USER-CENTERED ADAPTIVE INFORMATION RETRIEVAL

BY

XUEHUA SHEN

B.S., Nanjing University, 1999

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

This material is based upon work supported by the National Science Foundation under Grant No.
IIS-0347933. Any opinions, findings and conclusions or recommendations expressed in this
material are those of the author and do not necessarily reflect the views of the National Science
Foundation (NSF).
Abstract

Information retrieval systems are critical for overcoming information overload. A major deficiency of existing retrieval systems is that they generally lack user modeling and are not adaptive to individual users; information about the actual user and search context is largely ignored. Personalization is expected to break this deficiency and significantly improve retrieval accuracy. In this thesis, we study how to put the user in the center of information retrieval process for the personalized search.

We develop a decision-theoretic framework for optimizing interactive information retrieval based on eager user model updating, in which the system responds to every action of the user by choosing a system action to optimize a utility function. The framework emphasizes immediate and frequent feedback to bring maximum benefit of context to the user. In general, it serves as a roadmap for studying retrieval models for personalized search.

Using the general decision-theoretic framework described above, specific retrieval models for exploiting implicit user context based on statistical language model are developed to improve retrieval accuracy. Evaluation indicates that the user context information especially the clickthrough information can effectively and efficiently improve retrieval performance with no additional effort from the user.

Sometimes we need user effort to provide more information to improve the retrieval performance. In this scenario, we study how a retrieval system can perform active feedback, i.e., how to choose documents for relevance feedback so that the system can learn most from the feedback information. We frame the problem of active relevance feedback as a statistical decision problem, and examine several special cases in refining the framework. We derive several practical algo-
algorithms for active feedback. The experimental results indicate that the diversity in the presented documents is a desirable property.

On the result representation side, we study how to exploit a user’s clickthrough information to adaptively reorganize the clustering results and help a user find the relevant information more quickly. We propose four strategies for adapting clustering results based on user actions. The simulation experiments show that the adaptation strategies have different performance for different types of users. We also conduct a user study on one of the four adaptive clustering strategies to see if an adaptive clustering system using such a strategy can bring users better search utility than a static clustering system. The results show that there is generally no significant difference between the two systems from a user’s perspective.

We design and develop a client-side web search agent UCAIR on top of popular search engines for personalized search. UCAIR search agent captures and exploits implicit context information such as related immediately preceding query and viewed document summaries to immediately rerank any documents that have not yet been seen by the user. User studies show that the UCAIR search agent improves performance over a popular search engine, on which UCAIR search agent is built.
# Table of Contents

List of Tables .............................................................................................. viii
List of Figures .............................................................................................. x

Chapter 1  Introduction ............................................................................... 1

Chapter 2  Related Work ........................................................................... 7

Chapter 3  Optimization in Interactive IR ................................................ 12
  3.1  A decision-theoretic Framework ...................................................... 14
  3.2  User Models .................................................................................... 16
  3.3  Loss Functions ................................................................................ 17
  3.4  Implicit User Modeling .................................................................... 19

Chapter 4  Language Models for Context-sensitive IR ............................. 20
  4.1  Basic Retrieval Model ..................................................................... 22
  4.2  Fixed Coefficient Interpolation (FixInt) .......................................... 23
  4.3  Bayesian Interpolation (BayesInt) ................................................... 24
  4.4  Online Bayesian Updating (OnlineUp) ........................................... 24
    4.4.1  Bayesian Updating .................................................................... 25
    4.4.2  Sequential Query Model Updating ......................................... 26
  4.5  Batch Bayesian Updating (BatchUp) .............................................. 27
  4.6  Evaluation of Context-sensitive Retrieval Models ......................... 28
    4.6.1  Data Collection ....................................................................... 28
    4.6.2  Experiment Design ................................................................... 30
    4.6.3  Overall Effect of Search Context ............................................. 30
    4.6.4  Using Query History Only ....................................................... 32
    4.6.5  Using Clickthrough History Only .......................................... 34
    4.6.6  Additive Effect of Context Information ................................... 35
    4.6.7  Parameter Sensitivity ............................................................... 36

Chapter 5  A Context-Sensitive IR system – UCAIR Search Agent .......... 39
  5.1  Design ............................................................................................ 40
  5.2  Session Boundary Detection and Query Expansion ......................... 41
  5.3  Information Need Model Updating ................................................ 43
5.4 Result Reranking ................................................. 43
5.5 Evaluation of UCAIR Toolbar ................................. 44
  5.5.1 Sample Results ........................................... 44
  5.5.2 Quantitative Evaluation ................................. 45

Chapter 6  Active Feedback ....................................... 50
  6.1 Introduction ................................................... 50
  6.2 Active Feedback Framework ................................ 52
    6.2.1 Independent Loss ....................................... 53
    6.2.2 Dependent Loss ......................................... 56
  6.3 Experiment Methodology .................................... 59
    6.3.1 Data Set ................................................ 59
    6.3.2 Experiment Setup ....................................... 60
    6.3.3 Algorithm Description ................................ 60
    6.3.4 Evaluation Method ..................................... 61
  6.4 Experiment Result ........................................... 63
    6.4.1 Gapped Top K ........................................... 63
    6.4.2 K Cluster Centroid ..................................... 64
    6.4.3 Comparison of Different Algorithms ................. 65
    6.4.4 Performance Sensitivity of K ......................... 66
    6.4.5 Mixture Feedback Algorithm Parameter $\alpha$ Factor 66
  6.5 Conclusions and Future Work ............................... 67

Chapter 7  Adaptive Clustering of Search Results ............... 69
  7.1 Introduction ................................................... 70
  7.2 Related Work ................................................ 73
  7.3 Basic Algorithm Description ............................... 74
    7.3.1 Information Retrieval Method ......................... 74
    7.3.2 K-Medoid Clustering .................................. 75
    7.3.3 Query Updating (Feedback) ......................... 76
  7.4 Four Adaptive Strategies ................................. 77
    7.4.1 Reranking Based on Cluster Selection ............... 77
    7.4.2 Reranking Based on Document Selection ............ 77
    7.4.3 Merging Unselected Clusters ......................... 78
    7.4.4 Promoting “Near Miss” Documents ................... 78
  7.5 Simulation Study ............................................ 78
    7.5.1 Experiment Methodology ................................ 78
    7.5.2 Experiment Design ...................................... 80
    7.5.3 Linearization of Cluster Results .................... 82
    7.5.4 Experiment Results .................................... 82
  7.6 User Study .................................................. 86
    7.6.1 Experiment Design ...................................... 86
    7.6.2 Experiment Procedure ................................ 88
    7.6.3 Data Collection ........................................ 89
<table>
<thead>
<tr>
<th>Table Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Effect of using query history and clickthrough data for document ranking.</td>
<td>31</td>
</tr>
<tr>
<td>4.2</td>
<td>Effect of using query history only for document ranking.</td>
<td>32</td>
</tr>
<tr>
<td>4.3</td>
<td>Average precision of BatchUp using query history only</td>
<td>32</td>
</tr>
<tr>
<td>4.4</td>
<td>Effect of using clickthrough data only for document ranking.</td>
<td>33</td>
</tr>
<tr>
<td>4.5</td>
<td>BayesInt evaluated on unseen relevant documents</td>
<td>35</td>
</tr>
<tr>
<td>4.6</td>
<td>Effect of using only non-relevant clickthrough data</td>
<td>36</td>
</tr>
<tr>
<td>4.7</td>
<td>Additive benefit of context information</td>
<td>37</td>
</tr>
<tr>
<td>4.8</td>
<td>Sensitivity of $\mu$ in BatchUp</td>
<td>38</td>
</tr>
<tr>
<td>4.9</td>
<td>Sensitivity of $\nu$ in BatchUp</td>
<td>38</td>
</tr>
<tr>
<td>5.1</td>
<td>Sample results of query expansion</td>
<td>45</td>
</tr>
<tr>
<td>5.2</td>
<td>Table of average precision at top n documents for 32 query topics</td>
<td>48</td>
</tr>
<tr>
<td>6.1</td>
<td>Average performance of Gapped Top K with different gaps. The best performance is shown in bold.</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>Average performance of K Cluster Centroid with N. The best performance is shown in bold.</td>
<td>61</td>
</tr>
<tr>
<td>6.3</td>
<td>Average performance of different active learning algorithms. The best performance is shown in bold. A double star (**) and a single star (*) indicate that the performance is significantly better than that of Top K according to Wilcoxin signed rank test at the level of 0.05 and 0.1, respectively.</td>
<td>63</td>
</tr>
<tr>
<td>6.4</td>
<td>Average performance of different retrieval algorithms on AP90 data set. The best performance is shown in bold. A double star (**) indicates that the performance is significantly better than that of Top K according to Wilcoxin signed rank test at the level of 0.05.</td>
<td>63</td>
</tr>
<tr>
<td>6.5</td>
<td>Sensitivity of average performance of different active learning algorithms on K.</td>
<td>64</td>
</tr>
<tr>
<td>6.6</td>
<td>Average performance of different active learning algorithms on different $\alpha$.</td>
<td>67</td>
</tr>
<tr>
<td>7.1</td>
<td>Retrieval performance of smart user and dummy user at different stages.</td>
<td>82</td>
</tr>
<tr>
<td>7.2</td>
<td>Experiment results for different number terms for smart adaptive method</td>
<td>83</td>
</tr>
<tr>
<td>7.3</td>
<td>Study of individual query topics.</td>
<td>84</td>
</tr>
<tr>
<td>7.4</td>
<td>Experiment results for regroup clustering method</td>
<td>85</td>
</tr>
<tr>
<td>7.5</td>
<td>Experiment results of near miss promotion method</td>
<td>85</td>
</tr>
<tr>
<td>7.6</td>
<td>Number of search result pages viewed by subjects</td>
<td>90</td>
</tr>
<tr>
<td>7.7</td>
<td>Comparison of difference of two clustering systems</td>
<td>90</td>
</tr>
</tbody>
</table>
7.8 Comparison of overall experience of two clustering systems . . . . . . . . . . . . 90
7.9 Comparison of number of saved documents of two clustering systems . . . . . . 91
7.10 Comparison of number of saved documents of two clustering systems over differ-
ent system preference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92
7.11 Topic familiarity and number of saved documents . . . . . . . . . . . . . . . . . . 92
List of Figures

5.1  UCAIR architecture ......................................................... 41
5.2  Screen shots for result reranking ................................. 46
5.3  An example of TREC query topic, expressed in a form which might be given to a human assistant or librarian ................................................. 47
5.4  Precision at top 10 documents of UCAIR and Google ......................... 48
5.5  Precision at top 20 documents of UCAIR and Google ................. 49
5.6  Precision at top 20 result of UCAIR and Google ..................... 49
6.1  Evaluation procedure ......................................................... 60
7.1  Clustering interface of search results ............................... 87
7.2  An example of query Topics .............................................. 87
7.3  Time dynamics comparison of subject 5 vs. subject 6 .................. 93
8.1  Software architecture of personalized web Search .................. 105
8.2  UCAIR architecture .......................................................... 111
With the dramatic increase of online information in recent years, management of textual information is becoming increasingly important. Information Retrieval (IR) refers to finding information from large amounts of text, and is among the most useful technologies for overcoming information overload. For example, Web search engines are now essential tools for everyone to find information on the Web. Indeed, search capabilities are becoming more and more popular in virtually all kinds of information management applications.

Research in information retrieval has a long history dating back to the 1950’s. Over the decades, significant progress has been made in developing retrieval models, performing large scale empirical evaluation, and building useful systems.

However, although many information retrieval systems (e.g., web search engines and digital library systems) have been successfully deployed, the existing retrieval systems are far from optimal. This non-optimality is seen clearly in the following two cases.

First, different users may use exactly the same query to search for different information, but existing retrieval systems return the same results for these users. For example, the query ”IR applications” on Google returns a mixture of documents about ”information retrieval” applications and ”infrared” applications, as ”IR” can be an acronym for both information retrieval and infrared. Without considering the actual user it is inherently impossible to know which sense ”IR” refers to.

Second, a user’s information needs may change over time. The same user may sometimes use ”java” to mean the Java island and some other times use ”java” to mean the programming language. Without recognizing the search context, it would be again inherently impossible to recognize the correct sense.
As we can see, a major deficiency of existing retrieval systems is that they generally lack user modeling and are not adaptive to individual users [62]. Most existing retrieval systems regard user search activities as independent behaviors. After the user submits a query, the retrieval system will return some search results to the user only according to the submitted query by the user. However, information retrieval is an inherently interactive process and the retrieval system can be regarded as an *interactive information service system* which presents search results in some way, e.g., a ranked list or several clusters, given the submitted query by the user. In general, the retrieval results using the user’s initial query may not be satisfactory; often, the user would need to revise the query to improve the retrieval accuracy. For a complex or difficult information need, the user may need to modify his query and view ranked documents with many iterations before the information need is completely satisfied. In such an interactive information retrieval scenario, the information naturally available to the retrieval system is more than just the current user query and the document collection – in general, all the interaction history can be available to the retrieval system, including past queries, information about which documents the user has chosen to view, and even how a user has read a document (e.g., which part of a document the user spends a lot of time in reading). Furthermore, The retrieval system may also actively solicit relevance feedback from the user. For example, when the retrieval system is not clear about the user information need, it can present some documents or passages to the user for relevance judgment. Through the user relevance feedback, the retrieval system can have more clues about the user information need, reformulate the query to initiate another round of retrieval, and present new retrieval results with improved accuracy to the user.

It is therefore clear that an optimal retrieval system must incorporate as much user information besides the user query as possible into the retrieval decision process, i.e., provide personalized search. *Personalization* can be expected to break the limitations of existing retrieval methods and significantly improve retrieval accuracy and user satisfaction level. Indeed, personalized and contextual search has been identified as a major challenge in information retrieval research [5].

Due to its importance, personalization has attracted much attention recently in several research
communities such as information retrieval [96, 48, 90], Web information management [29, 42, 87], digital library [14], human computer interaction [27, 23], and machine learning communities [11, 66].

However, there are three major limitations of the existing work. First, they have a limited notion of personalization. Most existing works are focused on ranking of information retrieval [29]. Nevertheless, there are several stages in interactive information retrieval, query formulation, ranking, result presentation and relevance feedback. At each stage, we can apply personalization techniques. Second, most works [42] are doing personalization at the server side, where privacy is a serious concern. When the personalization technique is applied at the server side, a lot of sensitive information about individual users will be stored at the server side. Users have no or few control of these sensitive data. Third, although many works study the user modeling or context-sensitive ranking separately, they do not design and develop practical systems to demonstrate the usefulness of personalized search.

In this thesis, we break these limitations and develop a full-fledged privacy-preserving personalized search system. We take a broader view of personalization and apply the personalization in the ranking, result representation and relevance feedback stages of interactive information retrieval. We present a decision theoretic framework and specific retrieval methods for the personalized search. We systematically examine the issue of privacy preservation in personalized search. We distinguish and define four levels of privacy protection, and analyze various software architectures for personalized search. We show that client-side personalization has advantages over the existing server-side personalized search services in preserving privacy, and envision possible future strategies to fully protect user privacy.

To demonstrate the usefulness of personalization in the real search activities, we design and develop a UCAIR (User-Centered Adaptive Information Retrieval) search agent, a privacy-preserving personalized search system which will incorporate the personalization into retrieval, result representation and relevance feedback stages of interactive information retrieval. UCAIR agent demonstrates the effectiveness of personalized search at different stages of interactive information re-
more specifically, we explore the privacy-preserving personalized search in the following five important directions.

- **Decision-theoretic framework**: We develop a general decision-theoretic framework for personalized search to model text and user context information. In order to **maximally** benefit the user of a retrieval system through user modeling, we propose to perform “eager feedback”. That is, as soon as we observe any new piece of evidence from the user, we would update the system’s belief about the user’s information need and respond with improved retrieval results based on the updated user model. We present a decision-theoretic framework for optimizing interactive information retrieval based on eager user model updating, in which the system responds to every action of the user by choosing a system action to optimize a utility function. In a traditional retrieval paradigm, the retrieval problem is to match a query with documents and rank documents according to their relevance values. As a result, the retrieval process is a simple independent cycle of “query” and “result display”. In the proposed new retrieval paradigm, the user’s search context in interactive information retrieval plays an important role and the inferred user model is exploited immediately to benefit the user.

- **Retrieval methods using implicit feedback**: To minimize the user effort, ideally, we could exploit user context naturally available during the interaction. Indeed, a lot of such information as past queries and clickthrough information could be useful. *Implicit feedback* has attracted much attention recently [29, 96, 87, 48]. We define implicit feedback broadly as exploiting all such naturally available interaction history to improve retrieval results.

Specific retrieval models for exploiting implicit user context based on statistical language model will be proposed to improve retrieval accuracy. Specifically, we develop models for using implicit user context information such as query and clickthrough history of the current search session to improve retrieval accuracy. We use the KL-divergence retrieval
model [105] as the basis and propose to treat personalized retrieval as estimating a query language model based on the current query and any user context information. We propose several statistical language models to incorporate query and clickthrough history into the KL-divergence model.

- **Active feedback**: Although no user effort is ideal, sometimes we need user effort to provide more information to improve the retrieval performance. For example, in the scenario of intelligence analysis or high-recall retrieval, the user is willing to provide explicit relevance feedback in order to get high-quality retrieval results. When explicit relevance feedback is possible, a natural question for the retrieval system is how to intelligently propose questions so that it can learn most from the user’s explicit answers while at the same time the user overhead is minimized. A question could be whether a document or passage is relevant, or whether a term describes the user’s information need.

In this scenario, a basic question is how the retrieval system should intelligently propose the questions so that it can learn most from the user’s answers to these questions. We study how a retrieval system can perform active feedback, i.e., how to choose documents for relevance feedback so that the system can learn most from the feedback information. We frame the problem of active relevance feedback as a statistical decision problem, and examine several special cases in refining the framework. We derive several practical algorithms for active feedback, including the Top K, Gapped Top K and K Cluster Centroid algorithm.

- **Personalized clustering representation**: Most current search engines present the user a ranked list given the submitted user query. Top ranked search results generally cover few aspects of all search results. However, in many cases, the users are interested in the main themes of search results besides the ranked list so that the user will have a global view of search results. This goal is often achieved through clustering approaches. Personalized search studies ranking or reranking the search results based on implicit feedback. The personalized search system will infer user information need based on user search engine
interaction and rerank the search results. Same as the ranked search result lists, clusterings of search results intuitively should also be dynamically tuned according to user search system interaction. Thus it brings interesting clustering challenges in the personalized search framework. Clustering results should change dynamically to reflect the personalized ranking of search results. However, traditional static clustering algorithms based on document similarity cannot achieve this goal. In this work, we study how to dynamically update the cluster representation based on user’s implicit feedback.

- **UCAIR search agent**: We design and develop a client-side web search agent UCAIR (User-Centered Adaptive Information Retrieval) on top of popular search engines such as Google and Yahoo! for personalized search. UCAIR search agent captures and exploits two types of implicit context information: (1) identifying related immediately preceding query and using the query and the corresponding search results to select appropriate terms to expand the current query, and (2) exploiting the viewed document summaries to immediately rerank any documents that have not yet been seen by the user. UCAIR search agent will also provide the functionality of active feedback. When the user is willing to provide the relevance feedback, the agent will provide several examples (e.g., passages or documents) for the user relevance judgment using the active feedback algorithm we study. The agent will incorporate the user feedback into the next round retrieval.

The rest of the thesis is organized as follows. In Chapter 2, the related work in this area is presented. A general framework for personalized search is introduced in Chapter 3, followed by the specific context-sensitive models based on statistical language model in Chapter 4. In Chapter 5, a prototype of a client-side intelligent search agent is described. In Chapter 6, we introduce a decision-theoretic framework for active feedback, describe several active feedback algorithms and empirical evaluation. In Chapter 7, we describe our work of adaptive clustering search result representation. In Chapter 8, we discuss the privacy preservation in personalized search system. In Chapter 9, the future work is presented.
Chapter 2

Related Work

There have already been several works done in different research communities including Information Retrieval (IR), Digital Library (DL), Human Computer Interaction (HCI) and Machine Learning (ML). These four research communities have different foci in their research of personalized search. In the information retrieval community, researchers are more focused on modeling and exploiting the useful information to improve retrieval accuracy and building personalized retrieval systems. In digital library community, researchers are more interested in providing personalized information access to individual or groups for online museums, libraries and archives. While in human computer interaction community, researchers are more interested in designing a personalized or customizable interface to the user so that the user can have personalized interaction with the underlying system, which is expected to improve the user experience. In machine learning community, researchers focus on how to apply machine learning techniques to user modeling and intelligent user interface [6, 84].

In a recent workshop in 2002 about challenges in information retrieval and language modeling [5], personalized and contextual search is considered as one of the two grand challenges in information retrieval.

There are some studies of user models about interactive information retrieval. The ASK model [9] and berry-picking model [8] are proposed to model the user behavior of the information seeking. TREC interactive track [37] also study issues of the interaction of information retrieval. Scatter/Gather [24, 36] study a cluster-based document browsing method in the interactive information retrieval. Recently, there are many studies about implicit feedback. In [49], a bibliography of implicit feedback is provided.
Besides works mentioned in [49], there are several additional research work about implicit feedback. In [19], a special web browser *Curious Browser* is developed to record user actions on web pages (implicit feedback) including dwelling time, mouse click, mouse movement, scrolling and elapsed time and user explicit rating of web pages (relevance feedback). The results show that the dwelling time on a page, amount of scrolling on a page and the combination of time and scrolling have a strong correlation with explicit relevance ratings while the individual scrolling methods and mouse clicks are ineffective in predicting explicit interests. In [30], implicit feedback, particularly the combination of clickthrough, dwelling time and how a user exit a result or end a search session, is also found to be associated with user explicit rating. However, in [48], the effect of the task on the display time and potential impact of this relationship on the effectiveness of display time as the implicit feedback is studied. The results show that there is no general direct relationship between display time and usefulness. Moreover, the display time depends on the specific tasks and specific users.

Following the work of studying how clickthrough data can be interpreted as implicit feedback [45], user search logs are partitioned into the query chains, from which relative relevance information is extracted to learn a better ranking formula of a library search system [66]. It is found that using evidence of query chains that is present in search engine logs can learn a better ranking formula compared with the traditional ranking formula and the ranking formula simply considering relative relevance from query logs [44].

In [87], authors use web browsing history in past N days for personalized search. They partition the browsing history data into three categories according to the time stamp, i.e., persistent data (before today), today data (today but before the current session) and current session data. They found that the performance of using web browsing history is competitive with that using relevance feedback.

In [96], from the unobtrusively tracked user interaction with the search system, several implicit feedback models (e.g., binary vote model and Jeffrey’s condition model) are constructed based on the different weights of document representations (e.g. title and query dependent summary of
relevant documents and top-ranked sentences extracted from top-ranked documents) and relevance path. The terms in the implicit feedback models are used to do query expansion.

Our retrieval framework integrates implicit user modeling with the interactive retrieval process, while the previous work either studies implicit user modeling separately from retrieval [19] or only studies specific retrieval models for exploiting implicit feedback to better match a query with documents [96]. Our work will also extend the existing research in several ways as mentioned in Chapter 1, i.e., decision theoretic framework, statistical language models using implicit feedback, and privacy-preserving personalization.

When implicit feedback is not available, user explicit feedback, also known as relevance feedback, sometimes can be solicited. There are also quite a few research works on explicit feedback. Relevance feedback is known to be effective for improving retrieval performance [71, 74, 34]. Previous works on relevance feedback focuses on query updating techniques such as query term reweighting and query expansion. The issue of choosing documents for relevance feedback has not been well addressed. Traditionally, relevance feedback methods just choose the top ranked documents for feedback, which is not necessarily the best strategy from the learning perspective. For example, if the top two documents have identical contents, the learning benefits of these two documents will be nearly equal to that of any one of them. Thus a very interesting research question is how to select appropriate documents for user judgment to maximize the learning benefits, which is the focus of the study in this work.

Our active feedback work is essentially an application of active learning in ad hoc information retrieval. Active learning has been extensively studied in machine learning [72, 91, 20]. It has been applied to text categorization in several previous studies [56, 59, 92], and recently to adaptive information filtering [108]. But there has been little work on applying it to ad hoc retrieval, partly because there are two special challenges in applying active learning to ad hoc retrieval. First, in ad hoc retrieval, we do not have any training examples available to guide the retrieval system for actively selecting the documents for feedback; the query is the only information that can be exploited. Second, it is unclear how we can define an objective function that optimizes ranking
performance rather than classification accuracy. An interesting recent work on applying active learning to ad hoc retrieval is [41], where a user is assumed to iteratively choose clusters, and the active learning task for the system is to design good clusters, a different task from active feedback. The TREC HARD Track [4] has stimulated some recent work along the line of active feedback including [68, 80].

In order to show the effectiveness of personalization techniques, we need to build retrieval systems and apply the personalization techniques into these practical systems. According to the modules of retrieval systems where personalization techniques are applied, there are different personalization strategies. The personalization strategies can be exploited in the document collection [33], query formulation [87], ranking algorithm [66], result representation [27] and feedback [49]. Most existing personalized search works focus on only one strategy, e.g., personalization in ranking. In our work, we take a broader view of personalization and apply personalized strategies in ranking, result representation and relevance feedback.

There are a few existing retrieval systems which apply the personalization techniques. Some representative systems are listed as follows.

SearchPad [10] is an extension of web browser and allows users to explicitly keep track of search context information such as queries submitted, the web pages believed to be relevant and the time of user actions.

In Watson [13] project, it is believed that user interactions with computer applications provide rich context information, which indicates the current user information need and can help provide user just-in-time information access. An application adaptor for each kind of application, e.g., Microsoft Word, is used to get the document internal representation. Watson analyzes documents being edited or viewed and anticipates user current information need from user actions using a task model, which results in information request being sent to external information repository when needed in contrast to Remembrance Project [67], which recommends documents only from local corpus. However, it is not clear what the task model of Watson is and how it is used to infer user information need. A list of results are returned, filtered and then presented to the user.
IntelliZap [29] postulates that search is initiated from a text query marked by the user in a document he is viewing and considers as context a body of words surrounding a user-selected phrase in contrast to existing approaches such as Watson [13], which analyzes the whole documents.

Stuff I’ve Seen [26] integrates and reuses information which have already been created or accessed by the user. The system builds a unified index for information stored in different types (e.g., emails, web pages and Microsoft Word documents) on the local hard disks, which helps the user search these information effectively and efficiently. Furthermore, because there are a lot of meta data on the local disk about the seen information, the search interface (top view or side view) and result representation (rank or date sort) can be tuned for different user preferences. In [90], authors use desktop search index as the user profile for personalized search. They consider the user profile as the implicit feedback and incorporate them into the ranking of web search results.

Different from these existing personalized retrieval systems, our UCAIR search agent is a full-fledged privacy-preserving personalized retrieval system. It takes a broader view of personalization and applied personalization strategies at different stages of interactive information retrieval. Moreover, it does the personalization in the privacy-preserving way.
In general, the retrieval results using the user’s initial query may not be satisfactory; often, the user would need to revise the query to improve the retrieval/ranking accuracy [38]. For a complex or difficult information need, the user may need to modify his query and view ranked documents with many iterations before the information need is completely satisfied. In such an interactive retrieval scenario, the information naturally available to the retrieval system is more than just the current user query and the document collection – in general, all the interaction history can be available to the retrieval system, including past queries, information about which documents the user has chosen to view, and even how a user has read a document (e.g., which part of a document the user spends a lot of time in reading).

To exploit these user context for personalized search in a general way, the retrieval problem is viewed as a decision problem, in which all contextual information and the normally available query and documents should be considered together to optimize the retrieval decision. In general, in response to every user action, the system would choose an optimal system action to take. For example, a user’s action may be submitting a query and the system’s response may be returning a list of 10 document summaries.

An advantage of treating retrieval generally as a decision-making problem is that we may also treat a user’s viewing a document in the search results as a user action, to which the system can respond with updating its own user model about the user’s information need. Although, in this case, such a response does not affect the user immediately, we may imagine that after the user views the document and returns to see more search results, the system can choose to rerank any unseen search results based on the updated user model. Indeed, to bring maximum benefit of context to
the user, we would like to exploit context as soon as it is available and respond immediately based on any new piece of context information.

We propose a decision-theoretic framework for optimizing interactive information retrieval based on eager user model updating [79], in which the system responds to every user action by choosing some system action to optimize a utility function. Specifically, as soon as we observe any new piece of evidence from the user, the system would attempt to perform two tasks: (1) compute the current user model to update its belief about the user’s information need (2) choose a response that minimizes a loss function. For example, immediately after the user views a document, we could use the knowledge that the viewed document summary is probably relevant to rerank the unseen results so as to minimize a loss function that favors a decision to rank relevant documents above irrelevant ones.

In the traditional retrieval paradigm, the retrieval problem is cast as matching a query with documents and rank documents according to their relevance values. As a result, the whole retrieval process is a simple independent cycle of “query submission” and “result display”, which is inadequate for exploiting context. The decision-theoretic framework we propose generalizes this traditional retrieval paradigm and allows us to exploit the user’s search context in a quite general way.

In interactive IR, a user interacts with the retrieval system through an “action dialogue”, in which the system responds to each user action with some system action. For example, the user’s action may be submitting a query and the system’s response may be returning a list of 10 document summaries. In general, the space of user actions and system responses and their granularities would depend on the interface of a particular retrieval system.

In principle, every action of the user can potentially provide new evidence to help the system better infer the user’s information need. Thus in order to respond optimally, the system should use all the evidence collected so far about the user when choosing a response. When viewed in this way, most existing search engines are clearly non-optimal. For example, if a user has viewed some documents on the first page of search results, when the user clicks on the “Next” link to fetch more
results, an existing retrieval system would still return the next page of results retrieved based on the original query without considering the new evidence that a particular result has been viewed by the user.

We propose to optimize retrieval performance by adapting system responses based on every action that a user has taken, and cast the optimization problem as a decision task. Specifically, at any time, the system would attempt to do two tasks: (1) User model updating: Monitor any useful evidence from the user regarding his/her information need and update the user model as soon as such evidence is available; (2) Improving search results: Rerank immediately all the documents that the user has not yet seen, as soon as the user model is updated. We emphasize eager updating and reranking, which makes our work quite different from any existing work. Below we present a formal decision theoretic framework for optimizing retrieval performance through implicit user modeling in interactive information retrieval.

### 3.1 A decision-theoretic Framework

Let $A$ be the set of all user actions and $R(a)$ be the set of all possible system responses to a user action $a \in A$. At any time, let $A_t = (a_1, ..., a_t)$ be the observed sequence of user actions so far (up to time point $t$) and $R_{t-1} = (r_1, ..., r_{t-1})$ be the responses that the system has made responding to the user actions. The system’s goal is to choose an optimal response $r_t \in R(a_t)$ for the current user action $a_t$.

Let $M$ be the space of all possible user models. We further define a loss function $L(a, r, m) \in \mathcal{R}$, where $a \in A$ is a user action, $r \in R(a)$ is a system response, and $m \in M$ is a user model. $L(a, r, m)$ encodes our decision preferences and assesses the optimality of responding with $r$ when the current user model is $m$ and the current user action is $a$. According to Bayesian decision theory, the optimal decision at time $t$ is to choose a response that minimizes the Bayes risk, i.e.,

$$r_t^* = \arg\min_{r \in R(a_t)} \int_M L(a_t, r, m_t) P(m_t | U, D, A_t, R_{t-1}) dm_t$$  \hspace{1cm} (3.1)
where \( P(m_t | U, D, A_t, R_{t-1}) \) is the posterior probability of the user model \( m_t \) given all the observations about the user \( U \) we have made up to time \( t \).

To simplify the computation of Equation 3.1, let us assume that the posterior probability mass \( P(m_t | U, D, A_t, R_{t-1}) \) is mostly concentrated on the mode \( m_t^* = \arg \max_{m_t} P(m_t | U, D, A_t, R_{t-1}) \). We can then approximate the integral with the value of the loss function at \( m_t^* \). That is,

\[
    r_t^* \approx \arg \min_{r \in R(a_t)} L(a_t, r, m_t^*)
\]

(3.2)

where \( m_t^* = \arg \max_{m_t} P(m_t | U, D, A_t, R_{t-1}) \).

Leaving aside how to define and estimate these probabilistic models and the loss function, we can see that such a decision-theoretic formulation suggests that, in order to choose the optimal response to \( a_t \), the system should perform two tasks: (1) compute the current user model and obtain \( m_t^* \) based on all the useful information. (2) choose a response \( r_t \) to minimize the loss function value \( L(a_t, r_t, m_t^*) \). When \( a_t \) does not affect our belief about \( m_t^* \), the first step can be omitted and we may reuse \( m_{t-1}^* \) for \( m_t^* \).

Note that our framework is quite general since we can potentially model any kind of user actions and system responses. In most cases, as we may expect, the system’s response is some ranking of documents, i.e., for most actions \( a \), \( R(a) \) consists of all the possible rankings of the unseen documents, and the decision problem boils down to choosing the best ranking of unseen documents based on the most current user model. When \( a \) is the action of submitting a keyword query, such a response is exactly what a current retrieval system would do. However, we can easily imagine that a more intelligent web search engine would respond to a user’s clicking of the “Next” link (to fetch more unseen results) with a more optimized ranking of documents based on any viewed documents in the current page of results. In fact, according to our eager updating strategy, we may even allow a system to respond to a user’s clicking of browser’s “Back” button after viewing a document in the same way, so that the user can maximally benefit from implicit feedback. These are precisely what our UCAIR system does.
3.2 User Models

A user model $m \in \mathcal{M}$ represents what we know about the user $U$, so in principle, it can contain any information about the user that we wish to model. We now discuss two important components in a user model.

The first component is a component model of the user’s information need. Presumably, the most important factor affecting the optimality of the system’s response is how well the response addresses the user’s information need. Indeed, at any time, we may assume that the system has some “belief” about what the user is interested in, which we model through a term vector $\bar{x} = (x_1, ..., x_{|V|})$, where $V = \{w_1, ..., w_{|V|}\}$ is the set of all terms (i.e., vocabulary) and $x_i$ is the weight of term $w_i$. Such a term vector is commonly used in information retrieval to represent both queries and documents. For example, the vector-space model, assumes that both the query and the documents are represented as term vectors and the score of a document with respect to a query is computed based on the similarity between the query vector and the document vector [75]. In a language modeling approach, we may also regard the query unigram language model [51, 105] or the relevance model [52] as a term vector representation of the user’s information need. Intuitively, $\bar{x}$ would assign high weights to terms that characterize the topics which the user is interested in.

The second component we may include in our user model is the documents that the user has already viewed. Obviously, even if a document is relevant, if the user has already seen the document, it would not be useful to present the same document again. We thus introduce another variable $S \subset \mathcal{D}$ ($\mathcal{D}$ is the whole set of documents in the collection) to denote the subset of documents in the search results that the user has already seen/viewed.

In general, at time $t$, we may represent a user model as $m_t = (S, \bar{x}, A_t, R_{t-1})$, where $S$ is the seen documents, $\bar{x}$ is the system’s “understanding” of the user’s information need, and $(A_t, R_{t-1})$ represents the user’s interaction history. Note that an even more general user model may also include other factors such as the user’s reading level and occupation.

If we assume that the uncertainty of a user model $m_t$ is solely due to the uncertainty of $\bar{x}$,
the computation of our current estimate of user model \( \mathbf{m}_t^* \) will mainly involve computing our best estimate of \( \tilde{x} \). That is, the system would choose a response according to

\[
r_t^* = \arg\min_{r \in R(a_t)} L(a_t, r, S, \tilde{x}^*, A_t, R_{t-1})
\]

where \( \tilde{x}^* = \arg\max_{\tilde{x}} P(\tilde{x}|U, \mathcal{D}, A_t, R_{t-1}) \). This is the decision mechanism implemented in the UCAIR system to be described later. In this system, we avoided specifying the probabilistic model \( P(\tilde{x}|U, \mathcal{D}, A_t, R_{t-1}) \) by computing \( \tilde{x}^* \) directly with some existing feedback method.

### 3.3 Loss Functions

The exact definition of loss function \( L \) depends on the responses, thus it is inevitably application-specific. We now briefly discuss some possibilities when the response is to rank all the unseen documents and present the top \( k \) of them. Let \( r = (d_1, \ldots, d_k) \) be the top \( k \) documents, \( S \) be the set of seen documents by the user, and \( \tilde{x}^* \) be the system’s best guess of the user’s information need. We may simply define the loss associated with \( r \) as the negative sum of the probability that each of the \( d_i \) is relevant, i.e., \( L(a, r, \mathbf{m}) = -\sum_{i=1}^{k} P(\text{relevant}|d_i, \mathbf{m}) \). Clearly, in order to minimize this loss function, the optimal response \( r \) would contain the \( k \) documents with the highest probability of relevance, which is intuitively reasonable.

One deficiency of this “top-k loss function” is that it is not sensitive to the internal order of the selected top \( k \) documents, so switching the ranking order of a non-relevant document and a relevant one would not affect the loss, which is unreasonable. To model ranking, we can introduce a factor of the user model – the probability of each of the \( k \) documents being viewed by the user, \( P(\text{view}|d_i) \), and define the following “ranking loss function”:

\[
L(a, r, \mathbf{m}) = -\sum_{i=1}^{k} P(\text{view}|d_i) P(\text{relevant}|d_i, \mathbf{m})
\]

Since in general, if \( d_i \) is ranked above \( d_j \) (i.e., \( i < j \)), \( P(\text{view}|d_i) > P(\text{view}|d_j) \), this loss function
would favor a decision to rank relevant documents above non-relevant ones, as otherwise, we could always switch \(d_i\) with \(d_j\) to reduce the loss value. Thus the system should simply perform a regular retrieval and rank documents according to the probability of relevance \([69]\).

Depending on the user’s retrieval preferences, there can be many other possibilities. For example, if the user does not want to see redundant documents, the loss function should include some redundancy measure on \(r\) based on the already seen documents \(S\).

Of course, when the response is not to choose a ranked list of documents, we would need a different loss function. We discuss one such example that is relevant to the search agent that we implement. When a user enters a query \(q_t\) (current action), our search agent relies on some existing search engine to actually carry out search. In such a case, even though the search agent does not have control of the retrieval algorithm, it can still attempt to optimize the search results through refining the query sent to the search engine and/or reranking the results obtained from the search engine. The loss functions for reranking are already discussed above; we now take a look at the loss functions for query refinement.

Let \(f\) be the retrieval function of the search engine that our agent uses so that \(f(q)\) would give us the search results using query \(q\). Given that the current action of the user is entering a query \(q_t\) (i.e., \(a_t = q_t\)), our response would be \(f(q)\) for some \(q\). Since we have no choice of \(f\), our decision is to choose a good \(q\). Formally,

\[
\begin{align*}
\mathbf{r}_i^* &= \arg\min_{r} L(a, r_i, \mathbf{m}) \\
&= \arg\min_{f(q)} L(a, f(q), \mathbf{m}) \\
&= f(\arg\min_{q} L(q_t, f(q), \mathbf{m}))
\end{align*}
\]

which shows that our goal is to find \(q^* = \arg\min_{q} L(q_t, f(q), \mathbf{m})\), i.e., an optimal query that would give us the best \(f(q)\). A different choice of loss function \(L(q_t, f(q), \mathbf{m})\) would lead to a different query refinement strategy. In UCAIR, we heuristically compute \(q^*\) by expanding \(q_t\) with terms extracted from \(r_{t-1}\) whenever \(q_{t-1}\) and \(q_t\) have high similarity. Note that \(r_{t-1}\) and \(q_{t-1}\) are
contained in \( m \) as part of the user’s interaction history.

### 3.4 Implicit User Modeling

Implicit user modeling is captured in our framework through the computation of
\[
\tilde{x}^* = \arg\max \tilde{x} P(\tilde{x} | U, D, A_t, R_{t-1}),
\]
i.e., the system’s current belief of what the user’s information need is. Here again there may be many possibilities, leading to different algorithms for implicit user modeling. We now discuss a few of them.

First, when two consecutive queries are related, the previous query can be exploited to enrich the current query and provide more search context to help disambiguation. For this purpose, instead of performing query expansion as we did in the previous section, we could also compute an updated \( \tilde{x}^* \) based on the previous query and retrieval results. The computed new user model can then be used to rank the documents with a standard information retrieval model.

Second, we can also infer a user’s interest based on the summaries of the viewed documents. When a user is presented with a list of summaries of top ranked documents, if the user chooses to skip the first \( n \) documents and to view the \( (n + 1) \)-th document, we may infer that the user is not interested in the displayed summaries for the first \( n \) documents, but is attracted by the displayed summary of the \( (n + 1) \)-th document. We can thus use these summaries as negative and positive examples to learn a more accurate user model \( \tilde{x}^* \). Here many standard relevance feedback techniques can be exploited [71, 74]. Note that we should use the displayed summaries, as opposed to the actual contents of those documents, since it is possible that the displayed summary of the viewed document is relevant, but the document content is actually not. Similarly, a displayed summary may mislead a user to skip a relevant document. Inferring user models based on such displayed information, rather than the actual content of a document is an important difference between UCAIR and some other similar systems.

In UCAIR, both of these strategies for inferring an implicit user model are implemented.
Chapter 4

Language Models for Context-sensitive IR

When instantiating the general decision-theoretic framework described above with specific retrieval methods, we obtain specific retrieval models that can rank documents based on search context. As a case study, we propose several different language models for using implicit feedback information in the same search session to improve retrieval accuracy in interactive information retrieval [78].

We use the KL-divergence retrieval model [105] as a basis and propose to treat context-sensitive retrieval generally as estimating a query language model based on the current query and any search context information. We proposed and tested several statistical language models to incorporate query and clickthrough history into the KL-divergence model, including linear interpolation with fixed coefficients, Bayesian interpolation, Online Bayesian updating and Batch Bayesian updating.

In general, the experiment results show that using implicit feedback information, especially the clickthrough data, can effectively and efficiently improve retrieval performance without requiring additional effort from the user at all [78].

There are two kinds of context information we can use for implicit feedback. One is short-term context, which is the immediate surrounding information which throws light on a user’s current information need in a single session. A session can be considered as a period consisting of all interactions for the same information need. The category of a user’s information need (e.g., kid or sports), previous queries, and recently viewed documents are all examples of short-term context. Such information is most directly related to the current information need of the user and thus can be expected to be most useful for improving the current search. In general, short-term context is most
useful for improving search in the current session, but may not be so helpful for search activities later. The other kind of context is *long-term context*, which refers to information such as a user’s education level and general interest, accumulated user query history and past user clickthrough information; such information is generally stable for a long time and is often accumulated over time. Long-term context can be applicable to all sessions, but may not be as effective as the short-term context in improving search accuracy for a particular session. In this work, we focus on the short-term context, though some of our methods can also be used to naturally incorporate some long-term context.

In a single information session, a user may interact with the search system several times. During interactions, the user would continuously modify the query. Therefore for the current query $Q_k$ (except for the first query of a search session), there is a *query history*, $H_Q = (Q_1, ..., Q_{k-1})$ associated with it that represents the preceding queries given by the same user in the current session. Note that we assume that the session boundaries are known in this work. In practice, we need techniques to automatically discover session boundaries, which have been studied in [39, 86]. Traditionally, the retrieval system only uses the current query $Q_k$ to do retrieval. But the short-term query history may also provide useful clues about the user’s current information need as seen in the “java” example given in the previous section. Indeed, the previous work [81] has shown that the short-term query history is useful for improving retrieval accuracy.

In addition to the query history, there may be other short-term context information available. For example, a user would presumably frequently click some documents to view. We refer to data associated with these actions as *clickthrough history*. The clickthrough data may include the title, summary, and perhaps also the content and location (e.g., the URL) of the clicked document. Although it is not clear whether a viewed document is actually relevant to the user’s information need, we may “safely” assume that the displayed summary/title information about the document is attractive to the user, thus conveys information about the user’s information need. Suppose we concatenate all the displayed text information about a document (usually title and summary) together, we will also have a *clicked summary* $C_i$ in each round of retrieval. In general, we may
have a history of clicked summaries $C_1, \ldots, C_{k-1}$. We will also exploit such clickthrough history $H_C = (C_1, \ldots, C_{k-1})$ to improve our search accuracy for the current query $Q_k$. Previous work has also shown positive results using similar clickthrough information [44].

Both query history and clickthrough history are implicit feedback information, which naturally exists in interactive information retrieval, thus no additional user effort is needed to collect them. In this work, we study how to exploit such information ($H_Q$ and $H_C$), develop models to incorporate the query history and clickthrough history into a retrieval ranking function, and quantitatively evaluate these models.

Intuitively, the query history $H_Q$ and clickthrough history $H_C$ are both useful for improving search accuracy for the current query $Q_k$. An important research question is how we can exploit such information effectively. We propose to use statistical language models to model a user’s information need and develop four specific context-sensitive language models to incorporate context information into a basic retrieval model.

### 4.1 Basic Retrieval Model

We use the Kullback-Leibler (KL) divergence method [105] as our basic retrieval method. According to this model, the retrieval task involves computing a query language model $\theta_Q$ for a given query and a document language model $\theta_D$ for a document and then computing their KL divergence $D(\theta_Q || \theta_D)$, which serves as the score of the document.

One advantage of this approach is that we can naturally incorporate the search context as additional evidence to improve our estimate of the query language model.

Formally, let $H_Q = (Q_1, \ldots, Q_{k-1})$ be the query history and the current query be $Q_k$. Let $H_C = (C_1, \ldots, C_{k-1})$ be the clickthrough history. Note that $C_i$ is the concatenation of all clicked documents’ summaries in the $i$-th round of retrieval since we may reasonably treat all these summaries equally. Our task is to estimate a context query model, which we denote by $p(w|\theta_k)$, based on the current query $Q_k$, as well as the query history $H_Q$ and clickthrough history $H_C$. We now
describe several different language models for exploiting \( H_Q \) and \( H_C \) to estimate \( p(w|\theta_k) \). We will use \( c(w, X) \) to denote the count of word \( w \) in text \( X \), which could be either a query or a clicked document’s summary or any other text. We will use \( |X| \) to denote the length of text \( X \) or the total number of words in \( X \).

### 4.2 Fixed Coefficient Interpolation (FixInt)

Our first idea is to summarize the query history \( H_Q \) with a unigram language model \( p(w|H_Q) \) and the clickthrough history \( H_C \) with another unigram language model \( p(w|H_C) \). Then we linearly interpolate these two history models to obtain the history model \( p(w|H) \). Finally, we interpolate the history model \( p(w|H) \) with the current query model \( p(w|Q_k) \). These models are defined as follows.

\[
\begin{align*}
p(w|Q_i) &= \frac{c(w, Q_i)}{|Q_i|} \\
p(w|H_Q) &= \frac{1}{k-1} \sum_{i=1}^{i=k-1} p(w|Q_i) \\
p(w|C_i) &= \frac{c(w, C_i)}{|C_i|} \\
p(w|H_C) &= \frac{1}{k-1} \sum_{i=1}^{i=k-1} p(w|C_i) \\
p(w|H) &= \beta p(w|H_C) + (1 - \beta) p(w|H_Q) \\
p(w|\theta_k) &= \alpha p(w|Q_k) + (1 - \alpha) p(w|H)
\end{align*}
\]

where \( \beta \in [0, 1] \) is a parameter to control the weight on each history model, and where \( \alpha \in [0, 1] \) is a parameter to control the weight on the current query and the history information.

If we combine these equations, we see that

\[
p(w|\theta_k) = \alpha p(w|Q_k) + (1 - \alpha)[\beta p(w|H_C) + (1 - \beta)p(w|H_Q)]
\]
That is, the estimated context query model is just a fixed coefficient interpolation of three models \( p(w|Q_k), p(w|H_Q), \) and \( p(w|H_C) \).

### 4.3 Bayesian Interpolation (BayesInt)

One possible problem with the FixInt approach is that the coefficients, especially \( \alpha \), are fixed across all the queries. But intuitively, if our current query \( Q_k \) is very long, we should trust the current query more, whereas if \( Q_k \) has just one word, it may be beneficial to put more weight on the history. To capture this intuition, we treat \( p(w|H_Q) \) and \( p(w|H_C) \) as Dirichlet priors and \( Q_k \) as the observed data to estimate a context query model using Bayesian estimator. The estimated model is given by

\[
p(w|\theta_k) = \frac{\alpha(w, Q_k) + \mu p(w|H_Q) + \nu p(w|H_C)}{|Q_k| + \mu + \nu}
\]

\[
= \frac{|Q_k|}{|Q_k| + \mu + \nu} p(w|Q_k) + \frac{\mu + \nu}{|Q_k| + \mu + \nu} \left[ \frac{\mu}{\mu + \nu} p(w|H_Q) + \frac{\nu}{\mu + \nu} p(w|H_C) \right]
\]

where \( \mu \) is the prior sample size for \( p(w|H_Q) \) and \( \nu \) is the prior sample size for \( p(w|H_C) \). We see that the only difference between BayesInt and FixInt is the interpolation coefficients are now adaptive to the query length. Indeed, when viewing BayesInt as FixInt, we see that \( \alpha = \frac{|Q_k|}{|Q_k| + \mu + \nu} \), \( \beta = \frac{\nu}{\nu + \mu} \), thus with fixed \( \mu \) and \( \nu \), we will have a query-dependent \( \alpha \). Later we will show that such an adaptive \( \alpha \) empirically performs better than a fixed \( \alpha \).

### 4.4 Online Bayesian Updating (OnlineUp)

Both FixInt and BayesInt summarize the history information by averaging the unigram language models estimated based on previous queries or clicked summaries. This means that all previous queries are treated equally and so are all clicked summaries. However, as the user interacts with the system and acquires more knowledge about the information in the collection, presumably,
the reformulated queries will become better and better. Thus assigning decaying weights to the
previous queries so as to trust a recent query more than an earlier query appears to be reasonable.
Interestingly, if we incrementally update our belief about the user’s information need after seeing
each query, we would naturally impose decaying weights on the previous queries. Since such an
incremental online updating strategy can be used to exploit any evidence in an interactive retrieval
system, we present it in a more general way.

In a typical retrieval system, the retrieval system responds to every new query entered by the
user by presenting a ranked list of documents. In order to rank documents, the system must have
some model for the user’s information need. In the KL divergence retrieval model, this means that
the system must compute a query model whenever a user enters a (new) query. A principled way
of updating the query model is to use Bayesian estimation, which we discuss below.

4.4.1 Bayesian Updating

We first discuss how we apply Bayesian estimation to update a query model in general. Let \( p(w|\phi) \)
be our current query model and \( T \) be a new piece of text evidence observed (e.g., \( T \) can be a query
or a clicked summary). To update the query model based on \( T \), we use \( \phi \) to define a Dirichlet prior
parameterized as

\[
Dir(\mu_T p(w_1|\phi), ..., \mu_T p(w_N|\phi))
\]

where \( \mu_T \) is the equivalent sample size of the prior. We use Dirichlet prior because it is a conjugate
prior for multinomial distributions. With such a conjugate prior, the predictive distribution of \( \phi \) (or
equivalently, the mean of the posterior distribution of \( \phi \) is given by

\[
p(w|\phi) = \frac{c(w, T) + \mu_T p(w|\phi)}{|T| + \mu_T} \quad (4.1)
\]

where \( c(w, T) \) is the counts of \( w \) in \( T \) and \( |T| \) is the length of \( T \). Parameter \( \mu_T \) indicates our
confidence in the prior expressed in terms of an equivalent text sample comparable with \( T \). For
example, $\mu_T = 1$ indicates that the influence of the prior is equivalent to adding one extra word to $T$.

### 4.4.2 Sequential Query Model Updating

We now discuss how we can update our query model over time during an interactive retrieval process using Bayesian estimation. In general, we assume that the retrieval system maintains a current query model $\phi_i$ at any moment. As soon as we obtain some implicit feedback evidence in the form of a piece of text $T_i$, we will update the query model.

Initially, before we see any user query, we may already have some information about the user. For example, we may have some information about what documents the user has viewed in the past. We use such information to define a prior on the query model, which is denoted by $\phi'_0$. After we observe the first query $Q_1$, we can update the query model based on the new observed data $Q_1$. The updated query model $\phi_1$ can then be used for ranking documents in response to $Q_1$. As the user views some documents, the displayed summary text for such documents $C_1$ (i.e., clicked summaries) can serve as some new data for us to further update the query model to obtain $\phi'_1$. As we obtain the second query $Q_2$ from the user, we can update $\phi'_1$ to obtain a new model $\phi_2$. In general, we may repeat such an updating process to iteratively update the query model.

Clearly, we see two types of updating: (1) updating based on a new query $Q_i$; (2) updating based on a new clicked summary $C_i$. In both cases, we can treat the current model as a prior of the context query model and treat the new observed query or clicked summary as observed data. Thus we have following updating equations:

$$p(w|\phi_i) = \frac{c(w,Q_i) + \mu_i p(w|\phi'_{i-1})}{|Q_i| + \mu_i}$$

$$p(w|\phi'_i) = \frac{c(w,C_i) + \nu_i p(w|\phi_i)}{|C_i| + \nu_i}$$

where $\mu_i$ is the equivalent sample size for the prior when updating the model based on a query, while $\nu_i$ is the equivalent sample size for the prior when updating the model based on a clicked
summary. If we set $\mu_i = 0$ (or $\nu_i = 0$) we essentially ignore the prior model, thus would start a completely new query model based on the query $Q_i$ (or the clicked summary $C_i$). On the other hand, if we set $\mu_i = +\infty$ (or $\nu_i = +\infty$) we essentially ignore the observed query (or the clicked summary) and do not update our model. Thus the model remains the same as if we do not observe any new text evidence. In general, the parameters $\mu_i$ and $\nu_i$ may have different values for different $i$. For example, at the very beginning, we may have very sparse query history, thus we could use a smaller $\mu_i$, but later as the query history is richer, we can consider using a larger $\mu_i$. But in our experiments, we set them to the same constants, i.e., $\forall i, j, \mu_i = \mu_j, \nu_i = \nu_j$.

Note that we can take either $p(w|\phi_i)$ or $p(w|\phi'_i)$ as our context query model for ranking documents. This suggests that we do not have to wait until a user enters a new query to initiate a new round of retrieval; instead, as soon as we collect clicked summary $C_i$, we can update the query model and use $p(w|\phi'_i)$ to immediately rerank any documents that a user has not yet seen.

To score documents after seeing query $Q_k$, we use $p(w|\phi_k)$, i.e.,

$$p(w|\theta_k) = p(w|\phi_k)$$

### 4.5 Batch Bayesian Updating (BatchUp)

The OnlineUp algorithm introduces a decaying factor – repeated interpolation would cause the early data to have a low weight. This may be appropriate for the query history as it is reasonable to believe that the user becomes better and better at query formulation as time goes on, but it not necessarily appropriate for the clickthrough information, especially because we use the displayed summary, rather than the actual content of a clicked document. One way to avoid applying a decaying interpolation to the clickthrough data is to do OnlineUp only for the query history $Q = (Q_1, ..., Q_{i-1})$, but not for the clickthrough data $C$. We first buffer all the clickthrough data together and use the whole chunk of clickthrough data to update the model generated through running OnlineUp on previous queries. The updating equations are as follows.
where $\mu_i$ has the same interpretation as in OnlineUp, but $\nu_i$ now indicates to what extent we want to trust the clicked summaries. As in OnlineUp, we set all $\mu_i$’s and $\nu_i$’s to the same value. And to rank documents after seeing the current query $Q_k$, we use

$$p(w|\theta_k) = p(w|\psi_k)$$

### 4.6 Evaluation of Context-sensitive Retrieval Models

#### 4.6.1 Data Collection

In order to quantitatively evaluate our models, we need a data set which includes not only a text database and testing topics, but also query history and clickthrough history for each topic. Since there is no such data set available to us, we have to create one. There are two choices. One is to extract topics and any associated query history and clickthrough history for each topic from the log of a retrieval system (e.g., search engine). But the problem is that we have no relevance judgments on such data. The other choice is to use a TREC data set, which has a text database, topic description and relevance judgment file. Unfortunately, there are no query history and clickthrough history data. We decide to augment a TREC data set by collecting query history and clickthrough history data.

We select TREC AP88, AP89 and AP90 data as our text database, because AP data has been used in several TREC tasks and has relatively complete judgments. There are altogether 242918 news articles and the average document length is 416 words. Most articles have titles. If not, we
select the first sentence of the text as the title. For the preprocessing, we only do case folding and do not do stopword removal or stemming.

We select 30 relatively difficult topics from TREC topics 1-150. These 30 topics have the worst average precision performance among TREC topics 1-150 according to some baseline experiments using the KL-Divergence model with Bayesian prior smoothing [106]. The reason why we select difficult topics is that the user then would have to have several interactions with the retrieval system in order to get satisfactory results so that we can expect to collect a relatively richer query history and clickthrough history data from the user. In real applications, we may also expect our models to be most useful for such difficult topics, so our data collection strategy reflects the real world applications well.

We index the TREC AP data set and set up a search engine and web interface for TREC AP news articles. We use 3 subjects to do experiments to collect query history and clickthrough history data. Each subject is assigned 10 topics and given the topic descriptions provided by TREC. For each topic, the first query is the title of the topic given in the original TREC topic description. After the subject submits the query, the search engine will do retrieval and return a ranked list of search results to the subject. The subject will browse the results and maybe click one or more results to browse the full text of article(s). The subject may also modify the query to do another search. For each topic, the subject composes at least 4 queries. In our experiment, only the first 4 queries for each topic are used. The user needs to select the topic number from a selection menu before submitting the query to the search engine so that we can easily detect the session boundary, which is not the focus of our study. We use a relational database to store user interactions, including the submitted queries and clicked documents. For each query, we store the query terms and the associated result pages. And for each clicked document, we store the summary as shown on the search result page. The summary of the article is query dependent and is computed online using fixed-length passage retrieval (KL divergence model with Bayesian prior smoothing).

Among 120 (4 for each of 30 topics) queries which we study in the experiment, the average query length is 3.71 words. Altogether there are 91 documents clicked to view. So on average,
there are around 3 clicks per topic. The average length of clicked summary is 34.4 words. Among 91 clicked documents, 29 documents are judged relevant according to TREC judgment file. This data set is publicly available \(^1\).

### 4.6.2 Experiment Design

Our major hypothesis is that using search context (i.e., query history and clickthrough information) can help improve search accuracy. In particular, the search context can provide extra information to help us estimate a better query model than using just the current query. So most of our experiments involve comparing the retrieval performance using the current query only (thus ignoring any context) with that using the current query as well as the search context.

Since we collected four versions of queries for each topic, we make such comparisons for each version of queries. We use two performance measures: (1) Mean Average Precision (MAP): This is the standard non-interpolated average precision and serves as a good measure of the overall ranking accuracy. (2) Precision at 20 documents (pr@20docs): This measure does not average well, but it is more meaningful than MAP and reflects the utility for users who only read the top 20 documents. In all cases, the reported figure is the average over all of the 30 topics.

We evaluate the four models for exploiting search context (i.e., FixInt, BayesInt, OnlineUp, and BatchUp). Each model has precisely two parameters ($\alpha$ and $\beta$ for FixInt; $\mu$ and $\nu$ for others). Note that $\mu$ and $\nu$ may need to be interpreted differently for different methods. We vary these parameters and identify the optimal performance for each method. We also vary the parameters to study the sensitivity of our algorithms to the setting of the parameters.

### 4.6.3 Overall Effect of Search Context

We compare the optimal performances of four models with those using the current query only in Table 4.1. A row labeled with $q_i$ is the baseline performance and a row labeled with $q_i + H_Q + H_C$.

\(^1\)http://sifaka.cs.uiuc.edu/ir/ucair/QCHistory.zip
Table 4.1: Effect of using query history and clickthrough data for document ranking.

<table>
<thead>
<tr>
<th>Query</th>
<th>FixInt (α = 0.1, β = 1.0)</th>
<th>BayesInt (μ = 0.2, ν = 5.0)</th>
<th>OnlineUp (μ = 5.0, ν = 15.0)</th>
<th>BatchUp (μ = 2.0, ν = 15.0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAP</td>
<td>pr@20docs</td>
<td>MAP</td>
<td>pr@20docs</td>
</tr>
<tr>
<td>q1</td>
<td>0.0095</td>
<td>0.0317</td>
<td>0.0095</td>
<td>0.0317</td>
</tr>
<tr>
<td>q2</td>
<td>0.0312</td>
<td>0.1150</td>
<td>0.0312</td>
<td>0.1150</td>
</tr>
<tr>
<td>q2 + H_Q + H_C</td>
<td>0.0324</td>
<td>0.1117</td>
<td>0.0345</td>
<td>0.1117</td>
</tr>
<tr>
<td>Improve</td>
<td>3.8%</td>
<td>-2.9%</td>
<td>10.6%</td>
<td>-2.9%</td>
</tr>
<tr>
<td>q3</td>
<td>0.0421</td>
<td>0.1483</td>
<td>0.0421</td>
<td>0.1483</td>
</tr>
<tr>
<td>q3 + H_Q + H_C</td>
<td>0.0726</td>
<td>0.1967</td>
<td>0.0816</td>
<td>0.2067</td>
</tr>
<tr>
<td>Improve</td>
<td>72.4%</td>
<td>32.6%</td>
<td>93.8%</td>
<td>39.4%</td>
</tr>
<tr>
<td>q4</td>
<td>0.0536</td>
<td>0.1933</td>
<td>0.0536</td>
<td>0.1933</td>
</tr>
<tr>
<td>q4 + H_Q + H_C</td>
<td>0.0891</td>
<td>0.2233</td>
<td>0.0955</td>
<td>0.2317</td>
</tr>
<tr>
<td>Improve</td>
<td>66.2%</td>
<td>15.5%</td>
<td>78.2%</td>
<td>19.9%</td>
</tr>
</tbody>
</table>

is the performance of using search context. We can make several observations from this table:

1. Comparing the baseline performances indicates that on average reformulated queries are better than the previous queries with the performance of q4 being the best. Users generally formulate better and better queries.

2. Using search context generally has positive effect, especially when the search context is rich. This can be seen from the fact that the improvement over q4 and q3 is generally more substantial compared with q2. Actually, in many cases with q2, using the search context may hurt the performance, probably because the history at that point is sparse. When the search context is rich, the performance improvement can be quite substantial. For example, BatchUp achieves 92.4% improvement in the mean average precision over q3 and 77.2% improvement over q4. (The generally low precisions also make the relative improvement deceptively high, though.)

3. Among four models using search context, the performances of FixInt and OnlineUp are clearly worse than those of BayesInt and BatchUp. Since BayesInt performs better than FixInt and the main difference between FixInt and BayesInt is that the latter uses an adaptive coefficient for interpolation, the results suggest that using adaptive coefficient is quite beneficial and a Bayesian style interpolation makes sense. The main difference between OnlineUp and BatchUp is that OnlineUp uses decaying coefficients to combine the multiple clicked summaries, while BatchUp
simply concatenates all clicked summaries. Therefore the fact that BatchUp is consistently better than OnlineUp indicates that the weights for combining the clicked summaries indeed should not be decaying. While OnlineUp is theoretically appealing, its performance is inferior to BayesInt and BatchUp, likely because of the decaying coefficient. Overall, BatchUp appears to be the best method when we vary the parameter settings.

Table 4.2: Effect of using query history only for document ranking.

<table>
<thead>
<tr>
<th>Query</th>
<th>FixInt $(\alpha = 0.1, \beta = 0)$</th>
<th>BayesInt $(\mu = 0.2, \nu = 0)$</th>
<th>OnlineUp $(\mu = 5.0, \nu = +\infty)$</th>
<th>BatchUp $(\mu = 2.0, \nu = +\infty)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MAP pr@20docs</td>
<td>MAP pr@20docs</td>
<td>MAP pr@20docs</td>
<td>MAP pr@20docs</td>
</tr>
<tr>
<td>$q_2$</td>
<td>0.0312 0.1150</td>
<td>0.0312 0.1150</td>
<td>0.0312 0.1150</td>
<td>0.0312 0.1150</td>
</tr>
<tr>
<td>$q_2 + H_Q$</td>
<td>0.0097 0.0317</td>
<td>0.0311 0.1200</td>
<td>0.0213 0.0783</td>
<td>0.0287 0.0967</td>
</tr>
<tr>
<td>Improve</td>
<td>-68.9% -72.4%</td>
<td>-0.3% 4.3%</td>
<td>-31.7% -31.9%</td>
<td>-8.0% -15.9%</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0.0421 0.1483</td>
<td>0.0421 0.1483</td>
<td>0.0421 0.1483</td>
<td>0.0421 0.1483</td>
</tr>
<tr>
<td>$q_3 + H_Q$</td>
<td>0.0261 0.0917</td>
<td>0.0451 0.1517</td>
<td>0.0444 0.1333</td>
<td>0.0455 0.1450</td>
</tr>
<tr>
<td>Improve</td>
<td>-38.2% -38.2%</td>
<td>7.1% 2.3%</td>
<td>5.5% -10.1%</td>
<td>8.1% -2.2%</td>
</tr>
<tr>
<td>$q_4$</td>
<td>0.0536 0.1933</td>
<td>0.0536 0.1933</td>
<td>0.0536 0.1933</td>
<td>0.0536 0.1933</td>
</tr>
<tr>
<td>$q_4 + H_Q$</td>
<td>0.0428 0.1467</td>
<td>0.0537 0.1917</td>
<td>0.0550 0.1733</td>
<td>0.0552 0.1917</td>
</tr>
<tr>
<td>Improve</td>
<td>-20.1% -24.1%</td>
<td>0.2% -0.8%</td>
<td>3.0% -10.3%</td>
<td>3.0% -0.8%</td>
</tr>
</tbody>
</table>

Table 4.3: Average precision of BatchUp using query history only

We have two different kinds of search context – query history and clickthrough data. We now look into the contribution of each kind of context.

4.6.4 Using Query History Only

In each of four models, we can “turn off” the clickthrough history data by setting parameters appropriately. This allows us to evaluate the effect of using query history alone. We use the same parameter setting for query history as in Table 4.1. The results are shown in Table 4.2. Here we see that in general, the benefit of using query history is very limited with mixed results. This is
different from what is reported in a previous study [81], where using query history is consistently helpful. Another observation is that the context runs perform poorly at $q_2$, but generally perform (slightly) better than the baselines for $q_3$ and $q_4$. This is again likely because at the beginning the initial query, which is the title in the original TREC topic description, may not be a good query; indeed, on average, performances of these “first-generation” queries are clearly poorer than those of all other user-formulated queries in in the later generations. Yet another observation is that when using query history only, the BayesInt model appears to be better than other models. Since the clickthrough data is ignored, OnlineUp and BatchUp are essentially the same algorithm. The displayed results thus reflect the variation caused by parameter $\mu$. A smaller setting of 2.0 is seen better than a larger value of 5.0. A more complete picture of the influence of the setting of $\mu$ can be seen from Table 4.3, where we show the performance figures for a wider range of values of $\mu$.

The value of $\mu$ can be interpreted as how many words we regard the query history is worth. A larger value thus puts more weight on the history and is seen to hurt the performance more when the history information is not rich. Thus while for $q_4$ the best performance tends to be achieved for $\mu \in [2, 5]$, only when $\mu = 0.5$ we see some small benefit for $q_2$. As we would expect, an excessively large $\mu$ would hurt the performance in general, but $q_2$ is hurt most and $q_4$ is barely hurt, indicating that as we accumulate more and more query history information, we can put more and

Table 4.4: Effect of using clickthrough data only for document ranking.

<table>
<thead>
<tr>
<th>Query</th>
<th>$\alpha = 0.1, \beta = 1$</th>
<th>$\mu = 0, \nu = 5.0$</th>
<th>$\mu_i - \mu_{i-1} = +\infty, \mu_i = 5.0, \nu = 15$</th>
<th>$\mu_i = 0, \nu = 15$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_2$</td>
<td>0.0312 0.1150</td>
<td>0.0312 0.1150</td>
<td>0.0312 0.1150</td>
<td>0.0312 0.1150</td>
</tr>
<tr>
<td>$q_2 + H_C$</td>
<td>0.0324 0.1117</td>
<td>0.0338 0.1133</td>
<td>0.0358 0.1300</td>
<td>0.0344 0.1167</td>
</tr>
<tr>
<td>Improve</td>
<td>3.8% -2.9%</td>
<td>8.3% -1.5%</td>
<td>14.7% 13.0%</td>
<td>10.3% 1.5%</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0.0421 0.1483</td>
<td>0.0421 0.1483</td>
<td>0.0421 0.1483</td>
<td>0.0420 0.1483</td>
</tr>
<tr>
<td>$q_3 + H_C$</td>
<td>0.0726 0.1967</td>
<td>0.0766 0.2033</td>
<td>0.0622 0.1767</td>
<td>0.0513 0.1650</td>
</tr>
<tr>
<td>Improve</td>
<td>72.4% 32.6%</td>
<td>81.9% 37.1%</td>
<td>47.7% 19.2%</td>
<td>21.9% 11.3%</td>
</tr>
<tr>
<td>$q_4$</td>
<td>0.0536 0.1930</td>
<td>0.0536 0.1930</td>
<td>0.0536 0.1930</td>
<td>0.0536 0.1930</td>
</tr>
<tr>
<td>$q_4 + H_C$</td>
<td>0.0891 0.2233</td>
<td>0.0925 0.2283</td>
<td>0.0772 0.2217</td>
<td>0.0623 0.2050</td>
</tr>
<tr>
<td>Improve</td>
<td>66.2% 15.5%</td>
<td>72.6% 18.1%</td>
<td>44.0% 14.7%</td>
<td>16.2% 6.1%</td>
</tr>
</tbody>
</table>

The value of $\mu$ can be interpreted as how many words we regard the query history is worth. A larger value thus puts more weight on the history and is seen to hurt the performance more when the history information is not rich. Thus while for $q_4$ the best performance tends to be achieved for $\mu \in [2, 5]$, only when $\mu = 0.5$ we see some small benefit for $q_2$. As we would expect, an excessively large $\mu$ would hurt the performance in general, but $q_2$ is hurt most and $q_4$ is barely hurt, indicating that as we accumulate more and more query history information, we can put more and
more weight on the history information. This also suggests that a better strategy should probably dynamically adjust parameters according to how much history information we have.

The mixed query history results suggest that the positive effect of using implicit feedback information may have largely come from the use of clickthrough history, which is indeed true as we discuss in the next subsection.

4.6.5 Using Clickthrough History Only

We now turn off the query history and only use the clicked summaries plus the current query. The results are shown in Table 4.4. We see that the benefit of using clickthrough information is much more significant than that of using query history. We see an overall positive effect, often with significant improvement over the baseline. It is also clear that the richer the context data is, the more improvement using clicked summaries can achieve. Other than some occasional degradation of precision at 20 documents, the improvement is fairly consistent and often quite substantial.

These results show that the clicked summary text is in general quite useful for inferring a user’s information need. Intuitively, using the summary text, rather than the actual content of the document, makes more sense, as it is quite possible that the document behind a seemingly relevant summary is actually non-relevant.

29 out of the 91 clicked documents are relevant. Updating the query model based on such summaries would bring up the ranks of these relevant documents, causing performance improvement. However, such improvement is really not beneficial for the user as the user has already seen these relevant documents. To see how much improvement we have achieved on improving the ranks of the unseen relevant documents, we exclude these 29 relevant documents from our judgment file and recompute the performance of BayesInt and the baseline using the new judgment file. The results are shown in Table 4.5. Note that the performance of the baseline method is lower due to the removal of the 29 relevant documents, which would have been generally ranked high in the results. From Table 4.5, we see clearly that using clicked summaries also helps improve the ranks of unseen relevant documents significantly.
Table 4.5: BayesInt evaluated on unseen relevant documents

<table>
<thead>
<tr>
<th>Query</th>
<th>BayesInt($\mu = 0, \nu = 5, 0$)</th>
<th>MAP</th>
<th>pr@20docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_2$</td>
<td>0.0263</td>
<td>0.100</td>
<td></td>
</tr>
<tr>
<td>$q_2 + H_C$</td>
<td>0.0314</td>
<td>0.100</td>
<td></td>
</tr>
<tr>
<td>Improve</td>
<td>19.4%</td>
<td>0%</td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td>0.0331</td>
<td>0.125</td>
<td></td>
</tr>
<tr>
<td>$q_3 + H_C$</td>
<td>0.0661</td>
<td>0.178</td>
<td></td>
</tr>
<tr>
<td>Improve</td>
<td>99.7%</td>
<td>42.4%</td>
<td></td>
</tr>
<tr>
<td>$q_4$</td>
<td>0.0442</td>
<td>0.165</td>
<td></td>
</tr>
<tr>
<td>$q_4 + H_C$</td>
<td>0.0739</td>
<td>0.188</td>
<td></td>
</tr>
<tr>
<td>Improve</td>
<td>67.2%</td>
<td>13.9%</td>
<td></td>
</tr>
</tbody>
</table>

One remaining question is whether the clickthrough data is still helpful if none of the clicked documents is relevant. To answer this question, we took out the 29 relevant summaries from our clickthrough history data $H_C$ to obtain a smaller set of clicked summaries $H'_C$, and re-evaluated the performance of the BayesInt method using $H'_C$ with the same setting of parameters as in Table 4.4. The results are shown in Table 4.6. We see that although the improvement is not as substantial as in Table 4.4, the average precision is improved across all generations of queries. These results should be interpreted as very encouraging as they are based on only 62 non-relevant clickthroughs. In reality, a user would more likely click some relevant summaries, which would help bring up more relevant documents as we have seen in Table 4.4 and Table 4.5.

### 4.6.6 Additive Effect of Context Information

By comparing the results across Table 4.1, Table 4.2 and Table 4.4, we can see that the benefit of the query history information and that of clickthrough information are “additive”, i.e., combining them can achieve better performance than using each alone. In Table 4.7, we show this effect for the BatchUp method.
### 4.6.7 Parameter Sensitivity

All four methods have two parameters, which control the relative weights when combining $H_Q$, $H_C$, and $Q_k$, though the parameterization is different from model to model. In this subsection, we study the parameter sensitivity for BatchUp, which appears to perform relatively better than others. BatchUp has two parameters $\mu$ and $\nu$.

We first look at $\mu$. When $\mu$ is set to 0, the query history is not used at all, and we essentially just use the clickthrough data combined with the current query. If we increase $\mu$, we will gradually incorporate more information from the previous queries. In Table 4.8, we show how the average precision of BatchUp changes as we vary $\mu$ with $\nu$ fixed to 15.0, where the best performance of BatchUp is achieved. We see that the performance is relatively insensitive to the change of $\mu$. The pattern is also similar when we set $\nu$ to other values. This low sensitivity may be because the performance is dominated by the influence from the clickthrough data, i.e., if $\nu$ is fixed to a relatively optimal value, the small differences caused by $\mu$ may not stand out. However, the best performance is generally achieved when $\mu$ is around 2.0, which means that the past query information is as useful as about 2 words in the current query. In general, there is some tradeoff between the current query and the previous queries and using a balanced combination of past queries and the current query achieves better performance than using each of them alone.
<table>
<thead>
<tr>
<th>Query</th>
<th>MAP</th>
<th>pr@20docs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_2$</td>
<td>0.0312</td>
<td>0.1150</td>
</tr>
<tr>
<td>$q_2 + H_Q$</td>
<td>0.0287</td>
<td>0.0967</td>
</tr>
<tr>
<td>Improve</td>
<td>-8.0%</td>
<td>-15.9%</td>
</tr>
<tr>
<td>$q_2 + H_C$</td>
<td>0.0344</td>
<td>0.1167</td>
</tr>
<tr>
<td>Improve</td>
<td>10.3%</td>
<td>1.5%</td>
</tr>
<tr>
<td>$q_2 + H_Q + H_C$</td>
<td>0.0342</td>
<td>0.1100</td>
</tr>
<tr>
<td>Improve</td>
<td>9.6%</td>
<td>-4.3%</td>
</tr>
<tr>
<td>$q_3$</td>
<td>0.0421</td>
<td>0.1483</td>
</tr>
<tr>
<td>$q_3 + H_Q$</td>
<td>0.0455</td>
<td>0.1450</td>
</tr>
<tr>
<td>Improve</td>
<td>8.1%</td>
<td>-2.2%</td>
</tr>
<tr>
<td>$q_3 + H_C$</td>
<td>0.0513</td>
<td>0.1650</td>
</tr>
<tr>
<td>Improve</td>
<td>21.9%</td>
<td>11.3%</td>
</tr>
<tr>
<td>$q_3 + H_Q + H_C$</td>
<td>0.0810</td>
<td>0.2067</td>
</tr>
<tr>
<td>Improve</td>
<td>92.4%</td>
<td>39.4%</td>
</tr>
<tr>
<td>$q_4$</td>
<td>0.0536</td>
<td>0.1930</td>
</tr>
<tr>
<td>$q_4 + H_Q$</td>
<td>0.0552</td>
<td>0.1917</td>
</tr>
<tr>
<td>Improve</td>
<td>3.0%</td>
<td>-0.8%</td>
</tr>
<tr>
<td>$q_4 + H_C$</td>
<td>0.0623</td>
<td>0.2050</td>
</tr>
<tr>
<td>Improve</td>
<td>16.2%</td>
<td>6.1%</td>
</tr>
<tr>
<td>$q_4 + H_Q + H_C$</td>
<td>0.0950</td>
<td>0.2250</td>
</tr>
<tr>
<td>Improve</td>
<td>77.2%</td>
<td>16.4%</td>
</tr>
</tbody>
</table>

Table 4.7: Additive benefit of context information

We now turn to the other parameter $\nu$. When $\nu$ is set to 0, we only use the clickthrough data; when $\nu$ is set to $+\infty$, we only use the query history and the current query. With $\mu$ set to 2.0, where the best performance of BatchUp is achieved, we vary $\nu$ and show the results in Table 4.9. We see that the performance is also not very sensitive when $\nu \leq 30$, with the best performance often achieved at $\nu = 15$. This means that the combined information of query history and the current query is as useful as about 15 words in the clickthrough data, indicating that the clickthrough information is highly valuable.

Overall, these sensitivity results show that BatchUp not only performs better than other methods, but also is quite robust.
### Table 4.8: Sensitivity of $\mu$ in BatchUp

<table>
<thead>
<tr>
<th>$q_2 + H_Q + H_C$</th>
<th>$\mu$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP</td>
<td>0.039</td>
<td>0.037</td>
<td>0.034</td>
<td>0.032</td>
<td>0.029</td>
<td>0.027</td>
<td>0.025</td>
<td>0.024</td>
<td>0.023</td>
<td>0.022</td>
<td>0.022</td>
<td></td>
</tr>
<tr>
<td>pr@20</td>
<td>0.133</td>
<td>0.123</td>
<td>0.110</td>
<td>0.103</td>
<td>0.102</td>
<td>0.093</td>
<td>0.093</td>
<td>0.077</td>
<td>0.078</td>
<td>0.077</td>
<td>0.075</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$q_3 + H_Q + H_C$</th>
<th>$\mu$</th>
<th>0.081</th>
<th>0.081</th>
<th>0.081</th>
<th>0.081</th>
<th>0.081</th>
<th>0.081</th>
<th>0.081</th>
<th>0.080</th>
<th>0.079</th>
<th>0.079</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr@20</td>
<td>0.210</td>
<td>0.215</td>
<td>0.207</td>
<td>0.205</td>
<td>0.207</td>
<td>0.205</td>
<td>0.207</td>
<td>0.207</td>
<td>0.205</td>
<td>0.202</td>
<td>0.200</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$q_4 + H_Q + H_C$</th>
<th>$\mu$</th>
<th>0.093</th>
<th>0.095</th>
<th>0.095</th>
<th>0.094</th>
<th>0.094</th>
<th>0.094</th>
<th>0.094</th>
<th>0.094</th>
<th>0.093</th>
<th>0.093</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr@20</td>
<td>0.218</td>
<td>0.222</td>
<td>0.225</td>
<td>0.223</td>
<td>0.227</td>
<td>0.228</td>
<td>0.233</td>
<td>0.233</td>
<td>0.235</td>
<td>0.235</td>
<td>0.233</td>
</tr>
</tbody>
</table>

### Table 4.9: Sensitivity of $\nu$ in BatchUp

<table>
<thead>
<tr>
<th>$q_2 + H_Q + H_C$</th>
<th>$\nu$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>30</th>
<th>100</th>
<th>300</th>
<th>500</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP</td>
<td>0.028</td>
<td>0.028</td>
<td>0.030</td>
<td>0.032</td>
<td>0.033</td>
<td>0.034</td>
<td>0.033</td>
<td>0.033</td>
<td>0.030</td>
<td>0.029</td>
<td></td>
</tr>
<tr>
<td>pr@20</td>
<td>0.093</td>
<td>0.095</td>
<td>0.095</td>
<td>0.100</td>
<td>0.105</td>
<td>0.110</td>
<td>0.115</td>
<td>0.098</td>
<td>0.097</td>
<td>0.097</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$q_3 + H_Q + H_C$</th>
<th>$\nu$</th>
<th>0.073</th>
<th>0.074</th>
<th>0.075</th>
<th>0.079</th>
<th>0.081</th>
<th>0.081</th>
<th>0.077</th>
<th>0.063</th>
<th>0.051</th>
<th>0.049</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr@20</td>
<td>0.192</td>
<td>0.193</td>
<td>0.195</td>
<td>0.210</td>
<td>0.200</td>
<td>0.207</td>
<td>0.202</td>
<td>0.178</td>
<td>0.160</td>
<td>0.155</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$q_4 + H_Q + H_C$</th>
<th>$\nu$</th>
<th>0.090</th>
<th>0.090</th>
<th>0.091</th>
<th>0.093</th>
<th>0.094</th>
<th>0.095</th>
<th>0.092</th>
<th>0.076</th>
<th>0.066</th>
<th>0.063</th>
</tr>
</thead>
<tbody>
<tr>
<td>pr@20</td>
<td>0.227</td>
<td>0.223</td>
<td>0.228</td>
<td>0.232</td>
<td>0.223</td>
<td>0.225</td>
<td>0.228</td>
<td>0.220</td>
<td>0.207</td>
<td>0.203</td>
<td></td>
</tr>
</tbody>
</table>
A client-side search agent (called UCAIR) embedded in a web browser has been developed, which can capture a user’s search context and perform implicit feedback [79]. As shown in Figure 5.1, the UCAIR search agent has 3 major components: (1) The (implicit) user modeling module captures a user’s search context and history information and infers search session boundaries. Currently the user modeling module captures the submitted queries and any clicked search results. (2) The query modification module selectively improves the query formulation according to the current user model. (3) The result re-ranking module immediately re-ranks any unseen search results whenever the user model is updated. The UCAIR search agent incorporates models and algorithms proposed in Section 4 to dynamically rerank the search results to reflect the most updated knowledge of the user’s information need whenever any new piece of implicit feedback becomes available.

We chose to do context-sensitive IR at the client side instead of the server side as it has three remarkable advantages. First, the user does not need to worry about privacy infringement, which is a big concern for personalized search [95]. Second, a richer category of user interactions such as mouse movement can be easily captured for implicit feedback. Third, the computation needed for personalization and the storage of the user profile are both done at the client side, so the server is not burdened [42].

Specific techniques are implemented to capture and exploit two types of implicit feedback information: (1) identifying any related immediately preceding query and using the query and its corresponding search results to select appropriate terms to expand the current query, and (2) exploiting the viewed document summaries to dynamically rerank any document that has not yet
been seen by the user.

User studies show that the UCAIR search agent improves performance over a popular search engine (Google), on which UCAIR search agent is built.

5.1 Design

In this section, we present a client-side web search agent called UCAIR, in which we implement some of the methods discussed in the previous section for performing personalized search through implicit user modeling. UCAIR is a web browser plug-in \(^1\) that acts as a proxy for web search engines. Currently, it is only implemented for Internet Explorer and Google, but it is a matter of engineering to make it run on other web browsers and interact with other search engines.

The issue of privacy is a primary obstacle for deploying any real world applications involving serious user modeling, such as personalized search. For this reason, UCAIR is strictly running as a client-side search agent, as opposed to a server-side application. This way, the captured user information always resides on the computer that the user is using, thus the user does not need to release any information to the outside. Client-side personalization also allows the system to easily observe a lot of user information that may not be easily available to a server. Furthermore, performing personalized search on the client-side is more scalable than on the server-side, since the overhead of computation and storage is distributed among clients.

As shown in Figure 5.1, the UCAIR toolbar has 3 major components: (1) The (implicit) user modeling module captures a user’s search context and history information, including the submitted queries and any clicked search results and infers search session boundaries. (2) The query modification module selectively improves the query formulation according to the current user model. (3) The result re-ranking module immediately re-ranks any unseen search results whenever the user model is updated.

In UCAIR, we consider four basic user actions: (1) submitting a keyword query; (2) viewing a

\(^1\)UCAIR is available at: http://sifaka.cs.uiuc.edu/ir/ucair/download.html
document; (3) clicking the “Back” button; (4) clicking the “Next” link on a result page. For each of these four actions, the system responds with, respectively, (1) generating a ranked list of results by sending a possibly expanded query to a search engine; (2) updating the information need model \( \bar{x} \); (3) reranking the unseen results on the current result page based on the current model \( \bar{x} \); and (4) reranking the unseen pages and generating the next page of results based on the current model \( \bar{x} \).

Behind these responses, there are three basic tasks: (1) Decide whether the previous query is related to the current query and if so expand the current query with useful terms from the previous query or the results of the previous query. (2) Update the information need model \( \bar{x} \) based on a newly clicked document summary. (3) Rerank a set of unseen documents based on the current model \( \bar{x} \). Below we describe our algorithms for each of them.

### 5.2 Session Boundary Detection and Query Expansion

To effectively exploit previous queries and their corresponding clickthrough information, UCAIR needs to judge whether two adjacent queries belong to the same search session (i.e., detect session boundaries). Existing work on session boundary detection is mostly in the context of web log analysis (e.g., [39]), and uses statistical information rather than textual features. Since our client-
side agent does not have access to server query logs, we make session boundary decisions based on textual similarity between two queries. Because related queries do not necessarily share the same words (e.g., “java island” and “travel Indonesia”), it is insufficient to use only query text. Therefore we use the search results of the two queries to help decide whether they are topically related. For example, for the above queries “java island” and “travel Indonesia”, the words “java”, “bali”, “island”, “indonesia” and ”travel” may occur frequently in both queries’ search results, yielding a high similarity score.

We only use the titles and summaries of the search results to calculate the similarity since they are available in the retrieved search result page and fetching the full text of every result page would significantly slow down the process. To compensate for the terseness of titles and summaries, we retrieve more results than a user would normally view for the purpose of detecting session boundaries (typically 50 results).

The similarity between the previous query $q'$ and the current query $q$ is computed as follows. Let $\{s_1', s_2', \ldots, s_n'\}$ and $\{s_1, s_2, \ldots, s_n\}$ be the result sets for the two queries. We use the pivoted normalization TF-IDF weighting formula [82] to compute a term weight vector $\bar{s}_i$ for each result $s_i$. We define the average result $\bar{s}_{avg}$ to be the centroid of all the result vectors, i.e., $(\bar{s}_1 + \bar{s}_2 + \ldots + \bar{s}_n)/n$. The cosine similarity between the two average results is calculated as

$$\frac{\bar{s}_{avg} \cdot \bar{s}_{avg}}{\sqrt{\bar{s}_{avg}^2 \cdot \bar{s}_{avg}^2}}$$

If the similarity value exceeds a predefined threshold, the two queries will be considered to be in the same information session.

If the previous query and the current query are found to belong to the same search session, UCAIR would attempt to expand the current query with terms from the previous query and its search results. Specifically, for each term in the previous query or the corresponding search results, if its frequency in the results of the current query is greater than a preset threshold (e.g. 5 results
out of 50), the term would be added to the current query to form an expanded query. In this case, UCAIR would send this expanded query rather than the original one to the search engine and return the results corresponding to the expanded query. Currently, UCAIR only uses the immediate preceding query for query expansion; in principle, we could exploit all related past queries.

5.3 Information Need Model Updating

Suppose at time $t$, we have observed that the user has viewed $k$ documents whose summaries are $s_1, ..., s_k$. We update our user model by computing a new information need vector with a standard feedback method in information retrieval (i.e., Rocchio [71]). According to the vector space retrieval model, each clicked summary $s_i$ can be represented by a term weight vector $\vec{s}_i$ with each term weighted by a TF-IDF weighting formula [75]. Rocchio computes the centroid vector of all the summaries and interpolates it with the original query vector to obtain an updated term vector. That is,

$$ \vec{x}' = \alpha \vec{q} + (1 - \alpha) \frac{1}{k} \sum_{i=1}^{k} \vec{s}_i $$

where $\vec{q}$ is the query vector, $k$ is the number of summaries the user clicks immediately following the current query and $\alpha$ is a parameter that controls the influence of the clicked summaries on the inferred information need model. In our experiments, $\alpha$ is set to 0.5. Note that we update the information need model whenever the user views a document.

5.4 Result Reranking

In general, we want to rerank all the unseen results as soon as the user model is updated. Currently, UCAIR implements reranking in two cases, corresponding to the user clicking the “Back” button and “Next” link in the Internet Explorer. In both cases, the current (updated) user model would be used to rerank the unseen results so that the user would see improved search results immediately.

To rerank any unseen document summaries, UCAIR uses the standard vector space retrieval
model and scores each summary based on the similarity of the result and the current user information need vector $\tilde{e}$ [75]. Since implicit feedback is not completely reliable, we bring up only a small number (e.g. 5) of highest reranked results to be followed by any originally high ranked results.

5.5 Evaluation of UCAIR Toolbar

We now present some results on evaluating the two major UCAIR functions: selective query expansion and result reranking based on user clickthrough data.

5.5.1 Sample Results

The query expansion strategy implemented in UCAIR is intentionally conservative to avoid misinterpretation of implicit user models. In practice, whenever it chooses to expand the query, the expansion usually makes sense. In Table 5.1, we show how UCAIR can successfully distinguish two different search contexts for the query “java map”, corresponding to two different previous queries (i.e., “travel Indonesia” vs. “hashtable”). Due to implicit user modeling, UCAIR intelligently figures out to add “Indonesia” and “class”, respectively, to the user’s query “java map”, which would otherwise be ambiguous as shown in the original results from Google on March 21, 2005. UCAIR’s results are much more accurate than Google’s results and reflect personalization in search.

The eager implicit feedback component is designed to immediately respond to a user’s activity such as viewing a document. In Figure 5.2, we show how UCAIR can successfully disambiguate an ambiguous query “jaguar” by exploiting a viewed document summary. In this case, the initial retrieval results using “jaguar” (shown on the left side) contain two results about the Jaguar cars followed by two results about the Jaguar software. However, after the user views the web page content of the second result (about “Jaguar car”) and returns to the search result page by clicking “Back” button, UCAIR automatically nominates two new search results about Jaguar cars (shown
Table 5.1: Sample results of query expansion

<table>
<thead>
<tr>
<th>Google result (user query = &quot;java map&quot;)</th>
<th>UCAIR result (user query = &quot;java map&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous query = &quot;travel Indonesia&quot;</td>
<td>expanded user query = &quot;java map Indonesia&quot;</td>
</tr>
<tr>
<td>expanded user query = &quot;hashable&quot;</td>
<td>expanded user query = &quot;java map class&quot;</td>
</tr>
<tr>
<td>1 Java map projections of the world ...</td>
<td>Lonely Planet - Indonesia Map</td>
</tr>
<tr>
<td><a href="http://www.btinternet.com/se16/js/mapproj.htm">www.btinternet.com/se16/js/mapproj.htm</a></td>
<td>Map (Java 2 Platform SE v1.4.2)</td>
</tr>
<tr>
<td>2 Java map projections of the world ...</td>
<td>INDOONESIA TOURISM - CENTRAL JAVA - MAP</td>
</tr>
<tr>
<td><a href="http://www.btinternet.com/se16/js/oldmapproj.htm">www.btinternet.com/se16/js/oldmapproj.htm</a></td>
<td><a href="http://www.indonesia-tourism.com/">www.indonesia-tourism.com/</a>...</td>
</tr>
<tr>
<td>3 Java Map</td>
<td>JAVA TOURISM - WEST JAVA - MAP</td>
</tr>
<tr>
<td><a href="http://www.java.sun.com/developer/">www.java.sun.com/developer/</a>...</td>
<td>An Introduction to Java Map Collection Classes <a href="http://www.oracle.com/technology/">www.oracle.com/technology/</a>...</td>
</tr>
<tr>
<td>4 Java Technology Concept Map</td>
<td>Indonesia Regions and Island Maps, Bali, Java ...</td>
</tr>
<tr>
<td><a href="http://www.java.sun.com/developer/onlineTraining/">www.java.sun.com/developer/onlineTraining/</a>...</td>
<td><a href="http://www.maps2anywhere.com/Map">www.maps2anywhere.com/Map</a>...</td>
</tr>
<tr>
<td>5 Science@NASA Home</td>
<td>Maps Of Indonesia</td>
</tr>
<tr>
<td>science.nasa.gov/Realtim/</td>
<td><a href="http://www.indonesia-tourism.com/">www.indonesia-tourism.com/</a>...</td>
</tr>
<tr>
<td>6 An Introduction to Java Map Collection Classes</td>
<td>Maps Of Indonesia</td>
</tr>
<tr>
<td><a href="http://www.oracle.com/technology/">www.oracle.com/technology/</a>...</td>
<td>Maps Of Indonesia</td>
</tr>
<tr>
<td>7 Lonely Planet - Java Map</td>
<td><a href="http://www.indostreets.com/maps/java/">www.indostreets.com/maps/java/</a></td>
</tr>
<tr>
<td><a href="http://www.lonelyplanet.com/indonesia/hello/">www.lonelyplanet.com/indonesia/hello/</a></td>
<td>An Introduction to Java Map Collection Classes <a href="http://www.theonsitevice.com/news/">www.theonsitevice.com/news/</a>...</td>
</tr>
<tr>
<td>8 ONjava.com Java API Map</td>
<td>Maps of Indonesia by Peter Loud</td>
</tr>
<tr>
<td><a href="http://www.onjava.com/pub/api/onjava/api_map/">www.onjava.com/pub/api/onjava/api_map/</a></td>
<td><a href="http://www.users.powernet.co.uk/">www.users.powernet.co.uk/</a>...</td>
</tr>
<tr>
<td>9 GTA San Andreas - Sam</td>
<td>tmap_pmel.noaa.gov/...</td>
</tr>
<tr>
<td><a href="http://www.indonesia-tourism.com/">www.indonesia-tourism.com/</a>...</td>
<td>tmap_pmel.noaa.gov/...</td>
</tr>
<tr>
<td>10 INDONESIA TOURISM - WEST JAVA - MAP</td>
<td><a href="http://www.indonesia-tourism.com/">www.indonesia-tourism.com/</a>...</td>
</tr>
<tr>
<td><a href="http://www.indonesiaphoto.com/">www.indonesiaphoto.com/</a></td>
<td>Maps Of Indonesia by Peter Loud</td>
</tr>
<tr>
<td>2 Text REtrieval Conference: <a href="http://trec.nist.gov/">http://trec.nist.gov/</a></td>
<td></td>
</tr>
</tbody>
</table>

on the right side), while the original two results about Jaguar software are pushed down on the list (unseen from the picture).

5.5.2 Quantitative Evaluation

To further evaluate UCAIR quantitatively, we conduct a user study on the effectiveness of the eager implicit feedback component. It is a challenge to quantitatively evaluate the potential performance improvement of our proposed model and UCAIR over Google in an unbiased way [35]. Here, we design a user study, in which participants would do normal web search and judge a randomly and anonymously mixed set of results from Google and UCAIR at the end of the search session; participants do not know whether a result comes from Google or UCAIR.

We recruited 6 graduate students for this user study, who have different backgrounds (3 computer science, 2 biology, and 1 chemistry). We use query topics from TREC ² 2004 Terabyte track [18] and TREC 2003 Web track [21] topic distillation task in the way to be described below.

An example topic from TREC 2004 Terabyte track appears in Figure 5.3. The title is a short phrase and may be used as a query to the retrieval system. The description field provides a slightly longer statement of the topic requirement, usually expressed as a single complete sentence or question. Finally the narrative supplies additional information necessary to fully specify the re-
Figure 5.2: Screen shots for result reranking

requirement, expressed in the form of a short paragraph.

Initially, each participant would browse 50 topics either from Terabyte track or Web track and pick 5 or 7 most interesting topics. For each picked topic, the participant would essentially do the normal web search using UCAIR to find many relevant web pages by using the title of the query topic as the initial keyword query. During this process, the participant may view the search results and possibly click on some interesting ones to view the web pages, just as in a normal web search. There is no requirement or restriction on how many queries the participant must submit or when the participant should stop the search for one topic. When the participant plans to change the search topic, he/she will simply press a button to evaluate the search results before actually switching to the next topic.

At the time of evaluation, 30 top ranked results from Google and UCAIR (some are overlapping) are randomly mixed together so that the participant would not know whether a result comes from Google or UCAIR. The participant would then judge the relevance of these results. We measure precision at top $n$ ($n = 5, 10, 20, 30$) documents of Google and UCAIR. We also evaluate precisions at different recall levels.
Figure 5.3: An example of TREC query topic, expressed in a form which might be given to a human assistant or librarian

Altogether, 368 documents judged as relevant from Google search results and 429 documents judged as relevant from UCAIR by participants. Scatter plots of precision at top 10 and top 20 documents are shown in Figure 5.4 and Figure 5.5 respectively (The scatter plot of precision at top 30 documents is very similar to precision at top 20 documents). Each point of the scatter plots represents the precisions of Google and UCAIR on one query topic.

Table 5.2 shows the average precision at top n documents among 32 topics. From Figure 5.4, Figure 5.5 and Table 5.2, we see that the search results from UCAIR are consistently better than those from Google by all the measures. Moreover, the performance improvement is more dramatic for precision at top 20 documents than that at precision at top 10 documents. One explanation for this is that the more interaction the user has with the system, the more clickthrough data UCAIR can be expected to collect. Thus the retrieval system can build more precise implicit user models, which lead to better retrieval accuracy.

The plot in Figure 5.6 shows the precision-recall curves for UCAIR and Google, where it is
Figure 5.4: Precision at top 10 documents of UCAIR and Google

<table>
<thead>
<tr>
<th>Ranking Method</th>
<th>prec@5</th>
<th>prec@10</th>
<th>prec@20</th>
<th>prec@30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>0.538</td>
<td>0.472</td>
<td>0.377</td>
<td>0.308</td>
</tr>
<tr>
<td>UCAIR</td>
<td>0.581</td>
<td>0.556</td>
<td>0.453</td>
<td>0.375</td>
</tr>
<tr>
<td>Improvement</td>
<td>8.0%</td>
<td>17.8%</td>
<td>20.2%</td>
<td>21.8%</td>
</tr>
</tbody>
</table>

Table 5.2: Table of average precision at top n documents for 32 query topics

clearly seen that the performance of UCAIR is consistently and considerably better than that of Google at all levels of recall.
Figure 5.5: Precision at top 20 documents of UCAIR and Google

Figure 5.6: Precision at top 20 result of UCAIR and Google
Chapter 6
Active Feedback

6.1 Introduction

In ad hoc information retrieval, a user often needs to interact with the retrieval system several times to obtain satisfactory results for one information need, which provides opportunities for the retrieval system to actively participate in this iterative retrieval process. Most traditional retrieval systems just passively respond to user queries and put the responsibility of refining/improving the search solely on the user. But there has been evidence showing that a retrieval system can play an active role in this process, e.g., obtaining user feedback explicitly or implicitly when the user browses these documents, and exploiting such information to improve the performance in the next round of search [44, 49]. Ideally, a retrieval system should collaborate with the user in the whole interactive search period to improve the accuracy and reduce the number of interactions.

When explicit feedback is possible, a natural way for the retrieval system to actively participate in the retrieval process is to clarify the user’s information need by probing the user with well-designed questions. A question could be whether a document or passage is relevant, or whether a term describes the user’s information need.

In this scenario, a basic question is how the retrieval system should intelligently propose the questions so that it can learn most from the user’s answers to these questions. In this work, we study how a retrieval system can perform active feedback, i.e., how to choose documents for relevance feedback so that the system can learn most from the feedback information.

Relevance feedback is known to be effective for improving retrieval performance [71, 74, 34]. Previous work on relevance feedback focuses on query updating techniques such as query term
reweighting and query expansion. The issue of choosing documents for relevance feedback has not been well addressed. Traditionally, relevance feedback methods just choose the top ranked documents for feedback, which is not necessarily the best strategy from the learning perspective. For example, if the top two documents have identical contents, the learning benefits of these two documents will be nearly equal to that of any one of them. Thus a very interesting research question is how to select appropriate documents for user judgment to maximize the learning benefits, which is the focus of the study in this work.

Active feedback is essentially an application of active learning in ad hoc information retrieval. Active learning has been extensively studied in machine learning [72, 91, 20]. It has been applied to text categorization in several previous studies [56, 59, 92], and recently to adaptive information filtering [108]. But there has been little work on applying it to ad hoc retrieval, partly because there are two special challenges in applying active learning to ad hoc retrieval. First, in ad hoc retrieval, we do not have any training examples available to guide the retrieval system for actively selecting the documents for feedback; the query is the only information that can be exploited. Second, it is unclear how we can define an objective function that optimizes ranking performance rather than classification accuracy. An interesting recent work on applying active learning to ad hoc retrieval is [41], where a user is assumed to iteratively choose clusters, and the active learning task for the system is to design good clusters, a different task from active feedback. The TREC HARD Track [4] has stimulated some recent work along the line of active feedback including [68, 80].

In this work, we frame the problem of active relevance feedback as a statistical decision problem, and examine several special cases in refining the framework. We derive several practical algorithms for active feedback, including the Top K, Gapped Top K and K Cluster Centroid algorithm. We empirically evaluate these three algorithms using the TREC2003 HARD data, AP88-89 and AP90. The results show that the performance of the Top K algorithm (i.e., the traditional way of relevance feedback) is consistently worse than that of Gapped Top K algorithm and K Cluster Centroid algorithm which present documents with more diversity. In general, with a diversity-based selection algorithm, we obtain fewer relevant documents, but these fewer documents have
more learning benefits.

The remaining sections are organized as follows. In Section 6.2, we present the active feedback framework and derive several practical algorithms. In Section 6.3, we describe our evaluation methods and three algorithms we tested. We discuss the experiment results in Section 6.4 and conclude our work in Section 6.5.

6.2 Active Feedback Framework

The problem of active feedback is essentially a decision problem in which we choose the best subset of documents for relevance judgment by the user. To formalize this problem, we follow the risk minimization framework for retrieval [51] and treat it as the following optimization problem:

$$D^* = \arg \min_D \int_{\Theta} L(D, \mathcal{U}, \theta)p(\theta|\mathcal{U}, q, C) \, d\theta$$

where $D = \{d_1, ..., d_k\}$ is a subset of the document collection $C$, $q$ is a query, $\mathcal{U}$ is a user variable, $\theta$ is the set of parameters of the query language model and document language models. $p(\theta|\mathcal{U}, q, C)$ is the posterior probability distribution of all the parameters, and $L(D, \mathcal{U}, \theta)$ is a loss function reflecting how much we can expect to learn by requesting relevance judgments on $D$ from user $\mathcal{U}$. In general, the loss function may also depend on other factors such as any relevance judgments available from previous iterations of retrieval, but here we ignore those factors for the convenience of presentation.

Without refining the language models $p(\theta|\mathcal{U}, q, C)$, which is not the focus of this work, we study how to define the loss function for active feedback. Clearly, the actual value of a set of documents $D$ for learning depends on not only $D$ but also the judgments the user would make. Let $\mathcal{J} = \{0, ..., m\}$ be the set of all possible relevance levels that a user may assign to each presented document (0 for “completely non-relevant”). For example, for binary judgments, $\mathcal{J} = \{0, 1\}$. The
loss function can now be written as

\[ L(D, \mathcal{U}, \theta) = \sum_{\vec{j} \in \mathcal{J}^k} l(D, \vec{j}, \theta) p(\vec{j} | D, \theta, \mathcal{U}) \]

where \( \vec{j} = (j_1, ..., j_k) \) and \( j_i \) is a possible judgment for document \( d_i \) in \( D \); \( p(\vec{j} | D, \theta, \mathcal{U}) \) is the probability that the user \( \mathcal{U} \) would assign judgments \( \vec{j} \) to all the documents in \( D \); and \( l(D, \vec{j}, \theta) \) is a loss function that indicates how much we can learn from the judgments \( \vec{j} \) on \( D \). In other words, \( l(.) \) tells us how good \((D, \vec{j})\) is as a set of labeled examples for learning.

Now assuming that the user would judge each document *independently*, we have

\[ L(D, \mathcal{U}, \theta) = \sum_{\vec{j} \in \mathcal{J}^k} l(D, \vec{j}, \theta) \prod_{i=1}^{k} p(j_i | d_i, \theta, \mathcal{U}) \]

Note that this assumption is reasonable if a user explicitly judges a document, but it is unlikely to hold when we infer a user’s judgments based on, say, clickthrough data [44], as obviously a user would not open a redundant (but relevant) document.

Thus our general framework for active feedback is the following decision rule:

\[ D^* = \arg \min_{D} \int_{\theta} \left[ \sum_{\vec{j} \in \mathcal{J}^k} l(D, \vec{j}, \theta) \prod_{i=1}^{k} p(j_i | d_i, \theta, \mathcal{U}) \right] p(\theta | \mathcal{U}, q, \mathcal{C}) \, d\theta \]

In the remaining part of the section, we discuss some interesting special cases. We will assume that the relevance judgments are all binary, though most derivations can be easily generalized to multi-level judgments.

### 6.2.1 Independent Loss

Let us first simplify the loss function by assuming that the value of each judged example for learning is *independent* of each other. The total value of a set of examples \((D, \vec{j})\) can thus be

53
written as the sum of the value of each individual example, i.e.,

\[ l(D, j, \theta) = \sum_{i=1}^{k} l(d_i, j_i, \theta) \]

where \( l(d_i, j_i, \theta) \) is the loss for a single judged document \((d_i, j_i)\).

After some algebraic manipulation, we have

\[ L(D, U, \theta) = \sum_{i=1}^{k} \sum_{j_i} l(d_i, j_i, \theta)p(j_i|d_i, \theta, U) \]

And the active feedback decision rule is

\[ D^* = \arg\min_D \int \sum_{i=1}^{k} \sum_{j_i} l(d_i, j_i, \theta)p(j_i|d_i, \theta, U)p(\theta|U, q, C) d\theta \]

The optimal set \( D \) can thus be obtained by ranking all the documents according to the following risk function and taking the \( k \) documents with the least risk:

\[ r(d_i) = \sum_{j_i} \int l(d_i, j_i, \theta)p(j_i|d_i, \theta, U)p(\theta|U, q, C) d\theta \]

which can be interpreted as the expected value of \( d_i \) for learning over all possible judgments.

We now examine the assumptions underlying two simple methods for defining \( r(d_i) \) — “Top K” and “Uncertainty Sampling”.

**Top K**

Let us assume that the loss of any relevant example (document) and that of any non-relevant example (document) are both constants. We further assume that the former is smaller than the latter, which is to say that a relevant example is more useful for learning than a non-relevant one. Formally, \( \forall d_i \in C \), we have \( l(d_i, 1, \theta) = C_1 \), \( l(d_i, 0, \theta) = C_0 \), and \( C_1 < C_0 \).
The risk function now becomes

\[
    r(d_i) = C_0 + (C_1 - C_0) \int_{\Theta} p(j_i = 1|d_i, \theta, U) p(\theta|U, q, C) \, d\theta
\]

Since \( C_1 - C_0 < 0 \), clearly the optimal set \( D^* \) is precisely the \( k \) documents with the highest probabilities of being judged as relevant (i.e., with the highest expected values of \( p(j_i = 1|d_i, \theta, U) \)). That is, we should simply rank all the documents in \( C \) according to the estimated relevance status of each document and select the top \( k \) documents that are most likely relevant for feedback.

We have thus obtained the “Top K” method as a special case under three assumptions: (1) independent loss function; (2) constant loss for any relevant (non-relevant) document; and (3) a relevant document has a smaller loss than a non-relevant one. The results are not really surprising because assumption 2 basically says that all relevant (non-relevant) examples are equally good for learning. However this analysis suggests that we may expect other methods to perform better than Top K if the underlying feedback algorithm does not satisfy all these three assumptions, e.g., independent loss function.

### Uncertainty Sampling

In [56, 55], a similar document selection problem is studied, though a set of documents are selected for labeling to train a text classifier instead of a ranking function. Authors propose to select the most uncertain documents for labeling. In [107], a similar idea, i.e., selecting most uncertain objects, is used to guide the hidden annotation for content-based image information retrieval. Using our general active feedback framework, we can derive the uncertainty sampling method by assuming the following loss function:

\[
    l(d_i, 1, \theta) = \log p(R = 1|d_i, \theta) \quad \forall d_i \in C
\]

\[
    l(d_i, 0, \theta) = \log p(R = 0|d_i, \theta) \quad \forall d_i \in C
\]

\(^1\)Top K as an active feedback method was first discussed in [54].
where $R \in \{0, 1\}$ is a binary relevance variable with 1 indicating “relevant”. This loss function essentially says that a relevant example is more useful if the predicted probability of relevance is smaller according to our current model, and similarly, a non-relevant example is more useful if the predicted probability of relevance is larger. In other words, an example is more useful if our prediction has less confidence.

With such a loss function, and assuming $p(R|d_i, \theta)$ is an approximation of $p(j_i|d_i, \theta, \mathcal{U})$, the risk function becomes
\[
r(d_i) = - \int_{\theta} H(R|d_i, \theta)p(\theta|\mathcal{U}, q, C) d\theta
\]
where $H$ is the entropy function. This means that, in order to obtain $D$, we should rank documents in the descending order of the expected entropy of the corresponding relevance variable $R$. That is, we would pick documents with the highest uncertainty.

We have thus obtained the “Uncertainty Sampling” method as a special case under two assumptions: (1) independent loss function; (2) an example is more useful for learning if our prediction of relevance is more uncertain. This method relies on explicitly predicting the probability of relevance, which is often not feasible in ad hoc retrieval.

### 6.2.2 Dependent Loss

Our assumption about an independent loss on each example is not realistic. For instance, if two examples are completely identical, their total value is clearly less than the sum of their individual values, and is probably close to the value of one of the examples. Thus we need to model the interactions between documents with a dependent loss function. Unfortunately, the exact form of such a loss function highly depends on the specific feedback algorithms. Nevertheless, intuitively, given a fixed size of $D$, increasing the representativeness of documents in $D$ appears to be always desirable. At the same time, we can also reasonably assume that relevant examples are more useful than non-relevant examples. Thus one possible way to refine a dependent loss function is to associate the value of $D$ for learning with the relevance status and diversity of $D$. That is, we
write our loss function as

\[ L(D, \mathcal{U}, \theta) \approx -\sum_{i=1}^{k} p(j_i = 1|d_i, \theta, \mathcal{U}) - \lambda \Delta(D, \theta) \]

where \( \Delta(D, \theta) \) is a function that measures the diversity of documents in \( D \) and \( \lambda \) is a parameter indicating the tradeoff between the relevance and diversity.

According to this loss function, the active feedback decision rule is

\[
D^* = \arg\min_D -\int_\Theta \sum_{i=1}^{k} p(j_i = 1|d_i, \theta, \mathcal{U}) p(\theta|\mathcal{U}, q, C) \, d\theta \\
-\lambda \int_\Theta \Delta(D, \theta) p(\theta|\mathcal{U}, q, C) \, d\theta
\]  

(6.1)

That is, we need to select \( D \) to simultaneously optimize both relevance (the first term) and diversity (the second term). A possible greedy algorithm is to first optimize the relevance term by selecting top \( N \) (\( N > K \)) documents according to relevance-based ranking, and then to further select \( K \) most diverse documents from the \( N \) documents. We now discuss several simple methods along this line.

**Gapped Top K**

Suppose we let \( N = (G + 1)K \), where \( G \) is a small positive integer. To capture the diversity, we partition the \( N \) documents into \( K \) clusters based solely on the relevance scores so that our first cluster would have the first \( G + 1 \) documents and the second one have the next \( G + 1 \) documents, and so on so forth. From each cluster, we then select a document with the highest relevance score to form our feedback document set \( D \). We refer to this method as “Gapped Top K” since it corresponds to selecting the top \( K \) documents with a gap of \( G \) documents in between any two documents. An interesting property of this method is that when \( G = 0 \), it is essentially the regular Top K method.
Maximal Marginal Relevance (MMR)

Maximal Marginal Relevance (MMR) ranking is a greedy algorithm for ranking documents based on relevance and at the same time avoiding redundancy \([15, 103]\). Specifically, we iteratively select a document which optimizes the following MMR function:

\[ r(d|D) = \alpha s(d) + (1 - \alpha) \max_{d' \in D} \text{sim}(d, d') \]

where \(s(d)\) is a relevance scoring function, \(\text{sim}(d, d')\) is a similarity function, and \(\alpha\) is a parameter for trading off between relevance and redundancy.

This method can also be regarded as performing an *implicit* clustering and then selecting a document from a cluster with the highest relevance value. The first document selected will be the top ranked one based on relevance. Since the next document to be selected must be far from this selected first document, we can interpret the first document as implicitly defining a cluster with the first document being the centroid, and none of the other documents in the cluster will be selected since they are all too close to the selected first document. As we select the second document, we again have another cluster which further excludes some documents from being selected. However, it is unclear what the clustering boundary is exactly as it is affected by not just the similarity function, but also the relevance scores of documents and the parameter \(\alpha\). The MMR method can also cover the Top K method as a special case when \(\alpha = 1\).

**Cluster Centroid**

A more direct method to maximize diversity is to perform explicit clustering. Specifically, we can first select the top \(N\) documents according to the relevance scores. Then we partition these \(N\) documents into \(K\) clusters and construct \(D\) by selecting one representative document from each cluster.

To optimize the relevance term in equation 6.1, we restrict \(N\) to a relatively small number. In this way, we ensure that each of the \(N\) documents has a reasonably high probability of relevance.
The diversity is ensured through clustering and choosing only one document from each cluster. There are several different ways to choose a representative document from each cluster. One is to choose the centroid document, which maximizes the average similarity between the chosen document and other documents in the cluster. Another choice is to choose the document with the highest relevance score.

### 6.3 Experiment Methodology

<table>
<thead>
<tr>
<th>Gap</th>
<th>HARD</th>
<th>MAP</th>
<th>0.3247</th>
<th>0.3277</th>
<th>0.3275</th>
<th>0.3289</th>
<th>0.3285</th>
<th><strong>0.3300</strong></th>
<th>0.3298</th>
<th>0.3262</th>
<th>0.3289</th>
<th>0.3267</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>pr@10</td>
<td>0.5271</td>
<td><strong>0.5563</strong></td>
<td>0.5438</td>
<td>0.5396</td>
<td>0.5521</td>
<td>0.5479</td>
<td>0.5479</td>
<td>0.5500</td>
<td>0.5417</td>
<td>0.5292</td>
<td></td>
</tr>
<tr>
<td></td>
<td>#APRel</td>
<td>3.0</td>
<td>3.1</td>
<td>3.0</td>
<td>2.7</td>
<td>2.8</td>
<td>2.6</td>
<td>2.7</td>
<td>2.7</td>
<td>2.5</td>
<td>1.9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>AP88-89</td>
<td>MAP</td>
<td>0.2284</td>
<td>0.2332</td>
<td>0.2320</td>
<td>0.2317</td>
<td>0.2323</td>
<td>0.2303</td>
<td>0.2284</td>
<td><strong>0.2344</strong></td>
<td>0.2303</td>
<td>0.2243</td>
</tr>
<tr>
<td></td>
<td>pr@10</td>
<td>0.3511</td>
<td>0.3837</td>
<td><strong>0.3913</strong></td>
<td>0.3826</td>
<td>0.3880</td>
<td>0.3859</td>
<td>0.3761</td>
<td>0.3891</td>
<td>0.3826</td>
<td>0.3609</td>
<td></td>
</tr>
<tr>
<td></td>
<td>#AFRel</td>
<td>2.1</td>
<td>1.9</td>
<td>1.8</td>
<td>1.8</td>
<td>1.6</td>
<td>1.5</td>
<td>1.7</td>
<td>1.5</td>
<td>1.2</td>
<td>0.9</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Average performance of Gapped Top K with different gaps. The best performance is shown in bold.

### 6.3.1 Data Set

We use two data sets for experiments. One is the Associated Press (AP) news data on TREC disks 1, 2, and 3. The other is the TREC2003 HARD (High Accuracy Retrieval from Documents) track data set [4]. TREC2003 HARD track puts search into context, which allows a retrieval system to actively infer a user’s information need and improve retrieval performance [80]. Our experiment process simulates two runs of the HARD track experiment setup [4]. For the HARD track data set, we use 48 topics that have relevance judgments. For the AP data set, we use 92 topics from topics 1-50 and 101-150, which have relevance judgments on both the AP88-89 and AP90 data set. We use the title of each topic as the query.
6.3.2 Experiment Setup

We use the Lemur toolkit as our retrieval system [53] and the KL-Divergence language retrieval model as our retrieval model [51, 104]. K is fixed to 6 in most experiments, and all parameters are set to default values [53] unless otherwise stated. Our baseline run is regular retrieval without any feedback. It allows us to test whether we can improve performance by performing feedback. From the baseline retrieval results, we use different active relevance feedback algorithms to select a set of documents for relevance feedback. Using the known relevance judgments available from these TREC data to simulate a user's judgments, we obtain relevance judgments on the selected documents. These judgments are then used to perform feedback using the mixture model approach implemented in Lemur [105]. This method only uses relevant documents for query model updating, which can be a limitation of our study. The retrieval results in the second run using different active feedback algorithms are compared for evaluation. This experiment procedure is illustrated in Figure 6.1.

6.3.3 Algorithm Description

As a first step of studying active feedback, we evaluate three representative active feedback algorithms discussed in Section 6.2. The first one is Top K, which chooses top K documents from the baseline run retrieval, and is also what existing retrieval systems would normally do. The second
one is Gapped Top K, which is to choose gapped top K documents from the baseline run results. For example, if we set the gap to 3 and K to 6, we will end up choosing the 1st, 5th, 9th, ..., 21st documents from the retrieval results. The third one is K cluster centroid, which represents the most direct way of modeling diversity. We use the K-Medoid clustering algorithm [47] to cluster the top N documents. And we use J-Divergence [57] of two documents as the distance function. J-Divergence is a divergence metric similar to KL-Divergence. But unlike the non-symmetry of KL-Divergence, J-Divergence is symmetric. The formula of J-Divergence is as follows.

\[
J(d_i||d_j) = \sum_w p(w|\theta_i) \log \frac{p(w|\theta_i)}{p(w|\theta_j)} + \sum_w p(w|\theta_j) \log \frac{p(w|\theta_j)}{p(w|\theta_i)}
\]

Evaluation of these methods allows us to examine whether presenting a diverse set of documents for feedback leads to more effective feedback than presenting the top k documents with the highest relevance values.

<table>
<thead>
<tr>
<th>N</th>
<th>6</th>
<th>20</th>
<th>40</th>
<th>60</th>
<th>80</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARD</td>
<td>MAP</td>
<td>0.3247</td>
<td>0.3280</td>
<td>0.3303</td>
<td>0.3277</td>
<td>0.3289</td>
</tr>
<tr>
<td></td>
<td>pr@10</td>
<td>0.5271</td>
<td>0.5563</td>
<td>0.5479</td>
<td>0.5583</td>
<td>0.5500</td>
</tr>
<tr>
<td></td>
<td>#AFRel</td>
<td>3.0</td>
<td>2.9</td>
<td>2.6</td>
<td>2.4</td>
<td>2.5</td>
</tr>
<tr>
<td>AP88-89</td>
<td>MAP</td>
<td>0.2284</td>
<td>0.2310</td>
<td><strong>0.2368</strong></td>
<td>0.2279</td>
<td>0.2318</td>
</tr>
<tr>
<td></td>
<td>pr@10</td>
<td>0.3511</td>
<td>0.3804</td>
<td><strong>0.3934</strong></td>
<td>0.3804</td>
<td>0.3739</td>
</tr>
<tr>
<td></td>
<td>#AFRel</td>
<td>2.1</td>
<td>1.9</td>
<td>1.3</td>
<td>1.3</td>
<td>1.4</td>
</tr>
</tbody>
</table>

Table 6.2: Average performance of K Cluster Centroid with N. The best performance is shown in bold.

### 6.3.4 Evaluation Method

To measure the performance of a ranking method, we use two standard ad hoc retrieval measures:

1. **Mean Average Precision (MAP):** This is the commonly used non-interpolated average precision and serves as a good measure of the overall ranking accuracy since it is sensitive to the rank of every relevant document.
2. **Precision at 10 documents (pr@10):** This measure does not average
well and only gives us the precision at one single cutoff point. But it reflects the utility perceived by a user who may only read up to top 10 documents. In all cases, the reported figure is the average over all the topics.

Since the task of active feedback involves identifying a certain number of relevant documents by the user, an interesting question is whether we should include such relevant documents when computing the retrieval precision of an active feedback algorithm. While this is also a problem for relevance feedback evaluation, it is especially a challenge for evaluating active feedback algorithms because the set of relevant documents used for feedback can usually be controlled in regular relevance feedback evaluation, but must vary in evaluating active feedback algorithms.

In our evaluation, we decided to include all the judged documents, including both relevant and non-relevant documents, because if we exclude them, we would have a potentially different test set for each method. In particular, it would be unfair for a method that tends to present more “easy” relevant documents for feedback; indeed, the retrieval task would become artificially harder for such a method due to the fact that more “easy to retrieve” relevant documents would be excluded.

However, including such judged documents also has a problem – it does not accurately reflect the actual utility of a method as perceived by a user. Indeed, a user would presumably not really care about where the judged feedback documents are ranked because the user has already seen them. Thus any improvement in the ranking of a seen relevant document does not really bring any real benefit to the user.

In order to see more clearly how much a method can improve the ranking of unseen documents, we can run the active feedback algorithms on one document database (i.e., the training database) to obtain relevance judgments and then use another similar document database (i.e., the testing database) to test the retrieval performance [85]. Thus, in addition to the regular evaluation on the HARD track data set and AP88-89 with all the judged documents included, we also use AP88-89 for training and AP90 for testing to compare different methods, assuming that the contents in these two databases are sufficiently similar.
Table 6.3: Average performance of different active learning algorithms. The best performance is shown in bold. A double star (**) and a single star (*) indicate that the performance is significantly better than that of Top K according to Wilcoxin signed rank test at the level of 0.05 and 0.1, respectively.

<table>
<thead>
<tr>
<th>Method</th>
<th>Baseline</th>
<th>K pseudo feedback</th>
<th>Top K</th>
<th>Gapped Top K</th>
<th>K Cluster Centroid</th>
</tr>
</thead>
<tbody>
<tr>
<td>HARD MAP</td>
<td>0.3076</td>
<td>0.3195</td>
<td>0.3247</td>
<td>0.3300**</td>
<td><strong>0.3318</strong></td>
</tr>
<tr>
<td>pr@10</td>
<td>0.5014</td>
<td>0.5146</td>
<td>0.5271</td>
<td>0.5479*</td>
<td><strong>0.5646</strong></td>
</tr>
<tr>
<td>#AFRel</td>
<td>/</td>
<td>/</td>
<td>3.0</td>
<td>2.6</td>
<td>2.4</td>
</tr>
<tr>
<td>AP88-89 MAP</td>
<td>0.2007</td>
<td>0.2184</td>
<td>0.2284</td>
<td>0.2344*</td>
<td><strong>0.2368</strong></td>
</tr>
<tr>
<td>pr@10</td>
<td>0.3255</td>
<td>0.3426</td>
<td>0.3511</td>
<td>0.3891**</td>
<td><strong>0.3934</strong></td>
</tr>
<tr>
<td>#AFRel</td>
<td>/</td>
<td>/</td>
<td>2.2</td>
<td>1.5</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Table 6.4: Average performance of different retrieval algorithms on AP90 data set. The best performance is shown in bold. A double star (**) indicates that the performance is significantly better than that of Top K according to Wilcoxin signed rank test at the level of 0.05.

<table>
<thead>
<tr>
<th>Method</th>
<th>Baseline</th>
<th>K pseudo FB</th>
<th>Top K</th>
<th>Gapped Top K</th>
<th>K Cluster Centroid</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP</td>
<td>0.2026</td>
<td>0.2196</td>
<td>0.2203</td>
<td>0.2219</td>
<td><strong>0.2232</strong></td>
</tr>
<tr>
<td>pr@10</td>
<td>0.2946</td>
<td>0.3174</td>
<td>0.3207</td>
<td><strong>0.3261</strong></td>
<td>0.325</td>
</tr>
</tbody>
</table>

6.4 Experiment Result

6.4.1 Gapped Top K

As we mentioned in Section 6.2.2, Top K can be considered as a special case of Gapped Top K (i.e. when the gap equals to 0). We do experiments varying the gap to test whether a non-zero gap can perform better than Top K. The results on the HARD data set and AP88-89 data set are shown in Table 6.1, where we show the MAP, the precision at 10 documents, and the number of judged relevant documents per query.

From the results, we can see Top K ($gap = 0$) is clearly not the best strategy. Actually, when we choose small gaps ($gap \leq 6$), the performance is consistently better than Top K, which strongly suggests that top K is really a poor choice for active relevance feedback. We may also note that, as we increase the gap, we obtain fewer relevant documents than we could obtain with Top K. But using these fewer relevant documents for feedback can achieve better retrieval performance, which
<table>
<thead>
<tr>
<th>Method</th>
<th>HARD</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Top K</td>
<td>Gapped Top K</td>
<td>K Cluster Centroid</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>pr@10</td>
<td>#AFDoc</td>
<td>MAP</td>
<td>pr@10</td>
</tr>
<tr>
<td>K=2</td>
<td>0.3235</td>
<td>0.5167</td>
<td>1.1</td>
<td>0.3239</td>
<td>0.5446</td>
</tr>
<tr>
<td></td>
<td>0.3204</td>
<td>0.5333</td>
<td>0.8</td>
<td>0.2216</td>
<td>0.3457</td>
</tr>
<tr>
<td></td>
<td>0.3204</td>
<td>0.5333</td>
<td>0.8</td>
<td>0.2184</td>
<td>0.3533</td>
</tr>
<tr>
<td>K=4</td>
<td>0.3253</td>
<td>0.5271</td>
<td>2.0</td>
<td>0.3263</td>
<td>0.5292</td>
</tr>
<tr>
<td></td>
<td>0.3299</td>
<td>0.5480</td>
<td>1.8</td>
<td>0.2228</td>
<td>0.3837</td>
</tr>
<tr>
<td></td>
<td>0.3299</td>
<td>0.5480</td>
<td>1.8</td>
<td>0.2145</td>
<td>0.3620</td>
</tr>
<tr>
<td>K=6</td>
<td>0.3247</td>
<td>0.5271</td>
<td>3.0</td>
<td>0.3275</td>
<td>0.5438</td>
</tr>
<tr>
<td></td>
<td>0.3264</td>
<td>0.5458</td>
<td>2.3</td>
<td>0.2285</td>
<td>0.3913</td>
</tr>
<tr>
<td></td>
<td>0.3264</td>
<td>0.5458</td>
<td>2.3</td>
<td>0.2249</td>
<td>0.3740</td>
</tr>
<tr>
<td>K=8</td>
<td>0.3248</td>
<td>0.5250</td>
<td>4.0</td>
<td>0.3270</td>
<td>0.5396</td>
</tr>
<tr>
<td></td>
<td>0.3360</td>
<td>0.5708</td>
<td>3.0</td>
<td>0.2307</td>
<td>0.3902</td>
</tr>
<tr>
<td></td>
<td>0.3360</td>
<td>0.5708</td>
<td>3.0</td>
<td>0.2346</td>
<td>0.3859</td>
</tr>
<tr>
<td></td>
<td>0.3360</td>
<td>0.5708</td>
<td>3.0</td>
<td>0.2334</td>
<td>1.6</td>
</tr>
<tr>
<td>K=10</td>
<td>0.3249</td>
<td>0.5271</td>
<td>5.1</td>
<td>0.3274</td>
<td>0.5500</td>
</tr>
<tr>
<td></td>
<td>0.3304</td>
<td>0.5563</td>
<td>3.6</td>
<td>0.2319</td>
<td>0.3924</td>
</tr>
<tr>
<td></td>
<td>0.3304</td>
<td>0.5563</td>
<td>3.6</td>
<td>0.2375</td>
<td>0.3740</td>
</tr>
<tr>
<td></td>
<td>0.3304</td>
<td>0.5563</td>
<td>3.6</td>
<td>0.2304</td>
<td>1.9</td>
</tr>
<tr>
<td>K=12</td>
<td>0.3256</td>
<td>0.5396</td>
<td>6.1</td>
<td>0.3282</td>
<td>0.5438</td>
</tr>
<tr>
<td></td>
<td>0.3341</td>
<td>0.5521</td>
<td>4.4</td>
<td>0.2339</td>
<td>0.3880</td>
</tr>
<tr>
<td></td>
<td>0.3341</td>
<td>0.5521</td>
<td>4.4</td>
<td>0.2374</td>
<td>2.8</td>
</tr>
<tr>
<td></td>
<td>0.3341</td>
<td>0.5521</td>
<td>4.4</td>
<td>0.2363</td>
<td>2.2</td>
</tr>
<tr>
<td></td>
<td>0.3341</td>
<td>0.5521</td>
<td>4.4</td>
<td>0.2363</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table 6.5: Sensitivity of average performance of different active learning algorithms on K.

means these fewer relevant documents have more learning benefits. The same phenomenon is also observed when active learning is applied in the classification problem [77]. One explanation of this phenomenon is that when we increase the gap, we obtain more diverse documents, thus the judgments become more informative.

6.4.2 K Cluster Centroid

Here we use the clustering algorithm to select more diverse documents for active relevance feedback. We cluster the top N documents into K clusters and choose the K cluster centroid for relevance feedback. When N = K, we again have Top K as a special case. We vary N for fixed K (= 6) to test if presenting documents with higher diversity is beneficial. The results are shown in Table 6.2.

The variation of N causes a different tradeoff point for relevance and diversity. If we choose a
bigger $N$, we pay more attention to diversity, while if we choose a smaller $N$, we pay more attention to relevance. We see that the optimal values are different for the two databases. Comparing Top K ($N = K$) with other results in the Table again shows that Top K is mostly the worst among all the results, suggesting that the relevance judgments obtained with clustering are more effective for feedback than those obtained using Top K. Moreover, with a large $N$, we actually obtain fewer judged relevant documents, but these fewer relevant documents are better examples for learning.

6.4.3 Comparison of Different Algorithms

Since the effectiveness of the underlying feedback mechanism (the mixture model method in our case) is an important factor that may affect our evaluation, we compare several different feedback algorithms with the non-feedback baseline in Table 6.3. The performance for the Gapped Top K and the K Cluster Centroid is the best performance from Table 6.1 and Table 6.2, respectively.

From these results, we can see that the performance of both active feedback and pseudo feedback are better than that of baseline retrieval. We also see that the Top K relevance feedback performs better than using the top K documents for pseudo feedback. All these results show that the underlying feedback mechanism is effective.

Among active feedback algorithms, K cluster centroid outperforms Gapped Top K algorithm, which in turn outperforms Top K algorithm, although the improvement appears to be quite small. A very interesting observation is that the K cluster centroid algorithm obtains the fewest number of relevant documents from user feedback, yet its performance is the best. This suggests that selecting diverse documents leads to more effective learning.

As mentioned in Section 6.3.4, when comparing different active feedback algorithms, it is more reasonable to use one document database for active feedback (training), and the other document database for measuring retrieval performance (testing). Thus we further compare these methods using AP88-89 as the training set and AP90 as the testing set. Specifically, we perform baseline retrieval on AP88-89 database, select a document subset for relevance feedback using different active relevance feedback algorithms, update the query model, all on AP88-89, and then retrieve
The results again show that the results of the Top K algorithm is the worst among three active relevance feedback algorithms. Although the performance difference is mostly insignificant according to the Wilcoxin signed rank test except in the case of pr@10 for Gapped Top K, there are more topics for which Gapped Top K and K Cluster Centroid are better than Top K than the other way in all cases. In the case of MAP, it is 42 topics vs. 31 topics (with 19 cases tied) for both Gapped Top K and K Cluster Centroid. In the case of pr@10, it is 12 topics vs. 3 topics (with 77 cases tied) and 9 topics vs. 5 topics (with 78 cases tied) for Gapped Top K and K Cluster Centroid, respectively. The large number of tied cases indicates that our query expansion feedback mechanism is conservative. Indeed, as we show later in Table 6.6, when we change the query expansion parameter to perform more aggressive query expansion, the performance improvement is generally amplified. The performance of all active feedback algorithms is also better than that of pseudo feedback and baseline retrieval.

### 6.4.4 Performance Sensitivity of K

The results shown so far are all obtained by fixing $K = 6$. We now examine how choosing a different $K$ may affect our conclusions. We compare Top K, Gapped Top K (gap=3), and K cluster centroid ($N = 100$) for several different values of $K$ in Table 6.5. The results show that our conclusion, i.e., the performances of Gapped Top K and K Cluster Centroid are better than that of Top K, is relatively insensitive to the choice of $K$. Indeed, the Top K results are almost always the worst among the three methods. Also, on the HARD data, the K cluster centroid method consistently outperforms the other two methods with fewer judged relevant documents.

### 6.4.5 Mixture Feedback Algorithm Parameter $\alpha$ Factor

In the results shown so far, the improvement of Gapped Top K and K Cluster Centroid over Top K is not so significant. We find that the feedback algorithm parameter is an important factor. In
[105], the new query model \( \hat{\theta}_{Q'} \) is

\[
\hat{\theta}_{Q'} = (1 - \alpha)\hat{\theta}_Q + \alpha \times \hat{\theta}_F
\]

Here, \( \alpha \) controls how much weight we give to feedback documents. In all the previous results, we set \( \alpha \) to 0.5. But since the feedback documents are judged to be relevant by users, we can give more weight to these feedback documents. So we did another set of experiments by varying \( \alpha \) and keeping all other parameters fixed. The results are shown in Table 6.6. From these results, we can see clearly that the \( \alpha \) can amplify the effect of feedback. And when \( \alpha \) is increased, the improvement of Gapped Top K and K Cluster Centroid over Top K is also amplified.

| Method | HARD | | | | AP88-89 | | | | | | | | | |
|--------|------|------|------|------|-------|------|------|------|------|------|------|------|------|------|------|
| \( \alpha = 0.5 \) | MAP 0.325 | Gapped Top K 0.328 | K Cluster Centroid 0.326 | Top K 0.228 | Gapped Top K 0.232 | K Cluster Centroid 0.225 | pr@10 0.527 | 0.544 | 0.546 | 0.351 | 0.391 | 0.374 |
| \( \alpha = 0.6 \) | MAP 0.332 | Gapped Top K 0.335 | K Cluster Centroid 0.340 | Top K 0.239 | Gapped Top K 0.244 | K Cluster Centroid 0.236 | pr@10 0.529 | 0.552 | 0.556 | 0.370 | 0.407 | 0.390 |
| \( \alpha = 0.7 \) | MAP 0.339 | Gapped Top K 0.344 | K Cluster Centroid 0.357 | Top K 0.251 | Gapped Top K 0.259 | K Cluster Centroid 0.250 | pr@10 0.544 | 0.575 | 0.594 | 0.387 | 0.418 | 0.409 |
| \( \alpha = 0.8 \) | MAP 0.348 | Gapped Top K 0.355 | K Cluster Centroid 0.348 | Top K 0.264 | Gapped Top K 0.277 | K Cluster Centroid 0.267 | pr@10 0.552 | 0.577 | 0.581 | 0.404 | 0.442 | 0.431 |
| \( \alpha = 0.9 \) | MAP 0.356 | Gapped Top K 0.368 | K Cluster Centroid 0.388 | Top K 0.275 | Gapped Top K 0.295 | K Cluster Centroid 0.273 | pr@10 0.544 | 0.602 | 0.640 | 0.421 | 0.472 | 0.442 |
| \( \alpha = 0.95 \) | MAP 0.350 | Gapped Top K 0.367 | K Cluster Centroid 0.341 | Top K 0.276 | Gapped Top K 0.300 | K Cluster Centroid 0.274 | pr@10 0.548 | 0.602 | 0.577 | 0.428 | 0.479 | 0.429 |
| \( \alpha = 0.98 \) | MAP 0.337 | Gapped Top K 0.350 | K Cluster Centroid 0.307 | Top K 0.270 | Gapped Top K 0.293 | K Cluster Centroid 0.263 | pr@10 0.527 | 0.598 | 0.546 | 0.423 | 0.471 | 0.436 |

Table 6.6: Average performance of different active learning algorithms on different \( \alpha \).

### 6.5 Conclusions and Future Work

This work presents the first serious study of the problem of active relevance feedback, in which a retrieval system actively chooses the best documents for relevance feedback. Ad hoc information
retrieval is largely an interactive process. Active relevance feedback allows a retrieval system to actively probe a user and clarify the user’s information need, thus can improve retrieval performance.

We formulate the problem of active feedback as a statistical decision problem and study several special cases. We analyze the assumptions made in each case. We derive three specific algorithms for active relevance feedback, i.e., Top K, Gapped Top K, and K Cluster Centroid algorithm. We evaluate these algorithms using the TREC2003 HARD data set, AP88-89 and AP90 data set. Experiment results show that the Top K algorithm, which is what an existing retrieval system normally uses for relevance feedback, is not optimal for active relevance feedback, and is actually often worse than both the Gapped Top K algorithm and the K Cluster Centroid algorithm. Compared with the Top K algorithm, Gapped Top K algorithm and K Cluster Centroid algorithm emphasize returning more diversified documents. The results show that with fewer judged relevant documents, both Gapped Top K and K Cluster Centroid outperform the Top K algorithm, suggesting that the diversity in the presented documents is a desirable property. Although the difference is generally small, the overall consistency strongly supports our conclusions.

Our work represents only a very preliminary exploration of this important topic. There are several interesting directions to explore. (1) It would be interesting to study how to learn from non-relevant documents judged by the user so as to make full use of user efforts and feedback. (2) Another interesting question is how to optimize performance over the entire search session, rather than just one iteration. (3) We may explore other approaches for selecting documents. For example, MMR strategy is also a promising strategy. (4) We can try to combine pseudo feedback and active feedback. Since those highly ranked documents are very likely relevant, we do not really need to present them for judgments; instead, we can propose documents ranked below a few top documents for feedback (essentially the uncertainty sampling strategy). In this way, feedback can be based on those top-ranked documents, which we assume to be relevant, and the obtained relevance judgments through active feedback.
Chapter 7

Adaptive Clustering of Search Results

Clustering of search results has been shown to be advantageous over the simple list presentation of search results. However, in most clustering interfaces, the clusters are not adaptive to a user’s interaction with the clustering results, and the important question “how to optimize the benefit of a clustering interface for a user” has not been well addressed in the previous work. In this work, we study how to exploit a user’s clickthrough information, which is naturally available when a user is interacting with a clustering interface, to adaptively reorganize the clustering results and help a user find the relevant information more quickly. We propose four strategies for adapting clustering results based on user actions, including reranking documents based on a selected cluster, reranking documents based on a viewed document, merging unselected clusters, and promoting “near-miss” documents. Evaluation of the utility of a cluster presentation of results is a challenging task. We propose a general method to simulate different kinds of users and linearize the cluster results so that we can compute regular retrieval measures. The method can be used to compare different clustering results as well as comparing clustering results with a regular ranked list of results. The simulation experiments show that the adaptation strategies have different performance for different types of users; in particular, they are effective for “smart users” who can correctly recognize the best clusters, but not effective for “dummy users” who follow system’s ranking of results. Besides the simulation study, we also conduct a user study on one of the four adaptive clustering strategies (i.e., promoting near-miss documents) to see if an adaptive clustering system using such a strategy can bring users better search utility and/or experience than a static clustering system. The results show that there is generally no significant difference between the two systems from a user’s perspective.

69
7.1 Introduction

The main goal of a search engine is to rank relevant documents above non-relevant ones. There has been a lot of research in developing effective retrieval models to help achieve this goal. For example, many effective retrieval models have been developed, including models for content-based ranking, such as the vector-space model [76, 83], classic probabilistic models [69, 94, 93, 31], language models [65, 106, 22], and algorithms for link-based ranking, such as PageRank [63] and HITS [50].

However, an equally important goal of a search engine is to present the search results effectively so that a user can find the relevant information from the results quickly. Compared with the huge amount of literature on retrieval models, there is little research on how to optimize result presentation for a user.

Indeed, most search engines present a ranked list of documents with brief summaries. In an ideal case when the system performs extremely well on a query, a user would be able to find most or all the relevant documents on the top of this list, and such a presentation would be fine. However, due to the inevitable mismatches between a query and documents, the search results are more often non-optimal. Indeed, it is quite common that a user may not find any relevant document among the top ranked ones. In such a case, the user would have to go through the many non-relevant documents in the list until eventually finding some relevant ones. Intuitively, a clustering view of the search results would be much more useful in such a case. Indeed, clustering of search results has been shown to be an effective way to present the search results [101, 27, 28, 16], and has been adopted by some search engines such as vivisimo\(^1\) and carrot2\(^2\).

Although clustering of search results has been studied [24, 36, 101, 28], in most existing work, the clusters are generally not adaptive to a user’s interaction with the clustering results. In Scatter/Gather\([24, 36]\), the authors proposed that re-clustering can be performed on the user-selected clusters (i.e., “scattering”), which can be regarded as attempting to adapt the clustering

\(^1\)http://vivisimo.com/
\(^2\)http://www.carrot2.org/
results to a user based on the user’s action. Unfortunately, this is as far as the work goes, and to the best of our knowledge, there has been no work that attempts to seriously study the important question “how to optimize the benefit of a clustering interface for a user.”

With non-adaptive clustered results, a user would generally select a cluster and examine the results in it. Thus the utility of the results is largely determined by the clustering algorithm. Intuitively, however, the clustering results may be improved as the system sees more user interactions when the user examines the results. For example, when a user opens a particular cluster for viewing, we can infer from this action that the user likes the content of this cluster better than others, and this knowledge can be exploited immediately to rerank the results inside the cluster to be viewed as well as the results in other clusters. Also, since the unselected clusters may not be so interesting to the user, we may consider merging all of them into one cluster. Moreover, since we now know the user likes the selected cluster better than other clusters, we could move some “borderline” documents that were put in other clusters into the selected cluster. Such adaptation can be expected to improve the utility of the clustered results in the sense that it would help a user find relevant information more quickly.

In this work, we study how to exploit a user’s clickthrough information, which is naturally available when a user is interacting with a clustering interface, to adaptively reorganize the clustering results and help a user find the relevant information more quickly. Specifically, we propose four strategies for adapting clustering results based on user actions, including reranking documents based on a selected cluster, reranking documents based on a viewed document, merging unselected clusters, and promoting “near-miss” documents. Evaluation of the utility of a cluster presentation of results is a challenging task. We propose a general method to simulate different kinds of users and linearize the cluster results so that we can compute regular retrieval measures. The method can be used to compare different clustering results as well as comparing clustering results with a regular ranked list of results. The simulation experiments show that the adaptation strategies have different performance for different types of users; in particular, they are effective for “smart users” who can correctly recognize the best clusters, but not effective for “dummy users” who simply
follow system’s ranking of results. Among the four proposed adaptation strategies, the strategy of reranking based on viewed document is shown to be most effective, but other strategies are also beneficial.

Besides the simulation study, we also conduct a user study to see if an adaptive clustering system can bring improved search utility and/or experience to users, compared with a static clustering system. To ensure that we have enough users to make meaningful conclusions, we focus on the strategy of promoting near-miss documents. We chose this strategy because it has been shown to be effective in the simulation study and also affects the clustering membership so that it is likely to make users feel some difference. We recruited 24 subjects and did a study of comparing a static clustering baseline system with an adaptive clustering system in Web search. The results show that there is generally no significant difference between the two systems from a user’s perspective. Specifically, more users say that they like the adaptive system better but users saved more relevant documents with the static interface than with the adaptive interface. While this result is a bit disappointing, it is also not very surprising because in general, a user tends not to feel much difference unless there is a significant difference in the interface.

The major contributions of this work are as follows:

- We study how to personalize clustering result presentation and propose four adaptation strategies to improve clustering results based on a user’s implicit feedback information.

- We propose a stochastic way to simulate a user’s browsing behavior and a method to evaluate clustering results quantitatively based on user simulation.

- We evaluate the proposed adaptation algorithms with simulation experiments and show that adaptive clustering, especially reranking of documents based on viewed document can be quite beneficial for “smart users” (i.e., those who can select good clusters and identify relevant documents effectively).

- We conduct a user study to compare an adaptive clustering system with a static clustering
system to see adaptive clustering strategies can bring users better search utility and experience. We found no significant difference between two clustering systems.

Overall, our study shows that adaptive clustering has a good potential for improving search utility for users, but a user may not perceive any significant difference in the system.

The rest of the chapter is organized as follows. We first discuss related work in Section 7.2. We then describe the basic retrieval method, clustering algorithm, and query updating method in Section 7.3. We propose four adaptive clustering strategies in Section 7.4, and present a simulation study of the four strategies in Sections 7.5. In Sections 7.6, we present our user study. Finally, we conclude in Section 7.7.

### 7.2 Related Work

Clustering of search results has been extensively studied. In [24, 36], Scatter/Gather is proposed and evaluated as a document browsing method. In the evaluation, they consider the cluster with the largest number of relevant documents as the best cluster and linearize the best cluster according to the rank of documents in the cluster. This short linearized ranked list is compared with the equivalent number of documents in the original ranked list. In [27], the effectiveness of grouping search results is studied and it is found that presenting search results in context (with category information) is more effective than simply presenting search results as a ranked list.

Recently, there are some studies of incorporating the prior knowledge or user feedback into clustering algorithms in the semi-supervised setting [43, 40]. In our work, we instead focus on the feedback information collected from the user’s interactive process, i.e., how to dynamically change the cluster representation according to the continuous user interaction. Thus our work is complementary, and the algorithms proposed in [43, 40] can be combined with our methods.

There are some studies of clustering web search results. In [101], a web search clustering algorithm is applied to a search engine for clustering search results. This work shows the feasibility of clustering web search results on-the-fly. In [28], the same clustering interface for web search
is implemented using a hierarchical clustering algorithm. There are some other studies about efficiency of the clustering algorithm such as [17]. Vivisimo is a meta-search engine doing search result clustering.

In [102], the clustering problem is modeled as a salient phrase ranking problem and a regression model trained on human labeled training data is used to assign documents to relevant salient phrases to form candidate clusters.

Our work differs from the previous work on clustering in that instead of presenting a static clustering results as done in most previous work, our clustering results will dynamically change during the user’s interaction with the search engine.

Our work is also related to some recent work on implicit feedback (e.g., [78]) where click-through information is used to rerank documents. The difference is that we exploit implicit feedback information collected while a user is browsing clustering results and we attempt to reorganize clustering results while the previous work is based on a simple list presentation of search results.

### 7.3 Basic Algorithm Description

#### 7.3.1 Information Retrieval Method

We use the vector space model with Okapi BM25 term frequency weighting as our baseline retrieval method [70]. Specifically, the query vector is

\[ \vec{q} = (q_1, ..., q_{|V|}) \]

where \( q_i \) is the weight for term \( w_i \) in our vocabulary \( V \), and is given by the following TF-IDF weighting formula:

\[ q_i = \frac{(k_3 + 1)c(w_i, q)}{k_3 + c(w_i, q)} \log \frac{N + 1}{df(w_i) + 0.5} \]
where \( c(w_i, q) \) is the count of word \( w_i \) in the query \( q \), \( N \) is the total number of documents in the collection, \( df(w_i) \) is the number of documents that contain the term \( w_i \), and \( k_3 \) is a parameter.

The document vector is

\[
\vec{d} = (d_1, ..., d_{|V|})
\]

where \( d_i \) is the weight for term \( w_i \) in our vocabulary \( V \), and is given by the following TF-IDF weighting formula:

\[
d_i = \frac{k_1 c(w_i, d)}{c(w_i, d) + k_1 (1 - b + b \cdot \frac{dl}{avdl}) \log \frac{N + 1}{df(w_i) + 0.5}}
\]

where \( c(w_i, d) \) is the count of word \( w_i \) in the document \( d \), \( dl \) is the document length, \( avdl \) is the average document length, and \( k_1 \) and \( b \) are parameters.

The score of document \( d \) w.r.t. query \( q \) is computed as the dot product of \( \vec{q} \) and \( \vec{d} \). \( k_1, b \) and \( k_3 \) are parameters and set to 1.2, 0.75 and 1000, as recommended in [70].

### 7.3.2 K-Medoid Clustering

After the retrieval system retrieves a ranked list of results using the baseline retrieval method, the top \( N \) ranked results will be clustered into \( K \) clusters using the K-Medoids (a.k.a. PAM) algorithm, which is a partition based clustering algorithm [47]. A “Medoid” is the most centrally located object in a cluster. K-Medoids algorithm starts with an initial set of medoids and iteratively replaces one of the medoids by one of the non-medoids if it reduces the total distance of the resulting cluster. K-Medoids algorithm is more robust than K-Means in presence of noise and outliers because a medoid is less influenced by outliers than a mean.

Algorithm 1 outlines the document clustering process using a K-Medoids approach.

It takes \( O(K(N - K)^2) \) for each iteration in K-Medoids, where \( K \) is the number of clusters and \( N \) is the number of documents. Multiple iterations are needed before convergence. The output is \( K \) documents as representatives of each cluster center.
Algorithm 1 Document Clustering: K-Medoids

Input: A collection of document \( \{D_i\} \),
Number of representatives \( K \),
Output: A set of medoid documents \( D_{C_1}, \ldots, D_{C_K} \).

1: randomly select \( K \) documents as the initial cluster centers;
2: for each document \( D_i \) do
   assign its membership to the cluster \( C_j \) that has the largest similarity \( \text{sim}(D_i, D_{C_j}) \);
3: find the most centrally located document in each cluster;
4: repeat Lines 2-3 til small change in total sum of similarity;
5: return;

The similarity of two documents is computed as the dot product of two corresponding document vectors as defined above.

7.3.3 Query Updating (Feedback)

Given a selected cluster \( C \) by the user, we can update the query using all or a subset of documents in the selected cluster using a feedback method. When the user selects a document to view, we could also update the query vector based on just one single document for feedback. In our study, we use a modified Rocchio feedback method [71], which is defined as:

\[
\overrightarrow{q}' = \overrightarrow{q} + \alpha \frac{1}{M} \sum_{d \in C} \overrightarrow{d}
\]  

(7.1)

where \( \alpha \) is the interpolation coefficient. This coefficient can be chosen by running experiments on another set of query topics with known relevant documents in the collection. \( M \) controls how many documents we select from the cluster to update the query.

We set \( \alpha \) to 0.5 in all our experiments based on tuning on some training queries as described in Section 7.5. We set \( M \) to 6 in most experiments of this work. We also control the number of terms used for query updating by setting it to 20.
7.4 Four Adaptive Strategies

We propose four strategies for adapting clustering results based on user actions, including reranking documents based on a selected cluster, reranking documents based on a viewed document, merging unselected clusters, and promoting “near-miss” documents, which we will discuss below.

7.4.1 Reranking Based on Cluster Selection

When a user selects a cluster to view, we may infer that the user likes the selected cluster better than un-selected cluster(s). This information can be exploited to improve the ranking of documents within a cluster. The information about the selected cluster can be combined with the original query to rerank documents in other (unselected) clusters. Here the retrieval system does not change the cluster structure, i.e., the retrieval system does not move one document from one cluster to another cluster. Instead, documents within each cluster are reranked according to dot product scores of the updated query vector and document vectors, as described in Section 7.3.1. The updated query is computed according to Equation 7.1.

7.4.2 Reranking Based on Document Selection

When a user clicks on a document to view after selecting a cluster, the viewed document can presumably provide more information about what the user is interested in to the retrieval system and can thus be exploited to improve search results (i.e., implicit feedback [78]). In this strategy, the retrieval system would use the viewed document or snippet to update the query as:

\[
\overrightarrow{q'} = \overrightarrow{q} + \alpha \overrightarrow{d}
\]

(7.2)

where \( \overrightarrow{d} \) is the selected document vector. Here, the original query \( \overrightarrow{q} \) is the updated query \( \overrightarrow{q'} \) after the user selects a cluster. The updated query can be used to rerank the documents within each cluster. Here, the cluster structure does not change either.
7.4.3 Merging Unselected Clusters

When the user selects a cluster or views a document, the adaptive clustering algorithm can also restructure the clusters. In [24, 36], the retrieval system merges several relevant clusters according to user’s selection. After seeing a user selecting a cluster, it would be reasonable to assume that the user may not be so interested in the partitioning of search results in other clusters. Thus, the retrieval system can merge all unselected clusters into a big cluster and put this big cluster below the selected cluster. Then according to the updated query as computed by Equation 7.1, the retrieval system can rerank all the documents in the big cluster. In this strategy, the cluster structure is changed.

7.4.4 Promoting “Near Miss” Documents

In this strategy, when the user clicks on a cluster, the adaptive clustering algorithm presents not only the documents in the clicked cluster to the user, but also several documents from other clusters. Some (borderline) documents which were originally scattered into unselected clusters may be found to be relevant according to the user cluster selection. One specific strategy is that the retrieval system selects the most similar documents (“near miss” documents) to the updated query vector from each unselected cluster, and then adds these “near miss” documents to the bottom of the selected cluster. The updated query is computed using Equation 7.1. Using this strategy, the cluster structure is also changed.

7.5 Simulation Study

7.5.1 Experiment Methodology

Following the use of the simulation strategy in some previous work [36, 98], we also use the simulation strategy to evaluate the proposed adaptation strategies.

Different from previous work (e.g., [36]), our simulation distinguishes different types of users.
There are two types of extreme users. One type is “smart users”, who can always make intelligent decisions. Specifically, such a user would be assumed to always select the best cluster among several clusters according to the description of the cluster labels and always select a relevant document among a set of documents to view according to the snippet of each document. Here, we consider the cluster with the largest percentage of relevant documents as the best cluster, which is different from some previous work such as [36], where the authors considered the cluster with the largest number of relevant documents as the best cluster. We believe that using the percentage of relevant documents to select the best cluster makes more sense since the cluster with the largest number of relevant documents may contain many non-relevant documents as well.

Since we assume the smart user can select the best cluster according to the presented cluster information, e.g., cluster labels, we simulate the interaction of the smart user as follows. We compute the percentage of relevant documents in each of K clusters, sort K clusters according to the relevance percentage. We assume the smart user will select the top (also the best) cluster. We linearize or expand these K clusters into a ranked list (see Section 7.5.3 for an illustration) and evaluate the expanded ranked list with regular retrieval performance measures. We refer to the retrieval performance of this ranked list as the smart baseline.

After a smart user selects the cluster with the highest percentage of relevant documents, we consider this user interaction as offering an opportunity for implicit feedback. The retrieval system can thus update the user’s information need model. The original query is interpolated with the documents in the best cluster using Rocchio feedback. The number of documents in the best cluster, the number of terms and interpolation coefficients are parameters in this update process. The updated query is used to rerank documents in each cluster. We evaluate the performance of the new ranked list and refer to the result as the smart adaptive.

After the documents of the best cluster are presented, the smart user would select the best document in the best cluster to view. We simulate this behavior by selecting the top ranked relevant document in the best cluster. Immediately after the smart user selects the best document in the best cluster to view, the retrieval system will further update the user’s information need. The
current query, which has already been interpolated with documents in the best cluster, is further interpolated with the best document. We use the updated query to rerank documents in each cluster again. We refer to this result as the *smart reranking*.

The other type of extreme users is “dummy users.” A dummy user would always select top ranked cluster and top ranked document to view. That is, such a user would just passively follow a system’s ranked result. In real web search, it is found that a user’s behavior has a view bias and clickthrough can be biased according to the presented ranking order by a search engine [45]. The user tends to view or click on documents from the top.

As in the case of a smart user, we can also define *dummy baseline, dummy adaptive, and dummy reranking* similarly; the main difference is that the dummy user always chooses the top ranked cluster by the system to view and always chooses the top-ranked document within a cluster to view. For dummy users, clusters are ranked by averaging retrieval scores of all documents in each cluster.

In real world, a user can be considered as a mixture of smart user and dummy user. That is, on some occasions, the user can intelligently pick the best cluster to view, i.e. apply the strategy of the smart user; on other occasions, the user is influenced by the ranking of search engine and would simply open the top cluster to view, i.e., apply the strategy of the dummy user. We can use the probability of being smart user, which varies from 0.0 (dummy user) to 1.0 (smart user) to control the user interaction behavior.

### 7.5.2 Experiment Design

We use TREC8 ad hoc track data set (query topics 401-450) for empirical evaluation. For each query, we use vector space retrieval model to obtain baseline retrieval results. We then use K-Medoids clustering algorithm to cluster the top 100 documents into 6 clusters.

Before evaluating the adaptive clustering algorithms, we first run some basic experiments on TREC7 ad hoc track topics (350-400) to set values for some parameters (feedback coefficients, number of documents for feedback and number of terms for feedback), by which pseudo feedback
method can get best performance. TREC7 ad hoc track uses the same data collection as TREC8 but different query topics. Thus by training parameters on TREC7 query topics, we think the parameter values will also work on TREC8 so that the performance of adaptive clustering algorithms will not be affected by poor parameter setting. We use the mean average precision (MAP), precision at 0.1 (pr@0.1) and 0.2 (pr@0.2) recall levels, and precision at top 10 (pr@10d) and 20 (pr@20d) documents as the evaluation metrics.
7.5.3 Linearization of Cluster Results

In order to evaluate the performance of adaptive clustering representation, we employ a linearization method to “convert” any clustering results into a perceived ranked list. Specifically, The clusters are first sorted by either the percentage of relevant documents or the average retrieval score of documents in the cluster, depending on whether a smart user or a dummy user is assumed, respectively. In each cluster, the documents will be sorted by the retrieval scores. The idea is to simulate the perceived order of documents by a user when the user is browsing a clustering result. This way, we can evaluate a clustering result in the same way as evaluating a regular ranked list of result.

7.5.4 Experiment Results

Experiment Results of Reranking

Table 7.1 shows the retrieval performance of the experiment results for two types of extreme users at different stages, when we use 6 documents in the cluster and 20 terms in the feedback method and the interpolation coefficient is 0.5.

<table>
<thead>
<tr>
<th>Method</th>
<th>MAP</th>
<th>pr@0.1</th>
<th>pr@0.2</th>
<th>pr@10d</th>
<th>pr@20d</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseline</td>
<td>0.230</td>
<td>0.459</td>
<td>0.355</td>
<td>0.398</td>
<td>0.356</td>
</tr>
<tr>
<td>smart baseline</td>
<td>0.283</td>
<td>0.529</td>
<td>0.413</td>
<td>0.531</td>
<td>0.465</td>
</tr>
<tr>
<td>smart adaptive</td>
<td>0.282</td>
<td>0.559</td>
<td>0.418</td>
<td>0.539</td>
<td>0.470</td>
</tr>
<tr>
<td>smart reranking</td>
<td>0.294</td>
<td>0.580</td>
<td>0.428</td>
<td>0.551</td>
<td>0.467</td>
</tr>
<tr>
<td>dummy baseline</td>
<td>0.205</td>
<td>0.410</td>
<td>0.324</td>
<td>0.357</td>
<td>0.318</td>
</tr>
<tr>
<td>dummy adaptive</td>
<td>0.196</td>
<td>0.414</td>
<td>0.324</td>
<td>0.349</td>
<td>0.326</td>
</tr>
<tr>
<td>dummy reranking</td>
<td>0.202</td>
<td>0.418</td>
<td>0.331</td>
<td>0.353</td>
<td>0.320</td>
</tr>
</tbody>
</table>

From Table 7.1, we can see that for the smart user, the adaptive clustering strategy is effective and the performance of smart baseline is much better than that of baseline (we assume the user can smartly explore the results according to the cluster labels). For the smart user, smart reranking is
apparently better than smart adaptive and smart baseline while smart adaptive has a better pr@10d than smart baseline does. For the dummy user, however, the adaptive clustering strategy appears to be ineffective. The dummy baseline is not as good as the baseline, which means clustering presentation is not effective for dummy users. Dummy adaptive and dummy reranking have similar performances to dummy baseline. Thus the adaptive clustering representation is generally not effective for the dummy user.

**Number of Terms**

In the clustering presentation of search results, we need a way to represent a cluster. A common way to represent a cluster is to use some terms extracted from the documents in a cluster to label the cluster. These cluster labels will directly affect the user’s selection of cluster. Here we extract such terms using our feedback method (i.e., highly weighted terms in the Rocchio term vector will be selected). We vary the number of terms in the feedback method to simulate different number of terms used for labeling a cluster. Table 7.2 shows the experiment results of different number of terms for smart adaptive method.

<table>
<thead>
<tr>
<th>TermCount</th>
<th>MAP</th>
<th>pr@0.1</th>
<th>pr@0.2</th>
<th>pr@10d</th>
<th>pr@20d</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.281</td>
<td>0.526</td>
<td>0.413</td>
<td>0.529</td>
<td>0.462</td>
</tr>
<tr>
<td>2</td>
<td>0.284</td>
<td>0.539</td>
<td>0.421</td>
<td>0.514</td>
<td>0.455</td>
</tr>
<tr>
<td>3</td>
<td>0.285</td>
<td>0.546</td>
<td>0.416</td>
<td>0.516</td>
<td>0.459</td>
</tr>
<tr>
<td>5</td>
<td>0.285</td>
<td>0.542</td>
<td>0.414</td>
<td>0.527</td>
<td>0.461</td>
</tr>
<tr>
<td>10</td>
<td>0.283</td>
<td>0.547</td>
<td>0.413</td>
<td>0.527</td>
<td>0.461</td>
</tr>
<tr>
<td>20</td>
<td>0.284</td>
<td>0.568</td>
<td>0.415</td>
<td>0.535</td>
<td>0.464</td>
</tr>
</tbody>
</table>

The retrieval performance is not very sensitive to the number of terms used in the feedback.

**Individual Query Analysis**

We analyze the retrieval performance of individual queries for smart users to study what affects the performance of the adaptive strategy. The difference between smart adaptive strategy and smart
baseline strategy is computed and sorted for individual query topics. For each of the top 10 and bottom 10 queries topics, we then compute the number of relevant documents in the collection and the percentage of relevant documents in the best cluster respectively. The top 10 queries topics represent the topics on which the smart adaptive strategy has made the biggest improvement over the smart baseline strategy. The result is shown in Table 7.3. From Table 7.3, it can be seen that if we use low precision clusters to do adaptive clustering or there are few relevant documents in the collection, the retrieval performance would be improved only slightly or is even decreased sometimes (negative improvement for bottom 10 query topics). Only when we use high precision clusters to do adaptive clustering or there are many relevant documents in the data collection, would the retrieval performance increase.

Table 7.3: Study of individual query topics.

<table>
<thead>
<tr>
<th>Average value</th>
<th>Top 10 query topics</th>
<th>Bottom 10 query topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difference of MAP</td>
<td>0.117</td>
<td>-0.061</td>
</tr>
<tr>
<td>Number of relevant documents in the collection</td>
<td>87.5</td>
<td>25.4</td>
</tr>
<tr>
<td>Percentage of relevant documents in the best cluster</td>
<td>0.537</td>
<td>0.267</td>
</tr>
</tbody>
</table>

Experiment Results of Cluster Regrouping

In the cluster regrouping strategy, when the smart user selects one cluster, we will merge other clusters into one cluster so that there will be only two clusters – the active cluster and the unopened cluster. Immediately after the smart user selects a cluster to view, the retrieval system will update the user’s information need and use the updated query to rerank the documents in each cluster. We call this result the *regroup adaptive*. When the user selects the best document to view, we can then rerank the documents in each cluster; the corresponding result will be called *regroup reranking*. Table 7.4 shows the retrieval performance of regroup adaptive and regroup reranking.

From Table 7.4, we find that the retrieval performance of regroup adaptive and regroup reranking is not as good as the corresponding smart adaptive and smart reranking.
### Table 7.4: Experiment results for regroup clustering method

<table>
<thead>
<tr>
<th>Method</th>
<th>MAP</th>
<th>pr@0.1</th>
<th>pr@0.2</th>
<th>pr@10d</th>
<th>pr@20d</th>
</tr>
</thead>
<tbody>
<tr>
<td>regroup adaptive</td>
<td>0.261</td>
<td>0.557</td>
<td>0.401</td>
<td>0.533</td>
<td>0.45</td>
</tr>
<tr>
<td>regroup reranking</td>
<td>0.270</td>
<td>0.578</td>
<td>0.407</td>
<td>0.539</td>
<td>0.453</td>
</tr>
</tbody>
</table>

### Experiment Results of Near Miss Promotion

When we apply the promotion strategy to select a subset of documents from clusters other than the best cluster, we can use the original query $q$ to rank and select documents. We can also use the updated query $q'$, which is interpolated with the best cluster term vector, to rank and select documents. We tried both queries to promote documents. We promote one document from each cluster other than the best cluster and append it to the bottom of the best cluster. Table 7.5 shows the experiment results using this near miss promotion strategy. The first two rows show the experiment results using the original query to promote the documents and the third and fourth row show the experiment results using the updated query to promote the documents.

### Table 7.5: Experiment results of near miss promotion method

<table>
<thead>
<tr>
<th>Method</th>
<th>MAP</th>
<th>pr@0.1</th>
<th>pr@0.2</th>
<th>pr@10d</th>
<th>pr@20d</th>
</tr>
</thead>
<tbody>
<tr>
<td>promotion adaptive</td>
<td>0.282</td>
<td>0.561</td>
<td>0.421</td>
<td>0.537</td>
<td>0.457</td>
</tr>
<tr>
<td>promotion reranking</td>
<td>0.293</td>
<td>0.583</td>
<td>0.434</td>
<td>0.545</td>
<td>0.460</td>
</tr>
<tr>
<td>promotion adaptive</td>
<td>0.287</td>
<td>0.575</td>
<td>0.428</td>
<td>0.527</td>
<td>0.471</td>
</tr>
<tr>
<td>promotion reranking</td>
<td>0.291</td>
<td>0.582</td>
<td>0.428</td>
<td>0.549</td>
<td>0.452</td>
</tr>
</tbody>
</table>

From Table 7.5, we can see that promotion reranking consistently has better retrieval performance than the promotion adaptive strategy. The better performance of promotion reranking over promotion adaptive clearly comes from reranking documents based on the viewed document; this observation is consistent with what we observed in the performance comparison of smart adaptive and smart reranking.
7.6 User Study

7.6.1 Experiment Design

Besides the simulation study, we also conduct a user study by deploying two clustering systems to real users and study whether adaptive clustering strategy can bring better user experience in interacting with the search results than the static clustering strategy. Among the four adaptive clustering strategies, the strategy of promoting “near-miss” document strategy has shown some promising results. Promoting “near-miss” document strategy dynamically changes cluster structure, so it is likely to make a difference in the interface than simply reranking documents within a cluster. Thus we will use this strategy in the adaptive clustering system. In addition, we will also collect data from real user interactions and questionnaires to study whether there are other factors which may affect adaptive clustering strategies. For example, we can study whether the familiarity with topics has an impact on a user’s search experience.

We implement the clustering result presentation functionality in the UCAIR toolbar [79], which is an Internet Explorer (IE) plugin like Google toolbar. The adaptive clustering strategy is also implemented in the UCAIR toolbar. Thus we evaluate two systems with clustering result presentation, i.e. Adaptive System (AS) and Static System (SS). There is a menu of UCAIR toolbar, from which subjects can select to control clustering strategies (AS or SS) they use. The clustering interface is as shown in Figure 7.1.

We randomly select 6 query topics from TREC8 ad-hoc track topics 401-450. We modify the title of these query topics and remove the narrative to reflect the top ranking search results on WWW. All of these query topics are informational query [12]. One example of query topics is as in Figure 7.2. All subjects will just use the same title of each topic as the query to do the web search using one of clustering systems. We randomly divide 6 query topics into 2 groups. Each subject will use System SS to search 3 query topics and System AS to search the remaining 3 query topics. We vary the order of query topics, topic groups, and systems to remove the bias of the order and combination of system and query topics.
Figure 7.1: Clustering interface of search results

After the subjects submit title query to UCAIR toolbar, the UCAIR toolbar will return clustered results to the user by clustering top ranked 100 documents from Google into 6 clusters using K-Medoid clustering algorithm. We use the centroid document to represent each cluster. Subjects browse clusters and click one cluster to view snippets (title, summary, and URL) of documents which belong to the selected cluster. Subjects browse document snippets and then click the most interesting one to view the content of the document. If it is relevant, subjects will save it on the local disk. Subjects can go back to document list of the selected cluster or go back to cluster interface to select other clusters. There is no restriction about how many clusters subjects can
For each topic, we ask subjects to find as many relevant documents as possible and learn as much knowledge as possible about each topic in 15-minute period. Subjects are instructed not to save documents hastily. Subjects will save all relevant documents (html, txt, or pdf files) into one directory using the appropriate naming method.

### 7.6.2 Experiment Procedure

The user study procedure is described as follows.

1. Subjects read and sign the consent form and ask any questions that they may have, which take about 2 minutes.

2. Subjects have some warm-up and are shown a demo of how to do the search for a sample topic, which takes about 7 minutes.

3. Subjects fill out a questionnaire about the background, computer experience and previous searching experience, which takes about 3 minutes.

4. For each search, subjects fill out a pre-search questionnaire before subjects start each search. This takes about 2 minutes.

5. For each search, subjects perform information searches using one of two information retrieval systems. Subjects use around 15 minutes to conduct the search and save as many relevant documents as possible.

6. For each search, subjects complete a post-search questionnaire after completing each search, which takes about 3 minutes.

7. After subjects finish 3 searches per system, they are asked to complete a post-system questionnaire. This should take about 2 minutes for each system.
8. Subjects are asked to fill out an exit questionnaire after subjects have completed searching so that we can learn more about their experience with the systems. This step takes about 5 minutes.

We set three breakpoints at the end of first, third, and fourth search respectively, to make sure the search activities are conducted in the correct way.

All questionnaires are adopted from a user study [99, 100] conducted in Rutgers University and we make small changes.

7.6.3 Data Collection

We recruit 29 subjects to participate in the user study by posting advertisement at subject recruitment website of the department of psychology. We first conduct a pilot study for 3 users. Two of them are volunteers and Ph.D. students of Computer Science and Chemistry respectively. One of them is a paid undergraduate. After the pilot study, we polish the experiment design including questionnaires and procedure, topics, and system implementation.

During the formal user study, we ask 26 paid subjects (nearly all of them are undergraduate) to participate in the user study. At each session, at most 2 subjects participate in the user study. The user study are conducted in one week. All subjects spend 1.5 to 2 hours on the user study. After the user study, we collect saved relevant documents, questionnaires, and user interaction log. For each topic, subjects save some relevant documents on the local disk.

7.6.4 Experiment Results

Overview

We collect data (questionnaires, log, and saved documents) from the first 24 subjects in the formal user study for analysis. There are 12 experiment settings, varying the order of systems, topic groups, and topic orders. Thus the experiment setting of subject 13 is the same as that of subject 1, the experiment setting of subject 14 is the same as that of subject 2, etc.
All but one subjects are between 16 and 25 years old. There are 16 female and 8 male subjects. Subjects have very diverse majors including nursing, social science, business, physical science, and engineering. All of them use computer daily and have high level expertise with searching. When they use search engines to find information, the number of search results pages they view if one result page has 10 search results is shown in Table 7.6.

Table 7.6: Number of search result pages viewed by subjects

<table>
<thead>
<tr>
<th>Pages</th>
<th>1</th>
<th>2</th>
<th>3-5</th>
<th>6-10</th>
<th>more than 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>2</td>
<td>8</td>
<td>12</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

From Table 7.6, we can see that nearly all subjects will view less than 10 result pages, which shows that it makes sense that we only cluster top ranked 100 search results.

We collect the exit questionnaires and compare two systems by overall experiences including difference of two systems, helpfulness of systems in completing tasks, easiness of learning to use, easiness of using, and overall preference. The results are listed as Table 7.7 and Table 7.8.

Table 7.7: Comparison of difference of two clustering systems

<table>
<thead>
<tr>
<th>Difference</th>
<th>1 (Not at all)</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7 (Extremely)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
<td>2</td>
<td>6</td>
<td>5</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.8: Comparison of overall experience of two clustering systems

<table>
<thead>
<tr>
<th>Comparison</th>
<th>AS is better</th>
<th>SS is better</th>
<th>No difference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Helpfulness</td>
<td>10</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>Easy to Learn</td>
<td>1</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Easy to Use</td>
<td>7</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>Overall Preference</td>
<td>11</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

From Table 7.7, we can see that most subjects do not think they see much difference between two clustering systems. But many subjects (22 subjects) actually see some differences. Indeed, Two clustering systems have very similar, but subtly different user interfaces. Since System AS moves some documents from other clusters into the selected cluster, the user will generally see
more documents in the selected cluster in AS than in SS. It is found that subjects can notice System SS sometimes generates clusters with very small number of search results while subjects are generally insensitive to large clusters which have a great number of search results. Since one side effect of promoting “near-miss” document strategy is to increase the cluster size, subjects do not have experience of selecting small clusters using System AS. However, if the subject views multiple clusters during the interaction, he/she may see the same document at multiple clusters.

From Table 7.8, we can see that System AS is a little better than SS in the aspect of being helpful in completing tasks and easy to use. Both systems are equally easy to learn since their interface is nearly identical. For the overall preference, System AS seems to be better than System SS. However, there is no clear indication that System AS is better than System SS.

**Number of Saved Documents**

We compute the number of saved document for each topic for each subject. Then we compute the average and standard deviation of each topic for two clustering systems, which is listed as Table 7.9.

<table>
<thead>
<tr>
<th>Topic</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg (AS)</td>
<td>6.75</td>
<td>4.5</td>
<td>5.92</td>
<td>5.33</td>
<td>6</td>
<td>4.912</td>
</tr>
<tr>
<td>Avg (SS)</td>
<td>9.75</td>
<td>5.5</td>
<td>7.92</td>
<td>5.33</td>
<td>5.83</td>
<td>5.58</td>
</tr>
<tr>
<td>Std (AS)</td>
<td>3.02</td>
<td>2.71</td>
<td>3.15</td>
<td>1.83</td>
<td>2.92</td>
<td>3.03</td>
</tr>
<tr>
<td>Std (SS)</td>
<td>5.21</td>
<td>4.01</td>
<td>3.90</td>
<td>2.23</td>
<td>2.72</td>
<td>2.39</td>
</tr>
</tbody>
</table>

From Table 7.9, it looks that similar numbers of documents are saved for each topic using two different clustering systems. Although more documents are saved with the static interface, the t-test value of average numbers of saved documents is 0.08, which is not statistically significant.

We compute the average number of saved documents for each system when the users have different preferences of two clustering systems. The results are shown in Table 7.10.

<table>
<thead>
<tr>
<th>Topic</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg (AS)</td>
<td>6.75</td>
<td>4.5</td>
<td>5.92</td>
<td>5.33</td>
<td>6</td>
<td>4.912</td>
</tr>
<tr>
<td>Avg (SS)</td>
<td>9.75</td>
<td>5.5</td>
<td>7.92</td>
<td>5.33</td>
<td>5.83</td>
<td>5.58</td>
</tr>
<tr>
<td>Std (AS)</td>
<td>3.02</td>
<td>2.71</td>
<td>3.15</td>
<td>1.83</td>
<td>2.92</td>
<td>3.03</td>
</tr>
<tr>
<td>Std (SS)</td>
<td>5.21</td>
<td>4.01</td>
<td>3.90</td>
<td>2.23</td>
<td>2.72</td>
<td>2.39</td>
</tr>
</tbody>
</table>

From Table 7.10, we can see that there is no apparent correlation between the number of saved
Table 7.10: Comparison of number of saved documents of two clustering systems over different system preference

<table>
<thead>
<tr>
<th>Topic</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>SS</td>
<td>8.5</td>
<td>5.17</td>
<td>7</td>
<td>4.8</td>
<td>5.4</td>
<td>4.8</td>
</tr>
<tr>
<td>AS</td>
<td>5.8</td>
<td>4.2</td>
<td>5</td>
<td>4.83</td>
<td>4.83</td>
<td>4</td>
</tr>
<tr>
<td>SS</td>
<td>14.67</td>
<td>7.33</td>
<td>11.67</td>
<td>5</td>
<td>8</td>
<td>6</td>
</tr>
<tr>
<td>AS</td>
<td>6.5</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>SS</td>
<td>7.33</td>
<td>4.33</td>
<td>6</td>
<td>6</td>
<td>5.4</td>
<td>6.2</td>
</tr>
<tr>
<td>AS</td>
<td>7.8</td>
<td>3.4</td>
<td>5.6</td>
<td>5.67</td>
<td>5.33</td>
<td>4.67</td>
</tr>
</tbody>
</table>

documents using specific clustering systems and user preference of clustering systems. It looks that no matter what preferences users have, they generally save a few more documents using System SS than using System AS, which is especially true for the first three topics. Further analysis is needed to understand why.

**Topic Familiarity Factor**

The number of saved documents may be related with the topic familiarity level of subjects. Thus we compute the average and standard deviation of each topic familiarity level, which varies from 1 (not at all) to 7 (extremely familiar). We also compute the average number of saved document and standard deviation for each topic without differentiating clustering systems used, shown in Table 7.11.

Table 7.11: Topic familiarity and number of saved documents

<table>
<thead>
<tr>
<th>Topic</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Topic Familiarity Avg</td>
<td>3.42</td>
<td>2.71</td>
<td>3.79</td>
<td>1.21</td>
<td>1.21</td>
<td>1.79</td>
</tr>
<tr>
<td>Saved Document Avg</td>
<td>8.25</td>
<td>5</td>
<td>6.92</td>
<td>5.33</td>
<td>5.92</td>
<td>5.25</td>
</tr>
<tr>
<td>Topic Familiarity Std</td>
<td>1.35</td>
<td>1.27</td>
<td>1.47</td>
<td>0.41</td>
<td>0.72</td>
<td>1.25</td>
</tr>
<tr>
<td>Saved Document Std</td>
<td>4.44</td>
<td>3.39</td>
<td>3.61</td>
<td>1.99</td>
<td>2.76</td>
<td>2.69</td>
</tr>
</tbody>
</table>

From Table 7.11, we can clearly see that topic 1 (Parkinson disease treatment), 2 (wildlife preserve poaching impact), and 3 (curbing population growth) are more familiar to subjects than topic 4 (legal Pan am 103), 5 (Schengen agreement border control), and 6 (new steel production).
However, there is no apparent correlation between the topic familiarity level and number of saved documents.

**Time Dynamics of Document Saving**

We record the timestamp of document saving actions by subjects to investigate whether subjects would be able to save documents more quickly with one system than with the other. For each document saving action, we compute the elapsed time (minutes) since the beginning of user study. Subjects $2n - 1$ and $2n$ ($n = 1, 2, ..., 12$) have same experimental settings except that subjects $2n - 1$ first use SS System while subjects $2n$ first use AS system. We plot curves of document saving action over elapsed time for each pair of subjects $2n - 1$ and $2n$ to compare the time dynamics of them. A representative plot for subject 5 and subject 6 is shown in Figure 7.3.

![Time Dynamics of Subject 5 and Subject 6](image)

**Figure 7.3:** Time dynamics comparison of subject 5 vs. subject 6.

From Figure 7.3, we could not see the significant difference between each pair of subjects. Thus it is likely that both systems have similar impact on the pace of document seeking activities of users.

During the user study, we explicitly asked subjects to do each task for around 15 minutes and communicated with them after they finished the first, third, and fourth tasks. Thus it is not meaningful to study the total time spent for each task.
7.7 Conclusion and Future Work

In this work, we explore adaptive clustering presentation in interactive information retrieval and propose four adaptation strategies (reranking documents based on a selected cluster, reranking documents based on a viewed document, merging unselected clusters, and promoting “near-miss” documents) to improve clustering results based on a user’s implicit feedback information.

We propose a stochastic way to simulate a user’s browsing behavior and propose a method for evaluating clustering results quantitatively based on user simulation. We evaluate the proposed adaptation algorithms with simulation experiments and show that adaptive clustering, especially reranking of documents based on viewed document is effective for smart users who would intelligently identify and view a high precision cluster and pick a relevant document to view, though such strategies are not effective for dummy users who simply follow a system’s ranking of clusters and documents.

Besides the simulation study, we also conduct a user study to see if an adaptive clustering system can bring improved search utility and/or experience to users, compared with a static clustering system. The results show that there is generally no significant difference between the two systems from a user’s perspective. Specifically, more users say that they like the adaptive system better but users saved more relevant documents with the static interface than with the adaptive interface. Overall, our study shows that adaptive clustering has a good potential for improving search utility for users, but a user may not perceive any significant difference in the system.

There are a lot of interesting future directions to explore. First, we can explore other strategies of adaptive clustering in interactive information retrieval. Second, we can study adaptive clustering algorithms for mixed user behavior in the multiple runs of the same task. Currently, we just propose a stochastic way to simulate the same user in different search tasks. From some retrieval task, the user has to have several interactions with the retrieval system to get the satisfactory results. In this scenario, the user will act differently during a sequence of actions. It will be interesting to study the user interaction in multiple interactions for the same search task. Third,
we can study the adaptive clustering algorithms for different categories of queries. For example, the user interaction will be different for easy tasks and complex tasks. There are also other factors that can affect the user interaction behavior [97]. Thus the adaptive clustering algorithm ideally should use different adaptive strategies according to the characteristics of the task. Fourth, we will study the factor of the underlying clustering algorithms. In this study, we only apply one clustering algorithm (K-Medoid algorithm). It would be interesting to explore whether different clustering algorithms such as spectral clustering or hierarchical clustering algorithms will affect the performance of adaptive clustering algorithms. Finally, although the user study comparing one method of adaptive clustering to static clustering showed no significant user performance or preference differences, this could be due to a variety of factors which we were unable to investigate in this study. Such factors include suitability of the clustering technique to the specific retrieval task; combining user behavior evidence for ranking as well as clustering; and combining several adaptation strategies, rather than using only one.
Chapter 8

Privacy Protection in Personalized Search

Personalized search is a promising way to improve the accuracy of web search, and has been attracting much attention recently. However, effective personalized search requires collecting and aggregating user information, which often raise serious concerns of privacy infringement for many users. Indeed, these concerns have become one of the main barriers for deploying personalized search applications, and how to do privacy-preserving personalization is a great challenge. In this chapter, we systematically examine the issue of privacy preservation in personalized search. We distinguish and define four levels of privacy protection, and analyze various software architectures for personalized search. We show that client-side personalization has advantages over the existing server-side personalized search services in preserving privacy, and envision possible future strategies to fully protect user privacy.

8.1 Introduction

Although search engines have been successfully deployed to serve users’ information needs, they are far from optimal. A major deficiency of existing search engines is that they follow the model of “one size fits all” and are not adaptive to individual users. This causes inherent non-optimality as is seen clearly in the following two cases: (1) Different users may use exactly the same query (e.g., “Java”) to search for different information (e.g., the Java island in Indonesia or the Java programming language), but existing search engines return the same results for these users. (2) A user’s information needs may change over time. The same user may use “Java” sometimes to mean the Java island in Indonesia and sometimes to mean the programming language. Existing
search engines are unable to distinguish such cases.

Clearly, without using more user information and/or the search context of a user it is impossible for a search engine to know which sense “Java” refers to in a query. In order to optimize search accuracy, we must use more user information and personalize search results according to each individual user [64]. To see how personalized search may help improve search accuracy, consider the query “Java” again. The intended meaning of “Java” can often be easily determined by exploiting some naturally available information about a user. Indeed, any of the following additional information about the user could help determine the intended meaning of “Java” in the query: (1) The user is a computer science student as opposed to a travel agent. (2) Before entering this query, the user had just bookmarked or viewed a web page with many words related to the Java programming language, such as “programming” and “applet”. (3) The previous query that the user entered is “object-oriented programming” as opposed to “cheap flight ticket”. Exploiting such user information to optimize the ranking of search results for a particular user is very appealing because it does not require any extra effort from the user. In general, personalized search is considered as one of the most promising techniques to break the limitation of current search engines and improve the quality of search results.

Despite the attractiveness of personalized search, we have not yet seen large scale uses of personalized search services. This is not because such services are not available, but likely because users are not comfortable with the lack of protection of user privacy [46, 73]. Google, for example, has deployed a personalized search system \footnote{http://www.google.com/psearch}. However, to the best of our knowledge, it has not been widely adopted by users yet.

Indeed, there is an inherent tension between providing personalized search and privacy preservation since personalized search requires collecting and aggregating a lot of user information. Specifically, in order to personalize search, a user profile or user model must be constructed to accurately represent a user’s information need. To build a precise user profile, a lot of user information including query and clickthrough history is often aggregated. However, from a user’s
privacy perspective, such a user profile can reveal a gamut of user’s private life such as political inclination, family life, and hobbies, which is clearly a serious concern for users. Thus there appears to be a dilemma: high-accuracy Web search requires accurate user modeling which increases the risk of privacy infringement. Indeed, the privacy concern is one of the major barriers in deploying serious personalized search applications, and how to achieve personalized search while preserving users’ privacy is a great challenge to be solved.

In this chapter, we systematically examine the issue of privacy preservation in personalized search. We distinguish and define four levels of privacy protection, and analyze various software architectures for personalized search. We show that client-side personalization has advantages over the existing server-side personalized search services in preserving privacy, and envision possible future strategies to fully protect user privacy. We also propose several research questions related with the “identifiability” in privacy-preserving personalized search.

The remaining sections are organized as follows. In Section 8.2, the search privacy problem is defined in a formal way. In Section 8.3, different levels of privacy are defined and analyzed. In Section 8.4, possible software architectures of personalized web search system are described and analyzed from the privacy protection perspectives. In Section 8.6, some research questions related with privacy-preserving personalized search is defined. Section 8.7 is a summary of our vision of privacy preservation in personalized search.

### 8.2 Privacy Concern in Web Search

Search involves interactions between two parties, a user \((U)\) and a search engine \((S)\).

There are two basic interaction cycles between a user and a search engine: (1) Search: A user \(U\) composes and submits a query \(q\) to search engine \(S\), and the search engine \(S\) would return some search results \(R = \{R_1, \ldots, R_n\}\) to the user. (2) Browse: A user \(U\) chooses to view a result \(R_i \in R\), and the search engine would bring the user the content of \(R_i\).

In a search process involving many such interaction cycles, a user thus potentially reveals the
following three kinds of personal information:

1. User identity: This could be a personal user ID in the case when the user has to register an account, or the IP address of the machine that the user is using.

2. Queries: This includes all the queries the user has submitted to the search engine.

3. Viewed results: This includes all the viewed web pages by the user.

Actually, the user also reveals some context information such as the time stamp, but such information is not central to the issue of privacy, thus we do not consider it in this paper.

Since such personal information can potentially reveal a gamut of user’s private life such as political inclination, family life, and hobbies, disclosing such information, especially in an aggregated fashion, would clearly raise serious concerns for users.

One may notice that there is a remarkable difference between user’s queries and clicked search results. Since queries are composed by users themselves, thus directly reveal the user’s information need, while the search results are composed by the Web page publishers. Thus in general, queries may contain much more personally identifiable information (PII) than viewed search results. However, from the viewpoint of privacy protection, both queries and viewed results can cause concerns for users and the difference appears to be not crucial. Thus in the rest of the chapter, we do not distinguish the queries and viewed search results; instead we refer to them together as descriptions of user’s information needs.

Thus in any search activity, the information a user $U$ potentially reveals when attempting to satisfy an information need $N$ can be represented as $(ID(U), TEXT(N))$, where $ID(U)$ is some ID revealed about the identity of the user (e.g., a user ID or an IP address), and $TEXT(N)$ is a text description of the information need $N$ (e.g., a set of related queries and/or viewed results).

When a user conducts a series of $k$ search activities, the sensitive personal information that the user may reveal can be represented as $P(U) = \{(ID(U, i), TEXT(N, i))\}$ where $i = 1, ..., k$.

The privacy concern of a user is that all or some of the information in $P(U)$ may be captured by some other people in the world. The concern may be less if $P(U)$ is revealed to some “trustable”
party (e.g., a search engine company that has a clearly written policy on privacy protection) than to some “untrustable” parties (e.g., any third party who has access to the web search log).

Note that $P(U)$ is precisely what is needed to help a search engine better understand the user’s information need. Thus performing personalized search in some sense “requires” a user to release $P(U)$. Such tension has created a barrier for deploying personalized search applications, and the main challenge of privacy-preservation personalized search is to exploit $P(U)$ to help improve the search service for $U$ while protecting $P(U)$ as much as we can from being known by anyone else in the world.

### 8.3 Levels of Privacy Protection in Personalized Search

Different users have different requirements of privacy protection. While some users may not want anyone else to know or hold any of their personal information, others may be willing to share some personal information for better search results or services. Thus the level of privacy protection may need to be tuned for different users to accommodate different preferences for the tradeoff of personalization and privacy protection. In this section, we define and analyze four levels of privacy protection in personalized search.

#### 8.3.1 Level I: Pseudo Identity

A personalized web search system has Level I privacy protection (Pseudo Identity) if:

a The user identity $ID(U)$ is replaced by a pseudo identity $ID^p(U)$ which contains less personally identifiable information than $ID(U)$ does.

b The description of user information needs $TEXT(N, i)$ can be aggregated according to $ID^p(U)$ at the search engine side.

$ID(U)$ can generally be mapped to a single or a small group of users (e.g., family members) with the help of public databases. For example, given an IP address, geographic information
such as city and state can be known through the whois service. With a pseudo identity $ID^p(U)$, such mapping is not available and some personal information such as the location of the user is protected.

From the viewpoint of personalized search, a pseudo identity $ID^p(U)$ can still be used to group all the descriptions of user information needs to build a user profile without needing $ID(U)$. The content of user profile such as queries and clickthrough is intact at the search engine side. This complete and clean descriptions of user information need can then be exploited to support personalize web search. For example, when AOL released their search engine log in August, 2006, they replaced IP addresses with pseudo identities [7].

Level I is the lowest level of privacy protection. Because of the removal of $ID(U)$, which may otherwise be used to directly identify a user, some people who do not care much about privacy may accept this level of privacy protection. Unfortunately, this level is not enough to protect a user’s privacy because it allows aggregation of all the information need descriptions of a user, which can in turn facilitate identification of the user. Since queries directly indicate a user’s interests, being able to group many queries from the same user makes it quite possible to identify a user. For example, a New York Times reporter identified a lady in Lilburn, Georgia according to the released AOL query logs.

### 8.3.2 Level II: Group Identity

A personalized web search system has Level II privacy protection (Group Identity) if:

a A group of users share a single user identity $ID(U)$.

b The description of user information needs $TEXT(N, i)$ is aggregated at the group level according to $ID(U)$.

This level of protection is achieved when a group of users send their profiles to the search engine in such a way that the search engine can only build a group user profile for the group instead of a user profile for each single user.
In this case, personalized web search can not be done at the individual user level, but is possible at the group level. This may reduce the effectiveness of personalization because a group’s information need description is used to model an individual user’s information need. However, if the group is appropriately constructed so that people with similar interests are grouped together, we may have much richer user information to offset the sparse description of individual user information needs. Thus the search performance may actually be improved because of the availability of more information from the group profile.

Level II has higher privacy protection than Level I. At this level, one cannot construct an individual user profile. Instead, only an aggregated profile for a group of users can be constructed. Since the identity information of an individual user \( ID(U) \) is lost in a group of identity, and the description of user information needs \( TEXT(N, i) \) is also mixed with those of other users, it is difficult to infer true information needs of any individual user if the group is appropriately constructed.

A common way to implement the Level II privacy protection is to set up a proxy for a group of users and all the users would communicate with the search engine through the proxy. Currently, there are many public proxy servers available on the Internet.

The obfuscation of query terms can also be considered as an indirect way to achieve Level II privacy protection. For example, TrackMeNot \(^2\), a Firefox web browser plug-in, protects web searchers’ identities by periodically issuing randomized search queries to search engines. This method can effectively mix the description of real user information needs with other people’s description of user information needs if the noisy queries are carefully constructed so as to resemble common queries. Thus the goal of sending noisy queries is to make a single user profile look like a group of user profiles, i.e., a user profile is undistinguished from a group of other user profiles. This method can also be considered as a way to realize k-anonymity [88].

\(^2\)http://mrl.nyu.edu/~dhowe/trackmenot/
8.3.3 Level III: No Identity

A personalized web search system has Level III privacy protection (No Identity) if:

a The user identity $ID(U)$ is not available to the search engine.

b The description of user information needs $TEXT(N, i)$ can not be aggregated on the search engine side, even at the group level.

At Level III, a search engine can not know $ID(U)$ of individual users at all, thus it has no way to aggregate the description of user information needs. At this level, however, it would be impossible to build a user profile on the search engine side, even at the group level. Since the search engine does not have the user profile, personalized search must be supported on a user’s own computer. Specifically, the user profile $P(U)$ can be kept on the personal computer of the user $U$. Personalized search can be achieved by combining general Web search with a local, personalized reranking of results.

A possible way to implement Level III privacy protection is through the anonymous network. For example, The web browser Torpark 3, which is based on Tor (The Onion Router) 4, enables the user to communicate anonymously on the Internet. When the user searches the Web using Torpark, the search engine would not be able to decide where the search originally comes from, but the search results can still be returned to the correct user through Tor network.

Level III has a higher privacy protection than Level II. At Level III, it is impossible for the search engine to aggregate any information about the individual user, even at the group level. However, some user information is still kept at the search engine side. For example, the original user queries may be kept at the search engine side. Although a user’s query generally does not explicitly contain personal identity $ID(U)$, it sometimes contains quite sensitive information (It is known that some queries contain social security numbers.) It is thus still possible to infer a user’s identity just from a query. We will further discuss this issue in Section 8.6.

---

3http://www.torrify.com/
4http://tor.eff.org/
8.3.4 Level IV: No Personal Information

A personalized web search system has Level IV privacy protection (No Personal Information) if:

a Neither the user identity $ID(U)$ nor the description of user information need $TEXT(N)$ is available to the search engine.

At Level IV, a search engine does not know $ID(U)$ of an individual user or the description of user information need $TEXT(N)$ at all. However, the search engine can still return the normal search results to the correct user. Thus the user privacy is fully protected.

On the surface, it appears to be impossible to achieve this level of privacy protection. However, cryptography methodology [32] may be applied to realize this ultimate level of privacy protection. For example, the search engine can release the index to a trusted third party; the user sends the query to the trusted third party and the third party does the search and returns the results to the user. Nevertheless, it is a challenge to design a communication protocol to make sure the ultimate privacy is guaranteed on both the search engine side and the third party side.

Another possibility for achieving the Level IV privacy protection is that a search engine would be required by law to guarantee that it does not store any user information ($ID(U)$ or $TEXT(N)$). That is, the search engine will have no memory of any activity of a user, even though it would still respond to a user search request directly. This scenario can be considered to be equal to the scenario that the search engine does not know any information about the user. As in the case of Level III privacy protection, since a search engine cannot construct any kind of user profile, personalized search must be supported on the user’s computer.

Level IV has the highest level of privacy protection for personalized search. However, it may also have the highest cost due to higher communication cost and encryption/decryption cost, which will delay real-time response. In another form, the cost is that the search engine gives up the logging of any user information, which could otherwise be useful for other purposes such as anti-spam or detection of attacks.
8.4 Software Architecture for Personalized Search

For Web search applications, server-client architecture, as shown in Figure 8.1(a), is commonly adopted, where a client (often the web browser) sends queries to a server (the search engine). The search engine analyzes the user information need, looks up its index structure of documents, and returns a ranked list of search results to the client for a user to view. A search engine generally stores user search logs for various kinds of purposes including personalization and anti-spam. Thus it is to the interest of search engines not to remove the search engine logs automatically. Indeed, they tend to keep the search engine logs indefinitely.

There are three kinds of software architectures that expand the basic server-client model of Web search to support personalized search. Their main differences lie in where personally identifiable information $P(U)$ is stored and how it is exploited for personalization. In this section, we describe these three kinds of software architectures and analyze what levels of privacy preservation can be achieved with these different architectures.

![Software architecture of personalized web Search](image)

Figure 8.1: Software architecture of personalized web Search
8.4.1 Server-side Personalization

For server-side personalization as shown in Figure 8.1(b), the personally identifiable information $P(U)$ is stored on the search engine side. The search engine builds and updates the user profile either through the user’s explicit input (e.g., asking the user to specify personal interests) or by collecting the user’s search history implicitly (e.g., query and clickthrough history). Both approaches require the user to create an account to identify himself. But the latter approach requires no additional effort from the user and contains richer description of user information need.

The advantage of this architecture is that the search engine can use all of its resources (e.g., document index, common search patterns) in its personalization algorithm. Also, the client software generally requires no changes. This architecture is adopted by some general search engines such as Google Personalized 5.

Currently most personalized search systems with server-side personalization architecture require the user to give consent before his/her search history can be collected and used for personalization. If the user gives the permission, the search engine will hold all the personally identifiable information possibly available on the server side. Thus from the user perspective, it even does not have level I privacy protection. Since many users fear its potential privacy infringement by search engines, this has hindered the wide adoption of personalization with this architecture.

However, if the search engine decides to voluntarily replace the user identity $ID(U)$ with a pseudo user identity $ID^p(U)$, Level I privacy protection can be achieved. When the search engines release the search engine logs to the public or a group of researchers, they generally replace user identity $ID(U)$ by a pseudo user identity $ID^p(U)$. To the third parties receiving these search engine logs, which may use it for personalized search purpose, the user will have Level I privacy protection.

If the user decides to use a proxy to communicate with the search engine, all user information going through the same proxy will be combined in a user profile. Through this method, Level II privacy protection can be achieved. However, this method does not always work: When the

5http://www.google.com/psearch
search engine uses the user login ID to collect user information, this method will not achieve Level II privacy protection; when the search engine only uses the IP address to aggregate the user information, this method works. Sometimes, search engines group users randomly or according to some criteria before they release the search engine logs. Then the user will also have Level II privacy protection to those third parties which receive the search engine logs.

It is impossible to implement Level III or Level IV privacy protection if personalization is done on the server side.

### 8.4.2 Client-side Personalization

For client-side personalization as shown in Figure 8.1(c), the personally identifiable information is always stored on a user’s personal computer. As in the case of server-side personalization, the user profile can be created from user specification explicitly or search history implicitly. The client sends queries to the search engine and receives results, which is the same as in the general web search scenario. But given a user’s query, a client-side personalized search agent can do query expansion to generate a new query before sending the query to the search engine. The personalized search agent can also rerank the search results to match the inferred user preferences after receiving the search results from the search engine.

With this architecture, not only the user’s search behavior but also his contextual activities (e.g., web pages viewed before) and personal information (e.g., emails, browser bookmarks) could be incorporated into the user profile, allowing for the construction of a much richer user model for personalization. The sensitive contextual information is generally not a major concern since it is strictly stored and used on the client side. Another benefit is that the overhead in computation and storage for personalization can be distributed among the clients. A main drawback of personalization on the client side is that the personalization algorithm cannot use some knowledge that is only available on the server side (e.g., PageRank score of a result document). UCAIR [79] adopts the client-side personalization.

With proxy functionality applied to the client side, Level II privacy protection can be achieved.
If the client side uses an anonymous network such as Tor to communicate with the search engine, Level III privacy protection can also be achieved. In order to achieve Level IV privacy protection, additional cooperation of the search engine would be needed as we described in Section 8.3.

### 8.4.3 Client-Server Cooperative Personalization

For the client-server cooperative personalization as shown in Figure 8.1(d), it is a compromise between the previous two kinds of architectures. The user profile is still stored on the client side, but the server also participates in personalization. At query time, the client extracts contextual information from the user profile, and sends it to the search engine along with the query. The search engine then does personalization with the received context. Compared with client-side personalization, this architecture has an advantage of allowing for the use of a search engine’s internal resources.

The contextual information sent to the server specifies the user’s search preferences (e.g., query expansion terms, topic weight vector). It is extracted from the user profile (e.g., the weight vector can be learned from search history), and is only relevant to a particular query. Therefore, it is a condensed version of the whole user profile (generally a few terms or a weight vector from a user’s search history), thus the architecture can minimize the personal information obtained by the search engine.

A main drawback is that the condensed contextual information may not be as powerful as the whole user profile. We have not yet seen any personalization products in this category, probably due to the relatively complex architecture.

This architecture provides the same level of privacy protection as server-side personalization. However, the personally identifiable information collectable on the server side is less than in the case of pure server-side personalization.
8.5 Privacy Protection in Current Web Search Systems

Currently, there are a variety of search engines on WWW – general search engines such as Google and Yahoo!, meta-search engines such as dogpile and ixquick, special search engines such as cluster search engine vivisimo, and personalized search systems such as UCAIR [79]. In this section, we analyze privacy protection for some of these typical search paradigms.

8.5.1 Autonomous Search Engines

When people do web search with an autonomous search engine such as Google, Yahoo, or MSN, both the IP address and query terms are stored on the search engine side unless the user uses a proxy or anonymous communication system additionally. Although Google has a strict and clear privacy policy, the personally identifiable information $P(U)$ is stored on Google servers and the users have no full control of their personal information. According to the levels of privacy protection described in Section 8.3, it does not even satisfy Level I privacy protection unless the user applies some privacy protection measures to strengthen the privacy protection themselves.

Users are generally not comfortable with counting on others to protect their privacy. Recent history has witnessed several privacy infringement incidents when some companies accidentally or willingly had violated such trust and were facing bankruptcy courts, civil subpoenas or lucrative mergers [1].

8.5.2 Meta Search Engines

There are quite a few meta search engines on the Web such as Dogpile, Looksmart and ixquick. A meta-search engine sends user requests to several autonomous search engines and reranks search results returned from each one. When people use the meta search engines, autonomous search engines only receive all user queries from the single meta search engine. Thus there is the Level III

---

6http://www.google.com/privacy.html
7http://www.ixquick.com/
privacy protection to those underlying autonomous search engine. However, there is no automatic privacy protection for the users of these meta search engines, which is the same as the scenario when people directly use autonomous search engines.

Interestingly, the meta search engine ixquick guarantees that it removes the IP addresses of users and keep no other unique identity. Thus $ID(U)$ of personally identifiable information is not stored on the server side although $TEXT(N)$ still is. It provides Level III privacy protection for the users of this meta search engine, but ixquick has no personalization functionality.

8.5.3 Client-side Personalized Search Tools

There are also some client-side personalized search tools such as Stuff I’ve Seen [26], Phlat [23] and UCAIR [79]. These client-side personalized search tools are installed on a personal computer and build rich user profiles for individual users. They communicate with autonomous search engines when they do web search.

Authors have designed and developed a privacy-preserving personalized search system (UCAIR), which resides on the client side and greatly alleviates the privacy concerns while doing personalized search. (See [58, 25] for two related systems.) A user’s personal information including user queries and clickthrough history resides on the user’s personal computer, and is exploited to better infer the user’ information need and provide more accurate search results. UCAIR is implemented as a web browser plug-in\(^8\). The software architecture of the system is as Figure 8.2. As shown in Figure 8.2, the UCAIR personalized search system has three major components: (1) The implicit user modeling module captures a user’s search context and history information, including the submitted queries and any clicked search results and infers search session boundaries. (2) The query modification module selectively improves the query formulation according to the current user model. (3) The result reranking module immediately reranks any unseen search results whenever the user model is updated. For example, when the user clicks on a search result to view the corresponding web page, UCAIR would assume that the clicked result summary is appealing to the

---

\(^8\)UCAIR is available at: http://sifaka.cs.uiuc.edu/ir/ucair/download.html
8.6 Identifiability of Information Need Descriptions

In the previous sections, we have mostly discussed the protection of privacy from the perspective of directly identifying a user. However, we may also indirectly identify a user based on the description of the user’s information needs $TEXT(N, i)$, especially if the descriptions are aggregated. In this section, we discuss the identifiability of information need descriptions. Since queries are far more likely to help infer the identity of a user than viewed results, our discussion will be focused on queries, but it can be easily generalized to cover viewed results as well.
Given queries $Q = \{q_1, q_2, \ldots, q_n\}$ submitted by a user, some information about the user can be disclosed. A set of queries may give a panorama of the user’s information seeking activities, thus raise a lot of privacy issues. However, different queries $q_i$ have different amounts of personally identifiable information. For example, the query “wikipedia” contains nearly no personally identifiable information while the query “landscapers in Lilburn, Ga” probably can reveal that the user is living in or around Lilburn, Georgia.

The difference of the personally identifiable information also holds for other types of user information in the user profile, which can directly affect the user selection of levels of privacy protection in personalized web search and even the general web search. For example, in Level II privacy protection (Group Identity), a set of queries from one group may reveal a lot of personal information while a set of queries from another group may reveal little personal information. Thus if the user is not satisfied with the Level II privacy protection according to the user profile, he/she should consider adopting the Level III privacy protection (No Identity).

In the following subsections, we propose some research questions about the measurement and identifiability of the personally identifiable information in queries.

### 8.6.1 Identifiability of a Single Query

Intuitively, different queries contain different personally identifiable information. For example, apparently the query “wikipedia” and the query “landscapers in Lilburn, Ga” contain different amounts of personally identifiable information. Thus we need a way to measure the identifiability of a single query. Here is the formal description of the problem.

**Question 8.6.1** Given a query $q$, how can the identifiability of $q$: $I(q)$ be measured?

Currently, some researchers [3, 2] use different metrics to measure the privacy of the data. We think that the method proposed in [2], which uses the entropy in information theory to quantify the privacy of the data, is a sound way to measure the identifiability of a single query.
When we do personalized search, personalized search system will reformulate query. It is possible that the query reformulation process alters the value of identifiability. Here is the question about this *differential identifiability*.

**Question 8.6.2** *Given the original query* \( q \) *and the reformulated query* \( q' \), *how can we measure the difference of identifiability: \( \Delta(q, q') \)?*

In [2], mutual information is proposed to measure the additional information in the perturbed value. This method can also be used to measure the additional personally identifiable information given two versions of queries (the original query \( q \) and reformulated query \( q' \)).

### 8.6.2 Identifiability of a Set of Queries

Search by search, click by click, the identity of web user becomes more easier to discern. For example, a single query “landscapers in Lilburn, Ga” may not identify a unique person. However, several queries including “landscapers in Lilburn, Ga”, “homes sold in shadow lake subdivision gwinnett county georgia” have been used to identify a lady in Lilburn Georgia by New York Times reporters.

**Question 8.6.3** *Given a set of queries* \( Q = \{q_1, q_2, \ldots, q_n\} \), *how can we measure the identifiability of* \( Q \): \( I(Q) \)?

When we do personalized search, similar to the questions in Section 8.6.1, we have a differential privacy.

**Question 8.6.4** *Given the original query set* \( Q \) *and the reformulated query set* \( Q' \), *how can we measure the difference of identifiability: \( \Delta(Q, Q') \)?*
Some researchers try to protect privacy through sending noisy queries to the search engine, e.g., TrackMeNot\textsuperscript{9}. But it is a question whether this method can effectively enhance the user privacy. Here are two questions from opposite perspectives, both of which are directly related with the effectiveness of this method.

**Question 8.6.5** Given a query set $Q$ stored on a search engine, does there exist a boolean function $f : Q \rightarrow B$, such that $B$ is the boolean value, which is $T$ for the original query and $F$ for the noisy query?

**Question 8.6.6** Given the user’s original query set $Q_T$, how can the obfuscation method construct a noisy query set $Q_f^*$ so that it maximizes the probability of failure of the boolean function $f$ at the search engine?

These two research questions are in some sense similar to the research questions about web spam and anti-spam.

### 8.7 Conclusions and Future Work

Personalized search is a promising way to improve the accuracy of web search, and has been attracting much attention recently. Because effective personalized search requires collecting and aggregating user information, it raises serious concern of privacy infringement for many users. In this chapter, we systematically examine the issue of privacy preservation in personalized Web search. We define and analyze four levels of privacy protection. We explore different kinds of software architectures of personalized search and their levels of privacy protection. We also investigate the privacy protection of current search systems.

From the above analysis, we show that client-side personalization has advantages over the existing server-side personalized search services in preserving privacy, and envision possible future

\textsuperscript{9}http://mrl.nyu.edu/~dhowe/trackmenot/
strategies to fully protect user privacy. Applying client-side personalization paradigm, Level I, Level II and Level III privacy protection can be easily achieved using various existing technologies. For example, when we combine UCAIR, a client-side personalized search system with Tor, an anonymous communication system, we can achieve Level III privacy protection. When a search engine is willing to share the index with a trusted third party and an appropriate communication protocol is designed, client-side personalized search system can even be used to achieve Level IV privacy protection.

We further discuss identifiability of the description of user information needs, and define some relevant research questions. Privacy concern is a serious issue that has become a major barrier for deploying serious personalized search applications. There are many research challenges to be solved before we can achieve the ultimate Level IV privacy protection. We believe that in the future there will likely be different levels of privacy protection provided by search engines depending on a user’s preference for the tradeoff between the privacy concern and the improved search service quality.

There are already much work done on private information retrieval by cryptography researchers [32]. Some research methods and results can be applied in privacy-preserving personalized search.
Chapter 9

Conclusion

9.1 Summary

Search accuracy is directly related to how well a search engine understands a user’s information need, thus user modeling and personalizing search service are extremely important for improving search accuracy. Unfortunately, little user modeling has been attempted in the current search engines; indeed, the only information from a user that the current search engines use for ranking documents is the few keywords in the user’s query. The lack of user modeling makes the current generation of search engines inherently non-optimal. For example, different users may use exactly the same query (e.g., “Java”) to search for different information (e.g., the Java island in Indonesia or the Java programming language), but existing search engines return the same results for these users.

In this thesis, we have studied the problem of personalized search systematically, including proposing a general decision-theoretic framework for personalized search, developing specific statistical language models for exploiting both short-term and long-term search history to improve retrieval accuracy [78, 89], developing a framework and algorithms for seeking active feedback from a user, and developing a personalized search agent UCAIR 1.

In the traditional retrieval paradigm, the retrieval problem is cast as matching a single query with documents and ranking documents according to their relevance values. As a result, the whole retrieval process is a simple independent cycle of “query submission” and “result display”, which is inadequate for exploiting user context. The decision-theoretic framework we proposed generalizes

1http://sifaka.cs.uiuc.edu/ir/ucair/download.html

116
this traditional retrieval paradigm and treats retrieval as an iterative decision-making process, in which the system would respond to every user action by choosing some system action to optimize a utility function. Based on Bayesian decision theory, the framework provides a solid basis for modeling users, incorporating user information into retrieval process, and optimizing retrieval performance in interactive retrieval. The framework emphasizes immediate and frequent feedback to bring maximum benefit of context to the user. In general, it serves as a roadmap for studying retrieval models for personalized search.

Using the general decision-theoretic framework described above, we developed several statistical language models that utilize intra-session and inter-session search context to improve retrieval accuracy. To exploit intra-session search context, not only the current query but also any search context information is used to estimate a query language model to better represent a user’s information need. We proposed and evaluated four statistical language models to incorporate query and clickthrough history into the query language model. Experimental results showed that using intra-session information, especially the clickthrough data, can effectively and efficiently improve retrieval performance with no additional effort from the user. To exploit inter-session search context, we proposed mixture models to represent a user’s information need and apply statistical language modeling techniques to discover and exploit relevant context from the search history. The mined search context can then be combined with the original query to improve search accuracy.

When the search results are poor, it may be more effective to first obtain some explicit feedback from individual users. Assuming that a user is willing to provide some feedback information to the system, an important research question is how a retrieval system should actively propose questions to the user so that it can obtain maximum benefits from the feedback on these questions. We studied a specific instance of this problem – active feedback, i.e., how to choose documents for relevance feedback so that the system can learn most from the feedback information. Active feedback can be considered as an application of active learning in information retrieval with two special challenges. First, we do not have any training examples available to guide the retrieval system for actively selecting the documents for feedback; the query is the only information that can be exploited.
Second, it is unclear how we can define an objective function that optimizes ranking performance rather than classification accuracy. We presented a general framework for such an active feedback problem, and developed several practical algorithms as special cases. Empirical evaluation of these algorithms showed that the performance of traditional relevance feedback (presenting the top K documents) is consistently worse than that of presenting documents with more diversity. With a diversity-based selection algorithm, we obtain fewer relevant documents, however, these fewer documents have more learning benefits.

To demonstrate the usefulness of personalized search, we developed a client-side search agent (called UCAIR). Compared with some existing retrieval systems which put personalization on the server side, UCAIR emphasizes client-side personalization, which has several advantages such as alleviating the concern of privacy infringement, utilizing more contextual information only available at the client side, and distributing among clients the computation and storage overhead due to personalization. The UCAIR search agent incorporates proposed models and algorithms, and dynamically reranks search results to reflect the most current knowledge of the user’s information need whenever any new piece of information becomes available.

Clustering of search results has been shown to be advantageous over the simple list presentation of search results. However, in most clustering interfaces, the clusters are not adaptive to a user’s interaction with the clustering results, and the important question “how to optimize the benefit of a clustering interface for a user” has not been well addressed in the previous work. We study how to exploit a user’s clickthrough information that is naturally available when a user is interacting with a clustering interface to adaptively reorganize the clustering results and help a user find the relevant information more quickly. We propose four strategies for adapting clustering results based on user actions, including reranking documents based on a selected cluster, reranking documents based on a viewed document, merging unselected clusters, and promoting “near-miss” documents. The simulation experiments show that the adaptation strategies have different performance for different types of users; in particular, they are effective for “smart users” who can correctly recognize the best clusters, but not effective for “dummy users” who follow system’s ranking of results. Besides
the simulation study, we also conduct a user study on one of the four adaptive clustering strategies (i.e., promoting near-miss documents) to see if an adaptive clustering system using such a strategy can bring users better search utility and/or experience than a static clustering system. The results show that there is generally no significant difference between the two systems from a user’s perspective.

We also systematically examine the issue of privacy preservation in personalized search. We distinguish and define four levels of privacy protection, and analyze various software architectures for personalized search. We show that client-side personalization has advantages over the existing server-side personalized search services in preserving privacy, and envision possible future strategies to fully protect user privacy.

9.2 Future Work

Due to the importance of user modeling in all search applications, technologies for personalized search will be essential for the next-generation search engines. The personalized search research can be further extended in the following directions.

**Modeling Search Context:** It is very important to develop retrieval models to incorporate more useful context information such as user interaction information. By better utilizing user context information, we can build more accurate representation of a user’s information need, thus improving retrieve performance. For example, it is interesting to study user familiarity with topics [60] and user tasks such as user preference of document genre [61] in context-sensitive retrieval. There are two questions related with it. First, how can we detect such context information? If the user does not explicitly specify these context information, we need to find a way to detect context information. Second, how can we model such context information? In our proposed decision-theoretic framework, although we can easily incorporate some user context information such as familiarity into the user model, we still need to address how to quantitatively combine different pieces of context information.
**Progressive Personalization:** Personalization does not always improve the search performance. There are some complaints that sometimes the personalized search result will deteriorate the original results. So it is interesting to study whether we can do personalization in a progressive way, i.e., we will guarantee with a high probability that the personalization will at least not cause the quality of search results to decrease. We will decide at the specific situation whether it will be helpful to do personalization or not. It is interesting to apply statistical decision theory to develop a principled way to gradually incorporate more and more user information as the system gains more confidence in understanding the user’s information need.

**Search Interface:** The user interface of a retrieval system directly impact the user experience, which is especially true for the personalized search system. We should not confuse the user when we apply the personalization technology. Instead, we need to design a friendly user interface so that the user can easily understand and control the search interface. Thus it is extremely important to explore intelligent user interface and other human-computer interaction issues in personalized search systems.

**Personalization in Applications:** Besides the general web search engine which can benefit from the personalized search research, many applications such as digital library, relational database, and online education system can also improve the user experience through the personalization. In some applications, the system can even build a richer user profile so that the personalization can be better utilized.
References


Author’s Biography

Xuehua Shen was born in Yancheng, China, on January 13, 1977. His main research interests are in information retrieval, database, and data mining. In addition, he is also broadly interested in some other areas in computer science, especially machine learning, human computer interaction, and information privacy. He has published papers in conferences of ACM SIGIR, ACM CIKM, ACM SIGKDD, WWW, and IEEE ICDE. He received his B.S. in Computer Science from Nanjing University, China in 1999. He did summer interns at Microsoft Research Redmond and IBM T. J. Watson Research Center. Recently, he served on the program committee of Symposium on Information Interaction in Context (IIiX’2006 and IIiX’2008). He has two patents related with personalized search filed.