Key Technologies for Information Retrieval and Design of High-Performance Retrieval Systems: Postprint

Authors: Yu Xiaoming, Guo Jiafeng, Zhu Xiaofei, Guan Feng, Cheng Xueqi

Date: 2017-03-09T09:01:33+00:00

Abstract

The rapid development of network technologies has led to a dramatic increase in the scale of accessible data, making it challenging to locate required information from massive datasets. Information retrieval technology serves as an effective solution, enabling users to quickly and efficiently find desired information. This paper introduces key retrieval technologies, including index organization, retrieval models, and query analysis, along with FirteX, a high-performance open-source retrieval system developed and maintained by our research group.

Full Text

Key Technologies in Information Retrieval and Design of High-Performance Retrieval Systems

Yu Xiaoming, Guo Jiafeng, Zhu Xiaofei, Guan Feng, Cheng Xueqi

Abstract

The rapid development of network technologies has led to an explosive growth in accessible data, making it increasingly difficult to locate needed information from massive datasets. Information retrieval technology provides an effective solution to this challenge, enabling users to quickly and efficiently find relevant information. This paper introduces key technologies in retrieval systems, including index organization, retrieval models, and query analysis, as well as FirteX—a high-performance open-source retrieval system developed and maintained by our research group.

Keywords: information retrieval, retrieval model, query analysis, learning to rank, FirteX

1 Introduction

With the exponential growth of online information, information retrieval has become an increasingly important means of information access in daily life. Broadly defined, information retrieval encompasses text, image, audio, and video retrieval; in its narrow sense, it refers specifically to text or document retrieval, particularly for unstructured or semi-structured text. The core task is to retrieve text relevant to user needs from a relatively stable text collection. This paper focuses primarily on key technologies related to text retrieval.

Specifically, information retrieval operates by identifying, from a document collection, the subset of documents most closely matching a user's query request. [Figure 1: see original paper] illustrates the general processing flow of an information retrieval system. The system first builds an index for the text collection to improve retrieval efficiency. During retrieval, users submit queries to the system, which then searches using the pre-constructed index and returns documents ranked by relevance according to a specific algorithm.

In information retrieval, a query represents a user's description of their information need—an external manifestation of that need. Documents are the fundamental retrieval objects or granularity units, typically consisting of unstructured free text or semi-structured text described in natural language, such as web pages, news articles, academic publications, product descriptions, or blog posts. A document collection refers to a set of such documents, which may change at varying frequencies depending on document type. For instance, digital libraries might add or remove books every few days, while forum posts, blog articles, or product listings may update within minutes. Based on update speed and pattern, document collections can be categorized as static, incremental, or dynamic.

In web-oriented information retrieval systems, two additional important modules are typically included beyond the general process shown above: information collection and information extraction, both preparatory steps for obtaining the document collection used for retrieval. Information collection involves gathering information from the web, usually accomplished through web crawlers. Information extraction transforms the collected semi-structured data into structured data for indexing.

The research scope of information retrieval is extensive, encompassing information collection, representation, organization, storage, access, and search [1]. In recent years, with the popularity of cloud computing, increasing numbers of researchers have begun designing information retrieval systems based on cloud platforms. While these systems effectively meet large-scale data retrieval needs, their fundamental principles remain largely unchanged from the general process shown in Figure 1, so we will not delve into further details.

This paper introduces key technologies for information retrieval systems from the perspectives of index organization, retrieval models, and query analysis.

Section 2 discusses different index organization methods for various document collections. Section 3 presents commonly used retrieval models and introduces advances in learning-to-rank technology. Section 4 summarizes query analysis techniques. Finally, Section 5 describes the design of FirteX, a high-performance open-source retrieval software developed by our research group.

2 Index Organization

As mentioned in Section 1, building indexes for document collections accelerates retrieval tasks [1]. Common index structures include signature files, bitmap files, and inverted indexes, with inverted indexes being the most prevalent form in information retrieval.

A signature file is a text index structure based on probabilistic methods [2], representing each document with a signature consisting of a series of bit masks that partially describe document content. A bitmap is a simple index structure where each term in the dictionary is represented by a bit vector, with each bit corresponding to a document (set to 1 if the term appears in that document, 0 otherwise). An inverted index is a word-oriented indexing mechanism consisting of a vocabulary and a posting list collection. Each element in the vocabulary is called a term or index item, while the posting list collection contains inverted lists (also called posting chains), with each term corresponding to one such list. An inverted list typically includes the document IDs where the term appears, term frequency, and position information. In essence, while the original document representation shows document-term relationships, inverted indexing reverses this to term-document relationships—hence the name "inverted." [Figure 2: see original paper] presents a simplified example of an inverted index.

Inverted index construction requires substantial CPU, memory, and disk resources. To improve retrieval efficiency and reduce disk operations, an inverted list is typically stored in contiguous disk space. In practice, document collections are often too large for their indexes to fit entirely in memory. Therefore, we must leverage disk space to build indexes for large-scale collections. Below, we discuss inverted index construction methods for different types of document collections.

2.1 Indexing for Static Document Collections

In static collections where documents do not change, index construction can be completed in one or multiple passes. Three primary methods exist for building inverted indexes for large-scale static collections using disk space: sort-based construction, multi-pass construction, and single-pass in-memory construction.

Witten et al. [2] view inverted index construction as essentially a sorting process. Raw documents can be represented as units of triples $\langle t, d, f \rangle$, where t denotes a term, d a document ID, and f the term frequency in document d. If term positions are considered, each triple corresponds to a set of term position sequences $\langle t, d, p \rangle, \ldots$. The entire document collection can thus be viewed

as multiple ordered triples $\langle t, d, f \rangle, \ldots$. Constructing an inverted index simply requires sorting these triples by t, and for identical t, by d.

The multi-pass method was initially proposed by Fox and Lee [3] and later improved by Witten et al. [2]. This approach builds inverted indexes through multiple scans of the document collection. The first scan generates the complete dictionary and collects detailed statistics about each term. After one full pass, the system has both the complete dictionary in memory and accurate knowledge of each term's posting list length, enabling pre-allocation of disk storage space for each list. Subsequent scans fill these pre-allocated spaces directly. This method ensures that each term's posting list is stored contiguously on disk.

In single-pass in-memory construction, each term maintains a linked list or dynamically growing array to record all $\langle t, d, f \rangle$ triples. During processing, posting lists grow continuously. When memory is exhausted, all terms and their posting lists in memory are written to disk as temporary sub-indexes (called "runs"), memory is freed, and the process continues for the next batch of documents. After all documents are processed, multiple temporary sub-indexes exist on disk, which are then merged using multi-way merging to form the final inverted index. This method was first proposed by Heinz and Zobel [4]. Studies in [2], [4], and [5] compare these methods and conclude that single-pass in-memory construction is the most efficient and highly scalable, capable of handling datasets of any size.

2.2 Indexing for Dynamic Document Collections

Dynamic document collections undergo additions, deletions, and modifications over time. To ensure correct retrieval results, indexes must be updated accordingly. Since updates typically occur while providing retrieval services, update strategies must balance indexing efficiency with retrieval efficiency. The most straightforward update method is to discard the old index and rebuild it from scratch by rescanning the document collection—this is called index reconstruction. Other common methods include in-place index updates and merge-based updates.

The in-place update approach updates only the posting lists that have changed when documents are modified, leaving unchanged lists intact. In practice, index data for documents first accumulates in memory. When memory is insufficient, all posting lists in memory are updated to their corresponding disk locations. In-place updates require careful space management for posting lists. Accumulating updates in memory significantly reduces disk update frequency. This method performs well with few document updates but degrades sharply as data scales grow and update frequencies increase [6].

Merge-based update methods first build in-memory inverted indexes for documents. When memory is exhausted, they sequentially read each term's posting list from disk, merge it with the corresponding in-memory list, and write the result to a new disk location. After merging all posting lists, a new inverted

index is formed on disk. This approach reads all disk-based posting lists regardless of whether they were updated, but the resulting new index stores all posting lists contiguously in term order. During query processing, a single disk seek and read operation retrieves a term's complete posting list, maximizing query performance. Merge-based updates can be categorized by merge strategy and the maximum number of allowed disk sub-indexes: immediate merge (allowing only one sub-index), no-merge (allowing multiple sub-indexes), and intermediate strategies like logarithmic merge and geometric partitioning.

(1) Immediate Merge Immediate merge, also called re-merge, combines the in-memory sub-index with the disk sub-index immediately when the memory index needs to be migrated to disk, generating a new index to replace the old one. This approach maintains only one sub-index on disk at all times, with each term's posting list stored sequentially and contiguously, thus maximizing query processing performance. However, each merge operation must read and process the entire disk sub-index, regardless of which posting lists were actually updated, making index construction costly. While this might seem inefficient, Lester et al. [6][7] experimentally compared immediate merge with in-place updates and concluded that in many practical scenarios, immediate merge actually outperforms in-place updates.

(2) No-Merge Index Update In the no-merge approach, when memory is exhausted, the in-memory inverted index is directly written to disk as a new sub-index without any merging, maintaining multiple sub-indexes on disk. During retrieval, a term's posting list is obtained by reading the corresponding lists from all disk sub-indexes. This method requires only one write operation per disk sub-index, optimizing index update performance. However, a term's posting list may be distributed across multiple sub-indexes, requiring multiple disk seeks and reads during query processing and negatively impacting retrieval performance.

(3) Logarithmic Index Update Immediate merge and no-merge represent two extremes in online index updates, each optimizing only one aspect of performance. Dynamic text indexing often requires balancing both query processing and index update performance. Büttcher [8] proposed a logarithmic merge method introducing the concept of index "generations." A sub-index migrated from memory to disk is called generation 0; one merged from generation 0 sub-indexes is generation 1, and so forth. No two sub-indexes of the same generation may coexist—if they do, a merge event is triggered to create a new generation sub-index. If this new sub-index again creates two sub-indexes of the same generation, another merge is triggered, and these two merges can be combined into one operation. Analysis in [8] examines the trade-offs between update and query processing costs for logarithmic index updates.

(4) Geometric Partitioning Index Update Lester proposed a similar strategy called geometric partitioning [9]. The basic idea divides the dataset to be indexed into multiple controllable partitions following a geometric progression. Assuming memory can hold b documents and introducing parameter r, each level of partition follows this rule: a level-k partition contains at most $r^k b$ documents. At level k, a partition is either empty or contains at least $r^{k-1}b$ documents. Adjusting parameter r can balance index update and query processing performance. Like logarithmic merge, geometric partitioning avoids overly frequent merges while preventing too many sub-indexes on disk, ensuring both reasonable update performance and acceptable query performance. Detailed analysis can be found in [9].

(5) Dynamic Balancing Tree-Based Index Update Guo Ruijie [10][33] proposed a dynamic balancing tree-based update strategy. A dynamic balancing tree is an m-ary tree where the size of nodes at level $k+1$ is approximately c times that of nodes at level k (m and c are algorithm parameters). A tree satisfying this condition for all nodes is called balanced. In this algorithm, newly added sub-indexes are placed at the appropriate tree level based on size. If adding a new index causes some nodes to violate the balancing condition, the tree is rebalanced by merging sub-indexes. Merge operations occur primarily between nodes at the same level, ensuring that participating sub-indexes have similar sizes and thus maintaining merge efficiency. Experimental analysis in [10] demonstrates that this algorithm achieves better performance than the aforementioned merge algorithms.

3 Retrieval Models

Document ranking is the core task in information retrieval. "Ranking" involves scoring each document in a collection for a given query using a retrieval model, then sorting documents by score from highest to lowest, where the order represents relevance between documents and the query. Ranking methods originated from traditional text retrieval approaches such as the Vector Space Model, probabilistic models (e.g., Okapi BM25), and Language Models. Link analysis (e.g., PageRank, HITS [12][13]) dramatically improved web retrieval effectiveness. In recent years, Learning to Rank has become a hot research topic.

3.1 Traditional Ranking Models

Representative traditional methods include the Vector Space Model, probabilistic models (Okapi BM25), Language Models, and link analysis models (PageRank, HITS).

3.1.1 Vector Space Model The Vector Space Model represents documents as term vectors and determines relevance by computing vector similarity. During retrieval, both queries and documents are converted to term vectors, and

similarity (e.g., cosine similarity) is calculated to measure query-document relevance. Term weights are typically computed using TF-IDF, where TF is term frequency in the document and IDF is inverse document frequency. The model's advantage is computational simplicity; its disadvantage is reliance on empirical formulas lacking theoretical validation.

3.1.2 Probabilistic Models In probabilistic models, let R represent relevance between document D and query Q, taking values r (relevant) or $\bar{r}$ (non-relevant). The goal is to estimate $P(R = r|D, Q)$. Relevance is typically computed using the ratio of relevant to non-relevant probabilities: $\frac{P(R=r|D,Q)}{P(R=\bar{r}|D,Q)}$. Okapi BM25 is a famous probabilistic model with similarity formula:

$$\text{Score}(d, q) = \sum_{w \in q} \frac{(k_1 + 1)c(w, d)}{k_1((1 - b) + b\frac{|d|}{\text{avdl}}) + c(w, d)} \cdot \frac{(k_3 + 1)c(w, q)}{k_3 + c(w, q)} \cdot \log \frac{N - \text{df}(w) + 0.5}{\text{df}(w) + 0.5}$$

where d, q denote document and query, $|d|$ is document length, avdl is average document length, w is a feature term, $c(w, d)$ and $c(w, q)$ are term frequencies in document and query, N is total documents, $\text{df}(w)$ is document frequency, and k_1, k_3, b are manually tuned parameters. Probabilistic models have solid mathematical foundations but cannot handle long-distance dependencies in language.

3.1.3 Language Models Language models treat relevance as the probability of generating the query under the document's language model. For a document d with word sequence $w_1, w_2, ...$, a statistical language model computes $P(w_1, w_2, ...)$. By Bayes' theorem:

$$P(d|w_1, w_2, ...) = \frac{P(d)P(w_1, w_2, ... |d)}{P(w_1, w_2, ...)} \propto P(w_1, w_2, ... |d)$$

Typically using unigram models that assume word independence, $P(w_1, w_2, ... |d) = \prod_i P(w_i|d)$. For a given query, we compute each document's query generation probability $P(q|d)$ as the similarity measure. Due to data sparsity causing zero probabilities, smoothing techniques assign non-zero probabilities to unobserved terms. Common methods include Jelinek-Mercer smoothing, Dirichlet smoothing, and Absolute Discount smoothing, which reduces observed term probabilities by subtracting a constant.

3.1.4 Link Analysis Models (PageRank, HITS) Link analysis models propagate weights between web pages to determine importance, with more important pages ranking higher. Representative methods include PageRank and HITS. The difference lies in weight propagation: PageRank transfers authority directly between authoritative pages, while HITS propagates authority through hub pages.

3.2 Learning to Rank

As information retrieval research progressed, researchers realized numerous factors affect ranking, and text/link-only approaches have significant limitations. Treating these factors as features and using machine learning to learn an optimal ranking function can substantially improve effectiveness—this is the fundamental problem Learning to Rank addresses. Current trends evolve from pointwise to pairwise to listwise approaches, and from query-independent to query-dependent methods.

3.2.1 Pointwise Methods Pointwise methods treat query-document pairs as training samples, with relevance degrees as target scores or classes, transforming ranking into regression or classification. Representative methods include PRank [14] and MCRank [15].

PRank (Perceptron Rank) maps training samples to real values, where each relevance grade corresponds to a real-valued interval, enabling straightforward grade assignment. PRank minimizes ranking loss by adjusting a weight vector. Theoretically, it is order-preserving with a mistake bound, ensuring correctness and convergence.

MCRank treats ranking as a multi-classification problem, using ensemble classification results for final ranking. It classifies query-document pairs into K categories $\{1, \ldots, K\}$, then sorts by category. Samples in the same category can be arbitrarily ordered, causing instability. To avoid this, soft classification can produce category distributions, and expected relevance scores can rank samples.

3.2.2 Pairwise Methods Pairwise methods construct training samples from relative relevance relationships between document pairs. Each sample comprises a document pair; if the first document is more relevant than the second, the label is positive, otherwise negative. This enables classification of new pairs to determine relative relevance. Representative methods include RankNet [16], RankBoost [17], and Ranking SVM [18].

Burges and Shaked et al. proposed RankNet using a simple probabilistic cost function modeled by neural networks and optimized via gradient descent. Let $x \in \mathbb{R}^n$ be a feature vector and $f : \mathbb{R}^n \to \mathbb{R}$ map samples to real-valued scores. Let P_{ij} represent the probability that x_i ranks before x_j, modeled as $P_{ij} = \frac{\exp(o_i - o_j)}{1+\exp(o_i - o_j)}$ where $o_i = f(x_i)$. RankNet uses cross-entropy loss: $L = -\sum_{(i,j)} \tilde{P}_{ij} \log P_{ij} + (1 - \tilde{P}_{ij}) \log(1 - P_{ij})$, measuring divergence between predicted and target probabilities.

Freund et al. proposed RankBoost, derived from AdaBoost. It combines multiple weak rankers h_t into a final ranking function $f(x) = \sum_t \alpha_t h_t(x)$. To capture pairwise relative relevance, they define a feedback function $\Phi : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ where $\Phi(x_i, x_j) > 0$ indicates x_i should rank before x_j, $\Phi(x_i, x_j) < 0$ indicates the opposite, and $\Phi(x_i, x_j) = 0$ indicates no ordering. The magnitude $|\Phi(x_i, x_j)|$

represents importance. The optimization minimizes the number (or weighted sum) of misordered pairs.

Ranking SVM transforms ranking into binary classification solved by support vector machines. The process has two steps: (1) Use a ranking function $f(x) = \mathbf{w}^T\mathbf{x}$ (typically linear) to map samples to real values, defining pairwise relationships; (2) Solve the resulting binary classification problem via SVM optimization:

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_k \xi_k \quad \text{s.t.} \quad z_k(\mathbf{w}^T\mathbf{x}_i - \mathbf{w}^T\mathbf{x}_j) \geq 1 - \xi_k, \xi_k \geq 0$$

where the k-th document pair consists of documents i and j, l is the number of pairs, and z_k is the true label for pair k.

3.2.3 Listwise Methods Listwise methods use entire ranked lists as training samples, where some relationships are inherently encoded. These methods generally fall into two categories: (1) Loss function minimization, differing primarily in the loss function used—e.g., RankCosine uses cosine similarity between score lists, ListNet uses Kullback-Leibler divergence, and ListMLE uses negative log-likelihood; (2) Direct optimization of IR metrics (MAP, NDCG), such as AdaRank, SVM-MAP, SoftRank, and LambdaRank.

Listwise methods consider all documents for a query and can model relationships like similarity, enabling more effective ranking functions. By focusing on entire lists, they leverage position information to emphasize top-ranked documents.

4 Query Analysis

Queries serve as the primary means for users to express information needs, acting as a bridge between users and retrieval systems. Ideally, systems would correctly understand queries and return appropriate results. However, just as natural language understanding remains challenging for computers, understanding queries—users' interactive language—poses significant difficulties. Recent research has increasingly focused on query analysis and understanding.

4.1 Query Optimization

Query optimization automatically handles common query issues like spelling errors, improper word forms, missing operators (e.g., quotes), and inappropriate abbreviations to improve retrieval performance.

Spelling errors, common in text, also appear in queries. Since queries are typically short with limited context, correcting spelling errors presents new challenges. Cucerzan et al. [19] proposed using query logs for spelling correction, leveraging the implicit and explicit linguistic information in large query logs to iteratively refine queries via a noise-channel model. Li et al. [20] improved this

using distributional similarity, noting that correct words and their misspellings have high distributional similarity while unrelated words have low similarity—making it effective for correction. Guo et al. [21] modeled query optimization as structured prediction, proposing CRF-QR (Conditional Random Fields for Query Refinement), a unified discriminative model that mines query logs to correct various errors, overcoming previous methods' limitation of focusing on single optimization tasks.

4.2 Query Redundancy Analysis

User queries typically contain only 2-3 keywords, though average length is gradually increasing. Longer queries provide richer context but also contain noise (redundant/irrelevant information), reducing search effectiveness. Automatically creating more concise queries from long ones has become a focus for search engines. Redundancy removal techniques delete irrelevant information to improve performance.

Kumaran and Allan [22] proposed using mutual information between query terms to rank sub-queries. They construct a fully connected graph based on term mutual information from a corpus, then use a maximum spanning tree algorithm to compute and rank sub-query information entropy. Their subsequent work [23] combines query expansion and deletion, suggesting that selectively applying either based on the query can greatly improve performance.

4.3 Query Parsing

Queries contain concepts and knowledge. Parsing queries to identify key concepts and named entities enables better understanding and improves retrieval intelligence.

Bendersky and Croft [24] used machine learning with query-dependent, corpus-dependent, and corpus-independent features to identify key concepts in long queries, employing an AdaBoost.M1 meta-learner with C4.5 base classifiers. A probabilistic model integrates identified concepts into the ranking model. Guo et al. [25] studied named entity recognition in queries, modeling it as an optimal triple extraction problem using a probabilistic model combined with large-scale query log mining.

4.4 Query Expansion

Traditional methods include global techniques based on thesauri, local techniques based on (pseudo) relevance feedback, and hybrid approaches. Cui et al. [26] used query logs for expansion, modeling probabilistic relationships between query and document terms from session information to select high-quality expansion terms. Further work [27] explored effectiveness for different query types, showing significant improvements over traditional thesaurus and pseudo-relevance feedback methods.

4.5 Query Classification

Query classification includes intent and topic classification. Research has evolved from rule-based to machine learning methods using query logs.

Broder [28] categorized web search intent into three types: navigational, informational, and transactional—a significant but coarse-grained classification. Shen et al. [29] proposed a probabilistic framework using an intermediate taxonomy as a bridge to map queries to target categories automatically.

4.6 Query Recommendation

Query recommendation assists users in expressing intent and improves search experience. Early work focused on text content, but modern approaches rely heavily on query log data.

Fonseca et al. [30] mined association rules from query logs for recommendation. Zhang and Nasraoui [31] mined query sequence information, modeling user search behavior to describe similarity between queries in the same session while computing content-based similarity using cosine distance. Huang et al. [32] considered contextual information in query sessions, arguing that recommendations should be relevant to the entire session, not just the most recent query.

5 FirteX System Design

FirteX is a large-scale text online indexing and retrieval platform we developed. Its main distinction from other experimental systems is a dynamic text online indexing and retrieval framework that enables researchers to easily implement incremental and dynamic text indexing and retrieval. It currently includes several index update strategies such as dynamic balancing tree. FirteX supports retrieval models, query feedback, natural language processing, and facilitates implementation of index update algorithms, query processing algorithms, distributed IR, query caching, and memory management, supporting terabyte-scale data retrieval. [Figure 3: see original paper] shows the FirteX system architecture.

6 Conclusion

As information technology advances, data scales grow larger and require faster response to changes. Challenges in information retrieval include retrieving more, faster, and more accurately, while providing more intelligent techniques. This paper introduced key technologies including index organization, retrieval models, and query analysis. In response to these trends, we developed the efficient open-source FirteX system with online indexing support. Future work will further improve data scale support and retrieval models.

References

[1] Baeza-Yates, Ribiero-Neto. 1999. Modern Information Retrieval. New York: ACM Press.

[2] Witten, I. H., Moffat, A. & Bell, T. C., Managing Gigabytes: Compressing and Indexing Documents and Images, second edn, Morgan Kaufmann, San Francisco, California, 1999.

[3] E. Fox and W. Lee. FAST-INV: A fast algorithm for building large inverted les. Technical Report TR 91-10, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1991.

[4] S. Heinz and J. Zobel. Efficient single-pass index construction for text database. Jour. Of the American Society for Information Science and Technology, 54(8):731-729, 2003.

[5] A. Moffat and T. A. Bell. In situ generation of compressed inverted files. Jour. of the American Society for Information Science, 46(7):537-550, 1995.

[6] Nicholas Lester. Efficient Index Maintenance for Text Database. PhD thesis, Department of Computer Science and Information Technology, RMIT, 2006.

[7] N. Lester, J. Zobel, and H.E. Williams, Efficient online index maintenance for text retrieval systems, Information Processing & Management, 42(4):916-933, July 2006.

[8] S. Büttcher and C. L. A. Clarke. Indexing time vs. query time trade-offs in dynamic information retrieval systems. In Proceedings of the 14th ACM Conference on Information and Knowledge Management (CIKM 2005). Bremen, Germany, November 2005.

[9] N. Lester, A. Moffat, and J. Zobel. Fast on-line index construction by geometric partitioning. In Proceedings of the ACM CIKM Conference on Information and Knowledge Management, pages 776-783, November 2005.

[10] Ruijie Guo, Xueqi Cheng, Hongbo Xu, Bin Wang. "Efficient On-line Index Maintenance for Dynamic Text Collections by Using Dynamic Balancing Tree". In Proceedings of the 16th ACM Conference on Information and Knowledge Management (CIKM 2007). Lisbon, Portugal, November, 2007.

[11] S. Robertson. Overview of the okapi projects. In Journal of Documentation, pages 275–281, 1998.

[12] Page, L., Brin, S., Motwani, R., Winograd, T.: The PageRank Citation Ranking: Bringing Order to the Web. Stanford InfoLab (1999)

[13] Kleinberg, Jon (1999). Authoritative sources in a hyperlinked environment. Journal of the ACM 46 (5): 604–632.

[14] K. Crammer and Y. Singer. Pranking with ranking. Advances on Neural Information Processing System, Pages 641-647, MIT Press, 2001.

[15] P. Li, C. Burges, et al. McRank: Learning to Rank Using Classification and Gradient Boosting, NIPS 2007.

[16] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank Using Gradient Descent. In ICML, pages 89-96, 2005.

[17] Y. Freund, R. Iyer, R.E. Schapire and Y. Singer. An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research, 4:933–969, 2003.

[18] R. Herbrich, Thore Graepel, and Klaus Obermayer, Large Margin Rank Boundaries for Ordinal Regression, MIT Press, Cambridge, MA, 2000.

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

[20] M. Li, M. Zhu, Y. Zhang, and M. Zhou. Exploring distributional similarity based models for query spelling correction. In Proceedings of COLING-ACL 2006, pages:1025-1032, 2006.

[21] J. Guo, G. Xu, H. Li, and X. Cheng. A Unified and Discriminative Model for Query Refinement. In SIGIR '08, pages 379–386, New York, NY, USA, 2008. ACM.

[22] G. Kumaran and J. Allan. A case for shorter queries, and helping users create them. In HLT-EMNLP Conference Proceedings, pages 220–227, Rochester, NY, 2007.

[23] Kumaran, G. and Allan, J. 2008. Effective and efficient user interaction for long queries. In SIGIR '08. ACM, New York, NY, pages:11-18.

[24] Bendersky, M. and Croft, W. B. 2008. Discovering key concepts in verbose queries. In SIGIR '08. ACM, New York, NY, pages:491-498.

[25] J. Guo, G. Xu, X. Cheng, and H. Li. Named Entity Recognition in Query. In SIGIR '09, pages 267–274, Boston, MA, USA, 2009. ACM.

[26] H, Cui, J.R. Wen, J.Y. Nie and W.Y. Ma. Probabilistic Query Expansion Using Query Log. In Proceedings of the 11th international conference on World Wide Web 2002, Honolulu, Hawaii, USA, May 07-11, 2002.

[27] Hang Cui, Ji-Rong Wen, and Jian-Yun Nie. Query expansion by mining user logs. IEEE Trans. on Knowl. and Data Eng., 15(4):829-839, 2003. Member-Ma„ Wei-Ying.

[28] Broder, A. A Taxonomy of Web Search. SIGIR Forum 36(2), 2002.

[29] Dou Shen, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Building Bridges for Web Query Classification. In SIGIR '06, August 6-11, 2006, Seattle, Washington, USA.

[30] Bruno M. Fonseca, Paulo B. Golgher, Edleno S. de Moura, and Nivio Ziviani. Using association rules to discover search engines related queries. In LA-WEB '03, page 66, Washington, DC, USA, 2003. IEEE Computer Society.

[31] Zhiyong Zhang and Olfa Nasraoui. Mining Search Engine Query Logs for Query Recommendation. In WWW'06, Edinburgh, Scotland May 23 - 26, 2006

[32] Chien-Kang Huang, Lee-Feng Chien, and Yen-Jen Oyang. Relevant term suggestion in interactive web search based on contextual information in query session logs. J. Am. Soc. Inf. Sci. Technol., 54(7):638-649, 2003.

[33] 郭瑞杰. 大规模动态文本在线索引技术研究 (博士论文),2008.

Authors:

Yu Xiaoming: Institute of Computing Technology, Chinese Academy of Sciences, Assistant Researcher, Email: yuxiaoming@software.ict.ac.cn

Guo Jiafeng: Institute of Computing Technology, Chinese Academy of Sciences, Assistant Researcher

Zhu Xiaofei: Institute of Computing Technology, Chinese Academy of Sciences, PhD Student

Guan Feng: Institute of Computing Technology, Chinese Academy of Sciences, Assistant Researcher

Cheng Xueqi: Institute of Computing Technology, Chinese Academy of Sciences, Professor

Note: Figure translations are in progress. See original paper for figures.