}$ (see Definition 4.5) for the tth observation of the training session O_n contain both s_i and s_j. In particular, s_j is a sibling state at position 1 of s_i if $\exists n$ such that the set of candidate states $Cand_{n,1}$ for the first observation of O_n contains both s_i and s_j.

LEMMA 4.11. *The initial state probability $P(s_i)$ is a constant value if state s_i does not have any sibling state at position 1.*

PROOF. To prove $P(s_i)$ is a constant, we actually need to prove Equation (4) is a constant. Since the denominator of Equation (4) is a constant, we only focus on the numerator. For any session O_n, $P(S_{m'}|O_n, \Theta^{r-1}) = 0$ holds for any $S_{m'} \notin \Gamma_n$ (Lemma 4.7), where Γ_n is the set of candidate state sequences for O_n. Therefore, $\sum_m P(S_m|O_n, \Theta^{(r-1)}) \cdot \delta(s_{m,1} = s_i)$ is equivalent to $\sum_{S_m \in \Gamma_n} P(S_m|O_n, \Theta^{(r-1)}) \cdot \delta(s_{m,1} = s_i)$.

The candidate state sequences S_m in Γ_n have two cases. In the first case, $s_i \notin Cand_{n,1}$. In this case, $\delta(s_{m,1} = s_i) = 0$ holds for all S_m in Γ_n, and thus $\sum_{S_m \in \Gamma_n} P(S_m|O_n, \Theta^{(r-1)})\delta(s_{m,1} = s_i) = 0$.

In the second case, $s_i \in Cand_{n,1}$. Since s_i does not have any sibling state at position 1, according to Definition 4.10, we have $Cand_{n,1} = \{s_i\}$, and $\delta(s_{m,1} = s_i) = 1$ holds for all $S_m \in \Gamma_n$. In this case, we can derive $\sum_{S_m \in \Gamma_n} P(S_m|O_n, \Theta^{r-1}) \cdot \delta(s_{m,1} = s_i) = \sum_{S_m \in \Gamma_n} P(S_m|O_n, \Theta^{r-1}) = \sum_m P(S_m|O_n, \Theta^{r-1}) = 1$.

In summary of the preceding two cases, if s_i does not have any sibling state at position 1, then for any session O_n, $\sum_m P(S_m|O_n, \Theta^{r-1}) \cdot \delta(s_{m,1} = s_i)$ is either zero when $s_i \notin Cand_{n,1}$ or one otherwise. Since $Cand_{n,1}$ for any session O_n is determined after the initialization process, if s_i does not have any sibling state at position 1, the numerator of Equation (4) is a constant. □

Similarly, we can prove both the numerator and the denominator of either Equation (5) or Equation (6) are both constants and derive the following lemma.

LEMMA 4.12. *The query and URL emission probabilities $P(q|s_i)$ and $P(u|s_i)$ are constant values if state s_i does not have any sibling state.*

Definition 4.13 (*Sibling State Sequences*). Let S_j be a state sequence with length t. A state sequence S_j is a *sibling state sequence* of S_i if (1) the length of S_j is t, and (2) $\exists n, t$ such that the set of candidate state sequences Γ_n^t (see Definition 4.6) for the first t observations of the training session O_n contains both S_i and S_j.

The following lemma can be proved similar to Lemma 4.11.

LEMMA 4.14. *The transition probability $P(s_i|S_j)$ is a constant value if the state sequence $S_j \circ s_i$ does not have any sibling state sequence.*

In our experimental data, the percentages of deterministic parameters among nonzero parameters are 91%, 65%, 57%, and 10% for initial state probabilities, query emission probabilities, URL emission probabilities, and state transition probabilities, respectively. As mentioned previously, the reason for the large percentages of deterministic parameters is that most queries are not ambiguous and most observations in sessions only map to unique states. Since the deterministic parameters do not need to be updated iteratively in the EM process, in our empirical study, the runtime for the

Table VI. Example Where Two Sessions Share Lots of Parameters

	Observations	States	Parameters
O_1	$(q_1, u_1) \Rightarrow (q_2, u_2)$	$s_1 \Rightarrow s_2$	$p(q_1\|s_1)$, $p(u_1\|s_1)$, $\mathbf{p(q_2\|s_2)}$, $p(u_2\|s_2)$, $p(s_1)$, $p(s_2\|s_1)$
O_2	$(q_1, u_1) \Rightarrow (q_3, u_2)$	$s_1 \Rightarrow s_2$	$p(q_1\|s_1)$, $p(u_1\|s_1)$, $\mathbf{p(q_3\|s_2)}$, $p(u_2\|s_2)$, $p(s_1)$, $p(s_2\|s_1)$

ALGORITHM 2: A Heuristic Data Partition Method

Input: Training sessions $\mathcal{X}$, the limit of main memory M_{max};
Output: A subset of training sessions $\mathcal{X}_k$ for each process node k;
Initialize: State list $L_k = \emptyset$, Size of used memory $M_k = 0$ for each process node pn_k;

```
for each training session O_n ∈ X do
  for each process node k do
    derive the set of states S_n from the candidate state sequences Γ_n by Definition 4.6;
    let S_k^+ = S_n \ L_k;
    compute the size M_k^+ of the parameters associated with the sates in S_k^+;
  if ∃k such that M_k + M_k^+ < M_max then
    assign O_n to the process node pn_k' with the minimum M_k^+;
    M_k' += M_k^+;
  else
    // starts a new round of partition
    for each process node pn_k do let L_k = ∅ and M_k = 0;
    randomly assign O_n to a process node pn_k';
    compute the size M^+ of the parameters associated with the sates in S_n;
    let M_k' += M^+;
```

E-step and M-step on the full data was reduced by 40% and 20%, respectively, with five process nodes used. Although we need an extra pre-processing step to detect deterministic parameters, the cost for that step is small compared to the saving in the E-step and M-step. For example, in our empirical study, the overall runtime, including that for the pre-processing step, the E-step, and the M-step, for the full training dataset was reduced by 34% when five process nodes were used (see the experiment in Section 6.4).

4.3.3. A Heuristic for Data Partitioning. During the training process of the vlHMM model, each process node carries out the E-step on a subset of training data and infers the probability distribution for the training sessions. Suppose a process node pn_k is assigned with a subset of training sessions $\mathcal{X}_k$. Based on the preceding discussion, we can determine the set of model parameters Θ_k needed by the inference procedure for $\mathcal{X}_k$. If the whole set of Θ_k can be loaded into main memory, then the node can directly carry out the E-step. Otherwise, the node can adopt a divide-and-conquer approach. To be specific, the node can divide $\mathcal{X}_k$ into several blocks $\mathcal{X}_{kb}$ such that the parameters needed by the inference procedure for each block can be loaded into main memory.

However, the divide-and-conquer approach does not utilize the overlapped parameters among different sessions. Consider the two sessions O_1 and O_2 in Table VI, where each session involves six parameters in the inference stage. An interesting observation is that most of the parameters for infering these two sessions are in common. Therefore, if these two sessions are assigned to the same process node, the node only needs to hold seven parameters to infer these two sessions, instead of $6 + 6 = 12$ parameters. Based on this idea, we adopt a heuristic approach and assign training sessions that share many common parameters into the same process node.

Algorithm 2 shows our heuristic data partition method. Each process node pn_k maintains a *state list* which traces the states whose parameters will be loaded into pn_k during the E-step. The data partition process scans the training sessions one by one.

For each training session O_n, the method derives the set of candidate state sequences Γ_n, as defined in Definition 4.6, and then determines the set S_n of states involved in Γ_n.

In the next step, the partition process checks the state list L_k for each process node pn_k and finds out how many extra states S_k^+ the process node needs to cover the whole set of S_n. In other words, $S_k^+ = S_n \setminus L_k$. According to Lemmas 4.1 to 4.4, any model parameter will remain zero if its initial value is zero. Therefore, each state s_i can be associated with four fixed sets of parameters $(\Psi_i, \Lambda_{qi}, \Lambda_{ui}, \Delta_i)$, where $\Psi_i = \{P^0(s_i)\}$ if $P^0(s_i) > 0$ or $\Psi_i = \emptyset$; otherwise, $\Lambda_{qi} = \{P^0(q|s_i)|P^0(q|s_i) > 0\}$, $\Lambda_{ui} = \{P^0(u|s_i)|P^0(u|s_i) > 0\}$, and $\Delta_i = \{P^0(s_i|S_j)|P^0(s_i|S_j) > 0\}$. Using this property, the partition process can calculate how much extra memory M_k^+ a process node needs to hold the parameters associated with the states in S_k^+. If the size of the parameters associated with the states in L_k of a process node pn_k already reaches the limit of main memory, the corresponding M_k^+ is set to ∞. The partition process then chooses the process node with the minimum M_k^+ to assign the training session O_n. The assignment of training sessions continues until no process node can add the parameters for a new session. At this point, the partition process clears the state list L_k for each process node and starts the next round of assignments. The whole partition process stops when all training sessions have been assigned.

Compared with the random partition method, the heuristic method can greatly reduce the number of blocks for each process node since it tries to maximize the sharing of model parameters in the inference stage. For example, in our experiments on the full data with five process nodes in the system, the maximum number of blocks to be processed by a process nodes reduces from five under random partition to only one under the heuristic partition method (see Section 6.4). Consequently, the runtime for the E-step was reduced by 40%. Note that since the heuristic data partition method does not reduce the number of parameters to be updated, the runtime for the M-step is the same. Moreover, unlike the random partition method, which actually does not need any computational cost to assign the training sessions, the cost by the heuristic method is nontrivial. However, the cost is still small compared to the reduction in runtime for the E-step. In our empirical study, the heuristic data partition method brought 18% reduction in the overall training time, including that for the heuristic partition time, E-step, and M-step, with the full data and five process nodes used.

4.4. Constructing Feature Vectors

Now we have derived a set of high-quality states after the E-M step, where each state contains a set of queries and a set of URLs. In the online stage of vlHMM model application, users may raise new queries which are not covered by the log data. To handle such new queries, we create two feature vectors for each state. Let S be a state and s be the corresponding cluster, the *URL feature vector* for s, denoted by $\overrightarrow{s}^{URL}$ is the L2-normalized vector in URL space. Particularly, the jth element of the feature vector of s is $\overrightarrow{s}^{URL}[j] = norm(P(u_j|s)) = \frac{P(u_j|s)}{\sqrt{\sum_{u'_j \in U}[P(u'_j|s)]^2}}$, where $norm(\cdot)$ is the L_2 normalization function and $P(u_j|s)$ is the final emission probability after iteration. Besides the URL feature vector, we also create a *term feature vector* for s based on the terms of the queries in s. To be specific, for each query $q_i \in Q_s$ (Q_s means the query set of the state s), we can represent q_i by its terms as $\overrightarrow{q}^{term}[t] = norm(tf(t, q_i) \cdot isf(t))$, where $tf(t, q_i)$ is the frequency of term t in q_i, $isf(t) = \log \frac{N_s}{N_s(t)}$ is the *inverse state frequency* of t, N_s is the total number of states, and $N_c(t)$ is the number of states containing term t. The term feature vector for state s is defined as $\overrightarrow{s}^{term}[t] = \frac{\sum_{q_i \in Q_s} \overrightarrow{q_i}^{term}[t]}{|S|}$, where $|Q_s|$ is the number of queries in S. We will describe the details of training a model to use these two feature vectors to handle new queries at the online stage in Section 6.5.

5. MODEL APPLICATION

The previous sections focus on the offline training of the vlHMM from search logs. In this section, we discuss how to apply the learned vlHMM to various search applications, including document reranking, query suggestion, and URL recommendation.

Suppose the system receives a sequence O of user events, where O consists of a sequence of queries $q_1, \ldots, q_t$, and for each query q_i $(1 \leq i < t)$, the user click on a set of URLs U_i. We first construct the set of candidate state sequences Γ_O, as described in Section 4.2, and infer the posterior probability $P(S_m|O, \Theta)$ for each state sequence $S_m \in \Gamma_O$, where Θ is the set of model parameters learned offline. We can derive the probability distribution of the user's current state s_t by $P(s_t|O, \Theta) = \frac{\sum_{S_m \in \Gamma_o} P(S_m|O,\Theta) \cdot \delta(s_{m,t}=s_t)}{\sum_{S_m \in \Gamma_o} P(S_m|O,\Theta)}$, where $\delta(s_{m,t} = s_t)$ indicates whether s_t is the last state of S_m $(= 1)$ or not $(= 0)$. One strength of the vlHMM is that it provides a systematic approach for not only inferring the user's current state s_t, but also predicting the user's next state s_{t+1}. Specifically, we have $P(s_{t+1}|O, \Theta) = \sum_{S_m \in \Gamma_o} P(s_{t+1}|S_m) \cdot P(S_m|O, \Theta)$, where $P(s_{t+1}|S_m)$ is the transition probability learned offline. To keep our presentation simple, we omit the parameter Θ in the remaining part of this section.

Once the posterior probability distributions of $P(s_t|O)$ and $P(s_{t+1}|O)$ have been inferred, we can conduct the following context-aware actions.

Document Reranking. Let $S_t = \{s_t|\ P(s_t|O) \neq 0\}$ and U be a ranked list of URLs returned by a search engine as the answers to query q_t. We compute the posterior probability $P(u|O)$ for each URL $u \in U$ by $\sum_{s_t \in S_t} P(u|s_t) \cdot P(s_t|O)$. Then, we rerank the URLs in the posterior probability descending order. Furthermore, we use Borda's ranking fusion method [Borda 1781] to combine the original ranking of search engine and current ranking of vlHMM. We combine original ranking of search engine R_0 with vlHMM-based ranking R_1. The final ranking scores are merged as: $score(u) = \alpha \cdot \frac{1}{R_0(u)} + (1 - \alpha) \cdot \frac{1}{R_1(u)}$, where $\alpha \in [0, 1]$ is set as 0.2 since we found that the re-ranking performance of all methods are nearly optimal while $\alpha \in [0.1, 0.3]$ for all methods.

URL Recommendation. Let $U_{t+1} = \{u|\ s_{t+1} \in S_{t+1}, P(u|s_{t+1}) \neq 0\}$. For each URL $u \in U_{t+1}$, we compute the posterior probability $P(u|O) = \sum_{s_{t+1} \in S_{t+1}} P(u|s_{t+1}) \cdot P(s_{t+1}|O)$, and recommend the top K_u URLs with the highest probabilities, where K_u is a user-specified parameter.

Query Suggestion. Let $S_{t+1} = \{s_{t+1}|\ P(s_{t+1}|O) \neq 0\}$ and $Q_{t+1} = \{q|\ s_{t+1} \in S_{t+1}, P(q|s_{t+1}) \neq 0\}$. For each query $q \in Q_{t+1}$, we compute the posterior probability $P(q|O) = \sum_{s_{t+1} \in S_{t+1}} P(q|s_{t+1}) \cdot P(s_{t+1}|O)$, and suggest the top K_q queries with the highest probabilities, where K_q is a user-specified parameter.

There are two issues in the online application of the vlHMM. First, users may raise new queries and click URLs which do not appear in the training data. In the ith $(1 \leq i < t)$ round of interaction, if either the query or at least one URL has been seen by the vlHMM in the training data, the vlHMM can simply ignore the unknown queries or URLs and still make the inference and prediction based on the remaining observations. As discussed in Section 4.4, given a new query q_t in the online session, we can build a URL feature vector $\overrightarrow{q_t}^{URL}$ for q_t based on the the clicked URLs by formula $\overrightarrow{q_t}^{URL}[j] = norm(Count(q_t, u_j))$, where $Count(q_t, u_j)$ denotes the click count on u_j by users after submitting q_t, $norm(\cdot)$ is the L_2 normalization function. If users do not click any URLs, we use top ten search results returned by the search engine to build $\overrightarrow{q_t}^{URL}$. Besides the URL feature vector, we can merge the snippets of the top ten search results of q_t and create a term feature vector $\overrightarrow{q_t}^{term}$ for q_i in Section 4.4. In the online stage, we propose a two-stage approach for mapping novel queries to known

state. First, we retrieve top K similar states as candidates based on Euclidian distance $||\overrightarrow{q_t}^{URL} - \overrightarrow{s}^{URL}||$ and $||\overrightarrow{q_t}^{term} - \overrightarrow{s}^{term}||$. Second, we extract features for query-state pairs and use a pre-trained model to judge which state the query should be mapped to. The detailed steps can be found in Section 6.5.

The second issue with the online application of our vlHMM is the strong requirement on efficiency. Given a user input sequence O, the major cost in applying the vlHMM depends on the sizes of the candidate sets Γ_O, S_t, S_{t+1}, Q_{t+1}, and U_{t+1}. In our experiments, the average numbers of Γ_O, S_t, and S_{t+1} are all less than 10 and the average numbers of Q_{t+1} and U_{t+1} are both less than 100. Moreover, the average runtime of applying the vlHMM to one user input sequence is only 0.1 milliseconds. In cases where the sizes of candidate sets are very large or the session is extremely long, we can approximate the optimal solution by discarding candidates with low probabilities or truncating the session. Since we only rerank top URLs returned by a search engine and suggest the top queries and URLs generated by vlHMM, such approximations will not lose much accuracy.

6. EXPERIMENTAL RESULTS

In this section, we report the results from a systematic empirical study using a large search log from a major commercial search engine. We examine the efficiency of our vlHMM training method and the effectiveness of using the learned vlHMM in three context-aware search applications.

6.1. Data Set and Preparation

To train a vlHMM, we use a large search log from a major commercial search engine collected from August 2009 to October 2009. We only focus on the Web searches in English from the U.S. market. The log data set contains 1.9 billion queries, 2.9 billion clicks, and 1.2 billion sessions. The data set involves 535 million unique queries and 396 million unique URLs. From the raw search log, we extract user sessions, as described in Section 3. The distribution of the session lengths follows the power law and about 50% of the sessions contain at least two rounds of interaction. These observations are consistent with those in previous studies (e.g., [Huang et al. 2003]).

Based on our observation, those infrequent (tail) query sequences are noisy and those long sessions with many queries also contain many noises. If we build our model based on them, then the suggestion performance will decrease. Since the focus of this article is to test the performance of all methods in context-aware search applications, we leave the fine tuning of noise reduction on session data as possible future work and conduct simple pruning strategies as follows. We remove a user session $\langle (q_1, U_1), \ldots, (q_T, U_T) \rangle$ only if the frequency of the query sequence $(q_1, \ldots, q_T)$ is less than a threshold min_sup. In our experiments, min_sup is set to 5. Consequently, 52% of the sessions in the log data set are pruned. After the pruning process, we manually inspect some sessions with lengths longer than 5, and find many of them contain meaningless query sequences. Since, the total number of sessions longer than five is actually small (9,885 distinct cases). Therefore, we remove all of them. Note we do not remove sessions with one query and corresponding clicks since those sessions can change the distribution of the parameters $P(q|s)$ and $P(u|s)$ in vlHMM.

Table VII shows the statistics of the dataset before and after the pre-processing. Although 97.6% unique queries and 87.8% unique URLs are removed by the pre-processing, the resulting dataset still keeps 38.1% of the original query occurrences, 34.0% of the original URL clicks, and 52.5% of the original user sessions. As shown in previous work (e.g., [Baeza-Yates and Tiberiet 2007]), this is because the query occurrences and URL clicks in search logs follow the power law distribution. In our search applications, sessions with length $=1$ can be used in document reranking, while

Table VII. Data Statistics Before and After Pre-processing

	Raw search log	Training data
Num. of unique queries	535,847,264	12,769,284
Num. of unique URLs	396,922,801	48,418,579
Num. of query occurrences	1,932,773,655	737,145,967
Num. of clicks	2,885,798,105	982,388,395
Num. of sessions	1,224,211,869	643,318,202

Table VIII. Distribution of Sessions w.r.t. the Length of Query Sequence

# of Queries	Distinct # of Sessions
2	10,734,074
3	407,846
4	44,313
5	10,950

sessions with length $\geq$2 can be used in URL recommendation and query suggestion. The distribution of sessions with respect to the lengths of query sequence is shown in Table VIII. In the experiments, about 0.5 millions of distinct sessions contain more than two queries. We use these sessions to evaluate the contextual assumptions and the vlHMM.

6.2. Verify the Context Difference Observatin

Before building the vlHMM for context-aware applications, we test the context difference here. Let q, u, q_s, and c denote the current query, the ranked URLs, the suggested query, and the context, respectively. Take URL ranking as an example—the ranked URLs for a given query q are generated by $P(u|q)$. When context information c is available, the rankings of URLs are generated by $P(u|q,c)$. Since we are using the top ranked URLs in our applications, we use generalized Kendall's tau [Fagin et al. 2003] to compare the difference between $P(u|q)$ and $P(u|q,c)$ instead of the distance between distributions, such as KL-divergence. In the experiments, we focus on the top five URLs or queries generated by different models.

Given two ordered lists π_1 and π_2, we can get all objects O from them. There are four cases for each object pair (o_i,o_j) from O, where $o_i \neq o_j$. (1) If they both exist in π_1 and π_2 and the preference of (o_i,o_j) is different in π_1 and π_2, then $K_{min} = K_{min} + 1$. (2) If both o_i and o_j exist in one rank list π_1 or π_2, and only one of o_i or o_j exists in another rank list, $K_{min} = K_{min} + 1$ if the higher ranked object in one list does not exist in another one. (3) If one of o_i or o_j exists in π_1 and another one exists in π_2, then $K_{min} = K_{min} + 1$. (4) If both o_i and o_j exist in one list and neither o_i nor o_j exists in another list, K_{min} keeps the same. Furthermore, we can normalize K_{min} to [0,1] and a bigger value of K_{min} indicates a bigger difference.

Based on the training dataset in Table VII, we compute different URL ranking and query suggestion models and use K_{min} to measure the difference. We compare in total four pairs of methods for a given q: (1) $P(u|q)$ and $P(u|q,c)$; (2) $P(u|q,c_1)$ and $P(u|q,c_2)$; (3) $P(q_s|q)$ and $P(q_s|q,c)$; (4) $P(q_s|q,c_1)$ and $P(q_s|q,c_2)$. Here we take query suggestion as example. Since $P(q_s|q) = \sum_c P(q_s,c|q) = \sum_c P(q_s|q,c) \cdot P(c|q)$, and $P(c|q)$ does not affect the suggestion while given context c and test query q, we compare $P(q_s|q,c_1)$ and $P(q_s|q,c_2)$ to unveil the differences among context models. Therefore, the comparisons of (2) and (4) are supplementary to the comparisons of (1) and (3), respectively.

The results are shown in Table IX. The value K_{min}^{avg} is obtained by the average between Model A and Model B, while K_{min}^{max} is obtained by maximum difference models from

Table IX. Generalized Kendall's tau for Top-5 URL Rankings and Query Suggestions of Different Context Observations

Application	Model A	Model B	K_{min}^{avg}	K_{min}^{max}
document Re-ranking	$P(u\|q)$	$P(u\|q,c)$	0.2952	0.3788
	$P(u\|q,c_1)$	$P(u\|q,c_2)$	0.2019	0.3810
Query Suggestion	$P(q_s\|q)$	$P(q_s\|q,c)$	0.3311	0.4638
	$P(q_s\|q,c_1)$	$P(q_s\|q,c_2)$	0.3370	0.6347

Note: K_{min}^{avg} is the average result among all contexts. K_{min}^{max} is the average result from maximum different contexts.

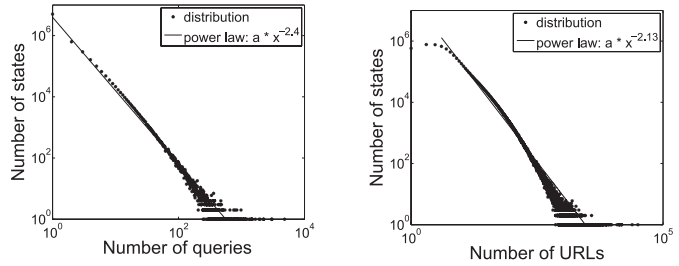


Fig. 3. The number of states with respect to the number of nonzero initial queries (left) and URL (right) emission probabilities.

Model A and Model B. For example, for $P(u|q)$ and $P(u|q,c)$, K_{min}^{avg} is obtained from the average of all c's, but K_{min}^{max} is obtained from the maximum different rankings between $P(u|q)$ and $P(u|q,c_{max})$ given a special context c_{max}. Take the pair $P(u|q)$ and $P(u|q,c)$ as an example—K_{min}^{avg} measures the average differences between context and non-context models, while K_{min}^{max} measures the maximum differences between context and non-context models. A value of K_{min} larger than 0.2 at top-5 positions can indicate certain differences in URL rankings or query suggestions.

From Table IX, we can find that (1) The top-5 URL rankings based on $P(u|q)$ and $P(u|q,c)$ are different, and the top-5 query suggestions based on $P(q_s|q)$ and $P(q_s|q,c)$ are different. These results verify that with or without context, users' behaviors are different. (2) The top-5 URL rankings based on $P(u|q,c_1)$ and $P(u|q,c_2)$ are different and the top-5 query suggestions based on $P(q_s|q,c_1)$ and $P(q_s|q,c_2)$ are different. The results indicate that users' querying and clicking behaviors recorded in the search logs are different for context-aware URL ranking and query suggestion. (3) The gap between methods in query suggestion is bigger than the gap in URL ranking. This observation indicates that for query suggestion, there are bigger differences between context and non-context methods than in URL ranking. Overall, it is necessary to capture high-order context information for context-aware search.

6.3. Initializing and Updating Model Parameters of vlHMM

To determine the number of states and assign initial parameter values, we apply the clustering algorithm [Liao et al. 2011] and derive 7,242,693 clusters of queries. We thus have 7,242,693 states in the vlHMM and then initialize the parameter values, as described in Section 4.2.

Figures 3(a) and 3(b) show the distributions of the number of states with respect to the number of nonzero initial query and URL emission probabilities, respectively. Both approximately follow the power law distribution. Let $\overline{N_{sq}}$ and $\overline{N_{su}}$ be the average numbers of nonzero parameters $P^0(q|s)$ and $P^0(u|s)$ in all states, respectively. In our

Table X. Comparison of the Actual Number, the Upper Bound, and the Whole Space of Parameters

	Actual Number	Upper Bound	Whole Space
$\#P(s_i)$	6,552,900	7,242,693	7,242,693
$\#P(q\|s_i)$	13,324,578	21,076,236	9.24×10^{13}
$\#P(u\|s_i)$	87,825,540	102,121,791	3.51×10^{14}
$\#P(s_i\|S_j)$	4,507,873	4,721, 451	1.99×10^{34}

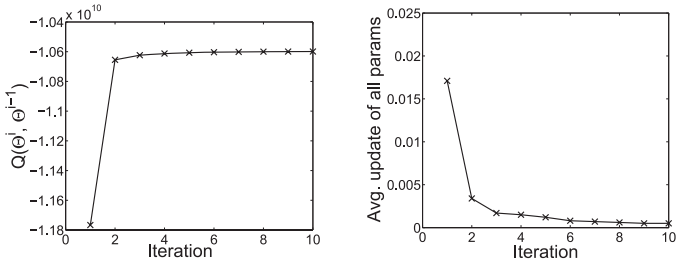


Fig. 4. $Q(\Theta, \Theta^{(i-1)})$ (left) and average difference of all parameters (right) in each iteration.

experiments, $\overline{N_{sq}} = 2.91$ and $\overline{N_{su}} = 14.1$. It means that on average, different users formulate 2.91 queries and click 14.1 URLs for a common search intent. We further compute the set of candidate state sequences Γ given the initialization of $P^0(q|s)$ and $P^0(u|s)$. For the 643, 318, 202 training sessions, there are only 11, 060, 773 unique candidate state sequences. The reason being that users with similar search intents often have similar search sequences and thus can be modeled by the same state sequence.

Table X compares the size of the whole parameter space, the actual number of parameters estimated in the training process, and the upper bound given by Theorem 4.8. Since we have 7 million states, 12 million unique queries, and 48 million unique URLs, the size of the whole parameter space can be calculated by multiplication. For example, the whole space of $P(q|s_i)$ is obtained by multiplying the size of queries and the size of states. However, based on Theorem 4.8, the size of all parameters $P(q|s_i)$ is bounded by $\overline{N_{sq}} \times N_s$, where $\overline{N_{sq}}$ is the average number of queries within one state and N_s denotes the number of states. For the upper bound of parameter $P(s_i|S_j)$, we obtain the average length of state sequence $\overline{T} = 2.047$ and $|\Gamma| = 4{,}507{,}873$ from the training sessions. Therefore, the upper bound of $P(s_i|S_j)$ is obtained by $|\Gamma| \cdot (\overline{T} - 1) = 4, 721, 451$. Clearly, the actual number of estimated parameters is significantly smaller than the size of the whole parameter space and the upper bound is tight. In particular, the actual number of estimated transition parameters is smaller than the size of the parameter space by a factor of 10^{26}, since the magnitude of the upper bound is 10^8 and that of the whole space is 10^{34}. There are two reasons for this factor. First, the session length follows the power law distribution, and a large part of sessions are short—of length one or two. Second, queries and clicked URLs in the same sessions are semantically related. Thus, the actual number of state sequences appearing in logs is significantly smaller than that of all possible combinations.

Figures 4(a) and 4(b) show the value of the object function $Q(\Theta, \Theta^{(i-1)})$ and the average difference of all parameters in one iteration, respectively. As a result, the training process converges fast and our vlHMM model can improve about 10% data likelihood.

6.4. Efficiency and Scalability of Training vlHMM

In this section, we first use Algorithm 1 as the baseline to test the efficiency and scalability of our distributed training process. In the baseline method, we adopt the

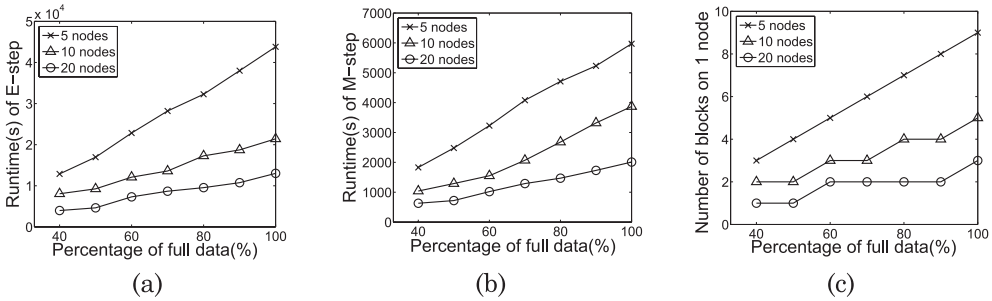


Fig. 5. The runtime of (a) E-step and (b) M-step, and (c) the average number of blocks on a process node for different sizes of training data.

random partition method and apply the divide-and-conquer approach if the size of the model parameters assigned to a process node is too large to be held in the main memory. After testing the baseline method, we incorporate the techniques developed in Section 4.3 into the baseline method and evaluate their effectiveness. We test all the methods on five, ten, and twenty process nodes, respectively. According to Figure 4, the EM iteration converges after ten rounds. Therefore, we carry out ten rounds of EM iterations for each test.

Figures 5(a) and 5(b) show the runtime of the E-Step and M-Step on 40%,50%, ..., and 100% of the full data. We can observe the following trends in Figure 5(a). First, the more process nodes used, the shorter the runtime for the E-step. For example, the runtime needed for the E-step on the full data by five, ten, and twenty process nodes is approximately in ratio 4:2:1. This suggests that our algorithm scales well with respect to the number of process nodes. Second, the increase of runtime is not linear with respect to the size of the data. For example, when ten process nodes are used, the runtime of the E-step increases sharply at 60%, 80%, and 100% of the data. The nonlinear increase can be explained by Figure 5(c), which shows the number of data blocks processed by a process node. In the divide-and-conquer approach, if the training data assigned to a process node involves too many parameters to be held in the main memory, the algorithm will split the training data into blocks such that the parameters needed by a block can be held in the main memory. Each time a process node processes a block of training sessions, it needs to scan the whole parameter file once. Since the parameter file is large (e.g., a 4.6G file containing 111,154,758 parameters for the full data), the disk I/O time becomes the major cost in the E-step. Consequently, the more blocks to be scanned, the longer runtime it takes. In Figure 5(c), when ten process nodes are used, the number of data blocks processed by a process node increases by one at 60%, 80%, and 100% of the data. Consequently, the runtime increases sharply at those points. In Figure 5(b), the runtime of M-step increases in $O(\frac{n}{K} \log \frac{n}{K})$, where n is the total number of key-value pairs and K is the number of process nodes. Comparing Figures 5(a) and 5(b), we can see the runtime of M-step is about one eighth that of E-step. In other words, the runtime of the training process is dominated by the E-step. Therefore, to improve the efficiency of the training process, it is more effective to speed up the E-step.

In the following, we evaluate the techniques developed in Section 4.3 to further scale up the baseline training method. Table XI shows the percentage of improvements in terms of running time in E-step, M-step, and overall at 100% of training data. We can see that all techniques improve the efficiency and combining them together achieves best performance. The improvements are due to the following reasons. Deterministic sessions consist of more than 80% of the whole sessions, deterministic parameters consist of more than 50% of all parameters, and number of blocks is reduced with heuristic

Table XI. Percentage of Improvements in Terms of Running Time by Different Scaling-up Methods on the Whole Training Data

Scale up techniques	% of in E-step	% of in M-step	% of Overall
Deterministic Sessions	75/70/65	35/30/15	65/60/52
Deterministic Parameters	40/35/40	20/30/25	34/31/36
Data Partition	40/25/25	(no improvements)	18/18/24
Combined	81/80/79	63/66/65	70/68/61

Note: The improvements are shown for 5/10/20 nodes.

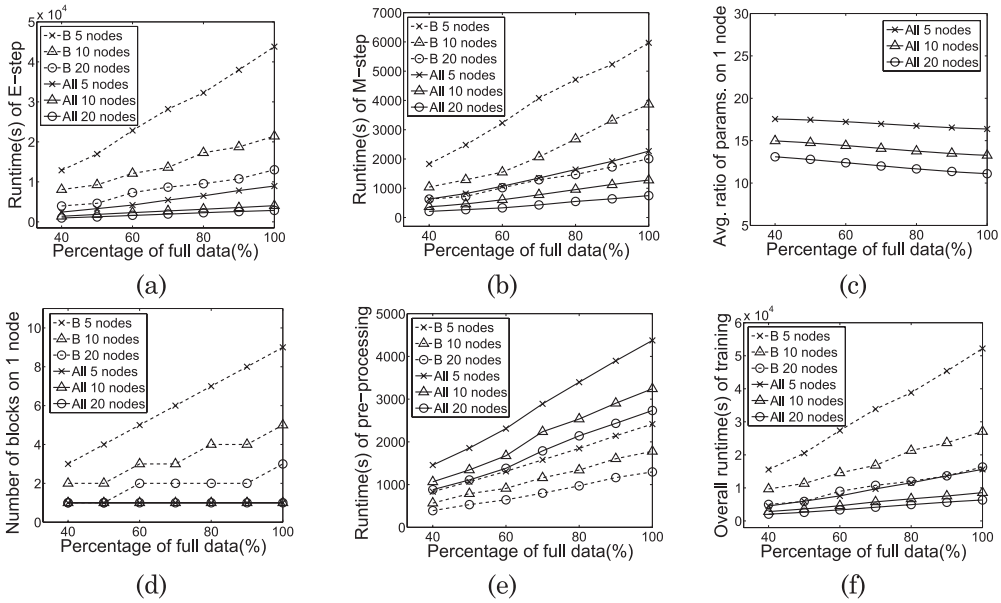


Fig. 6. The runtime of (a) E-step and (b) M-step, (c) average percentage of parameters of a process node with three techniques applied over those without the techniques applied, (d) average number of blocks processed by a process node, (e) the runtime of preprocessing steps, and (f) the total runtime of EM and preprocessing, for different sizes of training data. The solid and dotted curves correspond to the cases in which the combination of techniques is used and notused, respectively.

partition method. To give a better understanding of the improvements, we show the efficiency performance of combining all the techniques in Figure 6. Figures 6(a) and 6(b) compare the runtime of the E-step and M-step with respect to different sizes of training data before and after applying the combination of techniques. Figure 6(e) shows the runtime for the pre-processing step and Figure 6(f) shows the overall runtime with and without the combination of techniques. We can see that the extra cost for the preprocessing by the three techniques is much smaller than the gain in the E-step and M-step. Overall, the training time is reduced significantly.

6.5. Performance of Query State Mapping

We employ a two-stage method for mapping unknown queries into states in a vlHMM, as described in Section 5. The first stage is to retrieve candidate states with term and URL feature vectors, and the second stage is to employ a supervised learning method to make the decision on the mapping. Retrieving candidate states at the first stage can help exclude the states which are not relevant to the given query. We utilize a

Table XII. Coverage of vlHMM and vlHMM$^+$

Method	document re-ranking		Query Sug. / URL Rec.	
	Non-Context	Context	Non-Context	Context
vlHMM	69.12%	54.78%	46.31%	34.01%
vlHMM$^+$	84.63%	72.60%	49.27%	38.67%

human-labeled dataset consisting of query-state pairs to learn the model. Here we evaluate the query state mapping method.

We sample 137 queries which do not exist in the training data for learning the mapping model. We retrieve the top-5 candidate states by calculating the similarity between term feature vectors and top-5 candidate states by calculating the similarity between URL feature vectors. The similarity calculation is based on the Euclidian distance between term (or URL) feature vectors. Then the retrieved query-state pairs are labeled as relevant, partially relevant, and irrelevant. 97 queries (70%) have relevant candidate states. Many queries have multiple partially relevant states. Finally we take the relevant (205) and irrelevant (186) query-state pairs as positive and negative examples and train a linear SVM model with the query-based, state-based, and query-state pairwised features. The three-fold cross-validation results show that accuracy of classification is 78.77%, precision is 87.05%, and recall is 65.05%.

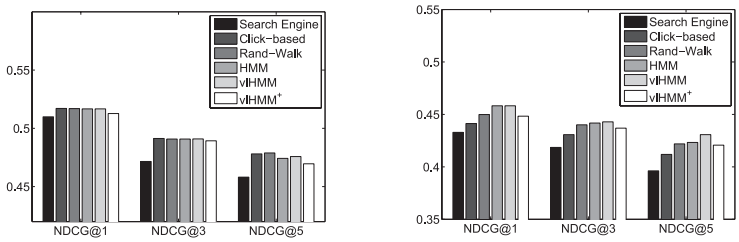
6.6. Improve the Coverage of vlHMM by vlHMM$^+$

If a query can be recognized by vlHMM, then one can perform reranking, suggestion, and recommendation by using the model. If it cannot be recognized, then one can employ the mapping model to assign it to a state of vlHMM. We use *vlHMM*$^+$ to denote the vlHMM with coverage enhancement hereafter. The learned mapping model in Section 6.5 is used for constructing the vlHMM$^+$. We conduct the coverage testing on both single query sessions and multiple-query sessions. The number of single query session is 100,000 and the number of multiple-query session is 43,105. The results are reported in Table XII.

From Table XII, we can observe the following trends. First, the coverage for document reranking is higher than that for query suggestion and URL recommendation, no matter whether context is utilized or not. This is because we only need to infer state s_t for document reranking. However, to provide query suggestion and URL recommendation, we not only need to infer state s_t, but also state s_{t+1}. In some cases, a state s_t does not transit to any other states, and thus no query suggestion or URL recommendation can be provided. The reason being that the states in vlHMM are derived from a click-through bipartite graph. Once s_{t+1} is inferred, there must exist at least one query q and one URL u such that $P(q|s_{t+1})$ and $P(u|s_{t+1})$ are nonzero. Note that the coverage for query suggestion and URL recommendation is the same.

Second, the coverage of document reranking, query suggestion, and URL recommendation with context is lower than those without context. For document reranking, the lower coverage indicates that the queries issued by users at the second, third, ..., positions in sessions tend to be more diverse. In other words, it is more likely to meet a novel query at those positions. For query suggestion and URL recommendation, the lower coverage comes from a second reason. That is, the more queries a user has typed in a search session, the more likely the user ends the session. Therefore, the training data available with contexts are less than those without contexts.

Finally, vlHMM$^+$ has broader coverage than vlHMM. For document reranking, vlHMM$^+$ can improve the coverage by 22% and 33% for non-context and context cases, respectively. For query suggestion and URL recommendation, vlHMM$^+$ increases the coverage by more than 6% and 13% for non-context and context cases, respectively.



(a) On single-query sessions. (b) On multiple-query sessions.

Fig. 7. The performance of reranking by vlHMM and baselines.

6.7. Performance of Document Reranking

We conduct reranking experiments on a human-labeled dataset and click-through log dataset. The baselines used in the experiments are: (1) the click-based method which utilizes the click frequencies of the query-URL pairs for reranking; (2) random walk with restart method [Gao et al. 2009; Craswell and Szummer 2007; Deng et al. 2009; Tong et al. 2006] which utilizes the click-through bipartite graph for ranking propagation; (3) HMM which is the first-order hidden Markov model. Other related works such as [Shen et al. 2005; Xiang et al. 2010] also conduct context-aware reranking. However, since they also use document but vlHMM does not, it is not possible to conduct a fair comparison. In a word, all compared methods are log-based methods.

To show how vlHMM performs document reranking, we give out some examples here. In a test session with only two queries $q_1 \rightarrow q_2$ which can be mapped to states $s_1 \rightarrow s_2$, the results of HMM and vlHMM are identical, since both of them utilize the distribution of $P(s_1)$, $P(s_2|s_1)$, and $P(u|s_2)$. If there are three testing queries with state sequence $s_1 \rightarrow s_2 \rightarrow s_3$, the performance of vlHMM and HMM are different because HMM will utilize the probabilities $P(s_1)$, $P(s_2|s_1)$, $P(s_3|s_2)$, and $P(u|s_3)$, but vlHMM will utilize the probabilities $P(s_1)$, $P(s_2|s_1)$, $P(s_3|s_1, s_2)$, and $P(u|s_3)$. Note that $P(\cdot|s_2)$ is different from $P(\cdot|s_1, s_2)$.

In the experiments on the human-labeled dataset, 500 test sessions with single queries and 500 test sessions with multiple queries are sampled, respectively. For each test session, we crawl the top ten results of the last query and utilize the log-based methods to conduct reranking using Borda's Fusion method described in Section 5. The top search results of the last query are labeled as five levels: perfect, excellent, good, fair, and bad. For single-query test sessions, the search results are labeled only based on the last queries. For multiple-query test sessions, the annotators are asked to label the search results based on the whole sessions.

NDCG@k, a widely used information retrieval measure, is employed in reranking: $NDCG@k = \frac{1}{Z} \sum_i \frac{2^{r(i)}-1}{log_2(i+1)}$, where Z is the normalization factor and $r(i)$ is the relevance degree of the document ranked at position i. The score of $r(i)$ is set as: perfect $= 5$, excellent $= 4$, good $= 3$, fair $= 1$, and bad $= 0$. We compare the final search rankings in terms of NDCG@1, NDCG@3, and NDCG@5. The experiment results are shown in Figure 7. We find that (1) log-based methods perform better than the search engine (p-value<0.01, t-test). (2) log-based methods have comparable performance in single-query sessions in which context information is not available. (3) When context information is available, HMM, vlHMM, vlHMM$^+$ perform better than the click-based method (p-value<0.05, t-test). (4) When there are more than three queries in the test sessions, vlHMM performs best among all approaches.

We further evaluate the reranking performance on a larger user click dataset. To conduct this experiment, we construct three testing datasets from another search log

Table XIII. Mean Click Positions (MCP) of All Methods on All Test Sets

Methods	S_1 (1 query) N=18,710	S_2 (2 queries) N=26,819	S_3 (>=3 queries) N=41,042
Search Engine	1.8166	2.2154	2.6127
Click-based	1.6514	2.0451	2.3017
Rand-Walk	**1.6508**	2.0107	2.2869
HMM	1.6828	**1.9824**	2.2148
vlHMM	1.6828	**1.9824**	**2.0297**
vlHMM$^+$	1.6835	1.9850	**2.0297**

which is not used in the training process: (1) S_1: randomly sampled 18,710 sessions with single query; (2) S_2: randomly sampled 26,819 sessions with two queries; (3) S_3: sampled 41,042 sessions with more than three queries. Note that sampled testing sessions with larger than three queries can be totally matched by vlHMM, since if the testing session cannot be matched, vlHMM will degenerate to lower-order HMM. Note that those long sessions are good to show the differences between all methods, since vlHMM can capture long contexts but other methods cannot.

Specifically, for testing query q_i and clicked URL set U_i, we calculate average clicked position by a reranking method on the basis of Borda's Fusion method, described in Section 5. Then all testing cases are aggregated to get the *mean average click position* [Xiang et al. 2010] for the method: $MCP = \frac{\sum_q \sum_{u_i \in U_i} R(u_i)}{\sum_i |U_i|}$, where rank R can be derived from the log data. Smaller values of MCP indicate better performance.

The results on the user click dataset are reported in Table XIII. From the results, we can see that while context information is available, the context-based approaches (HMM, vlHMM, vlHMM$^+$) are better than non-context approaches (click-based, Rand-Walk). For single-query sessions, the performances of non-context approaches (click-based, Rand-Walk) are slightly better than context approaches (HMM, vlHMM, vlHMM$^+$). The gap of performance between context and non-context approaches is bigger at S_3, since the testing sessions with length as one or two are randomly sampled, but testing sessions with three or more queries are sampled based on the vlHMM. As previously explained, those long sessions are good for showing the differences between all methods, since vlHMM can capture long contexts but other methods cannot. VlHMM$^+$ can cover 22% and 32% more queries than vlHMM on S_1 and S_2, respectively. The performance of vlHMM$^+$ on additional queries are 1.6951 and 1.9906 at S_1 and S_2, respectively. On S_3, since vlHMM can cover all cases, the performances of vlHMM and vlHMM$^+$ are the same.

To show the effectiveness of vlHMM over baselines, we give out two kinds of examples. The first kind of example shows the effectiveness of vlHMM over non-context methods. The second kind shows the effectiveness of vlHMM over HMM.

Table XIV shows the first example of rankings for query *"ritz"* by vlHMM with and without considering the context information. As we can see, the query *"ritz"* bears intentions of *digital camera* and *luxury hotel*. While there is no context information available, the webpage of digital cameras are ranked higher. However, if the user searched *"fredericksburg va"* before, then it is more likely the user is looking for the hotel. Since the pattern of searching a "ritz" hotel after a location exists in the training logs, vlHMM can boost the ranking of hotels higher. We can infer that the user is planning a trip and the query "fredericksburg va" represents the destination. In contrast, the click-based method cannot infer this intention.

Table XV shows the second example, where vlHMM performs better reranking than HMM. While the input sequence is "online dictionary → gmail → webster", vlHMM successfully boosts the ranking of URL `http://websters-online-dictionary.org`,

Table XIV. Ranking of vlHMM with and without Using Context Information

Without context		With context	
ritz		fredericksburg va ⇒ ritz ↓ http://fredericksburgva.gov	
1	**Digital Cameras / Electronics at...** Digital camera reviews, consumer electro... *http://www.ritzcamera.com*	1	**Luxury Hotels & Luxury Resorts...** The Ritz-Carlton features luxury hotels... *http://www.ritzcarlton.com*
2	**Luxury Hotels & Luxury Resorts...** The Ritz-Carlton features luxury hotels... *http://www.ritzcarlton.com*	2	**Digital Cameras / Electronics at...** Digital camera reviews, consumer electro... *http://www.ritzcamera.com*
3	**The Fun of RITZ!** Ritz it up with the best tasting Ritz ever! *http://www.nabiscoworld.com/ritz*	3	**Luxury London Hotels - The Ritz...** The Ritz London official website... *http://www.theritzlondon.com*

Table XV. Ranking of HMM and vlHMM for Testing Sequence "online dictionary → gmail → webster"

HMM		vlHMM	
1	**Dictionary and Thesaurus...** Free online dictionary, thesaurus, ... *http://www.merriam-webster.com*	1	**Dictionary and Thesaurus.....** Free online dictionary, thesaurus, ... *http://www.merriam-webster.com*
2	**Home | Webster University** Webster University, a worldwide institution... *http://www.webster.edu*	2	**Webster's Online Dictionary...** Our mission is to create the largest... *http://websters-online-dictionary.org*

which is an online dictionary. However, HMM cannot capture the high-order context.

6.8. Performance of URL Recommendation

In this section, we evaluate the effectiveness of vlHMM in the task of URL recommendation. We use a manually labeled small-scale session dataset and large-scale session dataset. We adopt two baselines: (1) Following-Click, which borrows the idea from White et al. [2007], where browse logs are used. Given a test query q, it counts in the training data the frequency of a URL following the searches of q, and recommends the top K URLs with the highest co-occurring frequencies. Following-Click does not consider the context of q. (2) First-order HMM. Note that for URL recommendation, models are built for predicting the next states given the current and previous states, and thus the recommended URLs are generated from next possible states but not the current state, which is different from document reranking.

In the manually labeled dataset, 200 single-query sessions and 200 multiple-query sessions are randomly sampled. All the recommendation methods generate at most five recommendations. The queries and URLs in the session as well as the content of recommended URLs are presented to the annotators. The recommended URLs are labeled as meaningful, not meaningful, and unknown. The unknown cases are those which are not easy to judge, and thus are discarded.

We calculate the quality of the URL recommendations of vlHMM, vlHMM$^+$, and the baselines by calculating the ratio of meaningful cases to not-meaningful cases in the manually labeled dataset. The results are presented in Figure 8. We can observe that the performances of vlHMM and vlHMM$^+$ are better than those of Following-Click and HMM, especially on multiple-query sessions. This is because the baseline methods do not utilize context information. Using a t-test and p-value threshold 0.05, we rejected

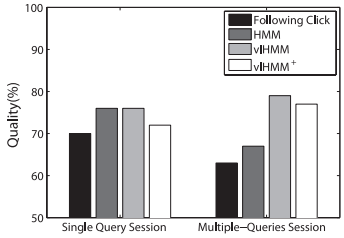
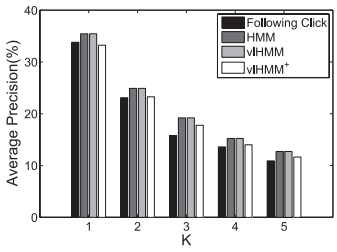
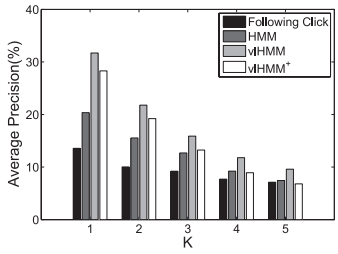


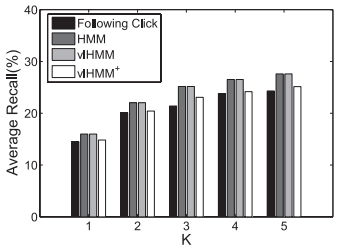
Fig. 8. Quality of URL recommendation on labeled data.



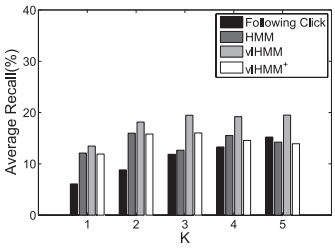
(a) Precision on single-query sessions.



(b) Precision on multiple-query sessions.



(c) Recall on single-query sessions.



(d) Recall on multiple-query sessions.

Fig. 9. Precision and recall of the URLs recommended by vlHMM and baselines.

the hypotheses that vlHMM and HMM have the same performance on multiple-query sessions and that vlHMM and Following-Click have the same performance.

Next, since human labeling is expensive, we also make use of user clicks in evaluation. Specifically, for each extracted session $O = \langle (q_1, U_1), \ldots, (q_T, U_T) \rangle$, we use q_{T-1} as the test query and consider U_T—the set of URLs clicked by the user—as the ground truth. Note that the clicked URLs are approximately taken as relevant URLs. Then we can measure the performance of the recommendation by calculating precision and recall of clicked URLs. Suppose that vlHMM recommends a set of URLs R, then precision is calculated as $\frac{|R \cap U_T|}{|R|}$ and recall is calculated as $\frac{|R \cap U_T|}{|U_T|}$. Figures 9(a) and 9(b) compare the precision of all the methods with respect to the number of recommendations K, while Figures 9(c) and 9(d) compare recalls. From these figures, we can see that precisions decrease and recalls increase when K becomes larger. This is because the larger the number of K, the more recommendations are provided. A larger number of recommendations can cover more clicked URLs but can also add more unclicked URLs. Figures 9(a) and (c) show that all the methods have comparable precisions and recalls in single-query sessions. In contrast, in multiple-query sessions, where the context information is available, vlHMM substantially outperforms the baselines.

Table XVI shows an example of URL recommendation for the query *"webster"* by vlHMM. When not utilizing context information, both Following-Click and vlHMM

Table XVI. URL Recommendation of vlHMM with and without Context Information

Without context		With context	
webster		citibank ⇒ webster ↓ https://online.citibank.com	
1	**Dictionary.com · Find the Meanings...** Dictionary.com - the largest and most... *http://dictionary.reference.com/*	1	**Bank of America · Home · Personal** Welcome to Bank of America... *https://www.bankofamerica.com*
2	**Thesaurus.com · Find Synonyms...** Thesaurus.com - the largest and most... *http://www.thesaurus.com*	2	**American Express Credit Cards...** American Express offers world-class... *https://www.americanexpress.com*
3	**Dictionary, Encyclopedia and...** Online Dictionary - Multiple dictionaries... *http://www.thefreedictionary.com*	3	**People's United Bank** Welcome to People's United Bank... *https://www.peoples.com*

conclude that the user is more likely to search for the Merriam-Webster online dictionary. Therefore, the top recommended websites are related to the online dictionary. However, if the user searched for *"citibank"* before, it is more likely to search for the Webster online bank. With the context information, vlHMM recommends websites about online banking, which seems to be more reasonable.

6.9. Performance of Query Suggestion

We follow the evaluation method in Cao et al. [2008] and Liao et al. [2011] to evaluate the performance of query suggestions made by the vlHMM methods. We compare vlHMM and vlHMM$^+$ with several baselines: (1) context-aware concept-based approach [Cao et al. 2008; Liao et al. 2011] (denoted as CACB), (2) first-order HMM (denoted as HMM), (3) state-based co-occurrence method [Huang et al. 2003] (denoted as Co-occur) and (4) set-based context method (denoted as Set-based). Note that HMM and vlHMM exploit sequential contexts, while co-occur and set-based approaches exploit co-occurring or set contexts.

Here we describe the training process of all models by representing the training data as a set of tuples (sequence, suggestion).

(1) Given a sequence ($s_1 \rightarrow s_2$), methods vlHMM, CACB, HMM, Set-based generate tuple (s_1, s_2), while method Co-occur generates tuples (s_1, s_2) and (s_2, s_1).
(2) Given a sequence ($s_1 \rightarrow s_2 \rightarrow s_3$), methods vlHMM and CACB generate tuples (s_1, s_2), ($s_1 \rightarrow s_2$,s_3), method HMM generates (s_1,s_2), (s_2,s_3), method Co-occur generates (s_i,s_j), where $i \neq j$, $1 \leq i \leq 3$, and $1 \leq j \leq 3$, and method Set-based generates (s_1, s_2), ($s_1 \rightarrow s_2$, s_3), ($s_2 \rightarrow s_1$, s_3).
(3) Given a sequence ($s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4$), methods vlHMM and CACB generate (s_1,s_2), ($s_1 \rightarrow s_2$, s_3), and ($s_1 \rightarrow s_2 \rightarrow s_3$, s_4), method HMM generates $s_i \rightarrow s_{i+1}$ where $1 \leq i \leq 3$; method Co-occur generates (s_i, s_j) where $i \neq j$, $1 \leq i \leq 4$, and $1 \leq j \leq 4$, and method Set-based generates (s_1, s_2), ($s_1 \rightarrow s_2$, s_3), ($s_2 \rightarrow s_1$, s_3), ($s_1 \rightarrow s_2 \rightarrow s_3$, s_4), ($s_1 \rightarrow s_3 \rightarrow s_2$, s_4), ($s_2 \rightarrow s_1 \rightarrow s_3$, s_4), ($s_2 \rightarrow s_3 \rightarrow s_1$, s_4), ($s_3 \rightarrow s_1 \rightarrow s_2$, s_4), ($s_3 \rightarrow s_2 \rightarrow s_1$, s_4).

After generating the (sequence, suggestion) tuples, we aggregate all pairs having the same sequence for query suggestion.

As described in Table VIII, we have 10,734,074, 407,846, 44,313, and 10,950 sessions of lengths 2, 3, 4, and 5, respectively. We report the number of candidate sequences generated by all models in Table XVII. We have following observations based on the results.

Table XVII. Number of Candidate Sequences Generated by All Models

candidate sequence	Co-occur	Set-based	HMM	vlHMM/CACB
(q_1, q_s)	24,665,980	12,078,444	11,726,505	11,197,183
$(q_1 \rightarrow q_2, q_s)$	—	1,344,370	—	463,109
$(q_1 \rightarrow q_2 \rightarrow q_3, q_s)$	—	528,678	—	55,263
$(q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4, q_s)$	—	262,800	—	10,950

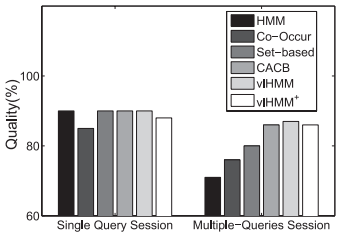


Fig. 10. Quality of query suggestion on labeled data.

(1) Both vlHMM and HMM can use context information to infer the hidden states, which is good for capturing the user intention of ambiguous queries. However, since in HMM $P(s_t|s_{t-1}, s_{t-2}) = P(s_t|s_{t-1})$, then HMM can only generate adjacent pairs based on training data. This leads to weak capability of leveraging contextual information.
(2) Both vlHMM and Set-based can use contextual information for providing suggestions. However, Set-based will generate more unseen sequences based on observed sequences. On the one hand, Set-based will cover more testing cases based on combinatorial observation generation. On the other hand, its precision will decrease due to the improvement of recall. That is because users are searching sequentially and previous queries may not be good suggestions for later queries. For example, users may begin a session with a typo keyword (such as "googgle"), the name of a search engine (such as "Bing"), or a frequently visited website (such as "Facebook", "Gmail"), while using these first queries, as suggestions by the Co-occur and Set-based methods may decrease the quality of suggestions.

Moreover, to improve the coverage of vlHMM in the test stage, we set a back-off if the given test sequence cannot be captured, as described in Section 5. For example, given a test case $(q_1 \rightarrow q_2)$, we use vlHMM to infer its state sequence $(s_1 \rightarrow s_2)$ and apply all models on it. If $s_1 \rightarrow s_2$ is not observed before, vlHMM backs off to s_2.

We randomly sample 500 single-query sessions and 500 multiple-query sessions as testing datasets. Each method is allowed to provide up to five suggestions for one testing case. Then the judges are given the test and suggested queries with their top search results and required to judge whether the suggestion is good or bad. If a suggestion is labeled as bad, the method is considered to make an error. The overall suggestion quality is evaluated by accuracy of suggestions. Figure 10 shows the labeling results. From the figure, we can observe that (1) while context information is not used, Co-occur performs worse than HMM, CACB, vlHMM, and vlHMM$^+$; (2) while context information is used, CACB, vlHMM, and vlHMM$^+$ perform better than HMM & Co-occur, and Co-occur performs better than HMM; (3) vlHMM, vlHMM$^+$, and CACB have similar performances. In our experiments, the coverage of CACB on multiple-query sessions is 29.45%, while vlHMM reaches 34.01% and vlHMM$^+$ further improves to 38.67%. The reason being that vlHMM utilizes the clusters which are the same as CACB, but vlHMM assigns more queries to new states, as shown in Section 4.2. (4) vlHMM$^+$ covers

Table XVIII. Query Suggestions by Co-occur and HMM

Test Case	Co-occur	HMM
007	007 games james bond james bond 007 movies oo7 007 actors	007 games james bond james bond 007 movies 007 actors daniel craig
"1 day discount disneyland tickets "	disneyland tickets disneyland ticket disneyland com disneyland discount tickets disneyland	discount single day disneyland ticket aaa discount disneyland ticket free disneyland tickets disneyland ticket prices disneyland tickets

Table XIX. Query Suggestions by vlHMM, Co-occur, and HMM

Method	Test Cases	
	edmonton journal → calgary herald	the sun → the mirror
HMM	edmonton journal google hotmail lethbridge herald calgary herald obituaries	the sun the mail daily star google digital spy
Co-occur	google hotmail the telegram vancouver province newspaper kijiji	the mail the daily star google the star digital spy
vlHMM	the telegram mississauga library medicine hat news janet charlton vancouver province	sky news uk chronicle echo shopping telly mr paparazzi the daily star

6% and 13% more queries than vlHMM in single and multiple testing cases. The overall qualities of vlHMM$^+$ are 0.88 and 0.86 at non-context and context test cases, and the qualities of vlHMM$^+$ on additional queries are 0.80 and 0.81 at non-context and context test cases. Compared to vlHMM, which has qualities of 0.90 and 0.87, vlHMM$^+$ sacrifices some accuracy for coverage. Using t-test and p-value threshold 0.05, we rejected the hypotheses that vlHMM and Co-occur have the same performance on single-query sessions, and that vlHMM, Co-occur, HMM, and Set-based have the same performance on multiple-query session. At the same time, the statistical significance test suggests that vlHMM, vlHMM$^+$, and CACB are not significantly different in performance.

In the following, we show three kinds of examples to compare different approaches. The first kind of example shows the effectiveness of HMM over Co-occur. The second shows the effectiveness of vlHMM over HMM. The third shows the effectiveness of vlHMM over the Set-based method.

Table XVIII shows the first kind of example. The Co-occur and HMM methods are compared to show why we should consider the order of queries for query suggestion. For example, for query "007", Co-occur suggests "oo7", which appears to be a typo of "007" and is not necessary. For query "1 day discount disneyland tickets", Co-occur suggests more general queries, which are not as good as suggestions from HMM.

Table XIX shows the second kind of example in which context information is available. From the table, we can observe that HMM tends to suggest queries which have

Table XX. Query suggestion by vlHMM and Set-Based

Method	Test Cases	
	ask jeeves → google	trade it → gmail
Set-Based	yahoo	bristol gumtree
	ask jeeves question	bristol homechoice
	ask jeeves kids	homeswapper
	google images	
	ask	
vlHMM	google images	gmail login
	google maps	gmail settings
	wikipedia	ebay
	yahoo	craigslist
	google services	gmail chat

been observed before. Co-occur tends to boost popular but irrelevant queries, such as "google". In contrast, vlHMM can suggest queries which are more relevant given the contexts. Therefore, the performances of vlHMM and vlHMM$^+$ are the best.

Table XX shows the third kind of example to compare vlHMM and the Set-based method. From the table, we can observe that the suggestions generated by vlHMM are better than those of set-based method. For example, given a query sequence "ask jeeves → google", we may conjecture that users will want to go to some search engine for further searching. Therefore, the suggestion "yahoo" seems to be reasonable in the Set-based and vlHMM methods. However, the set-based method will suggest suggestions, such as "ask jeeves question", since it observes the sequence "google → ask jeeves → ask jeeves question" from the training data. Based on the annotations, this kind of suggestions is not useful for sequence "ask jeeves → google". In contrast, vlHMM will provide better suggestions, such as "google maps" and "wikipedia". It is the same for the second example. Overall, vlHMM is better than Set-based method.

7. CONCLUSIONS

In this article, we proposed a general approach to context-aware search by learning a variable length hidden Markov model (vlHMM) from search sessions extracted from log data. We tackled the challenges of learning a large vlHMM with millions of states from hundreds of millions of search sessions by developing a strategy for parameter initialization which can greatly reduce the number of parameters to be estimated in practice. We devised a method for distributed vlHMM learning under the map-reduce model, and we further developed several techniques for improving the efficiency of the distributed learning substantially. Moreover, to handle new queries, we constructed both term and URL feature structures for the states in the learned vlHMM model.

In the experiments, we verified the observation that users' search behaviors are different in different context settings. We compared vlHMM with several baselines in document reranking, document recommendation, and query suggestions. We found that vlHMM works well with long context, especially when the test cases have more than two queries. Particularly, in document reranking, vlHMM can outperform traditional HMM when test cases have more than three queries. In query suggestion, vlHMM can outperform HMM when test cases have more than two queries. We validated that the sequential context models (e.g., HMM and vlHMM) can outperform the non-sequential models (e.g., Co-occur and Set-based approaches) in query suggestion. Besides, our acceleration approach can reduce 70% of training time for vlHMM. The term and URL feature structures for handling new queries can improve the coverage of vlHMM by about 20% for document reranking and about 10% for query suggestion.

There are several possible extensions of this work. First, we only modeled the search behaviors as context in this work, while we believe that the browsing behaviors are also important for context modeling. Second, the states of our vlHMM were initialized via the clustering results on a query-URL bipartite graph. The reason being that we want to obtain the states in an efficient way. However, there can be various ways to construct states. For examples, we can randomly assign queries or URLs into a fixed number of states and train the vlHMM or learn a topical model, such as LDA, on the session data. It is interesting to compare the efficiency and effectiveness of different approaches for state initialization. Third, our vlHMM is an unsupervised approach, where the model is trained without human supervision. It is better if we can leverage small amounts of labeled data to improve the model. One possible way is to try semisupervised or weakly-supervised methods. Last but not least, we trained vlHMM with a maximum order of 5 and pruned less frequent and longer sessions. The reason being that we want to improve the accuracy of the model by pruning noisy data. It is better if we can mine the cross-session contexts and extend our approach in even longer contexts. In addition, better denoising techniques are also preferred for processing the huge amount of search logs.

REFERENCES

Anagnostopoulos, A., Becchetti, L., Castillo, C., and Gionis, A. 2010. An optimization framework for query recommendation. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining (WSDM'10)*. ACM, New York, NY, 161–170.

Baeza-Yates, R. A. and Tiberiet, A. 2007. Extracting semantic relations from query logs. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*. 76–85.

Baeza-Yates, R. A., Hurtado, C., and Mendoza, M. 2004. Query recommendation using query logs in search engines. In *Proceedings of the 9th International Conference on Extending Database Technology (EDBT'04) Workshop on Clustering Information over the Web*. 588–596.

Baum, L., Petrie, T., Soules, G., and Weiss, N. 1970. A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains. *Ann. Math. Statist. 41,* 1, 164–171.

Beeferman, D. and Berger, A. 2000. Agglomerative clustering of a search engine query log. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*. 407–416.

Bilmes, J. 1998. A gentle tutorial of the EM algorithm and its application to parameter estimation for gaussian mixture and hidden markov models. Tech. rep. Intetrnational Computer Science Institute, Berkley, CA.

Boldi, P., Bonchi, F., Castillo, C., Donato, D., and Vigna, S. 2009. Query suggestions using query-flow graphs. In *Proceedings of the Workshop on Web Search Click Data (WSCD'09)*. 56–63.

Boldi, P., Bonchi, F., Castillo, C., Donato, D., Gionis, A., and Vigna, S. 2008. The query-flow graph: Model and applications. In *Proceeding of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*. 609–618.

Borda, J. C. 1781. Mémoire sur les élections au scrution. *Histoire de l'Académie Royal des Sciences*.

Cao, H., Jiang, D., Pei, J., Chen, E., and Li, H. 2009. Towards context-aware search by learning a very large variable length hidden Markov model from search logs. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*. 191–200.

Cao, H., Jiang, D., Pei, J., He, Q., Liao, Z., Chen, E., and Li, H. 2008. Context-aware query suggestion by mining click-through and session data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Chapelle, O. and Zhang, Y. 2009. A dynamic bayesian network click model for Web search ranking. In *Proceedings of the 18th International Conference on World Wide Web (WWW'09)*. 1–10.

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yn, Y., Bradski, G., Ng, A. Y., and Olukotun, K. 2006. Map-reduce for machine learning on multicore. In *Proceedings of the 20th Annual Conference on Neural Information Processing Systems (NIPS'06)*. MIT Press, Combridge, MA, 281–288.

Craswell, N. and Szummer, M. 2007. Random walks on the click graph. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*. 239–246.

CRASWELL, N., ZOETER, O., TAYLOR, M., AND RAMSEY, B. 2008. An experimental comparison of click position-bias models. In *Proceedings of the 1st ACM International Conference on Web Search and Data Mining (WSDM'08)*. 87–94.

DEAN, J. AND GHEMAWAT, S. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'04)*. USENIX Association, Berkeley, CA.

DEMPSTER, A. P., LAIRD, N. M., AND RUBIN, D. B. 1977. Maximal likelihood from incomplete data via the EM algorithm. *J. Royal Stat. Soci. Ser B,* 39, 1–38.

DENG, H., KING, I., AND LYU, M. R. 2009. Entropy-biased models for query representation on the click graph. In *Proceedings of the 32th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'09)*. 339–346.

DONATO, D., BONCHI, F., CHI, T., AND MAAREK, Y. 2010. Do you want to take notes?: Identifying research missions in Yahoo! search pad. In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*. ACM, New York, NY, 321–330.

DUPRET, G. E. AND PIWOWARSKI, B. 2008. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st Annual ACM SIGIR International Conference on Research and Development in Information Retrieval (SIGIR'08)*. 331–338.

DURBIN, R., EDDY, S. R., KROGH, A., AND MITCHISON, G. 1999. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, U.K.

ESTER, M., KRIEGEL, H., SANDER, J., AND XU, X. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*. 226–231.

FAGIN, R., KUMAR, R., AND SIVAKUMAR, D. 2003. Comparing top k lists. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'03)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 28–36.

FONSECA, B. M., GOLGHER, P., PÔSSAS, B., RIBEIRO-NETO, B., AND ZIVIANI, N. 2005. Concept-based interactive query expansion. In *Proceedings of the ACM CIKM International Conference on Information and Knowledge Management (CIKM'05)*. 696–703.

FOX, S., KARNAWAT, K., MYDLAND, M., DUMAIS, S., AND WHITE, T. 2005. Evaluating implicit measures to improve Web search. *ACM Trans. Inf. Syst. 23*, 147–168.

GAO, J., YUAN, W., LI, X., DENG, K., AND NIE, J.-Y. 2009. Smoothing clickthrough data for Web search ranking. In *Proceedings of the 32th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'09)*. 355–362.

GUO, F., LIU, C., AND WANG, Y.-M. 2009. Efficient multiple-click models in Web search. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining (WSDM'09)*. 124–131.

HASSAN, A., JONES, R., AND KLINKNER, K.-L. 2010. Beyond DCG: User behavior as a predictor of a successful search *(WSDM'10)*. ACM, New York, NY, 221–230.

HUANG, C., CHIEN, L., AND OYANG, Y. 2003. Relevant term suggestion in interactive Web search based on contextual information in query session logs. *J. Am. Soc. Inf. Sci. Technol. 54,* 7, 638–649.

JENSEN, E. C, BEITZEL, S., CHOWDHURY, A., AND FRIDER, O. 2006. Query phrase suggestion from topically tagged session logs. In *Proceedings of the 7th International Conference on Flexible Query Answering Systems (FQAS'06)*. Lecture Notes in Computer Science, vol. 4027, Springer, Berlin Heidelberg, 185–196.

JOACHIMS, T. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'02)*. ACM, New York, NY.

JOACHIMS, T., GRANKA, L., PAN, B., HEMBROOKE, H., AND GAY, G. 2005. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*. 154–161.

JONES, R. AND KLINKNER, K. L. 2008. Beyond the session timeout: Automatic hierarchical segmentation of search topics in query logs. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*. ACM, New York, NY, 699–708.

JONES, R., REY, B., MADANI, O., AND GREINER, W. 2006. Generating query substitutions. In *Proceedings of the 15th International Conference on World Wide Web (WWW'06)*. ACM, New York, NY, 387–396.

KOTOV, A., BENNETT, P., WHITE, R., DUMAIS, S., AND TEEVAN, J. 2005. Modeling and analysis of cross-session search tasks. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*.

LIAO, Z., JIANG, D., CHEN, E., PEI, J., CAO, H., AND LI, H. 2011. Mining concept sequences from large-scale search logs for context-aware query suggestion. *ACM Trans. Intell. Syst. Technol. 3*, 17:1–17:40.

Liao, Z., Song, Y., He, L.-W., and Huang, Y. 2012. Evaluating the effectiveness of search task trails. In *Proceedings of the 21st International Conference on World Wide Web (WWW'12)*. ACM, New York, NY, 489–498.

Lucchese, C., Orlando, S., Perego, R., Silvestri, F., and Tolomei, G. 2011. Identifying task-based sessions in search engine query logs. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining (WSDM'11)*. ACM, New York, NY, 277–286.

Mei, Q., Klinkner, K., Kumar, R., and Tomkins, A. 2009. An analysis framework for search sequences. In *Proceeding of the 18th ACM Conference on Information and Knowledge Management (CIKM'09)*. 1991–1996.

Mei, Q., Zhou, D., and Church, K. 2008. Query suggestion using hitting time. In *Proceeding of the 17th ACM Conference on Information and Knowledge Management (CIKM'08)*. 469–478.

Rabiner, L. R. 1989. A tutorial on hidden Markov models and selected applications inspeech recognition. *Proc. IEEE 77,* 2, 257–286.

Radlinski, F. and Joachims, T. 2005. Query chains: Learning to rank from implicit feedback. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'05)*. ACM, New York, NY.

Sadikov, E., Madhavan, J., Wang, L., and Halevy, A. 2010. Clustering query refinements by user intent. In *Proceedings of the International Conference on World Wide Web (WWW'10)*. 841–850.

Shen, X., Tan, B., and Zhai, C.-X. 2005. Context-sensitive information retrieval using implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'05)*. ACM, New York, NY, 43–50.

Tong, H., Faloutsos, C., and Pan, J.-Y. 2006. Fast random walk with restart and its applications. In *Proceedings of the 6th International Conference on Data Mining (ICDM'06)*. IEEE Computer Society, Washington, DC, 613–622.

Wang, Y., Zhou, L., Feng, J., Wang, J., and Lin, Z.-Q. 2006. Mining complex time-series data by learning Markovian models. In *Proceedings of the 6th International Conference on Data Mining (ICDM'06)*. IEEE Computer Society, Washington, DC, 1136–1140.

Wen, J., Nie, J., and Zhang, H. 2001. Clustering user queries of a search engine. In *Proceedings of the 10th International Conference on World Wide Web (WWW'01)*. 162–168.

White, R. W., Bailey, P., and Chen, L. 2009. Predicting user interests from contextual information. In *Proceedings of the 32nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'09)*. 363–370.

White, R. W., Bennett, P. N., and Dumais, S. T. 2010. Predicting short-term interests using activity-based search context. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM'10)*. 1009–1018.

White, R. W., Bilenko, M., and Cucerzan, S. 2007. Studying the use of popular destinations to enhance Web search interaction. In *Proceedings of the 30th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'07)*. 159–166.

Xiang, B., Jiang, D., Pei, J., Sun, X., Chen, E., and Li, H. 2010. Context-aware ranking in Web search. In *Proceedings of the 33rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'10)*. ACM, 451–458.

Zhao, M., Li, H., Ratnaparkhi, A., Hon, H.-W., and Wang, J. 2006. Adapting document ranking to users preferences using click-through data. In *Proceedings of the Asia Information Retrieval Symposium (AIRS'06)*. 26–42.

Received June 2011; revised February, August 2012, January, May 2013; accepted May 2013