Constructing a Multidimensional Topic-Concept Cube for OLAP on Large-Scale Search Logs

Zhen Liao[§] Daxin Jiang[§] Jian Pei[†] Dongyeop Kang[‡]
Xiaohui Sun[§] Ho-Jin Choi[‡] Hang Li[§]
[§]Microsoft Research Asia [†]Simon Fraser University
[‡]Korea Advanced Institute of Science and Technology
Email: [§]{v-zhliao,djiang, xiaos, hangli}@microsoft.com
[‡]jpei@cs.sfu.ca [†]{dykang,hojinc}@kaist.ac.kr

Abstract

In addition to search queries and the corresponding click-through information, search engine logs record multidimensional information about user search activities, such as search time, location, vertical, and search device. Multidimensional mining of search logs can provide novel insights and useful knowledge for both search engine users and developers. In this paper, we describe our topic-concept cube project, which addresses the business need of supporting multidimensional mining of search logs effectively and efficiently. We answer two challenges. First, search queries and click-through data are well recognized sparse, and thus have to be aggregated properly for effective analysis. Second, there is often a gap between the topic hierarchies in multidimensional aggregate analysis and queries in search logs. To address those challenges, we develop a novel topic-concept model that learns a hierarchy of concepts and topics automatically from search logs. Enabled by the topic-concept model, we construct a topic-concept cube that supports online multidimensional mining of search log data. A distinct feature of our approach is that, in addition to the standard dimensions such as time and location, our topic-concept cube has a dimension of topics and concepts, which substantially facilitates the analysis of log data. To handle a huge amount of log data, we develop distributed algorithms for learning model parameters efficiently. We also devise approaches to computing a topic-concept cube. We report an empirical study verifying the effectiveness and efficiency of our approach on a real data set of 1.96 billion queries and 2.73 billion clicks.

1 Introduction

Search logs in search engines record rich information about user search activities. In addition to search queries and the corresponding click-through information, the related information is also recorded on multiple attributes, such as search time, location, vertical, and search device. Multidimensional mining of such rich search logs can provide novel insights and useful knowledge for both search engine users and developers. Let us consider two multidimensional analysis tasks.

A **multidimensional lookup** (**lookup** for short) specifies a subset of user queries and clicks using multidimensional constraints such as time, location and general topics, and requests for the aggregation of the user search activities. For example, by looking up "the top-5 electronics that were most popularly searched by the users in the US in December 2009", a business analyst can know the common interests of search engine users on topic "Electronics". Moreover, search engine developers can use the results from the lookup to improve query suggestion, document ranking, and sponsored search. Multidimensional lookups can be extended in many ways to achieve advanced business intelligence analysis. For example, using multiple lookups with different multidimensional constraints, one may compare the major interests about electronics from users in different regions such as the US, Asia, and Europe.

A **multidimensional reverse lookup** (**reverse lookup** for short) is concerned about the multidimensional group-bys where one specific object is intensively queried. For example, using reverse lookup "What are the group-bys in time and region where Apple iPad was popularly searched for?", an iPad accessory manufacturer can find the regions where the accessories may have a good market. Using the results from the reverse lookup, a search engine can improve its service by, for example, locality-sensitive search. Again, reverse lookups can be used to compose advanced business intelligence analysis. For example, by organizing the results from the reverse lookup about iPad, one may keep track of how iPad becomes popular in time and in region, and also compare the trend of iPad with those of iPod and iPhone. This is interesting to both business parties and users.

As search engines have accumulated rich log data, it becomes more and more important to develop a service that supports multidimensional mining of search logs effectively and efficiently. To answer multidimensional analytical queries online, a data warehousing approach is a natural choice, which pre-computes all multidimensional aggregates offline. However, traditional data warehousing approaches only explore a series of statistical aggregates

1	ipad
2	apple ipad
3	ipad 32g
4	kindle
5	amazon kindle

(a)

1	ipad
2	kindle
3	iphone
4	xbox 360
5	wii

(b)

Table 1: Answers to "the top-5 electronics that were most popularly searched by the users in the US in December 2009" by (a) individual queries and (b) concepts.

such as `MIN`, `MAX`, and `AVG`; they cannot summarize the semantic information of user queries and clicks. In particular, multidimensional analysis on search log data presents two special challenges.

Challenge 1: sparseness of queries in log data. Queries in search engine logs are usually very sparse, since users may formulate different queries for the same information need [8]. For example, to search for Apple iPad, users may issue queries such as *"ipad"*, *"apple ipad"*, *"ipad 32g"*, *"i pad apple"*, and so on. Aggregating only on individual queries cannot summarize user information needs recorded in logs comprehensively. For example, when a business analyst asks for "the top-5 electronics that were most popularly searched by the users in the US in December 2009", a naïve method may simply count the frequency of the queries in the topic of "Electronics" and return the top-5 most frequently asked queries. Due to the sparseness of queries in the logs, the analyst may get an answer with many redundant queries, such as the one shown in Table 1(a). Instead, if we can summarize various query formulations of the same information need and provide non-duplicate answers (e.g., Table 1(b)), the user experience can be improved greatly. Similarly, in reverse lookup, when an iPad accessary manufacturer asks the question "What are the group-bys in time and region where Apple iPad was popularly searched for?", the system should consider not only aggregates of the query "Apple iPad" but also its various formulations. To address the sparseness of log data, we have to aggregate queries and click-through data in logs.

Challenge 2: mismatching between topic hierarchies used in analytics and those learned from log data. More often than not, people use different topic hierarchies in searching detailed information and summarizing analytic information. For example, when users search electronics on the web, often the queries are about specific products, brand names, or features. A query topic hierarchy automatically learned from log data

in a data-driven approach depends on the distribution and occurrences of user queries. "Apple products" may be a popular topic. When an analyst explores a huge amount of log data, she may bear in her mind a product taxonomy (e.g., a well adopted ontology), such as TV & video, audio, mobile phones, cameras & camcorders, computers, and so on being the first level categories. The analytic topic hierarchy may be very different from the query topic hierarchy learned from log data. For example, the "Apple products" in the query topic hierarchy corresponds to multiple topics in the analytic topic hierarchy. This mismatching in topic hierarchies is partly due to the different information needs in web search and web log data analysis. Web searches often opt for detailed information, while web log analysis usually tries to summarize and characterize popular user behavior patterns. To bridge the gap, we need to map the aggregates from logs to an analytic topic hierarchy.

In this paper, we describe our topic-concept cube project that builds a multidimensional service on search log data. We make the following contributions.

First, we tackle the sparseness of queries in logs and the gap between concept taxonomy in analytics and queries in logs by a novel concept-topic model. Figure 1 illustrates our idea. We first mine click-through information in search logs and group similar queries into concepts. Intuitively, users with the same information need tend to click on the same URLs. Therefore, various query formulations, for example, of Apple ipad, such as *"ipad"*, *"apple ipad"*, *"ipad 32g"*, and *"i pad apple"*, can be grouped into the same concept, since all of them lead to clicks on the web page `www.apple.com/ipad`. More interestingly, some misspelled queries, such as *"apple ipda"* and *"apple ipade"*, can also be clustered into this concept, since they also lead to clicks on the ipad page. Once we summarize queries and clicks into concepts, we will answer lookups and reverse lookups by concepts instead of individual queries. For each concept, we use the most frequently asked query as the representative of the concept. In this way, we can effectively avoid redundant queries in lookup answers. At the same time, we can effectively cover all relevant queries in reverse lookup answers.

Our concept-topic model further maps concepts to topics in a given taxonomy, which is essentially a query classification problem. For example, suppose a concept consists of queries *"apple ipad"*, *"ipad 32g"*, etc., we classify them into the topic "Electronics". Compared with classifying individual queries to topics, mapping concepts has several advantages. For example, for a misspelled query *"apple ipda"*, the classification problem becomes much easier once we know that this query belongs to a concept that also contains

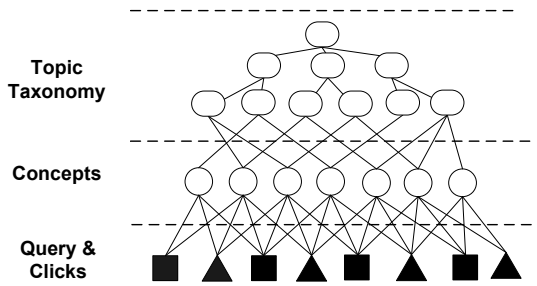


Figure 1: The hierarchy of topics, concepts, queries, and clicks.

other queries such as *"apple ipad"*. Moreover, through the content of the web pages that are commonly clicked as answers to the queries in the concept, we may further enrich the features to classify *"apple ipda"*.

Our concept-topic model provides the "semantic" aggregates for search log data. Those concepts and topics not only provide us a meaningful way to answer lookups and reverse lookups, but also serve as an important dimension for multidimensional analysis and exploration.

Second, to handle large volumes of search log data, which may contain billions of queries and clicks, we develop distributed algorithms to learn the topic-concept models efficiently. In particular, we develop a strategy to initialize the model parameters such that each machine only needs to hold a subset of parameters much smaller than the whole set. We further develop a heuristic data partition method to improve the efficiency of the training process.

Third, to serve online multidimensional mining of search log data, we build a topic-concept cube. In addition to the standard dimensions such as time and location, a topic-concept cube has a dimension of topics and concepts, such as "electronics" and "Apple iPad" used in the lookup and reverse lookup examples. We devise effective approaches for computing a topic-concept cube. In particular, queries are assigned to a hierarchy of concepts and topics in the materialization of the cube.

Finally, we conduct extensive experiments on a real log data set containing 1.96 billion queries and 2.73 billion clicks. We examine the effectiveness of the topic-concept model as well as the efficiency and scalability of our training algorithms. We also demonstrate several concrete examples of lookups and reverse lookups answered by our topic-concept cube system. The experimental results clearly show that our approach is effective and efficient.

The rest of the paper is organized as follows. We present the framework

Uid	Time Stamp	Location	Type	Value
U1	100605110843	Seattle, WA, US	Query	*"wsdm 2011"*
U2	100605110843	Vancouver, BC, CA	Query	*"you tube"*
U1	100605110846	Seattle, WA, US	Click	`wsdm2011.org`
...	...	...	...	...

Figure 2: A search log as a stream of query and click events with multidimensional information.

of our system in Section 2, review the related work in Section 3, describe the topic-concept model in Section 4, and develop the distributed algorithms for learning the topic-concept model from large-scale log data in Section 5. Section 6 discusses computing topic-concept cubes. We report the experimental results in Section 7, and conclude the paper in section 8.

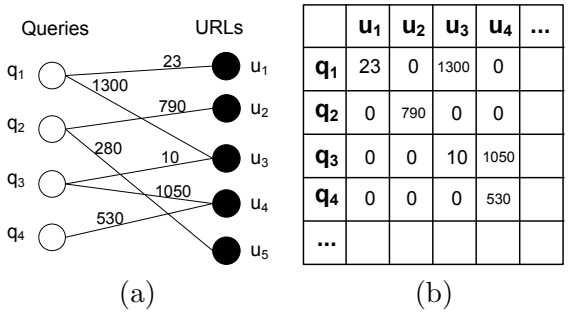
2 Our Framework

When a user raises a query to a search engine, a set of URLs are returned by the search engine as the search results. The user may browse the snippets of the top search results and selectively click on some of them. A *search log* can be regarded as a sequence of query-and-click events by users. For each event, a search engine may record the type and content of the event as well as some other information such as the time stamp, location, and device associated with the event. Figure 2 shows a small segment of a search log.

Some dimensions of search events may have a hierarchical structure. For example, the location dimension can be organized into levels of *country* $\rightarrow$ *state* $\rightarrow$ *city*, and the time dimension can be represented at levels of *year* $\rightarrow$ *month* $\rightarrow$ *day* $\rightarrow$ *hour*. Therefore, the multi-dimensional, hierarchical log data can be naturally organized into a raw log data cube [12], where each cell is a group-by using the dimensions. For example, a cell may contain all query-and-click events of time "February 2010" and location "Washington State".

We can aggregate the query-and-click events in a cell and derive a *click-through bipartite*, where each *query node* corresponds to a unique query in the cell and each *URL node* corresponds to a unique URL, as demonstrated in Figure 3(a). An *edge* e_{ij} is created between query node q_i and URL node u_j if u_j is a clicked URL of q_i. The *weight* w_{ij} of edge e_{ij} is the total number of times when u_j is a clicked result of q_i among all events in the cell.

A click-through bipartite can be represented as a *query-URL* matrix (*QU-matrix* for short), where each row corresponds to a query node q_i and



	u_1	u_2	u_3	u_4	...
q_1	23	0	1300	0	
q_2	0	790	0	0	
q_3	0	0	10	1050	
q_4	0	0	0	530	
...					

Figure 3: An example of (a) click-through bipartite and (b) QU-matrix.

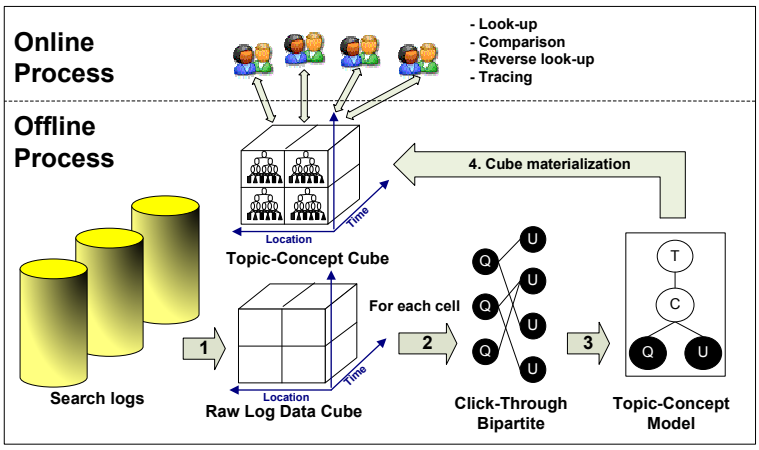


Figure 4: The framework of our system.

each column corresponds to a URL node u_j. The value of entry n_{ij} is simply the weight w_{ij} between q_i and u_j, as shown in Figure 3(b).

The QU-matrix at a cell is often sparse. Moreover, QU-matrix represents information at the level of individual queries and URLs. As discussed before, we need to summarize and aggregate the information in a QU-matrix to facilitate online multidimensional analysis. This will be achieved by the topic-concept model to be developed in Section 4.

Figure 4 shows the framework of our system. In the offline stage, we first form a raw log data cube by partitioning the search log data along various dimensions and at different levels. For each cell of the raw log data cube, we construct a click-through bipartite and derive the QU-matrix. Then, we materialize the cube by learning topic-concept models that summarize the

distributions of topics and concepts on the QU-matrix for each cell. The resulting data cube is called the *topic-concept cube*. In the online stage, we use the learned model parameters to support multidimensional lookups, reverse lookups, and advanced analytical explorations.

3 Related Work

Supporting multidimensional online analysis of large-scale search log data is a new problem. To the best of our knowledge, the most related work to our project is a query traffic analysis service provided by a major commercial search engine[1]. The service allows users to look up and compare the hottest queries in specified time ranges, regions, verticals, and topics. However, the service organizes the user interests at only two levels: the lower individual query level containing individual queries, and the higher topic level consisting of 27 topics such as "Health" and "Entertainment".

As will be illustrated in our experiment results, using only 27 topics seems insufficient to summarize user interests from time to time. Instead, a richer hierarchical structure of topics learned from search logs, as implemented in our project, is more effective in multidimensional analysis. For example, after browsing the hottest queries in topic "Entertainment", a user may want to drill down to a subtopic "Entertainment/Film". The current two layer structure in the existing project can only provide limited analysis power.

Moreover, using individual queries to represent user interests seems ineffective. It is well recognized that users may formulate various queries for the same information need. Therefore, the search log data at the individual query level may be sparse. For example, the system returns queries *"games"*, *"game"*, *"games online"*, and *"free games"* as the 1st, 2nd, 7th, and 8th hottest queries, respectively, on topic "Game" in the US. Clearly, those queries carry similar information needs. To make the analysis more effective, as achieved by the topic-concept model in our project, we need to summarize similar queries into concepts and represent user interests by concepts instead of individual queries.

To a broader extent, our project is related to the previous studies on search query traffic patterns, user interest summarization, data cube computation, and spatiotemporal mining.

Several previous studies explored the patterns of query traffic with respect to various aspects, such as time, locations, and search devices. For ex-

[1]Due to the policy of Microsoft, we do not reveal the name of the search engine mentioned here.

ample, Beitzel *et al.* [7] investigated how the web query traffic varied hourly. Backstrom *et al.* [4] reported a correlation between the locations referred in queries and the geographic focus of the users who issued those queries. Kamvar *et al.* [16] presented a log-based comparison on the distribution and variability of search tasks that users performed from three platforms, namely computers, iPhones, and conventional mobile phones. However, those studies mainly focused on the general trends of user query traffic without mining user interests from the log data.

Previous approaches to summarizing user search queries can be divided into two categories: the *clustering approaches* and the *categorization approaches*. A clustering approach groups similar queries and URLs in an unsupervised way. For example, Zhao *et al.* [22] identified events in a time-series of click-through bipartites derived from search logs. Each event consists of a set of queries and clicked URLs that evolve synchronously along the time-series. In [5, 6, 8, 20], the authors clustered the click-through bipartites and grouped similar queries into concepts. A categorization approach classifies queries into a set of pre-defined topics in a supervised way. For example, Shen *et al.* [19] leveraged the search results returned by a search engine and converted the query categorization problem into a text categorization problem. Both the clustering and categorization approaches are effective to summarize user interests into events, concepts, or topics. However, they do not consider how the interests vary with respect to various dimensions such as time and locations. Consequently, those methods cannot be directly used to support lookups and reverse lookups as well as advanced online multidimensional exploration.

Grey *et al.* [12] developed data cubes as the core of data warehouses and OLAP systems. A data cube contains aggregated numeric measures with respect to group-bys of dimensions. Zhang *et al.* [21] proposed a topic cube that extends the traditional data cube with a measure in a hierarchy of topics. Each cell in the cube stores the parameters learned from a topic modeling process. Users can apply the OLAP operations such as roll-up and drill-down along both standard dimensions and the topic dimension. The system was built on a single machine. There are several critical differences between our topic-concept cube and the topic cube [21]. The topic model pLSA [13] used in [21] targets at modeling documents, which involves only two types of variables, namely the terms as observed variables and the topics as hidden variables. However, to summarize the common interests in search log data, we have to consider more variables, especially, queries and clicked URLs as observed variables, and concepts and topics as hidden variables. Therefore, the traditional pLSA model cannot be applied in our project.

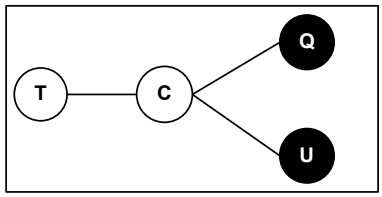


Figure 5: A graphical representation of TC-model.

Consequently, the methods to materialize our topic-concept cubes are very different from those to materialize the topic cubes. Moreover, we reported an empirical study on a much larger set of real data, containing billions of queries and clicks, and processed in a distributed environment.

Mei *et al.* studied mining temporal [17] and spatiotemporal [18] patterns from text data. The authors discovered interesting evolutionary theme patterns in documents with respect to time and locations. A major difference between those studies and our work is that the previous studies only considered a flat level of topics instead of a hierarchy of topics and concepts. Therefore, the previous studies cannot support users' roll-up and drill-down operations along the TC-dimension. Moreover, those previous studies also targeted at modeling documents and applied the traditional pLSA model. As mentioned above, the task of modeling log data involve much more complex variables and thus the traditional pLSA model cannot be directly applied in our work.

4 Topic-Concept Model

We propose a novel *topic-concept model* (*TC-model* for short), a graphical model as shown in Figure 5, to describe the generation process of a QU-matrix. Essentially, we assume that a user bears some search intent in mind when interacting with a search engine. The search intent belongs to certain topics and focuses on several specific concepts. Based on the search intent, the user formulates queries and selectively clicks on search results.

From the search log data, we can observe user queries q and clicks u. Following the convention of graphical models, these two observable variables are represented by black circles in Figure 5. Since user search intents cannot be observed, the topics t and concepts c are latent variables, which are represented by white circles.

Let Q and U be the sets of unique queries and unique URLs in a QU-matrix, respectively. Let C and T be the sets of concepts and topics to

model user interests. The training process of the topic-concept model is to learn four groups of model parameters $\Theta = (\Phi, \Delta, \Upsilon_Q, \Upsilon_U)$. Here, the *prior topic distribution* $\Phi = \{P(t_k)\}$, where $t_k \in T$ and $P(t_k)$ is the prior probability that a user's search intent involves topic t_k. The *concept generation distribution* $\Delta = \{P(c_l|t_k)\}$, where $c_l \in C$, $t_k \in T$, and $P(c_l|t_k)$ is the probability that topic t_k generates concept c_l. The *query generation distribution* $\Upsilon_Q = \{P(q_i|c_l)\}$, where $q_i \in Q$, $c_l \in C$, and $P(q_i|c_l)$ is the probability that concept c_l generates query q_i. The *URL generation distribution* $\Upsilon_U = \{P(u_j|c_l)\}$, where $u_j \in U$, $c_l \in C$, and $P(u_j|c_l)$ is the probability that concept c_l generates a click on URL u_j.

Given that a user bears a search intent on specific concepts c, we assume that (1) the formulation of queries is conditionally independent of the clicks on search results, i.e., $P(q, u|c) = P(q|c) \cdot P(u|c)$; and (2) both the formulation of queries and the clicks on search results are conditionally independent of the topics t of the search intent, i.e., $P(q, u|t, c) = P(q, u|c)$. Then, the likelihood for each entry (q_i, u_j) in the QU-matrix can be factorized as follows.

$$\begin{aligned} L(q_i, u_j; \Theta) &= \Big(\textstyle\sum_{t_k \in T} \sum_{c_l \in C} P(q_i, u_j, c_l, t_k; \Theta)\Big)^{n_{ij}} \\ &= \Big(\textstyle\sum_{t_k \in T} \sum_{c_l \in C} P(t_k) P(c_l|t_k) P(q_i|c_l) P(u_j|c_l)\Big)^{n_{ij}} \end{aligned} \tag{1}$$

where n_{ij} is the value of entry (q_i, u_j) in the QU-matrix. The likelihood for the whole QU-matrix D is $L(D; \Theta) = \prod_{q_i, u_j} P(q_i, u_j; \Theta)$.

Since the data likelihood is hard to be maximized analytically, we apply the Expectation Maximization (EM) algorithm [11]. The EM algorithm iterates between the E-step and the M-step. The E-step computes the expectation of the log data likelihood with respect to the distribution of the latent variables derived from the current estimation of the model parameters. In the M-step, the model parameters are estimated to maximize the expected log likelihood found in the E-step. We have the following equations for the E-step in the r-th iteration.

$$\begin{aligned} P^r(c_l|q_i, u_j) \propto \sum_{t_k} \big(&P^{r-1}(t_k) \cdot P^{r-1}(c_l|t_k) \\ &\cdot P^{r-1}(q_i|c_l) \cdot P^{r-1}(u_j|c_l)\big) \end{aligned} \tag{2}$$

$$\begin{aligned} P^r(t_k|q_i, u_j) \propto \sum_{c_l} \big(&P^{r-1}(t_k) \cdot P^{r-1}(c_l|t_k) \\ &\cdot P^{r-1}(q_i|c_l) \cdot P^{r-1}(u_j|c_l)\big) \end{aligned} \tag{3}$$

In the M-step of the r-th iteration, the model parameters are updated by

the following equations.

$$P^r(t_k) = \frac{\sum_{q_i,u_j} n_{ij} P^r(t_k|q_i,u_j)}{\sum_{t_{k'}} \sum_{q_i,u_j} n_{ij} P^r(t_{k'}|q_i,u_j)} \tag{4}$$

$$P^r(q_i|c_l) = \frac{\sum_{u_j} n_{ij} P^r(c_l|q_i,u_j)}{\sum_{q_{i'},u_j} n_{i'j} P^r(c_l|q_{i'},u_j)} \tag{5}$$

$$P^r(u_j|c_l) = \frac{\sum_{q_i} n_{ij} P^r(c_l|q_i,u_j)}{\sum_{q_i,u_{j'}} n_{ij'} P^r(c_l|q_i,u_{j'})} \tag{6}$$

$$P^r(c_l|t_k) = \frac{\sum_{q_i,u_j} n_{ij} P^r(c_l|q_i,u_j) P^r(t_k|q_i,u_j)}{\sum_{c_{l'}} \sum_{q_i,u_j} n_{ij} P^r(c_{l'}|q_i,u_j) P^r(t_k|q_i,u_j)} \tag{7}$$

5 Learning Large TC-models

Although the EM algorithm can effectively learn the parameters in TC-models, there are still several challenges to apply it on huge search log data. In Section 5.1, we will develop distributed algorithms for learning TC-models from a huge amount of data. In Section 5.2, we will discuss the model initialization steps. Last, in Sections 5.3 and 5.4, we will develop effective heuristics to reduce the number of parameters to learn in each machine.

5.1 Distributed Learning of Parameters

Search logs typically contain billions of query-and-click events involving tens of millions of unique queries and URLs. To address this challenge, we develop distributed algorithms for the E-step and M-step.

In our learning process, a QU-matrix is represented by a set of (q_i, u_j, n_{ij}) tuples. Since a query usually has a small number of clicked URLs, a QU-matrix is very sparse. We only need to record the tuples where $n_{ij} > 0$. We first partition the QU-matrix into subsets and distribute each subset to a machine (called a *process node*). Then we carry out the E-step and the M-step.

In the E-step of the r-th iteration (Algorithm 1), each process node loads the current estimation of the model parameters and scans the assigned subset of training data once. For each tuple (q_i, u_j, n_{ij}), the process node enumerates all the concepts c_l such that $P^{r-1}(q_i|c_l) > 0$ and $P^{r-1}(u_j|c_l) > 0$. For each enumerated concept c_l, the process node further enumerates each topic t_k such that $P^{r-1}(c_l|t_k) > 0$ and evaluates the value $v_{k,l} = P^{r-1}(t_k)P^{r-1}(c_l|t_k)P^{r-1}(q_i|c_l)P^{r-1}(u_j|c_l)$. The values of $v_{k,l}$ are summed up to estimate $P^r(c_l|q_i,u_j)$ and $P^r(t_k|q_i,u_j)$ using Equations 2

Algorithm 1 The r-th round E-step for each process node.

Input: the subset of training data S; the model parameters Θ^{r-1} of the last round

Load model parameters Θ^{r-1};
for each tuple (q_i, u_j, n_{ij}) in S **do**
  $\sigma_{ij} = 0$;
  for each topic $t_k \in T$ **do** $\sigma^t_{ijk} = 0$;
  let $C_{ij} = \{c_l | P^{r-1}(q_i|c_l) > 0 \;\&\&\; P^{r-1}(u_j|c_l) > 0\}$;
  for each concept $c_l \in C_{ij}$ **do**
    $\sigma^c_{ijl} = 0$;
    for each topic $t_k \in T$ such that $P^{r-1}(c_l|t_k) > 0$ **do**
      $v = P^{r-1}(t_k)P^{r-1}(c_l|t_k)P^{r-1}(q_i|c_l)P^{r-1}(u_j|c_l)$;
      $\sigma^c_{ijl} += v$; $\sigma^t_{ijk} += v$; $\sigma_{ij} += v$;
  for each concept $c_l \in C_{ij}$ **do**
    for each topic $t_k \in T$ such that $P^{r-1}(c_l|t_k) > 0$ **do**
      output$(q_i, u_j, c_l, t_k, n_{ij}, \sigma^c_{ijl}/\sigma_{ij}, \sigma^t_{ijk}/\sigma_{ij})$;

and 3, respectively. Finally, we output the probabilities for the latent variables. Those results will serve as the input of the M-step.

In the M-step, we estimate the model parameters based on the probabilities of the hidden variables. According to Equations 4-7, the estimation for each parameter involves a sum over all the queries and URLs. Since the matrix is distributed on multiple machines, the summation involves aggregating the intermediate results across machines, which is particularly suitable for a *Map-Reduce* system [10]. In general, Map-Reduce is a programming model for distributed processing of large data sets. In the *map* stage, each 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 the map stage of the M-step, each process node receives a subset of tuples $(q_i, u_j, c_l, t_k, n_{ij}, \sigma^c_{ijl}/\sigma_{ij}, \sigma^t_{ijk}/\sigma_{ij})$. For each tuple, the process node emits four key-value pairs as shown in Table 2. In the reduce stage, the process nodes simply sum up all the values with the same key and update the model parameters using Equations 4-7.

Key	Value	Key	Value
$\langle t_k \rangle$	$n_{ij} \cdot \sigma^t_{ijk}/\sigma_{ij}$	$\langle q_i, c_l \rangle$	$n_{ij} \cdot \sigma^c_{ijl}/\sigma_{ij}$
$\langle c_l, t_k \rangle$	$n_{ij} \cdot \sigma^c_{ijl} \cdot \sigma^t_{ijk}/\sigma^2_{ij}$	$\langle u_j, c_l \rangle$	$n_{ij} \cdot \sigma^c_{ijl}/\sigma_{ij}$

Table 2: The key/value pairs at the map stage of the r-th round of M-step.

5.2 Model Initialization

The Topic-Concept model consists of four sets of parameters, Φ, Δ, Υ_Q and Υ_U. We first initialize the query-and-click generation probabilities Υ_Q and Υ_U by mining the concepts from the click-through bipartite. We then initialize the prior topic probabilities Φ and the concept generation probabilities Δ by assigning concepts to topics.

To mine concepts from a click-through bipartite, we cluster queries from the query-URL bipartite graph by a two-step propagation approach [9]. For each query cluster Q_l, we find the set of URLs U_l such that each URL $u \in U_l$ is connected with at least one query in Q_l. In the first step of propagation, Q_l is expanded to Q'_l such that each query $q' \in Q'_l$ is connected with at least one URL $u \in U_l$. In the second step of propagation, U_l is expanded to U'_l such that each URL $u' \in U'_l$ is connected with at least one query $q' \in Q'_l$. Finally, we represent each concept c_l by the pair of query and URL sets (Q'_l, U'_l), and initialize the query and URL generation probabilities by

$$P^0(q_i|c_l) \propto \sum_{u_j \in U'_l} n_{ij}; \quad P^0(u_j|c_l) \propto \sum_{q_i \in Q'_l} n_{ij},$$

where n_{ij} is the value of entry (q_i, u_j) in the QU-matrix.

After deriving the set of concepts C, we consider the set of topics T. Although we may automatically mine topics by clustering concepts, in practice, there are several well-accepted topic taxonomies, such as Yahoo! Directory [3], Wikipedia [2], and ODP [1]. We use the ODP topic taxonomy in this paper, though others can be adopted as well.

The ODP taxonomy is a hierarchical structure where each parent topic subsumes several sub topics, and each leaf topic is manually associated with a list of URLs by the ODP editors. Given a set of topics at some level in the taxonomy, we can initialize the concept generation probabilities $P(c_l|t_k)$ as follows.

According to Bayes Theorem, $P(c_l|t_k) \propto P(c_l)P(t_k|c_l)$. The prior probability $P(c_l)$ indicates the popularity of concept c_l and the probability $P(t_k|c_l)$ indicates how likely c_l involves topic t_k. Suppose c_l is represented by the

query-and-URL sets (Q'_l, U'_l). The popularity of c_l can be estimated by $\hat{P}(c_l) \propto \sum_{q_i \in Q'_l, u_j \in U'_l} n_{ij}$, where n_{ij} is the value of entry (q_i, u_j) in the QU-matrix. To tell how likely c_l involves topic t_k, we merge the text content of the URLs $u \in U'_l$ into a pseudo-document d_l. Then, the problem of estimating $P(t_k|c_l)$ is converted into a text categorization problem, and $P(t_k|c_l)$ can be estimated by applying any text categorization techniques (e.g., [14, 15]) on the pseudo-document d_l. Based on the estimated $\hat{P}(c_l)$ and $\hat{P}(t_k|c_l)$, we initialize the parameters by

$$P^0(c_l|t_k) \propto \hat{P}(c_l)\hat{P}(t_k|c_l); \quad P^0(t_k) \propto \sum_{c_l} \hat{P}(c_l)\hat{P}(t_k|c_l).$$

Why do we still need the EM iterations given that we can estimate all the model parameters in the initialization stage? The EM iterations can improve the quality of concepts and topics by a mutual reinforcement process. In the TC-model, the probabilities $P(q|c)$ and $P(u|c)$ assign queries and URLs to concepts, while the probabilities $P(c|t)$ assign concepts to topics. In the initialization stage, those two types of probabilities are estimated independently. If two queries/URLs belong to the same concept, it is more likely that they belong to the same topic, and vice versa. Therefore, if we jointly consider those two types of probabilities, we may derive more accurate assignments of concepts and topics. In the EM iterations, the relationship between concepts and topics is captured by the probabilities $P(c|q, u)$ and $P(t|q, u)$, which contribute to the increase of the data likelihood. In our experiments on a real data set, the data likelihood is increased by 11% after the EM iterations.

5.3 Reducing Re-estimated Parameters

As described in Section 5.1, in the E-step, each process node estimates $P(c_l|q_i, u_j)$ and $P(t_k|q_i, u_j)$ on the basis of the last round estimation of parameters Φ, Δ, Υ_Q, and Υ_U. Let N_t, N_c, N_q, N_u be the numbers of topics, concepts, unique queries, and unique URLs, respectively. The sizes of the parameter sets are $|\Phi| = N_t$, $|\Delta| = N_t \cdot N_c$, $|\Upsilon_Q| = N_q \cdot N_c$, and $|\Upsilon_U| = N_u \cdot N_c$. In practice, we usually have tens of millions of unique queries and URLs in the search log data, which may form millions of concepts. For example, in the real data set in our experiments, we have 11.76 million unique queries, 9.5 million unique URLs, 4.71 million concepts, and several hundred topics. The total size of the parameter space reaches 10^{14}. Consequently, it is infeasible to hold the full parameter space into the main memory of a process node.

To reduce the number of parameters to be re-estimated, we analyze the cases when the model parameters remain zero during the EM iterations.

Suppose a process node receives a subset S of training data in the E-step, we give a tight superset $\Theta(S)$ of the nonzero model parameters that need to be accessed by the process node in the E-step. In our experiments, $|\Theta(S)|$ for each process node is several orders of magnitudes smaller than the size of full parameters space. Each process node only needs to process a subset of $\Theta(S)$.

Lemma 1. *The query generation probability at the r-th iteration $P^r(q_i|c_l) = 0$ if $P^0(q_i|c_l) = 0$.*

Proof. Let U be the whole set of unique URLs. From Equation 2, if $P^{r-1}(q_i|c_l) = 0$, then $P^r(c_l|q_i, u_j) = 0$ holds for every $u_j \in U$. According to Equation 5, if $P^r(c_l|q_i, u_j) = 0$ holds for every $u_j \in U$, then $P^r(q_i|c_l) = 0$. Therefore, we have $P^{r-1}(q_i|c_l) = 0 \Rightarrow P^r(q_i|c_l) = 0$. Using simple induction, we can prove $P^0(q_i|c_l) = 0 \Rightarrow P^0(q_i|c_l) = 0$. □

Similarly, we can prove the following lemma.

Lemma 2. *The URL generation probability at the r-th iteration $P^r(u_j|c_l) = 0$ if $P^0(u_j|c_l) = 0$.*

Let us consider the concept generation probabilities $P(c_l|t_k)$. We call a pair (q_i, u_j) *belongs to* concept c_l, denoted by $(q_i, u_j) \in c_l$, if $n_{ij} > 0$, $P^0(q_i|c_l) > 0$, and $P^0(u_j|c_l) > 0$. Two concepts c_l and $c_{l'}$ are *associated* if there exists a pair (q_i, u_j) belonging to both concepts. Trivially, a concept is associated with itself. Let $A(c_l)$ be the set of concepts associated with c_l, and $QU(c_l)$ be the set of pairs (q_i, u_j) that belong to at least one concept associated with c_l, i.e., $QU(c_l) = \{(q_i, u_j)|\exists c_{l'} \in A(c_l), (q_i, u_j) \in c_{l'}\}$. We have the following.

Lemma 3. *The concept generation probability at the r-th iteration $P^r(c_l|t_k) = 0$ if $\forall c_{l'} \in A(c_l)$, $P^{r-1}(c_{l'}|t_k) = 0$.*

Proof. According to the definitions, for any $(q_i, u_j) \notin c_l$, one of the following three predicates holds (1) $n_{ij} = 0$; (2) $P^0(q_i|c_l) = 0$; or (3) $P^0(u_j|c_l) = 0$. If $n_{ij} = 0$, from Equation 7, (q_i, u_j) does not contribute to $P^r(c_l|t_k)$. Otherwise, if $P^0(q_i|c_l) = 0$ or $P^0(u_j|c_l) = 0$, according to Lemmas 1 and 2, we have either $P^{r-1}(q_i|c_l) = 0$ or $P^{r-1}(u_j|c_l = 0)$. From Equation 2, if either $P^{r-1}(q_i|c_l) = 0$ or $P^{r-1}(u_j|c_l) = 0$, then $P^r(c_l|q_i, u_j) = 0$. Therefore, Equation 7 can be re-written as

$$P^r(c_l|t_k) \propto \sum_{(q_i,u_j)\in c_l} n_{ij} P^r(c_l|q_i, u_j) P^r(t_k|q_i, u_j). \tag{8}$$

Now we only need to focus on $P^r(t_k|q_i, u_j)$ *for pairs* $(q_i, u_j) \in c_l$. *According to the definition of* $A(c_l)$, *for any pair* $(q_i, u_j) \in c_l$ *and concept* $c_{l''} \notin A(c_l)$, *either* $P^0(q_i|c_{l''}) = 0$ *or* $P^0(u_j|c_{l''}) = 0$ *holds. Using Lemmas 1 and 2, we can rewrite Equation 3 for every pair* $(q_i, u_j) \in c_l$ *as*

$$P^r(t_k|q_i, u_j) \propto \sum_{c_{l'} \in A(c_l)} P^{r-1}(t_k) \cdot P^{r-1}(c_{l'}|t_k) \cdot P^{r-1}(q_i|c_{l'}) \cdot P^{r-1}(u_j|c_{l'}). \quad (9)$$

According to Equation 9, if $\forall c_{l'} \in A(c_l)$, $P^{r-1}(c_{l'}|t_k) = 0$, *then* $P^r(t_k|q_i, u_j) = 0$ *holds for every* $(q_i, u_j) \in c_l$. *Further according to Equation 8, if* $P^r(t_k|q_i, u_j) = 0$ *holds for every* $(q_i, u_j) \in c_l$, *then* $P^r(c_l|t_k) = 0$. *Therefore, if* $\forall c_{l'} \in A(c_l)$, $P^{r-1}(c_{l'}|t_k) = 0$, *then* $P^r(c_l|t_k) = 0$. □

Lemma 3 suggests that at each round of iteration, a concept c_l propagates its nonzero topics t_k (i.e., topics such that $P(c_l|t_k) > 0$) one step further to all its associated concepts.

To further explore the conditions for $P^r(c_l|t_k) = 0$, we build a *concept association graph* $G(V, E)$, where each vertex $v \in V$ represents a concept c, and two concepts c_a and c_b are linked by an edge $e_{ab} \in E$ if they are associated with each other. In the association graph, two concepts c_a and c_b are *connected* if there exists a path between c_a and c_b. The connected component $N^*(c_a)$ of concept c_a consists of all concepts c_b that are connected with c_a. The *distance* between two concepts c_a and c_b is the length of the shortest path between c_a and c_b in the graph. If c_a and c_b are not connected, the distance is set to ∞. The set of *m-step neighbors* $N^m(c_a)$ $(1 \leq m < \infty)$ of concept c_a consists of the concepts whose distance from c_a is at most m. We have the following lemma by recursively applying Lemma 3.

Lemma 4. *The concept generation probability at the* r*-th iteration* $P^r(c_l|t_k) = 0$ *if* $\forall c_{l'} \in N^m(c_l)$ $(1 \leq m \leq r)$, $P^{r-m}(c_{l'}|t_k) = 0$. *Moreover,* $P^r(c_l|t_k) = 0$ *if* $\forall c_{l'} \in N^*(c_l)$, $P^0(c_{l'}|t_k) = 0$.

Using Lemmas 1-4, we can give a tight superset of the parameters needed in the E-step for any subset S of training data. Let (q_i, u_j, n_{ij}) be a training tuple in S. In the E-step, we enumerate the concepts c_l such that $P^{r-1}(q_i|c_l) > 0$ and $P^{r-1}(u_j|c_l) > 0$. According to Lemmas 1 and 2, to process (q_i, u_j, n_{ij}), we can enumerate only those concepts $C'_{ij} = \{c_l|(q_i, u_j) \in c_l\}$.

We consider the nonzero parameters for each concept c_l. Using Lemmas 1 and 2, the nonzero query and URL generation probabilities are simply $\Upsilon^+_Q(c_l) = \{P(q_i|c_l)|P^0(q_i|c_l) > 0\}$ and $\Upsilon^+_U(c_l) = \{P(u_j|c_l)|P^0(u_j|c_l) > 0\}$,

respectively. Furthermore, let $T(c_l) = \{P(c_l|t_k)|P^0(c_l|t_k) > 0\}$ and $T^*(c_l) = \bigcup_{c_{l'} \in N^*(c_l)} T(c_{l'})$. Using Lemma 4, the nonzero concept generation probabilities are $\Delta^+(c_l) = \{P(c_l|t_k)|t_k \in T^*(c_l)\}$.

Let C'_S be the set of concepts that are enumerated for the training tuples in S, i.e., $C'_S = \bigcup_{s_{ij} \in S} C'_{ij}$. We summarize the above discussion as follows.

Theorem 1. *Let S be a subset of training data, the set of nonzero parameters need to be accessed in the E-step for S is a subset of $\Theta(S)$, where*

$$\Theta(S) = \left(\{P(t_k)\}, \bigcup_{c_l \in C'_S} \Upsilon^+_Q(c_l), \bigcup_{c_l \in C'_S} \Upsilon^+_U(c_l), \bigcup_{c_l \in C'_S} \Delta^+(c_l) \right).$$

In practice, a concept association graph can be highly connected. That is, for any two concepts c_a and c_b, there likely exists a path $c_a, c_{l1}, \ldots, c_{lm}, c_b$. In some cases, although each pair of adjacent concepts on the path are related to each other, the two end concepts c_a and c_b of the path may be about dramatically different topics. As discussed before, in the EM iterations, each concept propagates its nonzero topics to its neighbors. Consequently, after several rounds of iterations, two totally irrelevant concepts c_a and c_b may exchange their nonzero topics through the path $c_a, c_{l1}, \ldots, c_{lm}, c_b$. To avoid over propagation of the nonzero topics, we may constrain the propagation up to ς steps. Specifically, for each concept c_l, let $T(c_l) = \{P(c_l|t_k)|P^0(c_l|t_k) > 0\}$ and $T^\varsigma(c_l) = \bigcup_{c_{l'} \in N^\varsigma(c_l)} T(c_{l'})$, we constrain the concept generation probability $P(c_l|t_k) = 0$ if $t_k \notin T^\varsigma(c_l)$. In our experiments, we find that the nonzero topics propagated from the neighbors of more than one step away are often noisy. Therefore, we set ς to 1.

Theorem 1 greatly reduces the number of parameters to be re-estimated in process nodes in practice. For example, when we use 50 process nodes in our experiments, each process node only needs to re-estimate 62 million parameters, which is about 10^{-7} of the size of the total parameter space. In practice, 62 million parameters may still be too many for a machine with small memory, e.g., less than 2GB. In this case, the process node can recursively split the assigned training data S_n into smaller blocks $S_{nb} \subset S_n$ until the necessary nonzero parameters $\Theta(S_{nb})$ for each block can be loaded into the main memory. Then, the process node can carry out the E-step block by block. We report the details of the experiment in Section 7.2.

5.4 A Heuristic Data Partition Method

In the distributed training of the TC-model, we need to partition the whole data set into several slices and allocate one slice of data to one process node.

A straightforward method to partition the training data D is to randomly assign an equal size of subset $S_n \subseteq D$ to the n-th process node p_n, where $|S_n| = |D|/N$ and N is the total number of process nodes. If each process node can hold all the necessary nonzero parameters $\Theta(S_n)$ into the main memory, we simply carry out the E-step. Otherwise, we have to recursively split S_n into smaller blocks $S_{nb} \subset S_n$ until the necessary nonzero parameters $\Theta(S_{nb})$ for each block can be loaded into the main memory. Then, we can carry out the E-step block by block.

To carry out the E-step for each block, the process node p_n has to scan the whole file of the nonzero parameters and keep those in $\Theta(S_{nb})$ in main memory. In practice, the file of nonzero parameters is usually large, and the cost for disk reading is expensive. For example, in our experiments, the size of the parameter file is about 4GB and it takes about 420 seconds to scan the file once for loading parameters. Compared with the runtime of the inference process for a block of about 0.15 million training tuples, which is about 230 seconds, the efficiency of the E-step is heavily influenced by the number of times the parameter file has to be scanned, which is also the number of blocks to be processed. To improve the efficiency, we want to reduce the number of blocks for the process nodes.

To address the above challenges, let us consider two training tuples s_{ij1} and s_{ij2}. Suppose s_{ij1} involves concepts $C_{ij1} = \{c_1, c_2, c_3\}$ and s_{ij2} involves concepts $C_{ij2} = \{c_1, c_2, c_4\}$. Since C_{ij1} is heavily overlapped with C_{ij2}, the E-step on those two training tuples s_{ij1} and s_{ij2} share many parameters. If they are assigned to the same process node, the node actually only needs to keep the parameters for four concepts (i.e., c_1, c_2, c_3, c_4), instead of six concepts (i.e., $|C_{ij1}|+|C_{ij2}|$). Based on this idea, we adopt a greedy approach and try to assign the tuples which involve common concepts on the same process node.

Algorithm 2 shows our data partition method. The method has a master-slave structure, where each process node p_n acts as a slave. The master maintains two tables. The *concept size table* records the estimated size $|\Upsilon_Q^+(c_l)| + |\Upsilon_U^+(c_l)| + |\Delta^+(c_l)|$ of the nonzero parameters for each concept c_l. The *client memory table* traces the current size of the free memory for each client. Each slave p_n maintains a *concept ID list* which records the IDs of the concepts whose nonzero parameters will be loaded into the main memory by p_n during the E-step. Moreover, to facilitate the look up of the concept list C_{ij} for each training tuple (q_i, u_j, n_{ij}), we create two mapping tables. The *query mapping table* records for each query q_i the concepts c_l if $P^0(q_i|c_l) > 0$. Similarly, the *URL mapping table* records for each URL u_j the concepts c_l if $P^0(u_j|c_l) > 0$. Since these two tables are usually large,

Algorithm 2 A heuristic data partition method

Input: The whole set of training data D;
Output: A subset of training data for each process node;

Master Side: Host Program
Initialize: Concept size list: L_s and client memory list: L_m;

for each tuple $(q_i, u_j, n_{ij}) \in D$ **do**
  if no client has free memory **then**
    initialize L_m;
    broadcast message "Start E-step";
  // derive C_{ij};
  broadcast message "Lookup (i, j)";
  receive concept lists Q_i and U_j from each client;
  let $C_{ij} \Leftarrow$ intersect the concepts in Q_i and U_j;
  // select the node to process (q_i, u_j, n_{ij});
  broadcast message "Bid C_{ij}";
  receive the concept list C_n^+ from each client;
  for each process node **do**
    $M_n^+ = 0$;
    for each concept $c \in C_n^+$ **do** $M_n^+ += L_s[c]$;
  choose the client pn_{min} with the minimum M_n^+
  send message "Process $(q_i, u_j, n_{ij}, C_{ij})$" to client pn_{min};
  $L_m[pn_{min}] += \min\{M_n^+\}$;

Slave Side: Daemon Program
Input: messages from the server;
Initialization: The query mapping table M_q and the URL mapping table M_u, the subset of training data D_n, and the concept ID list L_c;

for each Master's command **do**
  if command == "Start E-step" **then**
    load parameters for the concepts in L_c;
    call E-step (Algorithm 1) on D_n;
    $L_c = \emptyset$;
  if command == "Lookup (i, j)" **then**
    if $q_i \in M_q$ **then return** $M_q[i]$;
    if $u_j \in M_u$ **then return** $M_u[j]$;
  if command == "Bid C_{ij}" **then**
    $C_n^+ = \emptyset$;
    for each $c \in C_{ij}$ **do if** $c \notin L_c$ **then** $C_n^+ \cup = c$;
    return C_n^+;
  if command == "Process $(q_i, u_j, n_{ij}, C_{ij})$" **then**
    $D_n \cup = (q_i, u_j, n_{ij}, C_{ij})$;
    for each $c \in C_{ij}$ **do if** $c \notin L_c$ **then** $L_c \cup = c$;

they can be distributed to the process nodes where each node is responsible for the mapping of a subset of queries and a subset of URLs.

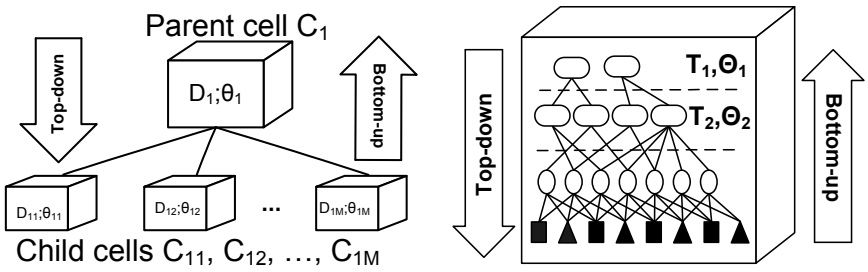
In the data partition process, the master scans the training tuples $s_{ij} = (q_i, u_j, n_{ij})$ one by one. For each tuple, the master sends a message (i, j) to all the process nodes. The process nodes will check their query and URL mapping tables and return the IDs of the concepts c_l to the master if $P^0(q_i|c_l) > 0$ or $P^0(u_j|c_l) > 0$. Then, the master derives an ID list $ID_{ij} = \{l|P^0(q_i|c_l) > 0 \;\&\&\; P^0(u_j|c_l) > 0\}$.

In the second round of communication, the master sends the ID list ID_{ij} to all the process nodes. Each process node p_n reports to the master a list C_n^+ of concept IDs which appear in ID_{ij} but do not exist in its concept ID table. This means that if s_{ij} is to be processed on p_n, the process node need to add the nonzero parameters for the concepts with IDs in C_n^+ into the memory. The master will look up the concept size table and find out for each p_n the memory cost M_n^+ for the nonzero parameters for the concepts with IDs in C_n^+. Finally, the master will assign s_{ij} to the process node with the smallest M_n^+. If that process node has reached the limit of memory, s_{ij} will be assigned to the process node which has the next smallest M_n^+, and so on. If the smallest M_n^+ is claimed by multiple process nodes, the master will assign s_{ij} to the one with the largest free memory.

Once the recipient node is determined, the master will send it a message $(q_i, u_j, n_{ij}, ID_{ij})$. The process node will save it for the future E-step. Finally, the master updates the client memory table and the recipient node updates the local concept table accordingly. At the point when all process nodes reach the limit of memory, the E-step is carried out based on the current assignment of training tuples. After the E-step, the same procedure continues until all the training data have been processed. An empirical study of the effectiveness of the heuristic data partition method is described in Section 7.2.

6 Cube Construction and Request Answering

Similar to a traditional data cube, a *topic-concept cube* (*TC-cube* for short) contains some standard dimensions such as time and locations. However, a TC-cube differs from a traditional data cube in several critical aspects. First, for each cell in a TC-cube, we learn the TC-models from the training data in the cell and use the model parameters as the measure of the cell. Those parameters allow us to effectively answer lookups and reverse lookups introduced in Section 1. Second, a TC-cube contains a special topic-concept dimension (*TC-dimension* for short) as shown in Figure 1.



(a) On standard dimension (b) On TC-dimension

Figure 6: The cube construction methods on (a) standard dimension and (b) TC-dimension.

To materialize a TC-cube, we need to address three questions. First, how to materialize on the standard dimensions? Second, how to materialize on the TC-dimension? Finally, how to combine the materialization on both the standard dimensions and the TC-dimension to materialize the whole TC-cube? In the following, we will address these three questions in detail.

6.1 Materialization on Standard Dimensions

As illustrated in Figure 6(a), in a standard dimension, the training data in a non-leaf level cell C_1 is split into its children cells $C_{21}, \ldots, C_{2M}$. For example, C_1 may contain the set of training tuples D_1 from the US, while each child cell C_{2m} $(1 \leq m \leq M)$ may contain the set of training tuples D_{2m} from one state of the US. In general, $D_{21}, \ldots D_{2M}$ form a partition of D_1. A naïve method to materialize the standard dimension is to follow the initialization steps in Section 5.2 for each cell and learn the TC-models from scratch. However, since the training data D_{2m} in a child cell is a subset of D_1, the topics and concepts may not differ dramatically between a child cell and a parent cell. Hence, we may develop two methods. In the top-down method, we may inherit the trained parameters Θ_1 for the parent cell C_1 to initialize the parameters for a child cell C_{2m}. Alternatively, in the bottom-up method, we may aggregate the trained parameters $\Theta_{21}, \ldots, \Theta_{2M}$ of the child cells to initialize the parameters for the parent cell C_1. In the following, we discuss these two methods in detail. We compare the performance of these two methods in Section 7.3.

6.1.1 A Top-down method

As illustrated in Figure 6(a), in the top-down method, high level cells are materialized before the low level ones. Without loss of generality, suppose we already compute the model parameters Θ_1 for a cell C_1 and want to materialize the parameters Θ_2 for a cell C_2 which is a child cell of C_1. Let D_1 and D_2 be the sets of query-and-click events in C_1 and C_2, respectively. Obviously, $D_2 \subseteq D_1$.

One way to materialize C_2 is to follow the initialization steps in Section 5.2 and learn the TC-model from scratch. Can we find a better initialization method such that the number of EM iterations can be reduced? Since $D_2 \subseteq D_1$, the topics and concepts in D_2 only partially differ from those in D_1. Therefore, we may consider using the trained parameters for C_1 to initialize the parameters for C_2. In the following, we first describe how to project the concepts in C_1 to C_2, and then discuss how to initialize the parameters for C_2 using those in C_1.

Let c_{1l} be a concept in C_1, which is represented by a pair of query and URL sets (Q_{1l}, U_{1l}). Let Q_2 and U_2 be the sets of queries and URLs in D_2, respectively. We can project c_{1l} on C_2 by calculating $Q_{2l} = Q_{1l} \cap Q_2$ and $U_{2l} = U_{1l} \cap U_2$. Then, the concept c_{2l} can be represented by the projected pair (Q_{2l}, U_{2l}). Correspondingly, the initial query and URL generation probabilities in cell C_2, denoted by $P_2^0(q_i|c_{2l})$ and $P_2^0(u_j|c_{2l})$, respectively, can be estimated by $P_2^0(q_i|c_{2l}) \propto \sum_{u_j \in U_{2l}} n_{2,ij}$ and $P_2^0(u_j|c_{2l}) \propto \sum_{q_i \in Q_{2l}} n_{2,ij}$, where $n_{2,ij}$ is the count of (q_i, u_j) pairs in D_2.

Let us consider how to initialize the concept generation probability $P_2^0(c_{2l}|t_k)$ and the topic prior probability $P_2^0(t_k)$ in cell C_2. We assume the projection of concepts from C_1 to C_2 does not change the meaning of the concepts. In other words, if a concept c_{1l} in C_1 involves a topic t_k, so does its projected image c_{2l} in C_2. Based on this assumption, the probability $P_2(t_k|c_{2l})$ of concept c_{2l} belonging to topic t_k can be inherited from $P_1(t_k|c_{1l}) \propto P_1(t_k)P_1(c_{1l}|t_k)$. Then the initial concept generation probability in cell C_2, denoted by $P_2^0(c_{2l}|t_k)$, can be estimated by $P_2^0(c_{2l}|t_k) \propto P_2(c_{2l})P_2(t_k|c_{2l})$, and the initial topic prior probability in cell C_2, denoted by $P_2^0(t_k)$, can be estimated by $P_2^0(t_k) \propto \sum_{c_{2l}} P_2(c_{2l})P_2(t_k|c_{2l})$, where the concept prior probability $P_2(c_{2l})$ can be estimated by the count of (q_i, u_j) pairs in D_2, i.e., $P_2(c_{2l}) \propto \sum_{q_i \in Q_{2l}, u_j \in U_{2l}} n_{2,ij}$. After initializing the parameters, we carry out the EM algorithm for C_2 as described in Section 5.

6.1.2 A Bottom-up method

In the bottom-up method, we want to estimate the parameters Θ_1 for a higher level cell C_1 (Figure 6(a)). In the similar spirit of the top-down method, we initialize Θ_1 based on the trained parameters $\Theta_{21}, \ldots \Theta_{2M}$ for C_1's child cells $C_{21}, \ldots, C_{2M}$.

Let the concepts $c_{2l1}, \ldots, c_{2lM}$ be the projected images of cell c_{1l} in child cells $C_{21}, \ldots, C_{2M}$ of parent C_1, respectively. The query generation probability of concept c_{1l}, denoted by $P_1(q_i|c_{1l})$, can be initialized by aggregating those of its projected images, i.e., $P_1^0(q_i|c_{1l}) \propto \sum_m n_{2lm} P(q_i|c_{2lm})$, where $n_{2lm} = \sum_{q_i,u_j} n_{2,ij,m}$ and $n_{2,ij,m}$ is the count of (q_i, u_j) pairs in cell C_{2m} $(1 \leq m \leq M)$. Similarly, we can initialize the URL generation probabilities.

For concept generation probabilities $P_1(c_{1l}|t_k)$, we again assume that the meaning of concepts does not change across different level of cells. We initialize $P_1(c_{1l}|t_k)$ in three steps. First, we estimate the probability $P_{2lm}(t_k|c_{2lm})$ for concept c_{2lm} in cell C_{2m} to involve topic t_k by $P_{2lm}(t_k|c_{2lm}) \propto P_{2m}(t_k) \cdot P_{2m}(c_{2lm}|t_k)$. In the second step, we estimate the probability $P_l(c_{1l}|t_k)$ for concept c_{1l} to involve topic t_k by aggregating those probabilities from c_{1l}'s projected images, i.e,. $P_1(t_k|c_{1l}) \propto \sum_m n_{2lm} P_{2m}(t_k|c_{2lm})$. Finally, the concept generation probability can be initialized by $P_1^0(c_{1l}|t_k) \propto P_1(c_{1l}) P_1(t_k|c_{1l})$, where $P_1(c_{1l}) \propto \sum_{q_i,u_j} n_{1,ij}$ and $n_{1,ij}$ is the count of (q_i, u_j) pairs in c_{1l}. The topic prior probability can be initialized by $P_1^0(t_k) \propto \sum_{c_{1l}} P_1(c_{1l}) P_1(t_k|c_{1l})$.

6.2 Materialization on TC Dimension

The topic-concept model assigns the concepts to a set of topics. Given a taxonomy of topics, such as ODP [1], the TC dimension organizes the queries and clicks into a hierarchy of topics and concepts (see Figure 1). To materialize the cube on the TC dimension, we can first learn a topic-concept model with respect to any level of topics in the hierarchy. Then, we can materialize the model parameters with respect to other levels of topics by either a top-down method or a bottom-up method.

Different from the standard dimensions, the topic-concept dimension has the same set of training data at different levels. As a result, the initial concepts derived from the clustering results on the training data are the same for all the levels. Therefore, when we roll up or drill down along the TC dimension, it is reasonable to assume that only the topic prior probabilities $P(t_k)$ and the concept generation probabilities $P(c_l|t_k)$ change substantially with respect to the different sets of topics at different level, while the query and URL generation probabilities, i.e., $P(q_i|c_l)$ and $P(u_j|c_l)$ do not change much at different levels. In the following, we only focus on the initialization

of $P(t_k)$ and $P(c_l|t_k)$.

Without loss of generality, let $T_1 = \{t_{1k}\}$ be the set of topics at some level of a given topic taxonomy, and $T_2 = \{t_{2kn}\}$ be the set of topics one level lower than T_1. In particular, t_{2kn} is a sub topic of t_{1k}, where $1 \leq n \leq N_{1k}$ and N_{1k} is the number of sub topics of t_{1k}.

As illustrated in Figure 6(b), the top-down method along the TC dimension materialize the model parameters Θ_1 with respect to T_1 before the materialization of parameters Θ_2 with respect to T_2. Given the parameters $P_1(t_{1k})$ and $P_1(c_l|t_{1k})$ with respect to T_1, the initialization of $P_2^0(t_{2kn})$ and $P_2^0(c_l|t_{2kn})$ is straightforward as follows. First, we can distribute the mass of prior probability $P_1(t_{1k})$ evenly to its children t_{2kn} $(1 \leq n \leq N_{1k})$, i.e., the initial topic prior probability $P_2^0(t_{2kn}) = P_1(t_{1k})/N_{1k}$. Second, we inherit the concept generation probability of topic t_{2kn} from that of its parent t_{1k}, i.e., $P_2^0(c_l|t_{2kn}) = P_1(c_l|t_{1k})$.

In the bottom-up method, we initialize the parameters for a higher level topic t_{1k} by aggregating those of its sub topics t_{2kn} (Figure 6(b)). To be specific, the topic prior probability can be initialized by $P_1^0(t_{1k}) = \sum_n P_2(t_{2kn})$ and the concept generalization probability can be initialized by $P_1^0(c_l|t_{1k}) \propto \sum_n P_2(c_l|t_{2kn})$.

6.3 Materializing the whole TC-cube

We have two alternative approaches to materializing the whole TC-cube that consists of both standard dimensions and the TC-dimension. The standard-dimension-first approach starts with a specific topic level in the TC-dimension and materializes a raw log data cube along the standard dimensions. Then, it materializes along the TC-dimension for each cell in the raw log data cube. The TC-dimension-first approach specifies a layer in the standard dimensions and processes the topic hierarchy level by level for each cell in the specified layer. In the second step, it materializes the cells in other layers of the standard dimensions. To better illustrate the difference between these two approaches, let us consider the following simple example.

Suppose the TC-dimension of a cube consists of only two levels T_1 and T_2. Further suppose there is only one standard dimension with two layers in the cube. The top layer contains a single cell C_{11} and the bottom layer consists of two cells C_{21} and C_{22}. Finally, suppose we adopt the top-down methods on both the standard dimension and the TC-dimension. The standard-dimension-first approach first learns the TC-model for C_{11} with respect to T_1. Then, it materializes C_{21} and C_{22} with respect to T_1 along the standard dimension. After that, it materializes along the TC-dimension for each cell. That means, it computes the model parameters in C_{11}, C_{21}, and C_{22} with

respect to T_2, respectively. The TC-dimension-first approach also starts with the TC-model for C_{11} with respect to T_1. However, in the second step, it does not materialize C_{21} and C_{22} with respect to T_1. Instead, it materialize C_{11} with respect to T_2 along the TC-dimension. In the third step, it materializes C_{21} and C_{22} with respect to T_1 and T_2, respectively, along the standard dimension. The key difference is as the following. In the standard-dimension-first approach, the TC-model for C_{21} and C_{22} with respect to T_2 are initialized by the parameters of C_{21} and C_{22} with respect to T_1, respectively. On the other hand, in the TC-dimension-first approach, the TC-model for C_{21} and C_{22} with respect to T_2 are initialized by the parameters of C_{11} with respect to T_2.

6.4 Request answering

After materializing the whole TC-cube, we answer the lookups and reverse lookups using the model parameters in the TC-cube. Since the number of model parameters can be large, we store the parameters distributively on a cluster of process nodes, where each node contains the parameters for a set of cells. When the system receives a lookup request, for example, "(time=Dec., 2009; location=US; topic=Games)", it will delegate the query to the process node where the model parameters of the corresponding cell are stored. Then the process node will select the top k concepts c with the largest concept generation probabilities $P(c|t = \text{Games})$. For each top concept, the process node will use the query q with the largest $P(q|c)$ as the representative query. Finally, the system returns a list of representative queries of the top concepts as the answer to the lookup request.

To answer the reverse lookups, we build inverted lists that map key words to concepts. The inverted list can be stored distributively on a cluster of process nodes, where each node takes charge of a range of key words. Suppose a user requests a reverse lookup about "hurricane Bill". The system will delegate the key words to the corresponding node that stores the inverted list for "hurricane Bill". The node retrieves from the inverted list the set of concepts $C_{\text{hurricane Bill}}$ where each concept is related to "hurricane Bill". The system then broadcasts the concepts in $C_{\text{hurricane Bill}}$ to all the nodes that store the model parameters. Each node checks the measures of all its cells and reports $(Dval, Count)$ for each cell, where $Dval$ consists of the corresponding values of the standard dimensions of the cell, and $Count$ is the frequency of the concepts in $C_{\text{hurricane Bill}}$ in the cell, i.e., $Count = \sum_{c \in C_{\text{hurricane Bill}}} \sum_{q_i, u_j \in c} n_{ij}$, where n_{ij} is the value of entry (q_i, u_j) in the QU-matrix of the cell. If the user specifies the levels of the standard dimensions, for example, $time@day; location@country$, the system returns

No.	baseline	TC cube	$P(c\|t)$
1	**games**	games	0.020
2	**game**	pogo	0.013
3	cheats	maxgames	0.012
4	wow	aol games	0.011
5	lottery	wow heroes	0.010
6	xbox	killing games	0.009
7	**games online**	addicted games	0.008
8	**free games**	age of war	0.008
9	wii	powder game	0.008
10	runescape	monopoly online	0.008

Table 3: The top ten queries returned by our TC-cube and the baseline for lookup "(time=ALL; location=US; topic=Games)".

the *Dvals* of the top k cells that match the specified levels of the standard dimension. If the user does not specify the levels, the system will answer the request at the default levels. The user can further drill down or roll up to different levels.

7 Experiments

In this section, we report the results from a systematic empirical study using a large search log from a major commercial search engine. The extracted log data set spans for four months and contains 1.96 billion queries and 2.73 billion clicks from five markets, i.e., the United States, Canada, United Kingdom, Malaysia, and New Zealand. In the following, we first demonstrate the effectiveness of our approach using several examples of the lookup and reverse lookup requests. Then, we examine the efficiency and scalability of our distributed training algorithms for the TC-model.

7.1 Lookups and Reverse Lookups Examples

In this subsection, we show some real examples for the lookups and reverse lookups answered by our system. We use the query traffic analysis service by a major commercial search engine as the baseline. Please refer to Section 3 for a more detailed description of the baseline.

Table 3 compares the results for the lookup request (time = ALL; location = US; topic = Games) returned by our system and the baseline. Since

card_games	$P(c\|t)$	gambling	$P(c\|t)$
pogo	0.020	sun bingo	0.004
gogirlsgames	0.004	wink bingo	0.004
solitaire	0.004	tombola	0.003
aol games	0.003	skybet	0.003
scrabble blast	0.003	ladbrokes	0.002
party_games	$P(c\|t)$	puzzles	$P(c\|t)$
tombola	0.003	pogo	0.006
oyunlar	0.003	sudoku	0.004
fashion games	0.003	meriam webster	0.003
drinking games	0.002	thesaurus com	0.003
evite	0.002	mathgames	0.002

Table 4: The top queries returned by TC-cube for four sub topics of "Games" in the US.

the baseline does not group similar queries into concepts, the top 10 results are quite redundant. For example, the 1st, 2nd, 7th, and 8th queries are similar. Our system summarizes similar queries into concepts and selects only one query as the representative for each concept. Consequently, the top 10 queries returned by our system are more informative. We further request the top results for four sub topics of "Games", namely "card games", "gambling", "party games", and "puzzles". The queries returned by our system are informative (Table 4). However, the baseline only organizes the user queries by a flat set of 27 topics; it does not support drilling down to sub topics.

As an example for reverse lookup, we asked for the group-bys where the search for Hurricane Bill was popular by a request "(time@day, location@state, keyword='hurrican bill')". Purposely we misspelled the keyword "hurricane" to "hurrican" to test the summarization capability of our TC-model. Our system can infer that the keyword "hurrican bill" belongs to the concept that consists of queries *"hurricane bill"*, *"hurrican bill"*, *"huricane bill"*, *"projected path of hurricane bill"*, *"hurricane bill 2009"* and some other variants. Therefore, the system sums up the frequencies of all the queries in the concept and answers the top five states during the days in August 2009 (Figure 7). Figure 8 visualizes the trend of the popularity of the whole concept according to the output of the reverse lookup. The dates in the figure indicate when the concept was most intensively searched in different states in the US. Interestingly, the trend shown in Figure 8 reflects well the

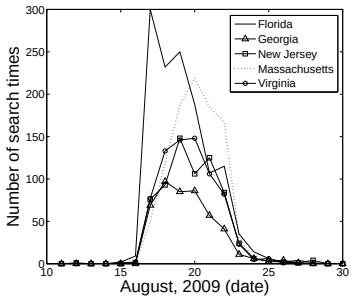


Figure 7: The top five states of US where Hurricane Bill was most intensively search in Aug. 2009.

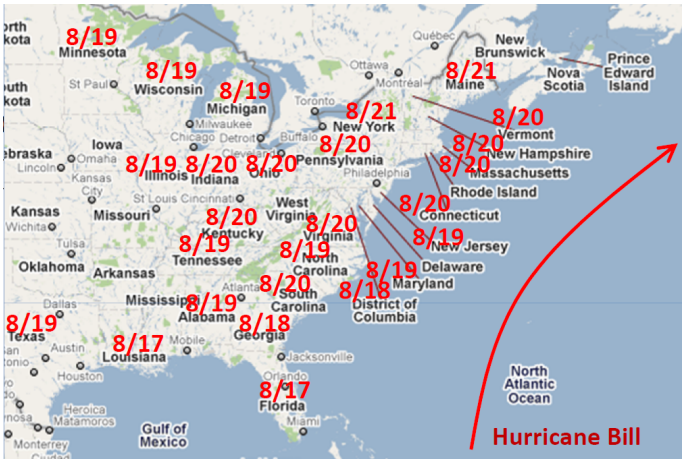
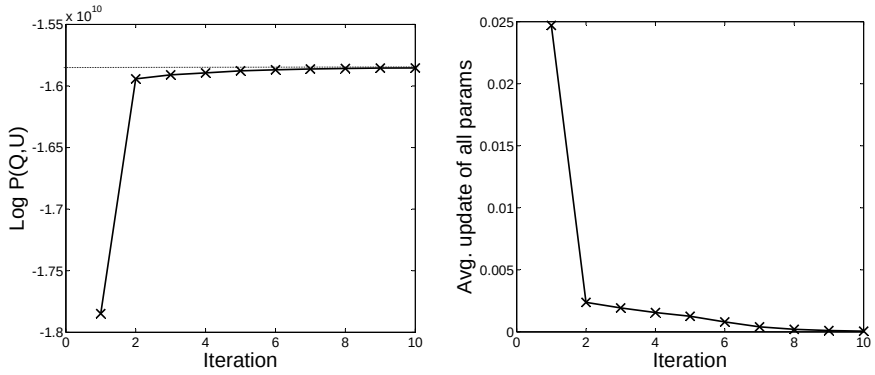


Figure 8: The trajectory of Hurricane Bill.

trajectory and the influence of the hurricane geographically and temporally, which indicates that the real world events can be reflected by the popular queries issued to search engines. However, when we sent the same request to the baseline, it answered that the search volume was not enough to show trend. The reason is that the baseline may only consider the query that exactly matches the misspelled keyword "hurrican bill", which may not be searched often.

7.2 Training TC-models

The TC-model was initialized as described in Section 5.2. We derived 4.71 million concepts, which involve 11.76 million unique queries and 9.5 million



(a) Data likelihood (b) Parameter changes

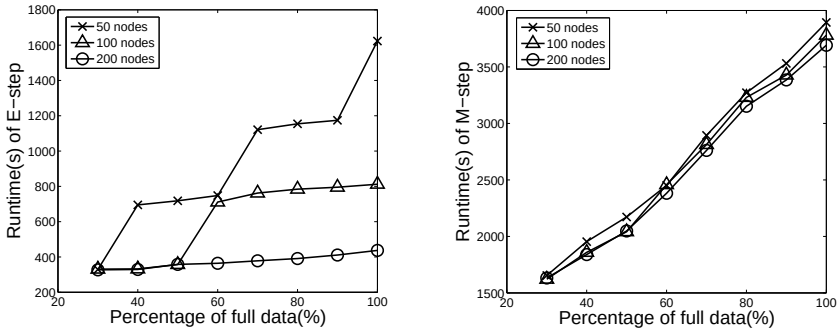
Figure 9: The data likelihood and the average percentage of parameter changes during EM iterations.

unique URLs. On average, a concept consists of 4.68 unique queries and 6.77 unique URLs. We further chose the second level of the ODP [1] taxonomy and applied the text classifier in [14] to categorize the concepts into the 483 topics. For each concept, we kept the top five topics returned by the classifier.

From the raw log data, we derived 23 million training tuples where each training tuple is in the form (q_i, u_j, n_{ij}) and n_{ij} is the number of times URL u_j was clicked on as answers to query q_i.

Figures 9(a) and (b) show the data likelihood and the average percentage of parameter changes with respect to the number of EM iterations. The iteration process converges fast; the data likelihood and parameters do not change much (less than 0.1%) after five iterations. The results suggest that our initialization methods are effective to set the initial parameters close to a local maximum. Moreover, the data likelihood increases by 11% after ten iterations. As explained in Section 5.2, this indicates that the EM algorithm is effective to improve the quality of the TC-model by jointly mining the assignments of concepts and topics in a mutual reinforcement process.

Figures 10(a) and (b) show the runtime of the E-step and the M-step with respect to the percentage of the complete data set with 50, 100, and 200 process nodes, respectively. Each process node has a four-core 2.67GHz CPU and 4GB main memory while 2GB is used in EM iteration and the rest for system usage. We observe the following in Figure 10(a). First, the more process nodes used, the shorter runtime for the E-step. The runtime needed for the E-step on the complete data set by 50, 100, and 200 process nodes



(a) E-step (b) M-step

Figure 10: The scalability of the E-step and the M-step.

is approximately in ratio 4:2:1. This suggests that our algorithm scales well with respect to the number of process nodes. Second, the more process nodes are used, the more scalable is the E-step. For example, when 50 process nodes were used, the runtime increased dramatically when 40%, 70%, and 100% of the data was loaded. As explained in Section 5.3, if the training data for a process node involves too many parameters to be held in the main memory, the algorithm recursively splits the training data into blocks until the parameters needed by a block can be held in the main memory. Therefore, the runtime of the E-step mainly depends on the number of disk scans of the parameter file, i.e., the number of blocks to be processed. When we used 50 process nodes, each node split the assigned training data into 2, 3, and 4 blocks when 40%, 70%, and 100% of the complete data set was used for training, respectively. This explains why the runtime increases dramatically at those points. When we used 200 nodes, each node can process the assigned data without splitting even for the complete data set. Consequently, the runtime increases linearly.

In Figure 10(b), the runtime of M-step increases almost linearly with respect to the data set size, indicating the good scalability of our algorithm. Interestingly, the runtime of the M-step does not change much with respect to the number of process nodes. This is because the major cost of the map-reduce process of the M-step is the merging of parameters, which is done on a single machine. This bottleneck costs the M-step much longer time than the E-step.

Table 5 evaluates the effectiveness of Theorem 1. We executed the E-step on the complete data set with 50, 100, and 200 process nodes, re-

# pn	$\|S\|$	$\|\Theta(S)\|$	# nonezero parameters	Ratio	# B
50	460,062	62,325,884	56,682,113	5.7 e-7	4
100	230,031	35,368,823	30,370,194	3.0 e-7	2
200	115,015	18,656,725	15,821,818	1.6 e-7	1

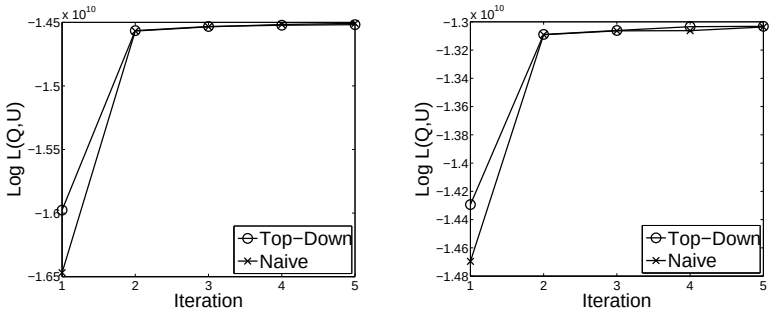
Table 5: The effectiveness of Theorem 1.

# pn	$\|S\|$	Random Partition			Our Partition Method		
		$\|\Theta(S)\|$	Ratio	# B	$\|\Theta(S)\|$	Ratio	# B
3	7,667,712	280,376,458	2.8002 e-6	15	91,379,849	9.1264 e-7	5
5	4,600,627	252,181,397	2.5186 e-6	13	40,606,079	4.0554 e-7	3
10	2,300,314	185,471,181	1.8523 e-6	10	14,444,786	1.4426 e-7	1

Table 6: A comparison of the random partition and our partition method.

spectively. For each setting, e.g., using 50 nodes, we recorded the average number of training tuples $|S|$ assigned to each process, the average number of the estimated nonzero parameters $\Theta(S)$ by Theorem 1, the average number of nonzero parameters after ten iterations, the ratio of the average size of $\Theta(S)$ over the size of the whole parameter space, and the number of blocks processed by each process node. Table 5 suggests the following. First, the average size of $\Theta(S)$ over the size of the whole parameter space is very small, in the order of 10^{-7}. This means Theorem 1 can greatly reduce the number of parameters to be held by each process node. Moreover, the size of the estimated nonzero parameters is close to that of nonzero parameters during the iterations. This indicates that the superset of nonzero parameters given by Theorem 1 is tight.

Table 6 compares the performance by random partition and our heuristic partition method (Section 5.4), where $|S|$ is the average size of training set assigned to each process node, $|\Theta(S)|$ is the average size of the estimated set of nonzero parameters needed by each process node, the column "Ratio" is the ratio of $|\Theta(S)|$ over the size of the whole parameter space, and column "# B" is the number of blocks need to be processed by each process node. Since our partition method tries to put the training examples which share the same nonzero parameters on the same process node as much as possible, the size of $\Theta(S)$ and the number of blocks to be processed for each process node is much smaller than those under random partition. As shown in Figure 10(a), the major cost of the E-step is the number of disk scan for each



(a) Date (b) Location

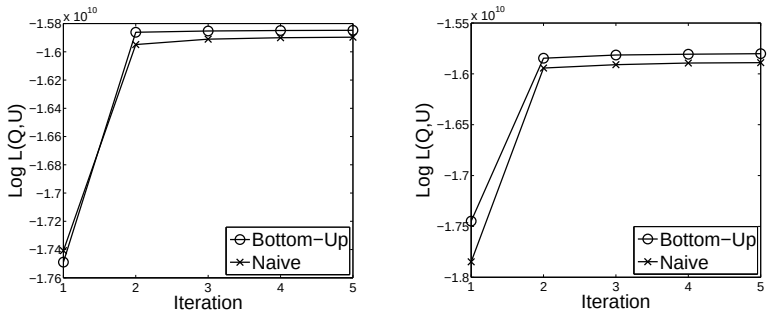
Figure 11: The top-down method *vs.* the naïve method on the standard dimensions: (a) date and (b) location.

block. Consequently, our data partition method can improve the efficiency of the E-step substantially, especially when a limited number of process nodes are available.

7.3 Empirical study on cube construction

In this subsection, we report the results of an empirical study on different options to materialize the standard dimensions, the TC-dimension, and the whole TC-cube. In this study, we used the time and location as two standard dimensions. The top cell C_1 consists of the full data set D_1. Along the standard time dimension, the full data set splits into four subsets $D^t_{21}, \ldots, D^t_{24}$, where each subset consists of one month data and constitutes one child cell $C^t_{2n_t}$ $(1 \leq n_t \leq 4)$. Along the standard location dimension, each child cell $C^l_{2n_l}$ $(1 \leq n_l \leq 5)$ corresponds to one of the five countries, i.e., the United States, Canada, United Kingdom, Malaysia, and New Zealand, and contains the corresponding subset $D^l_{2n_l}$ of training data. We adopted the top two levels of topics of ODP [1] in the TC-dimension, which consists of 16 and 483 topics, respectively.

Figure 11 compares the top-down method and the naïve method on the standard dimensions. As described in Section 6.1.1, in the top-down method, we inherit the model parameters from the parent cell, while in the naïve method, we initialize the model parameters from scratch (i.e., using the initialization method in Section 5.2). Figure 11(a) shows the log data likelihood with respect to the number of EM iterations summed over the four month cells $C^t_{21}, \ldots, C^t_{24}$. We can see the top-down method achieves higher initial likelihoods than that by the naïve method after initialization.



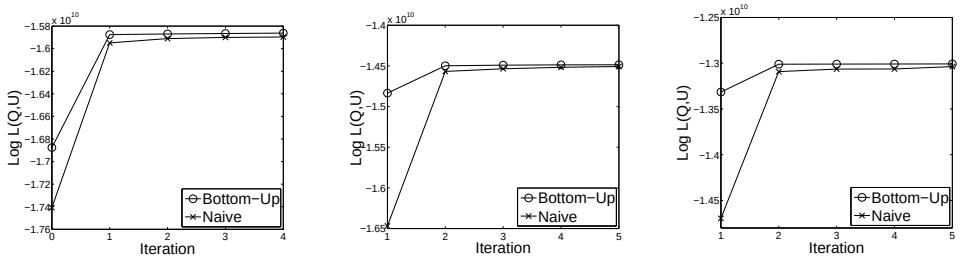
(a) Date (b) Location

Figure 12: The bottom-up method *vs.* the naïve method on the standard dimensions: (a) date and (b) location.

However, both methods needed about five iterations to converge, and thus took similar runtime. Moreover, both methods converged to comparable likelihoods. Therefore, we may choose any of them to materialize the standard time dimension. Figure 11(b) shows the log data likelihood summed over the five country cells $C_{21}^l, \ldots, C_{25}^l$ with respect to the number of EM iterations. This figure illustrates similar trends with those in Figure 11.

Figure 12 compares the bottom-up method and the naïve method on the standard dimensions. As described in Section 6.1.2, in the bottom-up method, we aggregate the model parameters of the child cells to initialize those for the parent cell, while in the naïve method, we initialize the model parameters from scratch. Figures 12 shows the log data likelihoods in the parent cell C_1 with respect to the number of EM iterations. The model parameters were initialized by aggregating those of month cells (Figure 12(a)) and country cells (Figure 12(b)) by the bottom-up method, respectively. The bottom-up method achieves higher initial likelihoods than that by the naïve method after initialization. Again, both methods needed about five iterations to converge, and they converged to comparable likelihoods. Therefore, these two methods have comparable performance. Considering both Figures 11 and 12, we may choose any of the bottom-up, top-down, and naïve methods to materialize the standard dimensions.

Figures 13(a), (b), and (c) compare the bottom-up method and the naïve method on the TC-dimension in (a) the full data cell, (b) the month cells, and (c) the country cells, respectively. As described in Section 6.2, in the bottom-up method, we initialize the model parameters for the parent topic level by aggregating those from the child topic level, while in the naïve method, we initialize the model parameters from scratch. In all the three



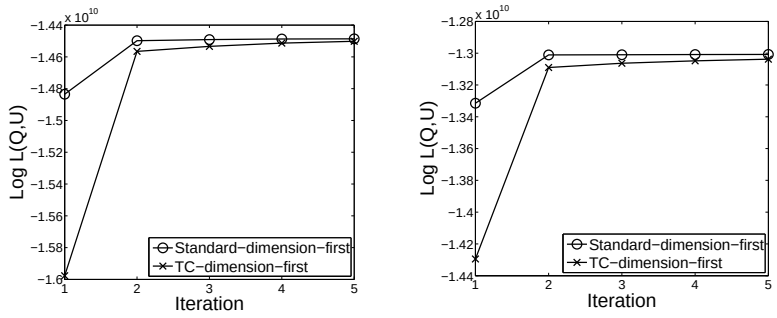
(a) Full data cell (b) Month cells (b) Country cells

Figure 13: The bottom-up method *vs.* the naïve method on the TC-dimension in (a) full data cell, (b) month cells, and (c) country cells.

figures, we observe similar trends. That is, the bottom-up method achieves higher initial data likelihood than that of the naïve method. However, both methods have comparable performance in efficiency and effectiveness.

We also compared the top-down method and the naïve method on the TC-dimension. The top-down method was much slower than the naïve method. The reason is that when we inherit the model parameters from the upper level topics, most of the concept generation probabilities $P(c|t)$ for the lower level topics are nonzero. In this case, the superset of nonzero parameters estimated by Theorem 1 can still be very large. Consequently, each process node needs to partition the assigned training tuples into many blocks and scan the large parameter file many times. Therefore, in the TC-dimension, we may consider either the bottom-up method or the naïve method.

Finally, we compare the two options to materialize the whole TC-cube, i.e., the standard-dimension-first method and the TC-dimension-first method. In our study, the parent cell C_1 consists of the full data set D_1. There are two types of child cells: the four month child cells $C^t_{2n_t}$ $(1 \leq n_t \leq 4)$ and the five country child cells $C^l_{2n_l}$ $(1 \leq n_l \leq 5)$. For each cell, there are two topic levels T_1 and T_2, which are the top two topic levels in the ODP taxonomy [1]. According to the above experimental results, we may choose any of the three options, i.e., the bottom-up, top-down, or naïve method, to materialize along the standard dimensions. Moreover, we may choose either the bottom-up or the naïve method to materialize along the TC-dimension. Therefore, to materialize the whole TC-cube, a possible standard-dimension-first approach consists of four steps: (1) materialize the second topic level T_2 of the parent cell C_1 from scratch (i.e., using the initialization method in Section 5.2); (2) materialize the second topic level T_2 of the child cells C_2 by the top-down method along the standard dimensions; (3) materialize



(a) Month cells (b) Country cells

Figure 14: The standard-dimension-first method *vs.* the TC-dimension-first method in (a) month cells and (b) country cells.

the first topic level T_1 of the parent cell C_1 by the bottom-up method along the TC-dimension; and (4) materialize the first topic level T_1 of the child cells C_2 by the bottom-up method along the TC-dimension. Analogously, a possible TC-dimension-first approach consists of the following four steps: (1) materialize the second topic level T_2 of the parent cell C_1 from scratch; (2) materialize the first topic level T_1 of the parent cell C_1 by the bottom-up method along the TC-dimension; (3) materialize the second topic level T_2 of the child cells C_2 by the top-down method along the standard dimensions; and (4) materialize the first topic level T_1 of the child cells C_2 by the top-down method along the standard dimensions. The only difference between the two approaches is in step (4). Therefore, we only focus on the fourth step of the two approaches.

Figures 14(a) and (b) show the log data likelihoods with respect to the number of EM iterations by the standard-dimension-first approach and the TC-dimension-first approach in month cells and country cells, where the log data likelihoods are the sums over those of the four month cells and the five country cells, respectively. Both figures show similar trend. First, the standard-dimension-first approach has higher initial log data likelihood than that of the TC-dimension-first approach. This suggests the aggregation of the model parameters learned from the same data set is more accurate than those directly inherited from the larger parent data set. Second, both approaches need the same number of iterations to converge, and they converge to comparable likelihoods. Therefore, it does not make much difference to materialize the standard dimensions first or the TC-dimension first.

8 Conclusion

In this paper, we described our topic-concept cube project that supports online multidimensional mining of search logs. We proposed a novel topic-concept model to summarize user interests and developed distributed algorithms to automatically learn the topics and concepts from large-scale log data. We also explored various approaches for efficient materialization of TC-cubes. Finally, we conducted an empirical study on a large log data set and demonstrated the effectiveness and efficiency of our approach. A prototype system that can provide public online services is under development.

References

[1] ODP: `http://www.dmoz.org`.

[2] Wikipedia: `http://en.wikipedia.org`.

[3] Yahoo! Directory: `http://dir.yahoo.com`.

[4] Backstrom, L., et al. Spatial variation in search engine queries. In *WWW'08*, 2008.

[5] Baeza-Yates, R.A., et al. Query recommendation using query logs in search engines. In *EDBT'04 Workshop*, 2004.

[6] Beeferman, D. and Berger, A. Agglomerative clustering of a search engine query log. In *KDD'00*, 2000.

[7] Beitzel, S.M., et al. Hourly analysis of a very large topically categorized web query log. In *SIGIR'04*, 2004.

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

[9] Cao, H., et al. Towards context-aware search by learning a very large variable length hidden markov model from search logs. In *WWW'09*, 2009.

[10] Dean, J., et al. MapReduce: simplified data processing on large clusters. In *OSDI'04*, 2004.

[11] 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.

[12] Grey, J., et al. Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *ICDE'06*, 1996.

[13] Hofmann, T. Probabilistic Latent Semantic Analysis. In *UAI'99*, 1999.

[14] Joachims, T. Text categorization with support vector machines: learning with many relevant features. In *ECML'98*, 1999.

[15] Joachims, T. Transductive inference for text classification using support vector machines. In *ICML'99*, 1999.

[16] Kamvar, M. et al. Computers and iphones and mobile phones, oh my!: a logs-based comparison of search users on different devices. In *WWW'09*, 2009.

[17] Mei, M., et al. Discovering evolutionary theme patterns from text - an exploration of temporal text mining. *KDD'05*, 2005.

[18] Mei, M., et al. A probabilistic approach to spatiotemporal theme pattern mining on weblogs. *KDD'06*, 2006.

[19] Shen, D. et al. Q^2c@ust: our winning solution to query classification in kddcup 2005. *KDD Exploration*, 7(2), 2005.

[20] Wen, J., et al. Clustering user queries of a search engine. In *WWW'01*, 2001.

[21] Zhang, D., et al. Topic cube: Topic modeling for olap on multidimensional text databases. In *SDM'09*, 2009.

[22] Zhao, Q., et al. Event detection from evolution of click-through data. In *KDD'06*, 2006.