World Wide Web (2013) 16:1–29
DOI 10.1007/s11280-011-0154-0

A novel two-box search paradigm for query disambiguation

David C. Anastasiu · Byron J. Gao · Xing Jiang · George Karypis

Received: 12 May 2011 / Revised: 2 December 2011 /
Accepted: 7 December 2011 / Published online: 28 December 2011

Abstract Precision-oriented search results such as those typically returned by the major search engines are vulnerable to issues of polysemy. When the same term refers to different things, the dominant sense is preferred in the rankings of search results. In this paper, we propose a novel two-box technique in the context of Web search that utilizes contextual terms provided by users for query disambiguation, making it possible to prefer other senses without altering the original query. A prototype system, Bobo, has been implemented. In Bobo, contextual terms are used to capture domain knowledge from users, help estimate relevance of search results, and route them towards a user-intended domain. A vast advantage of Bobo is that a wide range of domain knowledge can be effectively utilized, where helpful contextual terms do not even need to co-occur with query terms on any page. We have extensively evaluated the performance of Bobo on benchmark datasets that demonstrates the utility and effectiveness of our approach.

Keywords two-box search · query disambiguation · domain knowledge · web search

A preliminary version of this paper was published in the Proceedings of the 23rd International Conference on Computational Linguistics (COLING'10) [12].

D. C. Anastasiu · B. J. Gao (✉)
Texas State University-San Marcos, 601 University Drive, San Marcos, TX 78666, USA
e-mail: bgao@txstate.edu

D. C. Anastasiu
e-mail: da1143@txstate.edu

X. Jiang
Nanyang Technological University, 50 Nanyang Avenue,
Singapore 639798, Singapore
e-mail: jian0008@ntu.edu.sg

G. Karypis
University of Minnesota, 4-192 EE/CS Building, 200 Union Street SE,
Minneapolis, MN 55455, USA
e-mail: karypis@cs.umn.edu

1 Introduction

World Wide Web and search engines have become an indispensable part of everyone's everyday life. While Web search has come a long way over the past 10 years, it still has a long way to go to respond to the ever-increasing size of the Web and needs of Web surfers. Today, Web search is under intensive and active research, drawing unparalleled attention from both industry and academia.

Need of disambiguation One of the major challenges in Web search lies in unsatisfactory relevance of results caused by ambiguity. Query terms are inherently ambiguous due to polysemy, and most queries are short, containing 1 to 3 terms only [17]. Thus queries are in general prone to ambiguity of user intent or information needs, resulting in retrieval of many irrelevant pages. As the Web size grows at an increasing rate, ambiguity becomes ubiquitous and users are in greater need of effective means of disambiguation. The ambiguity issue and its consequences are demonstrated in Example 1.

Example 1 There are 17 entries in Wikipedia for different renown individuals under the same name of "Jim Gray", including a computer scientist, a sportscaster, a zoologist, a politician, a film director, a cricketer, and so on. Suppose we intend to find information about Jim Gray", the Turing award winner, we can issue a query of "Jim Gray" in Yahoo!.[1] For this extremely famous name in computer science, only 3 are relevant in the top 10 results. They are his Wikipedia entry, homepage at Microsoft Research, and DBLP entry.

Straightforward query refinement approach One intuitive way of disambiguation would be to apply available domain knowledge and refine the query by adding some confining contextual terms. This would generally improve precision and work well for navigational queries (queries that seek a single website or Web page of a single entity), but is problematic for informational queries (queries that cover a broad topic for which there may be thousands of relevant results), which account for the majority of Web queries [4, 31].

There are several inevitable problems in this approach. First, the improvement on precision is at the *sacrifice of recall*. For example, many Jim Gray" pages may not contain the added contextual terms and are thus excluded from the search results.

Second, the query is altered, leading to unfavorable ranking of results. Term proximity matters significantly in ranking [26]. Some good pages w.r.t. the original query may be ranked low in the new search results because of worsened term proximity and relevance w.r.t. the new query. Thus, with this straightforward approach only limited success can be expected at best, as demonstrated in Example 2.

Example 2 Suppose we know that Jim Gray" is a computer scientist, we can issue a query of "Jim Gray computer". All the top 10 results are about Jim Gray" and relevant. However, many of them are trivial pages, failing to include 2 of the 3 most important ones. His DBLP entry appears as the 27th result, and his homepage at Microsoft Research appears as the 51st result.

[1] Other choices of search engine in the examples would not change the validity of the observations. Also note that search results and their ranks returned by search engines may change over time.

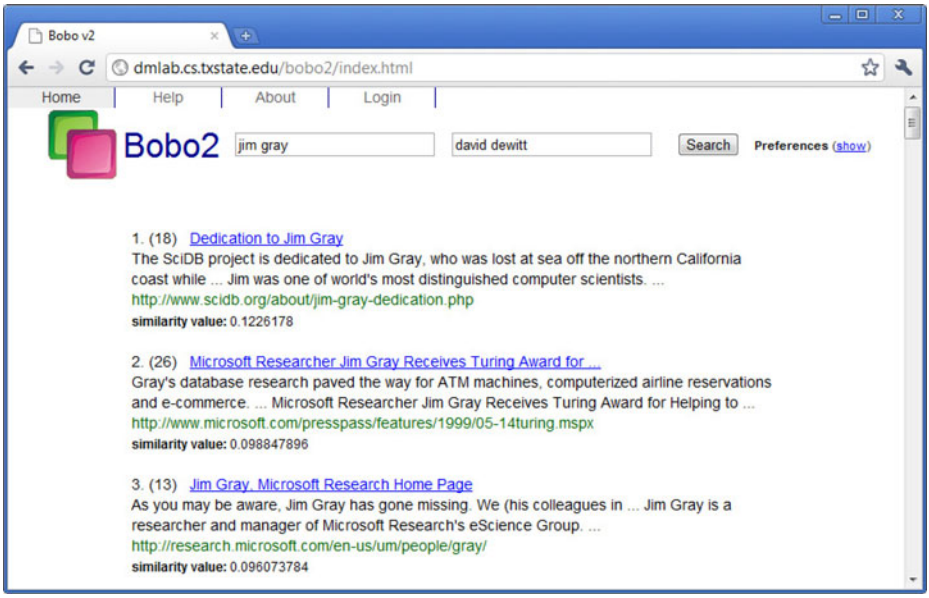

Figure 1 Snapshot of Bobo.

This limited success is achieved by using a carefully selected contextual term. "Computer" is a very general term appearing on most of the Jim Gray" pages. Also, there are no other competitively known computer people with the same name. Most other contextual terms would perform much worse. Thus a third problem of this straightforward query refinement approach is that only few contextual terms, which may not be available to users, would possibly achieve the limited success. Often, much of our domain knowledge would cause more damage than repair and is practically unusable, as demonstrated in Example 3.

Example 3 Suppose we know that Jim Gray" has David DeWitt as a close friend and colleague, we can issue a query of "Jim Gray David DeWitt". Again, all the top 10 results are about Jim Gray" and relevant. However, the theme of the query is almost completely altered. Evidently, the first result "Database Pioneer Joins Microsoft to Start New Database Research Lab", among many others, talks about David DeWitt. It is relevant to Jim Gray" only because the lab is named "Jim Gray Systems Lab" in his honor.

The Bobo approach Can we freely apply our domain knowledge to effectively disambiguate search intent and improve relevance of results without altering the original query? For this purpose, we propose and implement Bobo, a two-box search paradigm.

Figure 1 shows a snapshot of Bobo. For conceptual clarity, the default Bobo interface features two boxes. Besides a regular query box, an additional box is used to take *contextual terms* from users that capture helpful domain knowledge.

Contextual terms are used for disambiguation purposes. They do not alter the original query defined by query terms. Particularly, unlike in the straightforward approach, positive contextual terms are not required to be included in search results

and negative contextual terms are not required to be excluded from search results. Contextual terms help estimate relevance of search results, routing them towards a user intended domain, filtering out those not-in-domain, or irrelevant, results.

Bobo works in two rounds. In round I, a query is issued using by default the combination of query terms and contextual terms, or just the contextual terms if the query returns too few results. Then from the results, some top-ranked high-quality pages are (automatically) selected as *seeds*. In round II, a query is issued using the query terms. Then the results are compared with the seeds and their similarities are computed. The similarity values reflect the degree of relevance of search results to the user intent, based on which the results are re-ranked.

Example 4 reports the Bobo experiment using the same contextual terms as in Example 3.

Example 4 As in Example 3, suppose we know Jim Gray" has David DeWitt as a colleague. Then with Bobo, we can enter "Jim Gray" in the query box and "David DeWitt" in the auxiliary box. As a result with default preferences, all the top 10 results are relevant including all the top 3 important Jim Gray" pages. From the top 10, only 1 page, the DBLP entry, contains "David DeWitt" as they coauthored papers. The theme of the query is not altered whereas in Example 3, all the top 10 results contain "David DeWitt".

In Example 4, the selected seeds are relevant to Jim Gray". Observe that seeds can be useful if they are relevant to the user-intended domain, not only the user-intended query. Bobo works effectively with such seeds and thus can utilize a much expanded range of domain knowledge. Helpful contextual terms do not even need to co-occur with query terms on any page. They only need to occur, possibly separately, on some pages of the same domain, as demonstrated in Example 5.

Example 5 Using the criteria of being in the same community as Jim Gray" but co-occuring on no web pages, we randomly chose a student name, Flavia Moser. In Bobo, we entered "Jim Gray" in the query box, "Flavia Moser" in the auxiliary box, and used only the contextual terms for the round I query. As a result, 11 of the top 12 results were relevant including all the top 3 important Jim Gray" pages. Of course, none of the returned pages contain "Flavia Moser".

Note that, although we introduce the two-box paradigm in the context of Web search, the same idea can be applied to other information retrieval systems such as traditional archival information retrieval and multimedia information retrieval.

2 Related work

Disambiguating search intent, capturing information needs and improving search performance have been a fundamental research objective in information retrieval and have been studied from different perspectives. In this section we review related work in these areas.

Text classification and routing [41] shows that disambiguation cannot be easily resolved using thesauruses. The filtering problem [2, 35] views disambiguation as a binary text classification task assigning documents into one of the two categories,

relevant and irrelevant. The routing problem [36, 38] differs from text classification in that search results need to be ranked instead of just classified [14].

Contextual and personalized search Contextual search [11, 22, 23, 28], personalized search [16, 18, 39, 46], and implicit relevance feedback [19, 21, 43] generally utilize long-term or short-term [6] search history to build explicit or implicit user profiles. These profiles are used on a regular basis to guide *many* queries. Such approaches entail little extra user involvement in search and can be effective in some settings. However, building effective user profiles is a time-consuming process. There are also profile management costs and privacy issues. More importantly, these techniques are inflexible in context switching. In many cases, users may intend to fulfill occasional information needs that differ from their usual preferences stored in the profiles.

Relevance feedback Explicit and pseudo relevance feedback (RF) techniques [2, 26, 32, 45] are more related to our two-box approach in the sense that they do not build long-term profiles. Instead, they construct a one-time search context that is used only once to guide a *single* query each time. Such approaches enjoy the flexibility of being able to switch spontaneously from one domain to another in response to different information needs.

RF is regarded as the most popular query reformation strategy [2]. It iterates in multiple rounds, typically two, to modify a query step by step. Explicit RF asks explicit feedback from users, whereas pseudo (or blind) RF assumes relevance of top-ranked results. The problem of explicit RF is that it requires too much user involvement. Users are often reluctant to provide explicit feedback, or do not wish to prolong the search interaction. Web search engines of today do not provide this facility. Excite.com initially included but dropped it due to the lack of use [26].

Pseudo RF, first suggested by Croft and Harper [8] and since widely investigated, automates the manual part of RF, so that users get improved search performance without extended interactions. Psuedo RF has been found to improve performance in the TREC ad hoc task and Cornell SMART system at TREC-4 [5]. Unfortunately, pseudo RF suffers from a major flaw, the so-called *query drift* problem. Query drift occurs when the feedback documents contain few or no relevant ones. In this case, search results will be routed farther away from the search intent, resulting in even worse performance. Different approaches [24, 25, 27, 44] have been proposed to alleviate query drift but with little success. Some queries will be improved, others will be harmed [32].

Similarly to RF, two-box search works in two rounds. Similarly to pseudo RF, it uses top-ranked round I results as seeds (pseudo feedbacks). However, two-box search and RF differ fundamentally in various aspects.

First, two-box search is not a query reformation technique as RF. In RF, the *automatically generated* additional terms become part of the reformed query to be issued in round II, while in two-box search, the *user-input* contextual terms are *not* used in round II. The terms generated by RF may work well as contextual terms for two-box search but not the other way around. In general, effective contextual terms form a much larger set.

In query reformation, it is often hard to understand why a particular document was retrieved after applying the technique [26]. In two-box search, the original query is kept intact and only the ranking of search results is changed.

Second, in RF only query terms are used in round I queries. In two-box search, by default the combination of query terms and contextual terms, both entered by users, is used, leading to much more relevant seeds that are comparable to explicit RF. In this sense, two-box search provides a novel and effective remedy for query drift. Beyond that, two-box search can also use contextual terms only to obtain seeds that are relevant to the user-intended domain and not necessarily to the user-intended query, leading to effective utilization of a largely expanded range of domain knowledge.

Third, RF can have practical problems. The typically long queries (usually more than 20 terms) generated by RF techniques are inefficient for IR systems, resulting in high computing cost and long response time [26]. In two-box search, however, both query terms (1 to 3) and contextual terms (1 to 2) are short. A round I query combining the two would typically contain 2 to 5 terms only.

The most well-known algorithm for RF is Rocchio [30, 33]. Originally developed for query optimization, Rocchio is essentially a linear classifier and can be adapted to text categorization and routing problems. Rocchio uses prototypes or centroids of training instances to represent different classes. It is simple and efficient, but inaccurate if classes are not approximately spheres with similar radii. In later sections we will talk about Rocchio in more detail as well as its improvement in the presence of polymorphic (disjunctive) domains.

Query expansion and feedback forms Query expansion can be considered as an automatic query reformation technique that is useful for short queries. It works by adding expanding terms into the original query to better guide the search. The expanding terms are selected by using manually built [41] or automatically constructed thesauruses [29]. Query logs can also be employed to mine the relationship between query terms and document terms [9], and then the expanding terms can be chosen from those highly related document terms. Some recent query expansion methods use click-through and session data for selecting better expanding terms [7, 37].

In Section 1, we discussed the drawbacks of the straightforward query refinement approach. Query expansion would suffer from similar problems. While it can be effective for navigational queries, it trades significant recall for precision and is not good for informational queries. Unlike query expansion, two-box search is not a query reformation technique and it does not alter the original query.

Kelly et al. [20] investigates a technique encouraging users to be more loquacious. It elicits additional terms from users regarding their information needs in a feedback form when ambiguous queries are initially posed. Kelly et al. [20] found that search performance could be substantially improved showing that eliciting more information from the user to produce longer queries is a good technique. Different from RF, the terms are not automatically generated but elicited from users. However, this is again a query reformation technique, while two-box search is not.

Directory Another straightforward way of disambiguation would be to organize Web pages into different categories or domains, as in the Open Directory Project,[2] and search in a selected one. However, this approach suffers from various problems,

[2] www.dmoz.org/

e.g., inaccuracy of categorization, high maintenance cost, and inflexibility in search since the domains are predetermined. In addition, navigating in the domain hierarchy is much less favorable for users than keyword input.

3 Overview

In this section we provide an overview of the design and implementation of Bobo, our prototype system implementing two-box search.

3.1 Design overview

Bobo uses the vector space model, where both documents and queries are represented as vectors in a discretized vector space. Documents used in similarity comparison can be in the form of either full pages or snippets. Documents are preprocessed and transformed into vectors based on a chosen term weighting scheme, e.g., TF-IDF.

The architecture of Bobo is shown in Figure 2. Without input of contextual terms, Bobo works exactly like a mainstream search engine and the dashed modules will not be executed. Input of contextual terms is optional in need of disambiguation of user intent. Domain knowledge, directly or indirectly associated with the query, can be used as "pilot light" to guide the search towards a user-intended domain. The user may not even know what domain her search intent is in. In this case, based on our method, as long as she knows some terms that are in the same unknown domain, she can use them as contextual terms.

With input of contextual terms, Bobo works in two rounds. In round I, a query is issued using by default the combination of query terms and contextual terms, or just the contextual terms if they are unlikely to co-occur much with the query terms. Then from the results, the top κ documents (full pages or snippets) satisfying certain quality conditions, e.g., number of terms contained in each seed, are selected as seeds. Optionally, seeds can be *cleaned* by removing the contained query terms to reduce background noise of individual seeds, or *purified* by removing possibly irrelevant seeds to improve overall concentration. Contextual terms themselves can be used as an *elf seed*, which is a special document allowing negative terms, functioning as an explicit feedback.

In round II, a query is issued using the query terms. Then, each returned result (full page or snippet) is compared to the seeds to compute a similarity using a designated similarity measure, Jaccard coefficient or Cosine coefficient. In the computation, seeds can be *combined* to form a prototype as in Rocchio, or not

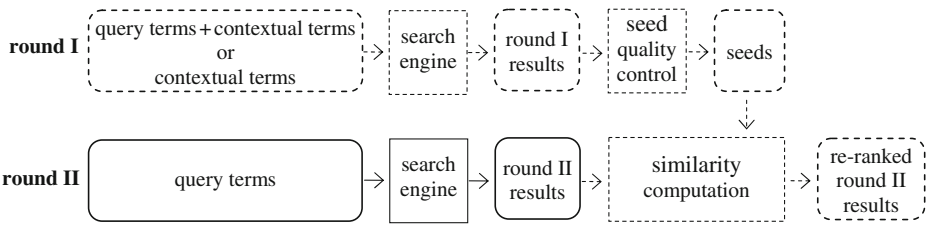

Figure 2 Architecture of Bobo.

combined, as in instance-based lazy learning, to better capture locality and handle polymorphic domains. Based on the assumption that seeds are highly relevant, the similarity values estimate the closeness of search results to the user intent, based on which the results are re-ranked.

3.2 Implementation overview

Bobo was implemented to take advantage of multiple search data sources, including search engines such as Google and Yahoo! and local Lucene-based indexes. For each query, Bobo retrieves up to 1000 results from the data source, though as little as 30 results are sufficient for Bobo to be effective. To evaluate how various factors affect performance, multiple user preferences are provided in Bobo, although most of them can be automated for a more user-friendly interface. Default preferences use conventional schemes whenever possible, e.g., TF-IDF term weighting and Cosine similarity, to make Bobo more transparent in emphasizing the key idea responsible for the performance improvement.

Retrieving query results In order to easily test Bobo with multiple search engines and data sources, we created a Web service, named *AbstractSearch*, responsible for hiding query execution details. It runs as a separate Java Enterprise Edition application and redirects query requests to Apache Lucene for local document inverted indexes, the Google AJAX Search API,[3] or Yahoo! Search API.[4]

AbstractSearch interprets received query parameters into parameters specific to the requested search source. For example, Google AJAX Search API expects a zero-based first requested result parameter, while Yahoo! Search API and Apache Lucene expect a one-based equivalent parameter.

AbstractSearch also speeds up query execution by parallelizing requests to the search source. While there is no limit to the number of results that can be requested from a local Lucene index, the Google API can retrieve a maximum of 8 results per request and a total of 64 results per query, while the Yahoo! API can retrieve 100 results per request and a total of 1000 results per query. The *multi-threaded parallel execution* of requests allows 500 Yahoo! results (executed using 5 Yahoo! requests) to be returned in less than 2 s instead of the 8 s it would take if the requests were executed sequentially. *AbstractSearch* returns the entire requested result set at once as either XML or JSON data.

Preprocessing Depending on user chosen preferences, Bobo uses the title and either the snippet or document body to represent a retrieved search result document. Each result document is analyzed with the Lucene *StandardAnalyzer*[5] and transformed into a bag of terms. The terms are then added to an index of collection *terms* spanning all retrieved search result documents, and each term is associated with a numeric index id. Document frequency and collection frequency are also recorded for each term. Further textual processing such as similarity computation is done using the assigned numeric term and document ids to increase efficiency.

[3] code.google.com/apis/ajaxsearch/

[4] developer.yahoo.com/search/web/V1/webSearch.html

[5] www.docjar.com/docs/api/org/apache/lucene/analysis/standard/StandardAnalyzer.html

Table 1 Bobo execution times for a typical query over multiple search sources.

Source	Doc. type	Total time	Retrieving results	Preprocessing	Similarity comp.
Yahoo!	snippets	0.956 s	0.950 s	0.005 s	0.598 ms
Google	snippets	0.274 s	0.267 s	0.006 s	0.982 ms
Lucene	snippets	0.090 s	0.074 s	0.015 s	0.609 ms
Yahoo!	full page	137.026 s	136.244 s	0.739 s	0.043 s
Google	full page	84.650 s	83.939 s	0.680 s	0.031 s
Lucene	full page	0.121 s	0.100 s	0.015 s	0.006 s

As in other average preprocessors, Bobo does not parse JavaScript, which can lead to missed dynamic content retrieved via AJAX calls in certain Web pages.

3.3 Usability

In terms of usability, the keyword input interface is very user-friendly. Moreover, Bobo is not picky on contextual terms. A large range of domain knowledge would help and users do not have to ponder over the choice of contextual terms. Note that, instead of using an additional auxiliary box, Bobo allows putting query terms and contextual terms in the same box and separate them with a "/" delimiter. Bobo uses two boxes by default only for better conceptual clarity.

In Bobo, for a typical two-box query, the time spent on everything except for page downloading and preprocessing is less than 1 s. Table 1 shows search times averaged over 10 executions for a 50 result query using three search data sources: Google Search, Yahoo! Search, and a local Lucene inverted index. Note that the majority of the time is spent retrieving results, while the parts of the algorithm Bobo is mainly responsible for, preprocessing and similarity computation, take roughly 0.01 s when using snippets and 0.7 s when using full pages.

When using an external search data source such as Google or Yahoo! the document type choice has direct impact on the Bobo execution time. Using snippets, a typical 50 result query in Bobo takes less than 1 s. When full pages are used with an external search data source, document download times can add more than 1 minute for 50 results, which most users are unwilling to wait for.

When using a local search data source Bobo can take advantage of off-line preprocessing and skip page downloading. Note that Lucene execution times are only slightly greater when using full pages (line 6) than when using snippets (line 3), whereas the Yahoo and Google executions using full pages take considerably more time than their snippet counterparts. Preprocessing for the Lucene search source consists of the time spent retrieving stored preprocessed documents and is virtually identical between the snippet and full page tests.[6] Similarity computation can also be done off-line, though it is very fast in Bobo taking only 0.15 s on average for 1000 full text documents.

[6]Lucene data source preprocessing takes longer than Google or Yahoo! preprocessing only due to our slow I/O, given a normal desktop PC with a 5400 rpm PATA disk. It would be much faster in a clustered search engine environment.

4 Principles and preferences

In this section, we introduce in detail the design principles and preferences of Bobo regarding the various key issues. We also discuss possible improvements of Bobo in these aspects.

4.1 Use of contextual terms

The use of contextual terms is the central idea in Bobo. How to use contextual terms has a fundamental impact on the behavior and performance of Bobo.

In round I By default, the combination of query terms and contextual terms are used in round I queries. This produces seeds that are relevant to the user-intended query. For instance, in Example 4, the seeds are relevant to Jim Gray". In the Web setting, there are often sufficient such relevant pages, resulting in high relevance of seeds. This usage of contextual terms actually provides a novel and effective remedy for query drift, thanks to the input of domain knowledge.

In one-box search, a large portion of domain knowledge cannot be utilized in a straightforward manner, due to the fact that contextual terms may co-occur with search terms in very few or no Web pages. However, as shown in Example 5, Bobo allows using only contextual terms for round I queries, enabling utilization of indirectly associated domain knowledge.

As elf seed Contextual terms can be considered to form a pseudo document, which can be optionally used as a seed. We call such a seed *elf seed* as it is actually a piece of explicit relevance feedback. Unlike normal seeds, an elf seed may contain positive as well as negative terms, providing a way of collecting positive as well as negative explicit feedback.

In actual implementation, an elf seed is divided into a positive seed containing all the positive contextual terms and a negative seed containing all the negative contextual terms. Negative seeds contribute negatively in similarity computation with search results.

Discussion The option of combing query terms and contextual terms in round I queries can be automated. The idea is to combine the terms first, then test the k^{th} result to see whether it contains all the terms. If not, only the contextual terms should be used in the query.

4.2 Quality of seeds

As in pseudo relevance feedback, quality of seeds plays a critical role in search performance. The difference is that in Boboinput of contextual terms is largely responsible for the much improved relevance of seeds. To provide further quality control, Bobo accepts several user-input thresholds, e.g., number of seeds and number of terms contained in each seed. Beyond that, Bobo also provides the following options.

Removing query terms By default, Bobo uses a combination of contextual terms and query terms in round I queries. Thus usually all the seeds contain the query

terms. Round II results contain the query terms as well. Then, in similarity computation against the seeds, those query terms contribute almost equally to each round II result. This amount of contribution then becomes background noise, reducing the sensitivity in differentiating round II results.

By default, Bobo removes query terms from seeds. Although a simple step, this option significantly improves performance in our experiments.

Purifying seeds Different approaches have been proposed to alleviate query drift by improving relevance of pseudo feedback, but with limited success [32]. In Bobo, due to the input of domain knowledge, we can well assume that the majority of seeds are relevant, based on which we can design simple mechanisms to purify seeds. Briefly, we first calculate the centroid of seeds. Then, we compute the similarity of each seed against the centroid, and remove those outlying seeds with poor similarities.

Purifying seeds is not a default option in Bobo. In our initial experiments, Bobo produces satisfactory quality of seeds in most cases due to input of contextual terms.

Discussion Current search engines take into account link-based popularity scores such as PageRank [3] in ranking search results. In Bobo, round I search results are not used to directly meet information needs of users. They are never browsed by users. Thus, different search engines with alternative ranking schemes may be used to better fulfill the purpose of round I queries.

Round I queries do not need to be issued to the same region as round II queries either. Working in a more quality region may help avoid spamming and retrieve better candidates for seed selection.

4.3 Term weighting

Bobo uses two term weighting schemes. The default one is the conventional TF-IDF. The other scheme, TF-IDF-TAI, uses term association to favor terms that show high co-occurrence with query terms. This scheme is tailored to Bobo, where documents are not compared in isolation, but being "watched" by a query. Thus it is natural to weight document terms according to their relationship with query terms. While TF-IDF can be considered global weighting independent of queries, TF-IDF-TAI can be considered local weighting.

In the TF-IDF-TAI scheme, the TF-IDF value of a term A is further weighted by $TAI_q(A)$, the term association index of A with respect to a query q, where

$$TAI_q(A) = \max_{T \in q} \{TAI(A, T)\}.$$

A query q is a set of terms. $TAI_q(A)$ takes the largest TAI value between A and $T \in q$. $TAI(A, T)$, the term association index between two terms A and T, is defined on $TAR(A, T)$, the term association ratio for A and T.

$$TAI(A, T) = 1 + \frac{1}{1 - \log_2 TAR(A, T)},$$

$$TAR(A, T) = \frac{DF(A, T)^2}{DF(A) \times DF(T)}.$$

In the formula, $DF(A)$ and $DF(T)$ are the document frequencies of A and T respectively. $DF(A, T)$ is the document frequency of the pair (A, T), i.e., the number of documents in which A and T co-occur.

Since $TAR(A, T) \in [0, 1]$, we have $TAI(A, T) \in [1, 2]$ and $TAI_q(A) \in [1, 2]$. Note that $TAR(T, T) = 1$ and $TAI(T, T) = 2$, then $TAI_q(T) = 2$ if $T \in q$. If A and $T \in q$ never co-occur in any Web page, then $TAR(A, T) = 0$, $TAI(A, T) = 1$, and $TAI_q(A) = 1$, in which case TF-IDF-TAI is equivalent to TF-IDF for term A.

IDF estimation IDF values can be accurately computed for local data sources. However, we do not have access to collection statistics for the Internet, thus IDF values must be estimated when using an external search engine data source. To estimate the term IDF values in this case, Bobo extracted term statistics from a set of 664,103 documents in the TREC Ad-hoc track dataset.[7] These documents can produce a reasonable approximation for term distribution in English language documents as they cover various domains such as newspapers, U.S. patents, financial reports, congressional records, federal registers, and computer related contents.

In particular, for a term A,

$$IDF(A) = \log_2 \frac{n}{DF(A)},$$

where $DF(A)$ is the document frequency of A in the TREC data set and $n = 664{,}103$.

TAR values are also estimated using the same TREC dataset as for IDF estimation. A technical issue is that in Bobowe consider about 110,000 terms, and managing $110,000 \times 110,000$ TAI values can be quite resource-consuming. The solution is that for each term A, we only store $TAI(A, T) \geq \delta$. $TAI(A, T) = 1$ if it is not stored.

4.4 Similarity computation

By computing similarities between round II results and seeds, Bobo estimates how close different results are to the search intent, based on which round II results are re-ranked.

Document type Seeds can either be in the form of snippets (including titles) or full pages. So it is with round II results. White et al. [42] reported that snippets performed even better than full texts for the task of pseudo RF. In our experiments, snippets also performed comparably to full pages. Thus, Bobo uses "snippet" as the default option for both the types of seeds and round II results for fast response time.

Similarity measure Bobo uses two standard similarity measures, Cosine coefficient (default) and Jaccard coefficient. Both performed very well in our experiments, with the default option slightly better.

Prototype-based similarity Bobo implements two types of similarity computation methods, prototype-based or instance-based, with the latter as the default option.

The prototype-based method is actually a form of the well-known Rocchio algorithm [30, 33], which is efficient but would perform poorly in the presence of

[7] trec.nist.gov/data/docs_eng.html

polymorphic domains. In this method, the seeds are combined and the centroid of seeds is used in similarity computation. Given a set S of seeds, the centroid $\mathbf{u}$ is calculated as follows:

$$\mathbf{u} = \frac{1}{|S|} \sum_{s \in S} \mathbf{s},$$

where $\mathbf{s}$ is the vector space representation of seed $s \in S$.

Recall that the original Rocchio algorithm for query reformation is defined as follows:

$$\mathbf{q}_e = \alpha \mathbf{q} + \beta \frac{1}{|D_r|} \sum_{\mathbf{d}_j \in D_r} \mathbf{d}_j - \gamma \frac{1}{|D_{ir}|} \sum_{\mathbf{d}_j \in D_{ir}} \mathbf{d}_j,$$

where q is the original query vector, q_e is the modified query vector, and D_r and D_{ir} represent the sets of known relevant and irrelevant document vectors respectively. α, β, and γ are empirically-chosen tuning parameters.

If we assign $\alpha = 0$ and $\gamma = 0$, the Rocchio formula agrees with our definition of centroid of seeds. We assign $\alpha = 0$ because `Bobo` does not target query reformation. We assign $\gamma = 0$ not because of the lack of negative feedback, which is not hard to identify from low-ranked round I query results. The reason is that even in explicit RF, there is no evidence that negative feedback improves performance [36].

Instance-based similarity Rocchio is simple and efficient. However, it over-generalizes training data and is inaccurate in the presence of polymorphic, or disjunctive, domains. In Figure 3, the 10 seeds labeled by "+" are split into two separate and rather distant sub-domains. The centroid of seeds labeled by "⊕" is not local to any sub-domain. Search result 1 is close to the centroid whereas result 2 is not. Rocchio would give high relevance score to result 1 and poor score to result 2. However, result 2 actually belongs to one of the two sub-domains whereas result 1 does not.

To handle polymorphic domains and capture locality, `Bobo` uses an instance-based approach, where the similarity of a document against each individual seed is computed, weighted, and aggregated. Let $sim(d, S)$ denote the similarity between a document d and a set S of seeds, then,

$$sim(d, S) = \sum_{s \in S} sim(d, s) \times sim(d, s).$$

Using this approach, result 2 will receive much higher relevance score than result 1 in Figure 3. Note that, this approach resembles instance-based lazy learning such as k-nearest neighbor classification. Lazy learning generally has superior performance

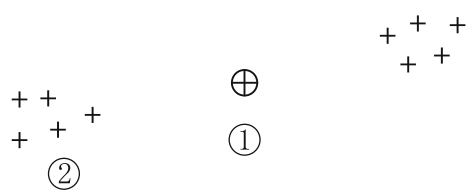

Figure 3 A polymorphic domain.

but would suffer from poor classification efficiency. This, however, is not a critical issue in our application because we do not have many seeds. The default number of seeds in Bobo is set to 10.

Discussion While Bobo adopts rather standard approaches, we are aware of the many other approaches proposed in the literature for pairwise Web page similarity computation. An interesting direction to investigate would be a link-based or hybrid approach. For example, [40] uses Web-graph distance for relevance feedback in web search. Link-based similarity computation can be prohibitively expensive and must be done off-line.

4.5 Re-ranking of results

A straightforward way of re-ranking round II results would be presenting them in decreasing order of similarity. However, this approach ignores page popularity information embedded in the original ranking, which is important for result browsing and good to be retained.

If we had a reliable threshold that can correctly classify the results as relevant or irrelevant, we would only need to present the relevant results with their original relative ranks retained. However, it is hard to find this decision boundary.

Bobo re-ranks results in a stratified manner. It takes a user-input parameter b, finds the $b-1$ most probable decision boundaries, and divides the results into b categories, or layers. Within each layer, results retain their original relative ranks. A simple approach is used to find the $b-1$ probable boundaries. First, search results are sorted in decreasing order of similarity. Then, the top $b-1$ positions with the largest similarity drops are selected. Note that if $b=1$, the re-ranked results are simply in their original order. If b equals the number of results, then the re-ranked results are in decreasing order of similarity.

It may not be intuitive for the users to choose b, just like in many cases, it is not easy to specify the number of clusters for clustering. Based on the observation that it is more intuitive to specify the maximum radius or diameter of a legitimate cluster, a possible improvement of the above approach is to solve a so-called converse k-center or converse pairwise clustering problem [13] that minimizes the number of clusters, where each cluster must satisfy a given radius or diameter constraint. For example, we can specify 0.2 to be the maximum allowable difference of relevance (diameter) for any pair of results in a cluster, then solving the converse pairwise clustering problem returns a minimized number of legitimate clusters (layers). We leave this improvement for future work.

5 Empirical evaluation

Bobo is maintained on a regular desktop PC with Intel 3.0 GHz Duo processor and 4GB memory. We evaluated the performance of Bobo using various data sources, including local indexes built from TREC datasets and Yahoo! Search. The results of experiments based on the Yahoo! data source confirm the trends seen in those performed on local index data sources. Search results returned from Yahoo! may vary with time. This, however, will not change the general trends revealed by our

empirical study. From the comprehensive evaluation we conclude that two-box search is a simple yet effective paradigm for query intent disambiguation without altering the original query and with maximized utilization of domain knowledge.

5.1 Methodology and metrics

To emphasize the Bobo idea, unless otherwise specified, we used default options in the experiments that implement conventional approaches, e.g., TF-IDF for term weighting and Cosine coefficient for similarity computation. By default, number of seeds was set to 10 with each seed having at least 10 terms. Cleaning seeds was set to yes. Combining and weighting seeds were set to no. The value of the number of layers parameter was set equal to the number of results, canceling the effect of this parameter.

Data sources Bobo was implemented to take advantage of multiple data sources, including Google AJAX Search API (code.google.com/apis/ajaxsearch), Yahoo! Search API (developer.yahoo.com/search/web/webSearch.html), and local Lucene indexes built on top of the New York Times Annotated Corpus [34] and several datasets from the TIPSTER (disks 1–3) and TREC (disks 4–5) collections (www.nist.gov/ tac/data/data_desc.html). Users are allowed to choose their preferred data source before executing the search. The Google API can retrieve a maximum of 8 results per request and a total of 64 results per query. The Yahoo! API can retrieve a maximum of 100 results per request and a total of 1000 results per query. Due to user licence agreements, the New York Times, TIPSTER and TREC datasets are not available publicly.

Local index experiments The local index experiments were based on TREC disks 4 & 5 (the Congressional Record data is not included). For each document, we built the index using the content of the headline and text fields which were analyzed with the Lucene *StandardAnalyzer*. Then, we conducted two experiments using TREC ad hoc track topics 351–400 and 401–450, which were designed for this dataset and used during the TREC-7 and TREC-8 proceedings.

Given users' propensity for short queries [17], we constructed our queries using the *title* field in the TREC topics, which is generally a short phrase, containing 1-3 terms. We manually chose 1–5 terms from the *narrative* and *description* TREC topic fields to be used as Bobo contextual terms. For topics that only provided a negative description of the search intent (what should not be included in the search results) we chose contextual terms from general terms associated with the query. The TREC topics along with the Bobo queries and contextual terms used in the experiments are available upon request.

In each experiment we compared Bobo results with those returned by the straightforward base Lucene query (labeled *Lucene*) and two search techniques which alter the returned result set: the first searches using the combined Bobo query and contextual terms (labeled *Combined*) and the second executes pseudo-relevance feedback search using the Bobo query terms (labeled *Prf*). We used standard Rocchio expansion [30] for our pseudo-relevance feedback implementation with α, β and γ parameters set to 1, 0.75 and 0.15 respectively as suggested in [26],

analyzing the first 10 returned documents and expanding the query up to a length of 100 terms.

For each query type we collected the top 1000 results. The results were then analyzed with the standard TREC evaluation tool, *trec_eval*,[8] using TREC conference submission *trec_eval* parameter defaults. The evaluation metrics we used were 11-point interpolated average precision graph, precision at k graph, mean average precision (MAP) and R-precision.

Yahoo! experiments This set of experiments shows `Bobo` performance in Web search and confirms the findings of the *local index experiments*. Additionally, we have tailored these experiments to emphasize the disambiguation capabilities of `Bobo` focusing on the important people search problem.

Finding information about people is one of the most common Web search activities. Around 30% of Web queries include person names [1]. Person names, however, are highly ambiguous, e.g., only 90,000 different names are shared by 100 million people according to the U.S. Census Bureau [15]. To test the disambiguation effectiveness of `Bobo`, we constructed 60 ambiguous name queries and 180 test cases from the Wikipedia disambiguation pages.[9]

In Wikipedia, articles about two or more different topics could have the same natural page title. Disambiguation pages are then used to solve the conflicts. From the various categories, we used the human name category, containing disambiguation pages for multiple people of the same name. For each name, the disambiguation page lists all the different people together with their brief introductions. For example, an Alan Jackson is introduced as "born 1958, American country music singer and songwriter".

Person names were chosen from the most common English first and last names for the year 2000 published on Wikipedia. The first ten male and first ten female given names were combined with the first ten most common last names to make a list of 200 possible names. From this list, names were chosen based on the following criteria. For each name, there are at least two distinct people with the same name, each having at least three relevant pages in the first 30 returned Yahoo! search results.

In total 60 names were chosen as ambiguous queries. For each query, the actual information need was predetermined in a random manner. Then, for this predetermined person, three contextual terms were selected from her brief introduction, or her Wikipedia page in case the introduction was too short. For example, for the above Alan Jackson query, "music", "singer", or "songwriter" can be selected as contextual terms. Contextual terms were used one at a time, thus there are three test cases for each ambiguous query. A subset of the names and contextual terms chosen are included in Table 2 as example.

The identification of relevance of search results was done manually by a third party, some Masters students in the English department at Texas State University-San Marcos. For each query, let R_{30} be the set of relevant pages w.r.t. the information need contained in the first 30 retrieved results. Most users only look through 1–3 short pages of results before abandoning the search or altering their search criteria

[8] trec.nist.gov/trec_eval/

[9] en.wikipedia.org/wiki/Category:Disambiguation_pages

Table 2 Subset of ambiguous name queries and test cases.

Query terms	Contextual terms (one at a time)		
John Adams	Song	Composer	Classical
David Adler	Author	Children	Jansen
Michael Asher	Artist	Conceptual	Exhibition
Larry Bell	Artist	Taos	Glass
Sheldon Brown	Cycling	Bicycle	Gear
Edward Clark	Governor	Texas	Confederate
John Dobson	Astronomer	Telescope	Sidewalk
Charles Garnier	Missionary	Jesuit	Catholic
Rachel Harrison	Sculptor	Artist	Installation
Thomas Hastings	Composer	Hymn	Rock
Paul Jansen	Piano	Bench	Artist
Andy Johnson	Football	English	Soccer
Kelly Johnson	Pitcher	Baseball	Baseman
Lonnie Johnson	Soaker	Invention	Nuclear
Michael Johnson	Music	Album	Guitar
Richard Lane	Actor	Robinson	Movie
Bryan Meyers	Author	Book	Programming
William Rush	Artist	Sculptor	Philadelphia
John Williams	Composer	Score	Film
George Winter	Artist	Frontier	Native

[26]. Therefore, R_{30} can be considered containing the most important relevant pages for the original query.

To compare with Bobo, two types of regular search methods were used. The base *Yahoo!* method uses the original query and performs the simplest Yahoo! Web search, returning the same set of results as Bobo but without re-ranking. The *Yahoo!-refined* method is the straightforward query refinement approach we previously discussed. It refines the original query by adding some contextual terms. The refined query alters the original query, leading to unfavorable ranking of results and failing to include many important relevant pages, i.e., R_{30} pages, in the top results.

We used ranking-aware evaluation measures to demonstrate the relevance improvement of Bobo over the *Yahoo!* method: 11-point interpolated average precision, precision at k, MAP and R-precision. We used the recall at k evaluation measure, which measures the fraction of relevant pages (here, ones in R_{30}) contained in the top k results, to demonstrate that the *Yahoo!-refined* method fails to include many important relevant pages.

Domain diversity experiments This series of experiments shows the disambiguation power of Bobo in a diverse set of ambiguous query domains. Additionally, by testing distinct domains for each query, we show Bobo is not only effective on dominant query senses. Rather, contextual terms in Bobo guide the search toward the preferred user sense.

We chose queries by randomly selecting multiple-sense terms from version 2.1 of WordNet [10]. For each query, we retrieved 30 results from *Google* and users were asked to ensure each query had at least two senses with a minimum of 3 relevant results in R_{30}. Classification of search result relevance was done manually by a third party, some Masters students in the English department at Texas State University-San Marcos. We kept the first 100 queries the users found. For each query, we

asked the users to create two relevance judgements for two distinct query senses. We asked other users to provide 1–5 domain terms suitable for each sense. For each query sense we compared Bobo with the base *Google* search method. In all, users classified $100 \times 2 \times 30 = 6{,}000$ results. We used the following standard ranking-aware evaluation measures to demonstrate the relevance improvement of Bobo over the *Google* base method: 11-point interpolated average precision, precision at k, MAP and R-precision.

Evaluation metrics The *11-point interpolated average precision* graph plots the interpolated precisions measured at the 11 recall levels 0.0, 0.1, ... , 1.0 averaged at each level over all executed queries. The interpolated precision at a certain recall level r is defined as the highest precision found for any recall level $r' \geq r$ [26]:

$$p_{interp}(r) = max_{r' \geq r} p(r').$$

The *precision at k* graph plots the proportion of relevant results among the first k retrieved results averaged for that k value over all executed queries.

For each query, the average precision (AP) is computed as the average of the precisions for the set of top k documents retrieved after each relevant document is retrieved. The *MAP* value for a set Q of executed queries is computed by averaging AP values over all queries in Q. In other words, given a query $q_j \in Q$, the set $\{d_1 ... d_j\}$ of relevant documents for q_j, and R_{jk}, the set of retrieved documents from top to d_k, and considering the precision of a set of retrieved documents with no relevant results to be 0,

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} Precision(R_{jk}).$$

R-precision computes the precision at rank R, where R is the total number of relevant documents. The value is averaged over all executed queries.

5.2 Local index experiment results

In this series of experiments, we evaluated the performance of Bobo by executing queries directly on a Web document inverted index. We compared Bobo with the

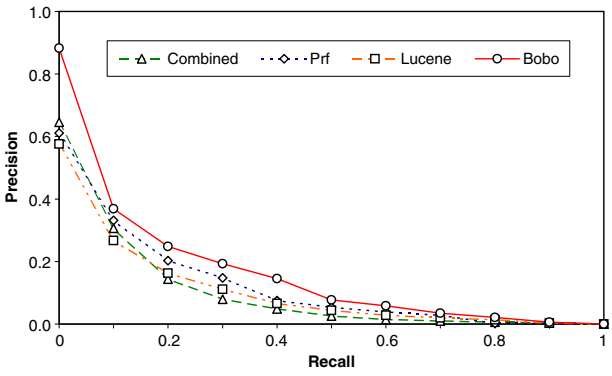

Figure 4 TREC topics 351–400 on 11-point interpolated average precision.

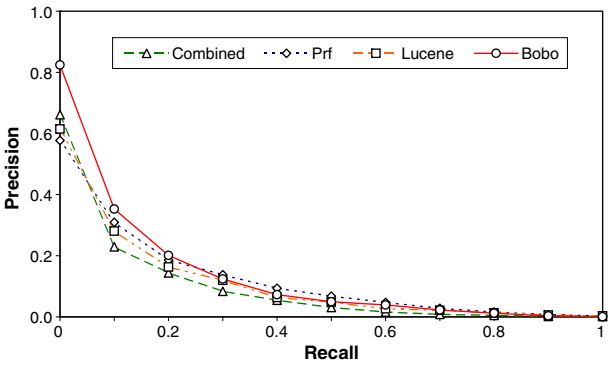

Figure 5 TREC topics 401–450 on 11-point interpolated average precision.

base Lucene search (labeled *Lucene*), the combined Bobo query and contextual terms search (labeled *Combined*) and pseudo-relevance feedback search using the same query terms as Bobo (labeled *Prf*).

Figure 4 shows the 11-point interpolated average precision graph for TREC topics 351–400. As can be seen, Bobo consistently improves on all three search methods it was compared to, with roughly 8–31% higher precision than *Lucene* for recalls higher than 0.5. The *Combined* and *Prf* methods start with higher precisions than *Lucene*, but for recalls higher than 0.2 perform similarly or worse than *Lucene*, likely due to query drift.

Figure 5 shows the 11-point interpolated average precision graph for our second test, using TREC topics 401–450. The performance is similar to our first test, with roughly 1–21% higher precision values than *Lucene* for recalls higher than 0.5. *Prf* is the only search method in this test able to provide slightly better (1–2%) precision than Bobo for recalls higher than 0.2, at the expense of possible query drift.

Figures 6 and 7 show the averaged precision at k for our two tests. Bobo offers a consistent improvement over *Lucene* in both tests. For the most important first few pages of results ($k \leq 30$), Bobo precision values are 9–26% higher than *Lucene*'s in the first and 3–18% in the second test. *Prf* is the only method which performs slightly better than Bobo for $k \geq 200$ in the first and $k \geq 500$ in the second test.

Table 3 shows the MAP and R-precision values for our two tests, which show consistent improvements of Bobo results over all three search methods compared

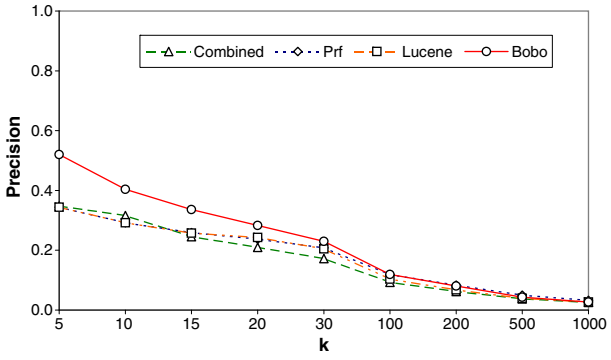

Figure 6 TREC topics 351–400 on averaged precision at k.

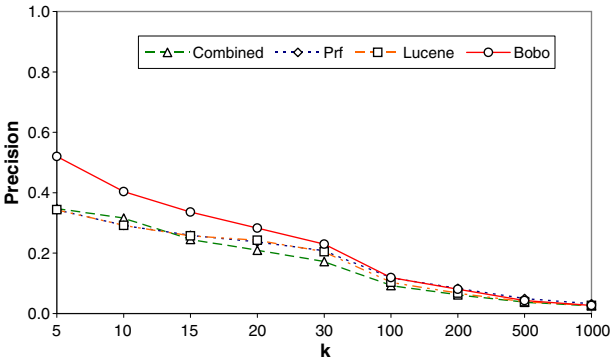

Figure 7 TREC topics 401–450 on averaged precision at *k*.

against. In the first test Bobo improves over the worst scoring competitor method, *Lucene*, by 74% (MAP, 49% R-precision) and 40% (MAP, 28% R-precision) over the best competitor method, *Prf*. In the second test Bobo improves over the worst scoring competitor method, *Combined*, by 51% (MAP, 28% R-precision).

Our two *local index experiments* show that Bobo consistently provides better results than the other three search methods compared against, especially in the top-ranked results, which are arguably the most important.

5.3 Yahoo! experiment results

In this series of experiments, we evaluated the performance of Bobo by comparing it with the *Yahoo!* and *Yahoo!-refined* search methods.

In Figures 8, 9, and 10, Bobo results using both Cosine similarity and Jaccard coefficient are shown. The two performed similarly, with the former (default) slightly better.

Bobo vs. Yahoo! Web search users would typically browse a few top-ranked results. From Figure 9 we can see that for $k = 15$, 10 and 5, the precision improvement of Bobo over *Yahoo!* is roughly 20–40%. In addition, the MAP and R-precision values for Bobo are 0.812 and 0.740 respectively, whereas they are 0.479 and 0.405 for *Yahoo!* respectively. The improvement of Bobo over *Yahoo!* is about 33% for both measures.

Table 3 MAP and R-precision values for TREC topics 351–400 and 401–450.

Run	Topics 351–400		Topics 401–450	
	MAP	R-precision	MAP	R-precision
Combined	0.0882	0.1399	0.0787	0.1267
Prf	0.1089	0.1531	0.1117	0.1571
Lucene	0.0875	0.1310	0.0940	0.1417
Bobo	0.1523	0.1954	0.1188	0.1624

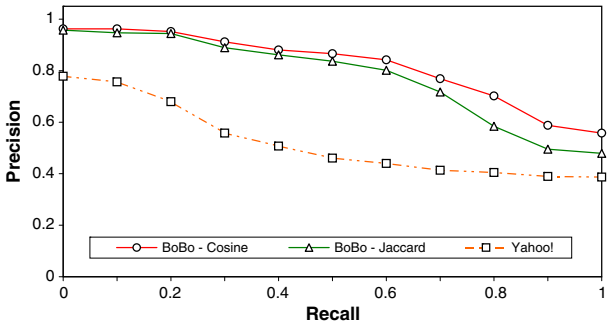

Figure 8 Bobo vs. Yahoo! on 11-point interpolated average precision.

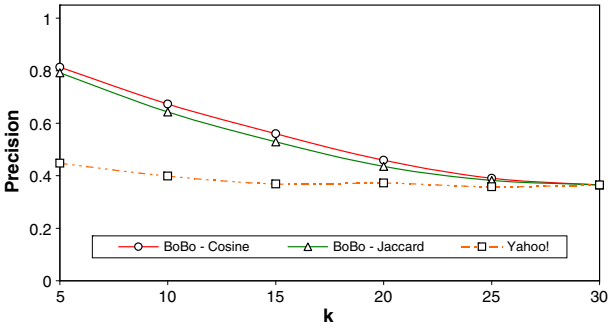

Figure 9 Bobo vs. Yahoo! on averaged precision at *k*.

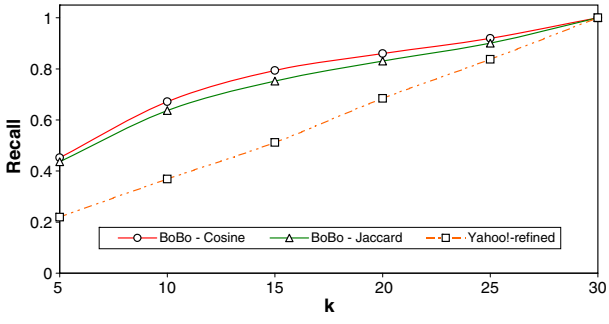

Figure 10 Bobo vs. Yahoo!-refined on averaged recall at *k*.

Table 4 Example WordNet diversity queries and user-chosen contextual terms.

Query terms	Contextual terms	
	Sense 1	Sense 2
Stillness	Calm silent	Lifelessness
Denote	Announce known inform	Designate assign
Center	Pass play field team	Organization place
Stir	Budge agitate fidget	Swirl
Way	Manner mode style	Passage
Wildly	Uncontrolled unrestrained	Greatly exaggerated
Turn back	Backtrack trail return	Clock

Bobo vs. Yahoo!-refined The recall at k graphs are presented in Figure 10. From the figure we can see that for $k = 15$, $k = 10$ and $k = 5$, the recall (of important R_{30} pages) improvement of Bobo over *Yahoo!* is roughly 30%.

The results demonstrate that, although the straightforward query refinement approach can effectively improve relevance, it fails to rank those important relevant pages high, as it alters the original query and changes the query theme. Bobo, on the other hand, overcomes this problem by using the contextual terms "in the background", effectively improving relevance while keeping the original query intact.

5.4 Domain diversity experiment results

We had two goals in executing the domain diversity experiment. First of all, by randomly choosing queries from WordNet we wanted to show that Bobo is effective in diverse domains, not just for the person disambiguation domain. While chosen queries had to meet some simple usefulness criteria, such as having relevant results within R_{30}, the words were chosen randomly and included nouns, verbs, and even adverbs and idiomatic phrases. Table 4 shows a few of the randomly selected queries and user-provided contextual terms.

Bobo performed well even when faced with obscure query senses, showing a 21–29% and 25–50% relative improvement over the *Google* baseline in MAP and R-precision values respectively. Table 5 shows the MAP and R-precision values for our test, reported for the individual query sense and for the combined sets.

It is also interesting to note that Bobo performed well even with little domain information. While we instructed users to choose up to five domain terms, they often chose few domain terms, the majority of the queries being executed with a single domain term. None of the users chose five domain terms. Figure 11 shows the distribution of the number of domain terms chosen by users for the randomly selected WordNet queries.

Table 5 MAP and R-precision values for Bobo and *Google* diversity queries.

Run	Bobo		*Google*	
	MAP	R-precision	MAP	R-precision
Sense 1	0.5118	0.6320	0.4221	0.4221
Sense 2	0.3624	0.5427	0.2815	0.4021
All	0.4371	0.5874	0.3518	0.4684

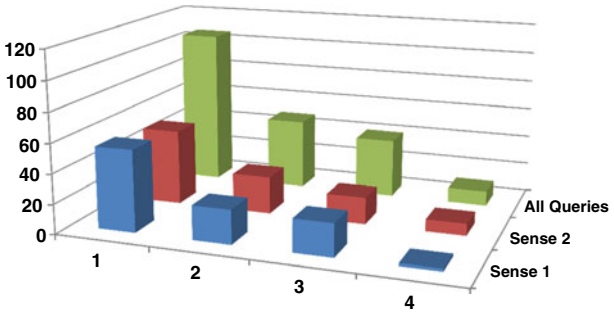

Figure 11 Distribution on number of contextual terms chosen by users for domain diversity queries.

Our second goal for the diversity experiment was to show that Bobo is not only effective for a given sense of a query, e.g. the dominant sense, but rather it guides the search towards the user's search intent. To show this, we independently tested two distinct senses of each query. Figues 12 and 13 show the 11-point interpolated precision recall and precision at k results for our experiment. The relative improvement over the baseline in the 11-point interpolated precision recall graph is above 20% for most recall values, in both the individual sense queries and overall. From the second figure we can see that, for $k = 15$, $k = 10$ and $k = 5$, the precision and recall (of important R_{30} pages) relative improvement of Bobo over *Google* is 20-40%. This shows that in most cases Bobo is able to successfully promote good results given simple domain terms provided by the user.

5.5 Other evaluations

In addition to Bobo performance evaluations we conducted two more sets of experiments to gauge Bobo flexibility when choosing contextual terms and preferences. The results show that Bobo can work effectively in cooperation with a wide range of domain knowledge and is not adversely affected by badly chosen preferences.

Domain knowledge utilization The performance improvement of Bobo has to do with the choice of contextual terms. In this series of experiments, we tested Bobo on the flexibility in choosing effective contextual terms.

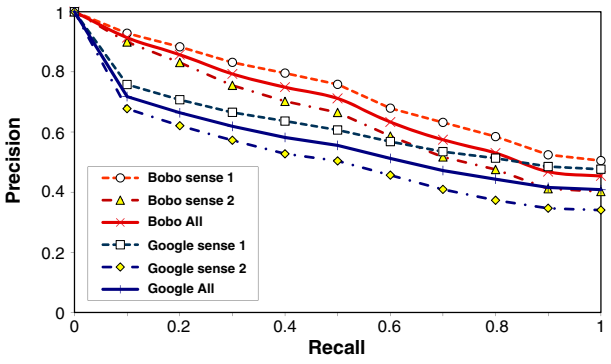

Figure 12 Bobo vs. *Google* on 11-point interpolated average precision.

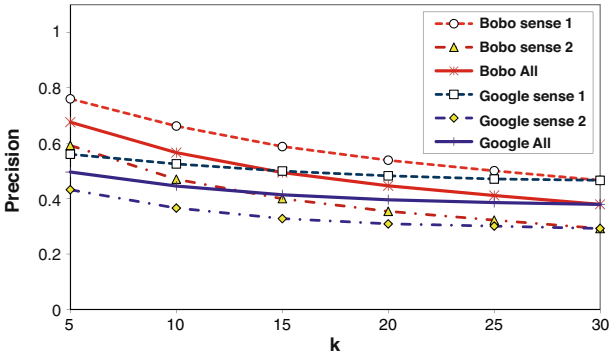

Figure 13 Bobo vs. *Google* on precision at k.

In the *performance evaluation* experiments we used not-carefully-crafted contextual terms for each query. Thus, we have initially demonstrated the flexible domain knowledge utilization of Bobo. Here, we focused on a single query, Jim Gray", to further the evaluation.

For Bobo, with "Jim Gray" as the query term, 10 different contextual terms were used. They were "computer", "science", "Microsoft", "research", "university", "David DeWitt", "Raghu", "AnHai", "Jiawei", and "Flavia Moser". The default preferences were used in the experiments. Precision and recall values were averaged over the 10 test cases.

For *Yahoo!*, out of the 30 search results for query "Jim Gray", 13 were relevant, so $|R_{30}| = 13$. Their ranks were 1, 2, 10, 12, 14, 16, 18, 19, 21, 25, 26, 27 and 29. For *Yahoo!-refined*, "Jim Gray" was combined with the above 10 contextual terms, one at a time, and the averaged recall values w.r.t. R_{30} were recorded.

Figure 14 shows the Bobo results compared to *Yahoo!* on precision at k and *Yahoo!-refined* on recall at k. From the results, we can see that Bobo worked effectively in disambiguation using varied contextual terms, significantly outperforming the two straightforward search methods.

Preference sensitivity In this series of experiments, we evaluated how the varied user preferences in Bobo would affect its performance in search intent disambiguation

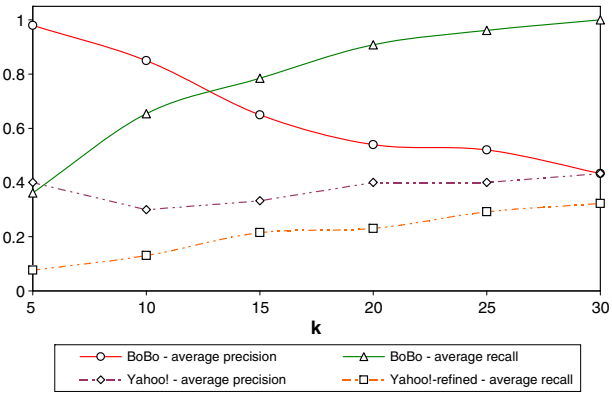

Figure 14 Bobo vs. Yahoo! on averaged precision at k & Bobo vs. Yahoo!-refined on averaged recall at k.

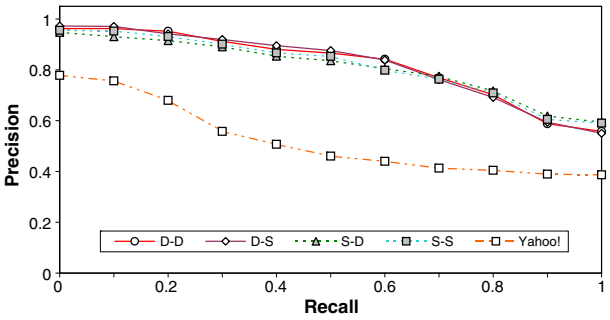

Figure 15 Document type on 11-point interpolated average precision.

and relevance improvement. We used a subset of 75 query and contextual term combinations from the *Yahoo!* performance experiment query pool and the same relevance judgements collected for those queries. Default `Bobo` parameter choices were used when executing each query with the exception of the parameter being tested.

For each query we compared the `Bobo` results ranking with that of the base *Yahoo!* method. All in all we tested 20 parameter combinations for each query, using two ranking methods, for a total of $75 \times 20 \times 2 = 3000$ executed queries. We report ranking-aware evaluation metrics 11-point interpolated average precision graph and precision at k graph. We also present the *Yahoo!* results in the Figures 15, 16, 17 and 18 for convenience of comparison.

Document type Both round II results and seeds can take the form of full document (D) or snippet (S). We tested the 4 combinations of D-D, D-S, S-D, and S-S. For example, D-S means the search results and seeds are in the forms of full document and snippet respectively.

From Figure 15, we can see that there is no significant difference among the 4 combinations. This confirms the observation reported previously in White et al. [42].

Cleaning seeds If cleaning seeds is set yes, query terms will be removed from seeds. As shown in Figure 16, this minor step surprisingly improves precision by 8.3% on average.

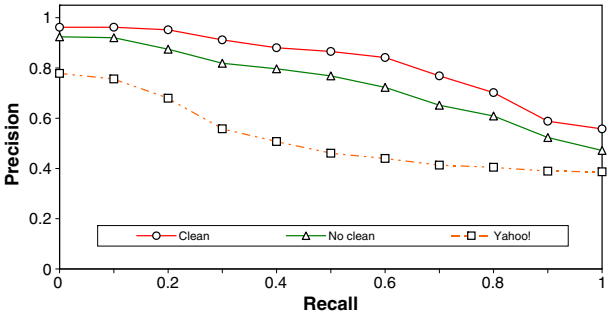

Figure 16 Cleaning seeds on 11-point interpolated average precision.

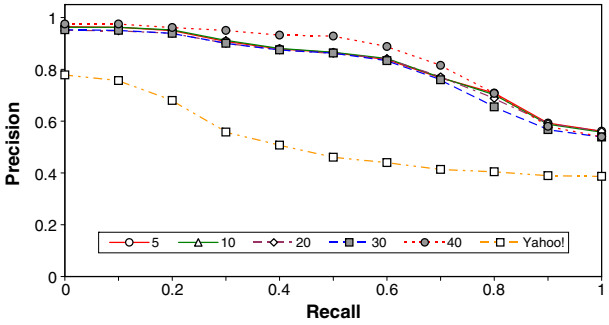

Figure 17 Minimum seed size on 11-point interpolated average precision.

Minimum seed size The size of a seed is the number of terms contained in the seed. As shown in Figure 17, we used 5, 10, 20, 30 and 40 for the minimum seed size threshold, and observed that the parameter is not sensitive. Thus, we choose a relatively small number of 10 as the default option.

Number of seeds We took choices of 1, 3, 5, 10, 15, 20 and 30 to see how the number of seeds affects the performance. From Figure 18 we can see that, while the choice of 1 performed slightly worse, other choices performed very similarly. The parameter is not sensitive and for efficiency concerns, we use a small number of 10 by default.

Similarity measure `Bobo` implemented two most popular similarity computation schemes, Cosine coefficient and Jaccard Coefficient. From Figures 8–10, we can see that they both performed very well, with Cosine coefficient, the default option, slightly better.

Other preferences There are some other preferences in `Bobo`, e.g, elf seed, purifying seeds, combining seeds, weighting seeds, and term weighting. Their variations did not lead to a significant difference in performance. All these preferences are "shut down" by default in `Bobo`.

The results of our *preference sensitivity* experiments show a definite improvement of `Bobo` over the base methods compared against regardless of parameter choices and provided us with best-choice parameter "defaults".

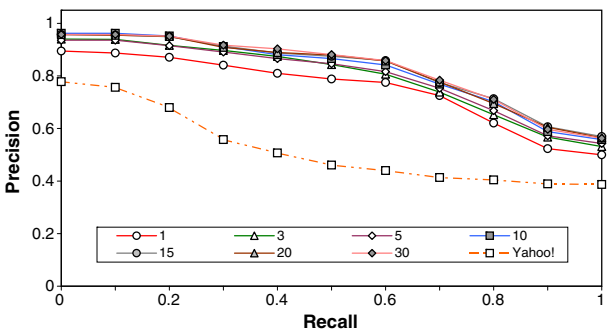

Figure 18 Number of seeds on 11-point interpolated average precision.

5.6 User study

We also performed a simple yet helpful user study collecting feedback on the usability and performance of Bobo.

A survey was distributed to a data mining undergraduate class with 26 students, among whom 14 replied. The following lists the survey questions and the corresponding received scores that are on a scale of 1 (worst) to 10 (best).

1. Is Bobo easy to learn? (range: 7–10; average: 8.5)
2. Is Bobo easy to use? (range: 5–10; average: 7.3)
3. Does Bobo help in disambiguating search intent? (range: 6–10; average: 8)

We received many inspiring and helpful comments. One in particular, showed some confusion between the way Bobo is implemented and how some of the users thought it was. About one third of the participants commented that, while Bobo did work, in some cases it did not return as many relevant results as the *Combined* search approach. An explanation for this confusion is that Bobo only re-ranks the *Lucene* query results. For a fair comparison with *Combined* queries in terms of number of relevant pages, Bobo should be implemented such that both the *Lucene* and *Combined* query results should be retrieved with their union re-ranked. This way, those additional relevant pages from *Combined* queries would most likely be included in the top Bobo results.

6 Conclusions

As the Web grows in size at an increasing rate, ambiguity becomes ubiquitous. For example, there are 87 Wikipedia entries for "Michael Smith" as of today. In this paper, we introduced a novel two-box search paradigm to achieve simple yet effective search intent disambiguation without altering the original query and with maximized domain knowledge utilization. Comparing to the straightforward query refinement, this approach is particularly useful for informational queries where users want to learn about something. Comprehensive empirical evaluation using both local and remote data sources demonstrated the utility and potential of our approach.

For future work, while there are many directions to explore, a particularly interesting one is to seamlessly integrate two-box search with personalized search. The integrated system would have combined advantages from the two, flexibility and convenience, in guiding search towards a user intended domain.

References

1. Artiles, J., Gonzalo, J., Verdejo, F.: A testbed for people searching strategies in the WWW. In: Proceeding of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 569–570 (2005)
2. Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval. Addison-Wesley (1999)
3. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: Proceedings of the 7th International World Wide Web Conference (WWW), pp. 107–117 (1998)
4. Broder, A.: A taxonomy of web search. SIGIR Forum **36**(2), 3–10 (2002)
5. Buckley, C., Amit, S., Mandar, M.: New retrieval approaches using smart: Trec 4. In: Proceeding of the 4th Text Retrieval Conference (TREC-4), pp. 25–48 (1995)

6. Cao, H., Jiang, D., Pei, J., Chen, E., Li, H.: Towards context-aware search by learning a very large variable length hidden markov model from search logs. In: Proceedings of the 18th International World Wide Web Conference (WWW), pp. 191–200 (2009)
7. Cao, H., Jiang, D., Pei, J., He, Q., Liao, Z., Chen, E., Li, H.: Context-aware query suggestion by mining click-through and session data. In: Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 875–883 (2008)
8. Croft, W., Harper, D.: Using probabilistic models of information retrieval without relevance information. J. Doc. **35**(4), 285–295 (1979)
9. Cui, H., Wen, J.R., Nie, J.Y., Ma, W.Y.: Probabilistic query expansion using query logs. In: Proceedings of the 11th International World Wide Web Conference (WWW), pp. 325–332 (2002)
10. Fellbaum, C.: WordNet: An Electronic Lexical Database. Bradford Books (1998)
11. Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., Ruppin, E.: Placing search in context: the concept revisited. ACM Trans. Inf. Sys. **20**(1), 116–131 (2002)
12. Gao, B.J., Anastasiu, D.C., Jiang, X.: Utilizing user-input contextual terms for query disambiguation. In: Proceedings of the 23rd International Conference on Computational Linguistics (COLING), pp. 329–337 (2010)
13. Gao, B.J., Ester, M., Cai, J.Y., Schulte, O., Xiong, H.: The minimum consistent subset cover problem and its applications in data mining. In: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 310–319 (2007)
14. Gkanogiannis, A., Kalamboukis, T.: An algorithm for text categorization. In: Proceeding of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 869–870 (2008)
15. Guha, R.V., Garg, A.: Disambiguating people in search. In: Proceedings of the 13th International World Wide Web Conference (WWW) (2004)
16. Haveliwala, T.H.: Topic-sensitive pagerank. In: Proceedings of the 11th International World Wide Web Conference (WWW), pp. 517–526 (2002)
17. Jansen, B.J., Spink, A., , Saracevic, T.: Real life, real users and real needs: A study and analysis of users queries on the web. Information Process. and Manage. **36**(2), 207–227 (2000)
18. Jeh, G., Widom, J.: Scaling personalized web search. In: Proceedings of the 12th International World Wide Web Conference (WWW), pp. 271–279 (2003)
19. Joachims, T., Granka, L., Pan, B., Hembrooke, H., Gay, G.: Accurately interpreting clickthrough data as implicit feedback. In: Proceeding of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 154–161 (2005)
20. Kelly, D., Dollu, V.D., , Fu, X.: The loquacious user: a document-independent source of terms for query expansion. In: Proceeding of the 18th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 457–464 (2005)
21. Kelly, D., Teevan, J.: Implicit feedback for inferring user preference: a bibliography. SIGIR Forum **37**(2), 18–28 (2003)
22. Kraft, R., Chang, C.C., Maghoul, F., Kumar, R.: Searching with context. In: Proceedings of the 15th International World Wide Web Conference (WWW), pp. 477–486 (2006)
23. Lawrence, S.: Context in web search. IEEE Data Eng. Bull. **23**(3), 25–32 (2000)
24. Lee, J.H.: Combining the evidencde of different relevance feedback methods for information retrieval. Information Process. and Manage. **34**(6), 681–691 (1998)
25. Lee, K.S., Croft, W.B., Allan, J.: A cluster-based resampling method for pseudo-relevance feedback. In: Proceeding of the 31st international ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 235–242 (2008)
26. Manning, C.D., Raghavan, P., Schutze, H.: Introduction to Information Retrieval. Cambridge University Press (2008)
27. Mitra, M., Singhal, A., Buckley, C.: Improving automatic query expansion. In: Proceeding of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 206–214 (1998)
28. Mizzaro, S., Vassena, L.: A social approach to context-aware retrieval. World Wide Web **14**(4), 377–405 (2011)
29. Qiu, Y., Frei, H.P.: Concept based query expansion. In: Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 160–169 (1993)
30. Rocchio, J.: Relevance Feedback in Information Retrieval. In The SMART Retrieval System—Experiments in Automatic Document Processing. Prentice-Hall (1971)
31. Rose, D.E., Levinson, D.: Understanding user goals in web search. In: Proceedings of the 13th International World Wide Web Conference (WWW), pp. 13–19 (2004)

32. Ruthven, I., Lalmas, M.: A survey on the use of relevance feedback for information access systems. Knowl. Eng. Rev. **18**(1), 95–145 (2003)
33. Salton, G., Buckley, C.: Improving Retrieval Performance by Relevance Feedback. Morgan Kaufmann (1997)
34. Sandhaus, E.: The New York Times Annotated Corpus. Linguistic Data Consortium, Philadelphia (2008)
35. Schapire, R.E., Singer, Y., Singhal, A.: Boosting and rocchio applied to text filtering. In: Proceeding of the 21st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 215–223 (1998)
36. Schutze, H., Hull, D.A., Pedersen, J.O.: A comparison of classifiers and document representations for the routing problem. In: Proceeding of the 18th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 229–237 (1995)
37. Shen, X., Tan, B., Zhai, C.: Context-sensitive information retrieval using implicit feedback. In: Proceedings of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 43–50 (2005)
38. Singhal, A., Mitra, M., Buckley, C.: Learning routing queries in a query zone. In: Proceeding of the 20th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 25–32 (1997)
39. Teevan, J., Dumais, S.T., Horvitz, E.: Personalizing search via automated analysis of interests and activities. In: Proceeding of the 28th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 449–456 (2005)
40. Vassilvitskii, S., Brill, E.: Using web-graph for relevance feedback in web search. In: Proceeding of the 29th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 147–153 (2006)
41. Voorhees, E.M.: Using wordnet to disambiguate word senses for text retrieval. In: Proceeding of the 16th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 171–180 (1993)
42. White, R.W., Clarke, C.L., Cucerzan, S.: Comparing query logs and pseudo-relevance feedback for web search query refinement. In: Proceeding of the 30th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 831–832 (2007)
43. White, R.W., Jose, J.M., Rijsbergen, C.J.V., Ruthven, I.: A simulated study of implicit feedback models. In: Proceeding of the 26th European Conference on Information Retrieval (ECIR), pp. 311–326 (2004)
44. Yu, S., Cai, D., Wen, J.R., Ma, W.Y.: Improving pseudo-relevance feedback in web information retrieval using web page segmentation. In: Proceedings of the 12th International World Wide Web Conference (WWW), pp. 11–18 (2003)
45. Zhang, H., Chen, Z., Li, M., Su, Z.: Relevance feedback and learning in content-based image search. World Wide Web **6**(2), 131–155 (2003)
46. Zhu, Y., Callan, J., Carbonell, J.: The impact of history length on personalized search. In: Proceeding of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pp. 715–716 (2008)