# Towards Context-Aware Search by Learning A Very Large Variable Length Hidden Markov Model from Search Logs

Huanhuan Cao[1][*] Daxin Jiang[2] Jian Pei[3] Enhong Chen[1] Hang Li[2]
[1]University of Science and Technology of China [2]Microsoft Research Asia [3]Simon Fraser University
[1]{caohuan, cheneh}@ustc.edu.cn [2]{djiang, hangli}@microsoft.com [3]jpei@cs.sfu.ca

## ABSTRACT

Capturing the context of a user's query from the previous queries and clicks in the same session may help understand the user's information need. A context-aware approach to document re-ranking, query suggestion, and URL recommendation may improve users' search experience substantially. In this paper, we propose a general approach to context-aware search. To capture contexts of queries, we learn a *variable length Hidden Markov Model* (*vlHMM*) from search sessions extracted from log data. Although the mathematical model is intuitive, how to learn a large vlHMM with millions of states from hundreds of millions of search sessions poses a grand challenge. We develop a strategy for parameter initialization in vlHMM learning which can greatly reduce the number of parameters to be estimated in practice. We also devise a method for distributed vlHMM learning under the *map-reduce* model. We test our approach on a real data set consisting of 1.8 billion queries, 2.6 billion clicks, and 840 million search sessions, and evaluate the effectiveness of the vlHMM learned from the real data on three search applications: document re-ranking, query suggestion, and URL recommendation. The experimental results show that our approach is both effective and efficient.

## Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process

## General Terms

Algorithms, Experimentation

## Keywords

Context-aware search, variable length Hidden Markov Model

## 1. INTRODUCTION

Capturing the context of a user's query from the previous queries and clicks in the same session may help understand the user's information need. A context-aware approach to document re-ranking, query suggestion, and URL recommendation may improve users' search experience substantially.

Example 1 (Motivation). Ada plans to buy a new car. Thus, she wants to compare the models, price, availability, and deals of various brands. If Ada does not know how to formulate an effective query to describe her information need, she may issue a series of queries to search for individual brands and models and browse many results one by one. Actually, what Ada needs is some review web sites such as `www.autohome.com`. This web site is returned as a search result when a user raises a query such as *"Ford new cars"* or *"Toyota new cars"*. However, since most search engines treat a user query individually and rank the documents based on the relevance to only the current query (e.g., *"Ford new cars"*), the review web site is usually ranked lower than the sites/pages dedicated to the brand in the query. Consequently, it is not easy for Ada to notice the review web site.

| SID | Search session |
|---|---|
| $S_1$ | Ford $\Rightarrow$ Toyota $\Rightarrow$ GMC $\Rightarrow$ Allstate<br>$\downarrow$<br>www.autohome.com |
| $S_2$ | Ford cars $\Rightarrow$ Toyota cars $\Rightarrow$ GMC cars $\Rightarrow$ Allstate<br>$\downarrow$<br>www.autohome.com |
| $S_3$ | Ford cars$\Rightarrow$ Toyota cars$\Rightarrow$ Allstate<br>$\downarrow$<br>www.allstate.com |
| $S_4$ | GMC $\Rightarrow$ GMC dealers<br>$\downarrow$<br>www.gmc.com |

**Table 1: Four search sessions.**

To understand why context-aware search may help, let us consider the search log data shown in Table 1 which contains four sessions. The '$\downarrow$' symbol indicates the user clicks a URL.

A pattern in the sessions is that *in* 50% *of the sessions, a user clicks on* `www. autohome. com` *after asking a series of queries about different brands of cars.* Using this pattern, a search engine should promote URL `www.autohome.com` in the ranked list of search results for Ada. Another pattern that can be found in the sessions is that *in* 75% *of the sessions, after a series of queries about different brands of cars, users will search for car insurance.* Using this context information, a search engine can provide the corresponding query and URL suggestions to improve users' search experience. ■

[*]The work was done when Huanhuan Cao was an intern at Microsoft Research Asia.

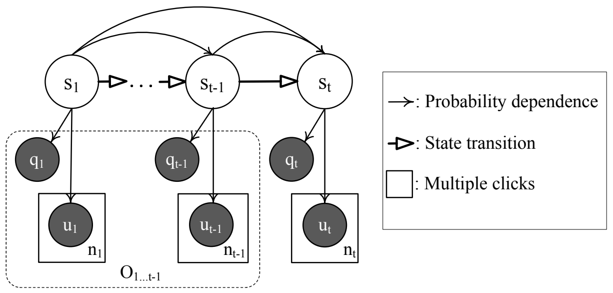


**Figure 1: Graphical structure of the vlHMM.**

Modeling query contexts by mining search sessions is a fundamental and challenging problem. Although some progress has been made by the previous studies [6, 15, 23], most of them only consider correlations within query pairs. Such a method cannot capture well the contexts exemplified in Example 1 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 comprehensive model which can be used simultaneously for multiple applications such as document re-ranking, query suggestion, and URL recommendation.

In this paper, we propose modeling query contexts by a *variable length Hidden Markov Model* (*vlHMM* for short) for context-aware search. As well recognized by many previous studies such as [12], a user often raises multiple queries and conducts multiple rounds of interaction with a search engine for an information need. For instance, in Example 1, Ada may decompose her general search task, comparing various cars, into several specific sub-tasks, such as searching the cars of Ford. In each sub-task, Ada may bear a particular search intent in mind and formulate a query to describe the intent. Moreover, Ada may selectively click on some related URLs to browse. A Hidden Markov Model (HMM for short) naturally describes the search process. We can model each search intent as a state of the HMM, and consider the query and clicked URLs as observations generated by the state. The whole search process can be modeled as a sequence of transitions between states as illustrated in Figure 1. It is possible that a state transits to itself.

Let $q_t$ be the current query. The vlHMM can re-rank the search results by the posterior probability distribution $P(s_t|q_t, O_{1\cdots t-1})$, where $s_t$ is the current search intent hidden in the user mind 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 of those queries. 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. For instance, in Example 1, given the current query *"GMC"* alone, the probability of Ada searching for the homepage of GMC is likely to be higher than that of searching for car review web sites. Therefore, the company homepage is ranked higher than `www.autohome.com` and suggestions such as *"GMC dealers"* are more likely to be served. However, given the context $O_{1\cdots t-1}$ that Ada has input a series of car companies and clicked corresponding homepages, the probability of searching for car review sites may significantly increase, while the probability of searching for the GMC homepage may decrease. Consequently, the vlHMM will boost the car review web sites, and provide suggestions about car insurance instead of the specific car dealers.

Learning Hidden Markov Models and variants has been well adopted in many applications. However, applying the Hidden Markov Models in context-aware search is far from trivial. Due to the extremely large search session data, it is impractical to apply the existing algorithms. In a commercial search engine, search logs may contain hundreds of millions of training examples. For instance, in the data set used in our experiments, there are 840 million search sessions. Moreover, search logs may contain millions of unique queries and URLs, e.g., 1.8 million unique queries and 8.3 million unique URLs in our experimental data, which makes the number of model parameters extremely large.

To tackle the problem, in this paper, we develop a strategy for parameter initialization in vlHMM learning which can greatly reduce the number of parameters to be estimated in practice. We also devise a method for distributed vlHMM learning under the *map-reduce* model [9]. Once the vlHMM is trained offline, it can be used online to support context-aware search in various applications such as document re-ranking, query suggestion, and URL recommendation.

We make the following contributions in this paper. First, we propose a novel model to support context-aware search. To the best of our knowledge, this is the first approach towards comprehensive context modeling for context-aware search. Second, we develop efficient algorithms and strategies for learning a very large vlHMM from huge log data. Finally, we test our approach on a real log data set containing 1.8 billion queries, 2.6 billion clicks, and 840 million search sessions. We examine the effectiveness of the trained vlHMM on three search applications: document re-ranking, query suggestion, and URL recommendation, as well as the efficiency of our training algorithms. The experimental results clearly show our approach is effective and efficient.

The rest of the paper is organized as follows. Section 2 briefly reviews the related work. In Section 3, we introduce the vlHMM model for search logs. The training methods of 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. The paper is concluded in Section 7.

## 2. RELATED WORK

Traditional approaches to query understanding often rely on explicit, implicit, or pseudo relevance feedback (e.g., [14, 18, 20]) or user profiles (e.g., [7, 19]). In terms of modeling contexts of queries, Bai *et al.* [3] constructed a language model using the context both around and within a query. Although these techniques are helpful to understand users' search intent, they do not use search log data and do not provide query or URL suggestions.

Some recent studies mined users' search or browsing logs and used wisdom of the crowds to improve users' search experience. For example, White *et al.* [23] proposed mining popularly visited web sites from browsing logs and recommending to a user the web sites frequently visited by other users with similar information needs. Huang *et al.* [13] mined query pairs frequently co-occurring in search sessions, and used the queries in those pairs as suggestions for each other. Fonseca *et al.* [12] and Jones *et al.* [15] extracted adjacent queries in sessions for query expansion and query substitution, respectively. Beeferman *et al.* [5], Wen*et al.* [22], and Baeza-Yates *et al.* [1] applied various clustering algorithms to find similar queries from click-through data and used them as candidates in query suggestion. Those studies can be regarded as initiatives to model partial contexts of queries using simple mechanisms. However, query contexts are not comprehensively modeled due to the limitation of the mechanisms which often consider only query pairs instead of a sequence of queries.

In [6], the authors proposed a context-aware and concept-based method *CACB* for query suggestion. CACB considers the user's previous queries in the same session as the context for query suggestion. Moreover, to make query suggestion robust, CACB summarizes similar queries into concepts, where each concept corresponds to a search intent.

The vlHMM approach developed in this paper is fundamentally different from CACB in three aspects. First, CACB constrains that each query represents only one search intent (concept). This limits the applicability of CACB since a query in practice (e.g., "java") may be used by different users for different search intents. By contrast, in the vlHMM approach, a query may be associated with multiple search intents with a probability distribution, which makes it more general and powerful than CACB.

Second, CACB only considers the previous queries in the same session as the context of a query. In the vlHMM approach, we consider both previous queries and previous clicks in the same session as the context. Therefore, vlHMM exploits more information available and thus can capture contexts of queries more accurately.

Finally, CACB is only able to provide query suggestions; it cannot handle other search applications such as document re-ranking and URL recommendation. By contrast, the vlHMM approach is a general model which can be used to support context-aware search in various applications simultaneously.

HMMs have been widely used in various domains such as speech recognition [17] and bioinformatics [11]. In [21], Wang *et al.* developed the notion of vlHMMs 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 paper. Our approach is critically different from [21]: 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 [4], which is essentially an EM algorithm [10]. 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.* [8] applied the *map-reduce* programming model [9] to a variety of learning algorithms. However, how to train a very large HMM from huge log data remains a challenging open problem.

## 3. VLHMM MODEL

We choose a Hidden Markov Model rather a Markov Chain to model query contexts because search intents are not observable. Different users may raise different queries to describe the same search intent. For example, to search for Microsoft Research Asia, queries "*Microsoft Research Asia*", "*MSRA*" or "*MS Research Beijing*" may be formulated. Moreover, even two users raise exactly the same query, they may choose different URLs to browse. If we model individual queries and URLs directly as states in a Markov Chain, we not only increase the number of states and thus the complexity of the model, but also lose the semantic relationship among the queries and the URLs clicked by the same search intent. 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 HMMs (1-HMMs) have been widely used in various applications such as speech recognition [17] and bioinformatics [11]. For context-aware search, we choose higher order HMMs. This is because the 1-HMM assumes 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 Ford cars at time point $t-1$, the probability that the user searches for GMC cars at the current time point $t$ still depends on the states $s_1 \ldots s_{t-2}$. As an intuitive instance, that probability will be smaller if the user searched for GMC cars at any time point before $t-1$. Therefore we consider higher order HMMs rather than 1-HMMs. In particular, we consider the vlHMM instead of the fixed-length HMM because the vlHMM is more flexible to adapt to variable lengths of user interactions in different search sessions.

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 vlHMM is a probability model defined as follows.

- The transition probability distribution $\Delta = \{P(s_i|S_j)\}$, where $S_j$ is a state sequence of 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 $s_{j,t}(1 \leq t \leq T_j)$ is the $t$-th 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 set of URLs $U$ from state $s_{j,T_j}$ after the user's $(T_j - 1)$ steps of transitions from state $s_{j,1}$ to $s_{j,T_j}$.

To keep the model simple, given a user is currently at state $s_{j,T_j}$, we assume the emission probability is independent of the user's previous search states $s_{j,1} \ldots s_{j,T_j-1}$, i.e., $P(q, U|S_j) \equiv P(q, U|s_{j,T_j})$. Moreover, we assume that query $q$ and URLs $U$ are *conditionally* independent given the state $s_{j,T_j}$, i.e., $P(q, U|s_{j,T_j}) \equiv P(q|s_{j,T_j}) \prod_{u \in U} P(u|s_{j,T_j})$. Under the above two assumptions, the emission probability distribution $\Lambda$ becomes $(\Lambda_q, \Lambda_u) \equiv (\{P(q|s_i)\}, \{P(u|s_i)\})$.

The 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 queries and click events. We can extract and sort each user's events and then derive sessions based on a widely-used method [23]: two consecutive events (either queries or clicks) are segmented into two sessions if the time interval between them exceeds 30 minutes. The sessions formed as such are then used as training examples. Let $\mathcal{X} = \{O_1, \ldots, O_N\}$ be the set of training sessions, where a session $O_n$ $(1 \le n \le N)$ of length $T_n$ is a sequence of pairs $\langle (q_{n,1}, U_{n,1}) \ldots (q_{n,T_n}, U_{n,T_n}) \rangle$, where $q_{n,t}$ and $U_{n,t}$ $(1 \le t \le T_n)$ are the $t$-th query and the set of clicked URLs among the query results, respectively. Moreover, we use $u_{n,t,k}$ to denote the $k$-th URL $(1 \le k \le |U_{n,t}|)$ in $U_{n,t}$.

We use the maximum likelihood method to estimate parameters $\Theta$. We want to find $\Theta^*$ such that

$$\Theta^* = \arg\max_{\Theta} \ln P(\mathcal{X}|\Theta) = \arg\max_{\Theta} \sum_n \ln P(O_n|\Theta) \quad (1)$$

Let $\mathcal{Y} = \{S_1 \ldots, S_M\}$ be the set of all possible state sequences, $s_{m,t}$ be the $t$-th state in $S_m \in \mathcal{Y}$ $(1 \le m \le M)$, and $S_m^{t-1}$ be the subsequence $s_{m,1}, \ldots, s_{m,t-1}$ of $S_m$. Then, the likelihood can be written as $\ln P(O_n|\Theta) = \ln \sum_m P(O_n, S_m|\Theta)$, and the joint distribution can be written as

$$\begin{aligned} P(O_n, S_m|\Theta) &= P(O_n|S_m, \Theta)P(S_m|\Theta) \\ &= \left( \prod_{t=1}^{T_n} P(q_{n,t}|s_{m,t}) \prod_k P(u_{n,t,k}|s_{m,t}) \right) \\ &\times \left( P(s_{m,1}) \prod_{t=2}^{T_n} P(s_{m,t}|S_m^{t-1}) \right). \end{aligned} \quad (2)$$

Since optimizing the likelihood function in an analytic way may not be possible, we employ an iterative approach and apply the *Expectation Maximization* algorithm (*EM* for short) [10].

At the *E-Step*, we have

$$\begin{aligned} Q(\Theta, \Theta^{(i-1)}) &= \mathbf{E}\left[ \ln P(\mathcal{X}, \mathcal{Y}|\Theta) | \mathcal{X}, \Theta^{(i-1)} \right] \\ &= \sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \ln P(O_n, S_m|\Theta)], \end{aligned} \quad (3)$$

where $\Theta^{(i-1)}$ is the set of parameter values estimated in the last round of iteration. $P(S_m|O_n, \Theta^{(i-1)})$ can be written as

$$P(S_m|O_n, \Theta^{(i-1)}) = \frac{P(O_n, S_m|\Theta^{(i-1)})}{P(O_n|\Theta^{(i-1)})}. \quad (4)$$

Substituting Equation 2 in Equation 4, and then substituting Equations 2 and 4 in Equation 3, we get

$$\begin{aligned} Q(\Theta, \Theta^{(i-1)}) \propto & \sum_{n,m} \left( \prod_{t=1}^{T_n} P^{(i-1)}(q_{n,t}|s_{m,t}) \prod_k P^{(i-1)}(u_{n,t,k}|s_{m,t}) \right) \\ \times & \left( P^{(i-1)}(s_{m,1}) \prod_{t=2}^{T_n} P^{(i-1)}(s_{m,t}|S_m^{t-1}) \right) \times \left( \sum_{t=1}^{T_n} \ln P(q_{n,t}|s_{m,t}) \right. \\ + & \left. \sum_{t=1}^{T_n} \sum_k \ln P(u_{n,t,k}|s_{m,t}) + \ln P(s_{m,1}) + \sum_{t=2}^{T_n} \ln P(s_{m,t}|S_m^{t-1}) \right). \end{aligned}$$

At the *M-Step*, we maximize $Q(\Theta, \Theta^{(i-1)})$ iteratively using the following formula until the iteration converges.

$$P(s_i) = \frac{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \delta(s_{m,1} = s_i)}{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)})} \quad (5)$$

$$P(q|s_i) = \frac{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \sum_t \delta(s_{m,t} = s_i \wedge q = q_{n,t})}{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \sum_t \delta(s_{m,t} = s_i)} \quad (6)$$

$$P(u|s_i) = \frac{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \sum_t \delta(s_{m,t} = s_i \wedge u \in U_{n,t})}{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \sum_t \delta(s_{m,t} = s_i)} \quad (7)$$

$$P(s_i|S_j) = \frac{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \delta(\exists t\ S_m^{t-1} = S_j \wedge s_{m,t} = s_i)}{\sum_{n,m} P(S_m|O_n, \Theta^{(i-1)}) \delta(\exists t\ S_m^{t-1} = S_j)} \quad (8)$$

In the above equations, $\delta(p)$ is a boolean function indicating whether predicate $p$ is true (= 1) or false (= 0).

## 4. TRAINING A VERY LARGE VLHMM

Although the EM algorithm has been widely used to train HMMs, there are still several challenges to apply 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 users' search intents, whose number is unknown. To address this challenge, we apply the mining techniques developed in [6] as a prior process to the parameter learning process. To be specific, we construct a click-through bipartite and derive a collection of query clusters as in [6]. 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. A duple of query and URL cluster $(Q, U)$ is considered to correspond to a hidden state. The total number of hidden states is determined by the total number of clusters. For example, Table 2 shows a state which is mined from a real data set.

| Queries | $P^0(q\|s)$ | $P(q\|s)$ |
|---|---|---|
| city of bothell | 0.52 | 0.43 |
| bothell wa | 0.20 | 0.27 |
| bothell washington | 0.14 | 0.19 |
| city of bothell washington | 0.07 | 0.05 |
| city of bothell wa | 0.06 | 0.05 |
| city bothell washington | 0.01 | 0.01 |
| **URLs** | $P^0(u\|s)$ | $P(u\|s)$ |
| ci.bothell.wa.us | 0.21 | 0.15 |
| bothellwashington.com | 0.17 | 0.14 |
| ci.bothell.wa.us/dept/pd/pdindex.html | 0.15 | 0.15 |
| beckwithconsult.com/bothellcityhall.html | 0.14 | 0.17 |
| explorebothell.com | 0.14 | 0.19 |
| allgetaways.com/city-guide.asp | 0.09 | 0.11 |
| nwmaps.net/bothell | 0.05 | 0.08 |
| ihsadvantage.com/h/hotels/bothell/wa/us | 0.02 | 0.01 |
| wecandoitall.com | 0.01 | 0.00 |
| dianasflowers.com | 0.01 | 0.00 |
| mrsc.org/Contracts/B67-SEWER.pdf | 0.01 | 0.00 |

**Table 2: An example of a hidden state and the emission probabilities before and after the training.**

Second, search logs may contain hundreds of millions of training sessions. It is impractical to learn a vlHMM from such a huge training data set using a single machine. To address this challenge, we deploy the learning task on a distributed system under the *map-reduce* programming model [9].

We will describe the *map* stage and the *reduce* stage in Section 4.1.

Last, although the distributed computation partitions the training data into multiple machines, each machine still needs to hold the values of all parameters to conduct local estimation. Since the log data usually contains millions of unique queries and URLs, the space of parameters is extremely large. For example, the real data set used in our experiments leads to more than $10^{30}$ parameters. Clearly, the EM algorithm in its original form cannot finish in practical time for even one round of iteration. To address this challenge, we develop a special initialization strategy based on the clusters mined from the click-through bipartite. We will show in Section 4.2 that, in practice, our initialization strategy reduces the number of parameters to be re-estimated in each round of iteration to a much smaller number. Moreover, theoretically the number has an upper bound.

## 4.1 Distributed Learning of Parameters

*Map-Reduce* is a programming model for distributed processing of large data sets [9]. 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 all intermediate values associated with the same intermediate key and outputs the final computation results.

In our learning process, 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 $p_{n,m} = P(S_m|O_n, \Theta^{(i-1)})$ by Equation 4 for each possible state sequence $S_m$ and emits the key/value pairs as shown in Table 3.

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

**Table 3: The key/value pairs emitted at the map stage.**

In the reduce stage, each process node collects all values for an intermediate key. For example, suppose the intermediate key $s_i$ is assigned to process node $n_k$. Then $n_k$ receives a list of values $\{(Value_{i,1}, Value_{i,2})\}$ $(1 \le i \le N)$ and derives $P(s_i)$ by $\frac{\sum_i Value_{i,1}}{\sum_i Value_{i,2}}$. 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 $\{P(s_i|S_j)\}$. Suppose the number of states is $N_s$, the number of unique queries is $N_q$, the number of unique URLs is $N_u$, and the maximal length of a training session is $T_{max}$. Then, $|\{P(s_i)\}| = N_s$, $|\{P(q|s_i)\}| = N_s \cdot N_q$, $|\{P(u|s_i)\}| = N_s \cdot N_u$, $|\{P(s_i|S_j)\}| = \sum_{t=2}^{T_{max}} N_s^t$, and the total number of parameters is $\mathcal{N} = N_s \cdot (1 + N_q + N_u + \sum_{t=2}^{T_{max}} N_s^{t-1})$. Since a search log may contain millions of unique queries and URLs, and there may be millions of states derived from the click-through bipartite, it is impractical to estimate all parameters straightforwardly. *Can we reduce the number of parameters that need to be re-estimated in each round of iteration?*

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 web site of Beijing Olympic 2008. Similarly, a user who raises query *"Beijing Olympic 2008"* is unlikely to click on the URL `http://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 $s_i$. As illustrated in Table 2, 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 $s_i$. As a possible method, we may 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.

Alternatively, we can conduct random walks on the click-through bipartite. $P(q|s_i)$ ($P(u|s_i)$) can be initialized as the average probability of the random walks that start from $q$ ($u$) and stop at the queries (URLs) belonging to cluster $C_i$. However, as indicated in [6], the click-through bipartite is highly connected – there may exist paths between two completely unrelated queries or URLs. Consequently, random walks may assign undesirable large emission probabilities to queries and URLs generated by an irrelevant search intent.

We design an initialization strategy to balance the above two approaches. We apply random walks up to a restricted number of steps. Such an initialization allows a query (as well as a URL) to represent multiple search intents, and at the same time avoids the problem of assigning undesirable large emission probabilities.

We 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, 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.

The initial emission probabilities have the following nice property.

LEMMA 1. *The query emission probability at the i-th round of iteration $P^i(q|s_i) = 0$ if the initial value $P^0(q|s_i) = 0$.*

PROOF. The denominator in Equation 6 is a constant. Thus, we only need to consider the numerator. For any pair of $O_n$ and $S_m$, if $O_n$ does not contain query $q$, the enumerator 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 $t_1$, i.e., $q_{n,t_1} = q$. If $s_{m,t_1} \neq s_i$, then the enumerator is zero since $\sum_t \delta(s_{m,t} = s_i \wedge q_{n,t} = q) = \delta(s_{m,t_1} = s_i \wedge q_{n,t_1} = q) = 0$.

Last, if $s_{m,t_1} = s_i$ and $q_{n,t_1} = q$, $P(O_n|S_m, \Theta^{(i-1)}) = P^{(i-1)}(q|s_i)\cdot\Big(\prod_{t\neq t_1} P^{(i-1)}(q_{n,t}|s_{m,t})\Big)\cdot\Big(\prod_t P^{(i-1)}(U_{n,t}|s_{m,t})\Big)$. Therefore, if $P^{(i-1)}(q|s_i)) = 0$, $P(O_n|S_m, \Theta^{(i-1)}) = 0$, and thus $P(S_m|O_n, \Theta^{(i-1)}) = 0$ (Equation 4).

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

Similarly, we can show the following.

LEMMA 2. *The URL emission probability at the i-th round of iteration* $P^i(u|s_i) = 0$ *if the initial value* $P^0(u|s_i) = 0$. ■

Based on Lemmas 1 and 2, for each training session $O_n$, we can construct a set of *candidate state sequences* $\Gamma_n$ which are likely to generate $O_n$. To be specific, let $q_{n,t}$ and $\{u_{n,t,k}\}$ be the $t$-th query and the $t$-th set of clicked URLs in $O_n$, respectively, and $Cand_{n,t}$ be the set of states $s$ such that $(P^0(q_{n,t}|s) \neq 0) \wedge (\forall_k P^0(u_{n,t,k}|s) \neq 0)$. From Equations 2 and 4 and Lemmas 1 and 2, we have $P(S_m|O_n, \Theta^{(i-1)}) = 0$ for any $S_m$ if $s_{m,t} \notin Cand_{n,t}$. Therefore, the set of candidate state sequences $\Gamma_n$ for $O_n$ can be constructed by joining $Cand_{n,1}, \ldots, Cand_{n,T_n}$. It is easy to see that for any $S_m \notin \Gamma_n$, $P(S_m|O_n, \Theta^{(i-1)}) = 0$. In other words, for each training session $O_n$, only the state sequences in $\Gamma_n$ are possible to contribute to the update of parameters in Equations 5-8.

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 $\Gamma^+ = \Gamma_1 + \ldots + \Gamma_N$, 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 $\Gamma^+$ that start with state $s_i$, subsequence $S_j$, and the concatenations of $S_j$ and $s_i$, respectively.

The above initialization limits the number of *active parameters* (i.e., the parameters updated in one iteration of the training process) to an upper bound $\mathcal{C}$ as indicated in the following theorem.

THEOREM 1. *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,* $\Gamma$ *is the set of unique state sequences in* $\Gamma^+$*, and* $\overline{T}$ *is the average length of the state sequences in* $\Gamma$.

PROOF. Let $\widetilde{\Psi^i}$, $\widetilde{\Delta^i}$, $\widetilde{\Lambda^i_q}$ and $\widetilde{\Lambda^i_u}$ be the sets of active initial state probabilities, transition probabilities, query and URL emission probabilities in the $i$-th iteration, respectively. Using Lemmas 1 and 2, we immediately have $|\widetilde{\Lambda^i_q}| \leq N_s \cdot \overline{N_{sq}}$ and $|\widetilde{\Lambda^i_u}| \leq N_s \cdot \overline{N_{su}}$. Moreover, from the construction of $\Gamma$, 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 1 is often much smaller than the size of the whole parameter space $\mathcal{N} = N_s \cdot (1 + N_q + N_u + \sum_{t=2}^{T_{max}} N_s^{t-1})$. For example, in our experimental data, $\overline{N_{sq}} = 4.5 \ll N_q = 1.8 \times 10^6$, $\overline{N_{su}} = 47.8 \ll N_u = 8.3 \times 10^6$, and $|\Gamma| \cdot (\overline{T} - 1) = 1.4 \times 10^6 \ll N_s \cdot \sum_{t=2}^{T_{max}} N_s^{t-1} = 4.29 \times 10^{30}$.

Our initialization strategy also enables an efficient training process. According to Equations 5-8, 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, e.g., 4.7 in our experiments. Although $N$ is a very large number (840 million 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.

## 5. MODEL APPLICATION

In this section, we discuss how to apply the learned vlHMM to various search applications including document re-ranking, 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 $\Gamma_O$ 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 $\Theta$ 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 to 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 $\Theta$ 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 re-ranking.** 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 $\sum_{s_t \in S} P(u|s_t) \cdot P(s_t|O)$. Then, we re-rank the URLs in the posterior probability descending order.

**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.

**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.

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 $i$-th $(1 \leq i < t)$ 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; otherwise, the vlHMM just skips this round. If the current query $q_t$ is unknown to the vlHMM, the vlHMM takes no action.

Second, the online application of our vlHMM may have a 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 $\Gamma_O$, $S_t$, $S_{t+1}$, $Q_{t+1}$, and $U_{t+1}$. In our experiments, the average numbers of $\Gamma_O$, $S_t$, 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 millisecond.

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 the candidates with low probabilities or truncating the session. Since we only re-rank the top URLs returned by a search engine and suggest the top queries and URLs generated by the 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 context-aware document re-ranking, query suggestion, and URL recommendation.

### 6.1 Data Set and Preparation

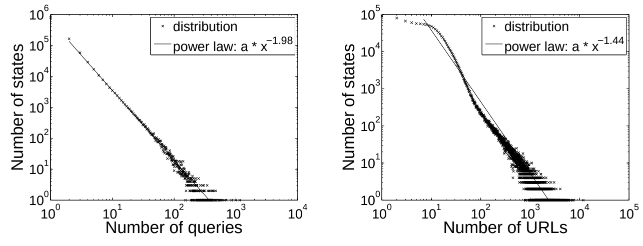
We use a large search log from a major commercial search engine to train a vlHMM. We only focus on the Web searches in English from the US market. The log data set contains 1.8 billion queries, 2.6 billion clicks, and 840 million sessions. The data set involves 151 million unique queries and 114 million unique URLs.

From the raw search log, we first extract user sessions as described in Section 3. Since we want to train a vlHMM to model the common search behavior of the crowds, infrequent sessions should be removed. However, we find that user sessions, especially long sessions, are extremely sparse. If we aggregate directly on the search sequences which include both queries and clicks, most of the sessions will be pruned. Therefore, instead of counting the frequencies of search sequences, we count the frequencies of query sequences. To be specific, 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, 48% of the sessions in the log data set are pruned.

We examine the distribution of the lengths of the surviving sessions and find it follows the power law. Moreover,

| | Raw search log | Training data |
|---|---|---|
| Num. of unique queries | 151,869,102 | 1,835,270 |
| Num. of unique URLs | 114,882,486 | 8,309,988 |
| Num. of query occurrences | 1,812,563,301 | 926,442,156 |
| Num. of clicks | 2,554,683,191 | 1,321,589,933 |
| Num. of sessions | 840,356,624 | 437,245,177 |

**Table 4: The data statistics before and after the pre-processing.**



(a) (b)

**Figure 2: The number of states with respect to the number of nonzero initial (a) query and (b) URL emission probabilities.**

more than 50% of the sessions contain at least two rounds of interaction. These observations are consistent with those in previous studies (e.g., [13]). We manually inspect some sessions with lengths longer than 5, and find many of them contain meaningless query sequences. We suspect that those sessions were generated by bots. To reduce those noise sessions, we further remove the sessions longer than 5. There are 22,919 such sessions, about 0.005% of the whole data set.

Table 4 shows the statistics of the data set before and after the pre-processing. Although 98.8% unique queries and 92.8% unique URLs are removed by the pre-processing, the resulting data set still keeps 51.1% of the original query occurrences, 51.7% of the original URL clicks, and 52% of the original user sessions. As shown in many previous works (e.g., [2]), this is because the query occurrences and URL clicks in search logs follow the power law distribution.

### 6.2 Efficiency of Training the vlHMM

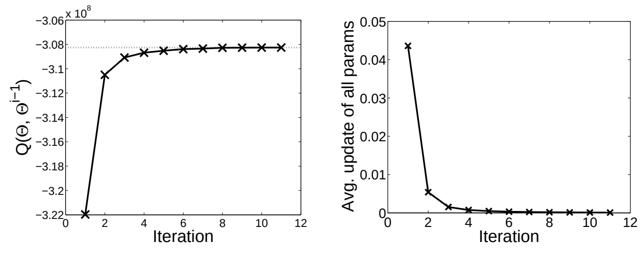
To determine the number of states and assign initial parameter values, we apply the clustering algorithm in [6] and derive 1,346,146 clusters of queries. We thus have 1,346,146 states in the vlHMM, and then initialize the parameter values as described in Section 4.2. As an example, Table 2 shows the initial emission probability distribution of state $s_{753}$. For all queries $q$ and URLs $u$ not in the table, $P^0(q|s_{753})$ and $P^0(u|s_{753})$ are set to 0.

Figures 2(a) and 2(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 experiments, $\overline{N_{sq}} = 4.5$ and $\overline{N_{su}} = 47.8$. It means that on average, different users formulate 4.5 queries and click 47.8 URLs for a common search intent.

We further compute the set of candidate state sequences

| | Actual number | Upper bound | Whole space |
|---|---|---|---|
| $\#P(s_i)$ | 1,146,346 | 1,346,146 | 1,346,146 |
| $\#P(q\|s_i)$ | 3,513,441 | 6,057,657 | $2.47 \times 10^{12}$ |
| $\#P(u\|s_i)$ | 56,624,285 | 64,345,778 | $1.12 \times 10^{13}$ |
| $\#P(s_i\|S_j)$ | 1,275,708 | 1,399,498 | $4.29 \times 10^{30}$ |

**Table 5: Comparison of the actual number, the upper bound, and the whole space of the parameters.**



(a) (b)

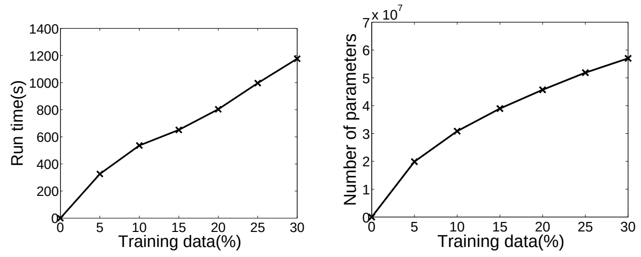
**Figure 3: The (a) value of $Q(\Theta, \Theta^{(i-1)})$ and (b) average difference of all parameters in one iteration.**

$\Gamma$ given the initialization of $P^0(q|s)$ and $P^0(u|s)$. For the 437, 245, 177 training sessions, there are only 2, 621, 854 unique candidate state sequences, since users with similar search intents often have similar search sequences, and thus can be modeled by the same state sequence. This verifies the power of a vlHMM as a compact statistical model to summarize the huge search logs.

Table 5 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 1. Clearly, the actual number of estimated parameters is dramatically 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 $10^{24}$. There are two reasons for this factor. First, as mentioned in Section 6.1, the session length follows the power law distribution and a large part of sessions are short – of length 1 or 2. Second, queries and clicked URLs in the same sessions are semantically related. Thus, the actual number of state sequences appeared in logs is dramatically smaller than that of all possible combinations.

Figures 3(a) and 3(b) show the value of the object function $Q(\Theta, \Theta^{(i-1)})$ and the average difference of all parameters in one iteration, respectively. Clearly, the training process converges fast under our initialization method.

Last, we test the runtime and scalability of the training algorithm. Since we run the training algorithm on a distributed system shared by multiple users, it is hard to measure the exact runtime of the training process. Thus, we simulate a process node in the distributed system by a standalone server using an Intel Core 2 2.0 GHZ ×2 CPU, 4 GB main memory, and test the runtime and the scalability on sampled subsets of the whole training data of different sizes. For each size, we randomly sample a subset 10 times and report the average result on the random samples in Figures 4. By using our initialization method, the training algorithm is efficient and scales up well.



(a) (b)

**Figure 4: The (a) runtime and (b) number of parameters of the training process on different sizes of sampled data.**

## 6.3 Effectiveness of vlHMMs

To evaluate the effectiveness of the trained vlHMM, we extract 100, 000 sessions as the test data set from a separate search log other than the one used as the training data set. We use each session $\langle(q_1, U_1), \ldots, (q_T, U_T)\rangle$ in the test set to test the vlHMM. First, the vlHMM performs re-ranking, suggestion, and recommendation, for query $q_1$. Second, $U_1$ is sent to the vlHMM as the URLs clicked by a user for $q_1$, and query $q_2$ is sent to the vlHMM as the next query. The vlHMM then performs re-ranking, suggestion, and recommendation for $q_2$ based on the context introduced by $q_1$ and $U_1$. Generally, a session of length $T$ is treated as a sequence of $T$ test cases where each case corresponds to a query.

We call a test case is *covered* by the vlHMM if the query can be recognized and the model can perform the corresponding re-ranking, suggestion, and recommendation. The coverage of the vlHMM is 58.3%. The uncovered queries are tail queries which either do not appear in the raw training log or have been removed in the pre-processing due to a very low frequency.

To better examine the effectiveness of context information, we further divide the test cases into two subsets: Test0 contains the cases of the first queries in the sessions, that is, no context is available; and Test1 contains the others. In Test1, the contexts of 25.5% of the covered cases can be recognized by the vlHMM. This indicates that, in many cases when context is available, the vlHMM is able to exploit such information.

In our implementation of the vlHMM, we use a set of queries $Q$ and a set of URLs $U$ to represent each state. Since the queries and URLs are very sparse in the log data, the coverage of the vlHMM and the percentage of recognized contexts are not high. In fact, we can simply expand the vlHMM learned from log data by building a language model [16] for each state based on the document text of the URLs in $U$. In this way, the ability of the vlHMM to generalize to unknown queries and URLs will be greatly enhanced, and the coverage as well as the percentage of recognized contexts will be improved substantially.

### 6.3.1 *Document Re-ranking*

We evaluate the quality of document re-ranking using the trained vlHMM on the top-10 results from a commercial search engine. We randomly select 500 re-ranked pairs of documents from the covered test cases in Test0 and 500 pairs from the cases in Test1 where the context can be recognized

| | Context | Test query | | Re-ranked document pairs |
|---|---|---|---|---|
| 1 | online games ↓ http://games.yahoo.com http://www.miniclip.com | Disney channel | ↑ | Games **Disney Channel** http://tv.disney.go.com/disneychannel/games/index.html |
| | | | ↓ | **Disney Channel** http://www.disney.go.com/disneychannel |
| 2 | ask.com ↓ http://www.ask.com | Ask Jeeves | ↑ | **Ask Jeeves**: Wikipeadia Free encyclopeida http://en.wikipedia.org/wiki/Ask_Jeeves#International |
| | | | ↓ | **Ask**.com Search Engine, Better Web Search http://www.ask.com |

**Table 6: Examples of re-ranked documents pairs by the vlHMM.**

by the model. For each test case, we present the test query to 3 judges and ask them to compare the relevance of the document pair. For cases from Test1, the judges are also presented with the past queries and clicked URLs of the test query. For a pair of documents $A$ and $B$, there are three labels: $A$ is more relevant than $B$, $A$ is less relevant than $B$, and unsure.

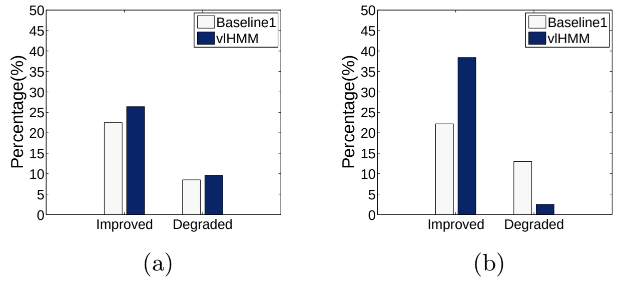


**Figure 5: The effectiveness of re-ranking by the vlHMM and Bbaseline1 on (a) Test0 and (b) Test1.**

The existing re-ranking methods either do not consider the click-through information (e.g., [24]) or combine click-through information with other features such as document text (e.g., [25]). It is not meaningful to make a direct comparison between our method and those existing methods. We use a baseline (denoted by *Baseline1*) which purely relies on click-through data, i.e., to re-rank documents A and B if the order of their click numbers with respect to the test query is reversed with their original order. One difference between Baseline1 and the vlHMM is that the former does not consider the context of the test query.

Figures 5(a) and 5(b) compare the quality of re-ranking performed by vlHMM and Baseline1. In the figures, the "Improved" category counts the cases where re-ranking improves the ordering of the documents, while "Degraded" counts the opposite cases. The unsure cases are not counted. The vlHMM has a comparable performance with Baseline1 for cases in Test0 where context information is not available (Figure 5(a)). However, in Test1, while the baseline achieves a similar performance as in Test0, the vlHMM shows a substantial gain (Figure 5(b)). This clearly indicates that the vlHMM is effective to model the context information and thus understands users' search intents better.

Table 6 shows two examples of re-ranked document pairs by the vlHMM. In the first example, when a user raises query *"Disney channel"*, the search engine ranks the homepage of Disney Channel higher than its game site. The vlHMM is able to consider the context that the user actually searches online games before this query, and accordingly boosts the game site on top of the homepage.

In the second example, the user inputs query *"Ask Jeeves"*. Unsurprisingly, `www.ask.com` is ranked higher than the wikipedia page about the company. However, the vlHMM notices that the user input query *"ask.com"* and clicked `www.ask.com` before query *"Ask Jeeves"*. This context provides the hint that the user may not be interested in the search service provided by `www.ask.com` but instead be interested in the background information of the company. Consequently, the vlHMM boosts the wikipedia page.

### *6.3.2 URL Recommendation and Query Suggestion*

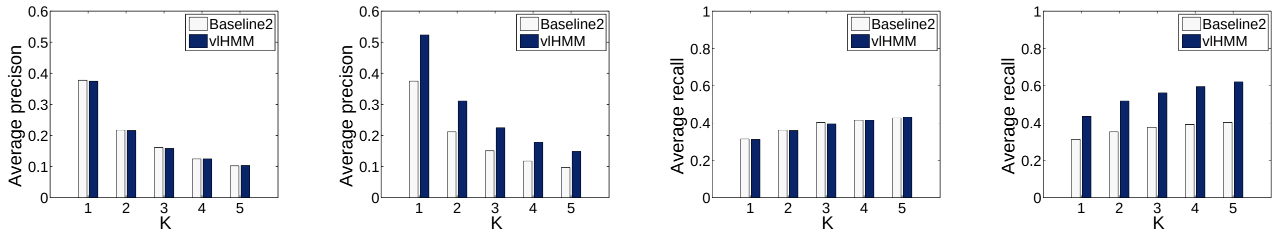
We evaluate the performance of the vlHMM on URL recommendation using the "leave-one-out" method. Specifically, for each extracted session $O = \langle (q_1, U_1), \ldots, (q_T, U_T) \rangle$, we use $q_{T-1}$ as the test query and consider $U_T$, the set of URLs really clicked by the user, as the ground truth. The performance is then measured by precision and recall. Suppose the vlHMM model recommends a set of URLs $R$, the precision is $\frac{|R \cap U_T|}{|R|}$ and the recall is $\frac{|R \cap U_T|}{|U_T|}$.

Since there has been little work on URL recommendation using search logs, we use a baseline (denoted by *Baseline2*) which borrows the idea from [23] where browsing logs rather than search logs are used. Given a test query $q$, Baseline2 counts in the training data the frequency of a URL occurring in the interactions following the round of $q$, and recommends the top $K$ URLs with the highest co-occurring frequencies. Baseline2 does not consider the context of $q$.

Figures 6(a) and 6(b) compare the precision of the vlHMM and Baseline2 with respect to the number of recommendations $K$ in Test0 and Test1, respectively, while Figures 6(c) and 6(d) compare the recall. In both methods and in both test sets, the precision drops and the recall increases when $K$ increases. Although the vlHMM and Baseline2 have comparable precision and recall in Test0, the vlHMM outperforms the baseline substantially in Test1, where the context information is available.

Table 7 shows an example of URL recommendation when the user inputs query *"Walmart"*. Without considering the context, Baseline2 recommends the homepage of Sears as the first choice. Although this recommendation is meaningful, if we consider the user searched *"circuit city"* before, the URL `www.bestbuy.com` recommended by the vlHMM looks a better choice.

Last, we follow the evaluation method described in [6] to compare the quality of query suggestions generated by



(a) Precision on Test 0 (b) Precision on Test1 (c) Recall on Test0 (d) Recall on Test1

**Figure 6: The precision and recall of the URLs recommended by the vlHMM and Baseline2.**

| | |
|---|---|
| Context: | circuit city → http://www.circuitcity.com |
| Test query: | Walmart |

| URL recommendation | |
|---|---|
| vlHMM | Baseline2 |
| http://www.bestbuy.com | http://www.sears.com |

**Table 7: An example of URL recommendation.**

vlHMM and CACB [6]. Since both methods consider the context information, we find the qualities of query suggestions by the two methods are comparable. We also compare the percentage of recognized contexts by the two methods. Among the covered test cases in Test1, the CACB can only recognize the contexts of 16.5% cases, while the vlHMM improves the coverage by 55%. The reason is that the vlHMM considers both queries and clicked URLs in the context, and is able to recognize the context when either the queries or clicked URL are seen in the training data. Moreover, as mentioned before, the coverage and the percentage of recognized contexts of vlHMM can be readily improved via the URLs in the states, while the expansion for CACB is not straightforward since CACB only considers queries.

In summary, the extensive empirical study using a large real data set from a major commercial search engine clearly verifies that our vlHMM method is effective in context-aware search, and is efficient in model learning.

## 7. CONCLUSIONS

In this paper, we propose a general approach to context-aware search by learning a vlHMM from search sessions extracted from log data. We tackle 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 also devise a method for distributed vlHMM learning under the map-reduce model. The experimental results on a large real data set clearly show that our context-aware approach is both effective and efficient.

## 8. REFERENCES

[1] Baeza-Yates, R.A., et al. Query recommendation using query logs in search engines. In *EDBT 2004 Workshop on Clustering Information over the Web*, pages 588–596, 2004.

[2] Baeza-Yates,R.A., et al. Extracting semantic relations from query logs. In *KDD'07*, pages 76–85, 2007.

[3] Bai, J., et al. Using query contexts in information retrieval. In *SIGIR'07*, pages 15–22, 2007.

[4] Baum, L.E., et al. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *Ann. Math. Statist.*, 41(1):164–171, 1970.

[5] Beeferman, D., et al. Agglomerative clustering of a search engine query log. In *KDD'00*, pages 407–416, 2000.

[6] Cao, H., et al. Context-aware query suggestion by mining click-through and session data. In *KDD'08*, pages 875–883, 2008.

[7] Chirita, P.A., et al. Personalized query expansion for the web. In *SIGIR'07*, pages 7–14, 2007.

[8] Chu, C.T., et al. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.

[9] Dean, J., et al. MapReduce: simplified data processing on large clusters. In *OSDI'04*, pages 137–150, 2004.

[10] Dempster, A.P., et al. Maximal Likelihood from Incomplete Data Via the EM Algorithm. *Journal of the Royal Statistical Society*, Ser B(39):1–38, 1977.

[11] Durbin, R., et al. *Biological sequence analysis: probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[12] Fonseca, B.M., et al. Concept-based interactive query expansion. In *CIKM'05*, pages 696–703, 2005.

[13] Huang, C., et al. Relevant term suggestion in interactive web search based on contextual information in query session logs. *Journal of the American Society for Information Science and Technology*, 54(7):638–649, 2003.

[14] Joachims, T., et al. Optimizing search engines using clickthrough data. In *KDD'02*, pages 133–142, 2002.

[15] Jones, R., et al. Generating query substitutions. In *WWW'06*, pages 387–396, 2006.

[16] Liu, X., et al. Cluster-based retrieval using language models. In *SIGIR'04*, pages 186–193, 2004.

[17] Rabiner, L.R. A tutorial on hidden Markov models and selected applications inspeech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[18] Rocchio, J. *Relevance feedback information retrieval.* Prentice-Hall Inc., 1971.

[19] Sugiyama, K., et al. Adaptive web search based on user profile constructed without any effort from users. In *WWW'04*, pages 675–684, 2004.

[20] Tao, T. and Zhai, C. A two-stage mixture model for pseudo feedback. In *SIGIR'04*, pages 486–487, 2004.

[21] Wang, Y., et al. Mining complex time-series data by learning Markovian models. In *ICDM'06*, pages 1136–1140, 2006.

[22] Wen, J., et al. Clustering user queries of a search engine. In *WWW'01*, pages 162–168, 2001.

[23] White, R.W., et al. Studying the use of popular destinations to enhance web search interaction. In *SIGIR'07*, pages 159–166, 2007.

[24] Xu, J., et al. AdaRank: A boosting algorithm for information retrieval. In *SIGIR'07*, pages 391–398, 2007.

[25] Zhao, M., et al. Adapting document ranking to users preferences using click-through Data. In *AIRS'06*, pages 26–42, 2006.