THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Search in Personal Spaces

Alex Penev

BE Software Eng. (Hons I), UNSW, 2005

Supervisor: A/Prof Raymond K. Wong

A thesis submitted in partial fulfillment of the requirements of the degree of

Doctor of Philosophy

August 2009

ORIGINALITY STATEMENT

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Signed .....

Date .....

Publications

1. **Penev, A.**, and Wong, R. K. Framework for timely and accurate ads on mobile devices. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management* (CIKM'09), ACM, 10 pages, 2009. To appear.

2. **Penev, A.**, and Wong, R. K. Using metadata for social tag search. In *Web Intelligence and Agent Systems (WIAS)*, IOS Press, 18 pages, 2009. To appear.

3. Gebski, M., **Penev, A.**, and Wong, R. K. Protocol identification of encrypted network streams. In *Complex Data Warehousing and Knowledge Discovery for Advanced Retrieval Development*, T. M. Nguyen (Ed.). IGI Global, ch. 15, pp. 328–341, 2009.

4. **Penev, A.**, and Wong, R. K. TagScore: Approximate similarity using tag synopses. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence* (WI'08), IEEE Computer Society, pp. 98–104, 2008.

5. **Penev, A.**, and Wong, R. K. Finding similar pages in a social tagging repository. In *Proceedings of the 17th International Conference on World Wide Web* (WWW'08), ACM, pp. 1091–1092, 2008.

6. **Penev, A.**, and Wong, R. K. Grouping hyperlinks for improved voice/mobile accessibility. In *Proceedings of the International Cross-disciplinary Conference on Web Accessibility (W4A'08)*, ACM, pp. 50–53, 2008.

7. Gebski, M., **Penev, A.**, and Wong, R. K. Grouping categorical anomalies. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence* (WI'08), IEEE Computer Society, pp. 411–414, 2008.

8. **Penev, A.**, Gebski, M., and Wong, R. K. Topic distillation in desktop search. In *17th International Conference on Database and Expert Systems Applications* (DEXA'06), vol. 4080, Springer, pp. 478–488, 2006.

9. Gebski, M., **Penev, A.**, and Wong, R. K. Classification of hidden network streams. In *Proceedings of the 8th International Conference on Data Warehousing and Knowledge Discovery* (DaWaK'06), vol. 4081, Springer, pp. 332–341, 2006.

10. Gebski, M., **Penev, A.**, and Wong, R. K. Protocol identification of encrypted network traffic. In *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence* (WI'06), IEEE Computer Society, pp. 957–960, 2006.

11. **Penev, A.**, and Wong, R. Shallow NLP techniques for internet search. In *Proceedings of the 29th Australasian Computer Science Conference* (ACSC'06), Australian Computer Society, pp. 167–176, 2006.

Abstract

Technology surrounds us with many daily Search tasks. However, there is a fundamental difference—one of user familiarity and control—that differentiates between search tasks in impersonal and personal search spaces. The World Wide Web itself is largely unknown, unfamiliar and impersonal to a user. In contrast, users regularly search in more 'personal' spaces, such as their own files, their web history, bookmarks, downloads, and so on. These spaces are personal because the user has more knowledge, familiarity and control over their content. A byproduct of these qualities is that search in personal spaces is typically navigational: to navigate through or to recover familiar information. This differs from web search, where very often a user is trying to discover new or unknown information. This important difference in search intent means that there are often few 'correct' results for a query in personal spaces, which is something we must keep in mind when implementing search algorithms.

This thesis leverages structure and metadata to build novel algorithms for improving search in several important personal search spaces: finding a file in a file hierarchy, website navigation on a mobile phone browser, tag-based search in an online bookmarking system, and sponsoring content on mobiles. The proposed methods are highly practical and applicable to current real-life search problems that affect millions of users.

Contents

Chapter 1

Search in personal spaces

1.1 Introduction

Search is probably the most important and visible manifestation of Information Retrieval (IR) research. Web search, in particular, has been responsible for driving much of the growth of the web. Today, five of the top-10 most used websites are search engines, and four of the remaining five are sites that organize data often searched for[1]. Thus it is fair to say that search is the most important activity on the web.

The three major search engines have recently become the largest online advertisers and now fund much of the web. They have also branched out into other online services such as email, chat, video hosting, web hosting, image hosting, blogs, online shopping, news, weather, stock quotes, document authoring, web browsers, web communities, social networking, collaborative bookmarking, online dating, digital libraries, language translation, maps, games, developer tools, mobile software and more. Some of these services are directly profitable while others maintain brand loyalty. Thus, there are many services that owe their existence to the initial success of web search technologies.

Of course, search is much older than the web. The classical term weight and vector space models were some initial breakthroughs in the field [118, 120, 121]. The next big

[1]Google, Yahoo, Microsoft Live, Baidu, Yahoo Japan; Wikipedia, YouTube, Facebook, Blogger [1].

breakthrough came decades later when these traditional methods were complemented with graph theoretic algorithms [69, 99] and scaled to very large datasets [19, 33, 41]. More recently we have witnessed the birth of web2.0, social networks and the Semantic Web, which may collectively represent the next major step in search. There is also an evolution of hardware, with more search applications becoming mobile and distributed.

Search is now everywhere on the web. Yet somewhere along the line, search in personal spaces missed out on the breakthroughs. Somewhere along the line it became easier to search and navigate the web and all of its foreign content than it was to search and navigate our own content. This thesis attempts to bridge some gaps in the inequality.

Many of us use the web daily, but it is a shared resource and largely impersonal: most of it is unfamiliar, unknown and anonymous to us, is out of our control, and rapidly changes without our say. *Personal search spaces* represent the opposite: content that is closer and more intimate to us. We do not need to author or be the sole reader of content for it be personal. Rather, we only need a certain degree of familiarity and control over it. As such, this thesis uses the concept of 'personal space' to refer to a finite collection of content that is under the user's jurisdiction. Personal search spaces are very commonly encountered (although perhaps not immediately noticed) and are related to the psychology concept of 'personal space' in proxemic theory, where spaces are concentric circles around an individual that represent their comfort zones[2].

Examples of personal search spaces include: the user's bookmarks, the user's friends in a social network, the user's shared uploads in tagging system, technical documents in the user's field of expertise[3], the contents of a book being read, the content of a website being viewed, the user's email, messages from a forum the user reads, files on the user's computer, music on their iPod and content on their mobile. In each example the user has an increased level of familiarity or control over the content and, as such, their search query intent is slightly different to a search in an impersonal space such as the web.

[2]The 4 proxemic spaces are intimate (e.g. physical contact), personal (e.g. talking with a friend), social (e.g. routine interactions with humans) and public (everything impersonal or anonymous) [54].

[3]E.g. an API for programmers, the USC for lawyers, the VLDB proceedings for database researchers.

Search intent on the web can be categorized as either informational, navigational or transactional [20]. An informational intent is a query seeking information on a subject, typically asking for sites that match a set of keywords. Such queries may have millions of 'correct' sites, or it may be answered without visiting any of the sites[4].

A navigational intent is a query asking for the URL of a particular site the user wants to reach. Such queries use the engine like a telephone operator, asking to be redirected[5].

A transactional intent is a query that asks for a web-mediated activity, typically a site where a further action can be performed such as shopping, downloading, finding a map or accessing some special database. Navigational queries are answered similarly to informational queries, but search engines now understand and handle various templates directly with a vertical search. This means that they provide non-typical results for seemingly typical queries by using external database[6].

Search engines are able to answer informational and navigational queries using a combination of content analysis and link analysis. They are also able to answer some transactional queries directly by matching templates. Each of the three query types is common on the web [20], but their relative use is likely to be much different in personal spaces. For instance, transactional intent would be very rare in personal spaces because the user knows that the domain is restricted. More importantly, because the user is expected to be somewhat familiar with the search space, more queries will be of a navigational nature: an intent to locate a file, to pinpoint a bookmark, to unearth an email, to flip to the right section in a book, to find a hyperlink, to find a forum post, and so on. The nature of navigational queries is that the user is looking for a specific result—not just any content-match will do. For these reasons it is important to look at personal spaces in closer detail and leverage any extra information that can help improve known-item search and navigational intent. This is not the same as simply adding a new layer of personalization, which

[4]E.g. the QA-style query "moon landing date" reveals the Apollo 11 date in the search snippets.

[5]E.g. the query 'dblp' finds the DBLP homepage among the millions of sites containing that keyword.

[6]E.g. Google's search engine 'understands' queries such as "*yhoo*" and shows a share price chart, or "*time tokyo*" (shows a timezone), "*70kg in lb*" (does a calculation), "*price nikon d300*" (lists camera prices), "*usain news*" (finds headlines on a current topic) and "*hotel sf*" (plots hotels on a map).

is possible for any system. Rather, this thesis is interested in making changes to the core method that is used to search or navigate the personal search space.

The scope of search and the scope of search in personal spaces are both overwhelmingly large, therefore this thesis focuses on a few selected domains. Each chapter presents a novel and practical method for a different personal space and tests the method's effectiveness for search or navigational tasks. The chapters are:

- Ch. 1: "*Search in personal spaces*" provides an introduction to personal spaces and outlines the scope and related work of the thesis.
- Ch. 2: "*Searching in a hierarchy*" starts with an older and ubiquitous personal space of searching for files in a hierarchy. It presents a method that exploits the hierarchical structure to improve known-item search and conducts experiments on real and simulated data.
- Ch. 3: "*Web navigation on constrained devices*" advances to the modern personal space of website content viewed on a mobile browser. It presents a method for on-demand link clustering to aid navigation, organization and advertising on high-level index pages, and conducts experiments using real data.
- Ch. 4: "*Searching in web2.0 tagging systems*" visits the more-modern personal space of online bookmarking to address a question that was encountered in Ch. 3. It presents a method for enriching tags to improve search accuracy in a tagging system and conducts experiments on real data.
- Ch. 5: "*Sponsoring mobile content*" returns to mobiles and investigates a new, emerging personal space of mobile content. It presents a method for offline client-side ad-serving and conducts experiments on real and simulated data.
- Ch. 6: "*Conclusions*" summarizes the thesis, highlights the contributions and potential future work, and gives a few closing remarks.

The remainder of this chapter describes the scopes and related works for the personal search spaces investigated in Chapters 2–5.

1.2 Searching in a hierarchy

The thesis begins with a classical personal search space: files and folders on a computer. It is well known that users generally organize files using extensive folder structures. In most cases a user searching for a file within such a folder structure will also have some degree of familiarity with the types of files that exist and therefore many of their search queries will have a navigational intent, such as wanting to find a particular file. Chapter 2 is dedicated to this personal space, where a novel method is proposed for exploiting the hierarchical structure to improve known-item search.

Apart from desktop and laptop computers, it is very common to find other types of content arranged in hierarchies. For example, technical documentation such as the DBLP bibliography or the Java API exist as folder structures. A book such as this thesis may look like a single document, but it also has a hierarchical arrangement of chapters, sections and subsections that organize its content.

Intuitively, a hierarchical structure should reveal *some* meaning that is exterior to the semantic meaning of the individual content nodes and this meaning should be possible to somehow capture. As an example, imagine a filesystem that contains a mystery file with no name and unknown format. This file may reveal some information about itself depending on where it is placed, e.g. if it is in a folder "travel/netherlands/", then perhaps it relates to a trip to the Netherlands. Even if the folder path is semantically unclear, such as "trav/ned09/", then it may still be possible to guess that the mystery file is about travel and the Netherlands based on the contents of other files nearby. In other words, the arrangement in a hierarchy can encode contextual information about the nodes that is external to their content.

Based on the premise that users organize their files in a meaningful way and generally place related files closer together, Penev & Wong [100] drew an analogy that folders play the role of 'hubs' and files of 'authorities', with a similar interpretation as HITS [69]. In line with Kleinberg's definitions, an authority would be a content node that contains query-relevant information and a hub would be a structural node that references such

authorities. In the case of HITS, these are both webpages. In a file system, authorities were suggested to be the individual files and hubs were suggested to be folders. This idea generalizes to other hierarchies, as well. For example, in a book the authorities may be paragraphs of text and hubs may be the chapters and sections.

The reinforcing relationship described by Kleinberg still holds for the desktop: good folders should contain good files and good files should be in good folders. The containment does not need to be direct, e.g. a folder can be a good hub even if has no files in it—it may contain only hubs, with the relevant authorities a few levels lower.

To relate hubs to authorities, a re-interpretation of what constituted a hyperlink—the web author's recommendation to another webpage—needed to be defined. Using the concept of "how many clicks apart" [82], the most suitable interpretation was node distance [100]. The atomic action of a mouse click on a web link is similar to pushing and popping in and out of folders in a hierarchy, thus nodes in the hierarchy could 'recommend' their content to each other based on distance. This is similar to how webpages recommend their relevance using inlinks/outlinks in HITS and PageRank.

In Chapter 2, a flexible and efficient algorithm called *Parameterized Filesystem HITS* (or PFH) is formulated to use the contextual information encoded within the hierarchy to improve known-item search. The chapter talks almost exclusively about files and folders, although the concepts and algorithms should generalize to other hierarchies because domain-specific information is avoided. Apart from investigating search accuracy, the experiments also observe changes in accuracy when the reliance between structure and content is varied. The work bridges three areas of IR research: file organization, intelligent Desktop Search software tools, and web connectivity analysis.

File organization

Window managers use the conventional desktop metaphor for organizing a person's files. Therefore it is expected of users to organize their personal document space using folders and subfolders. There is no particular reason that this should be the optimal way to

organize files, but it is the de facto standard among window managers. Yet, despite being such a common activity, multiple works have noted the lack of comprehensive research on how users organize and search file hierarchies [58, 113, 136]. Older studies have generally conducted small-scale interviews with users from a particular organization, but such studies have become oudated due to the many changes in modern window managers and user's changing needs.

Nevertheless, there have been several works that support the hypotheses that latent judgment and meaning is encoded in a folder hierarchy. Jones et al. [67] looked at the role of folders in organizing a user's task-related files and concluded that folders "*represent an emergent understanding of the associated information items and their various relationships*". Henderson [58] found that her subjects created folders to group files predominantly by their genre, task and topic. In Boardman & Sasse [14], subjects used "*extensive folder structures*" and filed items into folders immediately as they were created. Folders were used to group related files based on short-term "project" tasks, "document class" and long term "role". Additionally, Ravasio et al. [113] reported that "*a fair amount of effort was invested ... in creating elaborate file system structures and in labelling them adequately so as to support the unveiling of the content's meaning and relation*". They also noted that new subfolders were created "*where the user felt that it was important to keep an overview*" and for "*documents on the same subject.*"

Therefore there is evidence in the literature to support the intuition that a hierarchy has meaning and plays an important role in how humans organize files. Our goal for this personal search space is to harness this meaning for improving known-item search. Searching for files, particularly for personal files on a computer, is referred to as *Desktop Search* (DS).

Intelligent DS tools

Desktop Search tools are applications that help a user search through their own files. Being a personal space, queries on the desktop typically have a navigational intent. Due

to the known-item search nature of DS queries, maximizing Mean Reciprocal Rank (MRR) is preferable to maximizing other traditional metrics such as Recall, Precision, F-score or Mean Average Precision [131]. A well-known study into DS was by Dumais et al. [37], who observed more than 200 Microsoft employees and how they searched their personal files; some of their observations are referenced in context within the thesis. However, the works closest to PFH are two DS tools, Beagle++ and Connections.

Beagle++ [129] used a schema to introduce semantic relationships between files. Its main goal was to use these semantic relationships to improve Recall by retrieving files that a content-only tool would miss. Schematic relationships were assigned weights and evaluated using ObjectRank [8]. The strength of the approach is in specifying known file type relationships, such as linking an email attachment to its originating email. But the approach carries a limitation of being unable to help many searches, i.e. not all queries are for email attachments and some queries do not benefit from schemas. In such cases, the MRR of Beagle++ may be lower than a generic content-only search because it introduces noise to the results.

The PFH algorithm differs to Beagle++ in two ways. Firstly, PFH does not affect recall and therefore does not add noise to results. Instead, PFH inherits the top-k search results of an existing retrieval engine and merely rearranges them based on analysis of structure. Since it rearranges only the top results, all of PFH's results are already considered to be query-relevant (at least according to the underlying engine). Any existing tool can be used as the underlying retrieval engine, as long as it reports its content analysis scores. PFH uses Lucene [3] as the underlying engine, as well as an experimental baseline.

Secondly, PFH is not a fully-fledged DS tool like Beagle++ but a lightweight rearrangement algorithm. It is general and stateless: it has no persistent store, retrieves whatever files the underlying engine can retrieve, and knows nothing about the user. As such, PFH is versatile because it can be used on top of many existing DS tools, can have various layers added on top (e.g. personalization), and can be used in other domains such as books and technical documents.

Connections [126] was a DS tool that traced file system calls to maintain a file-file relationship graph. Standard content-only search results were extended by adding related files from this graph. Two files were considered related if they used each other for I/O or were opened within the same window of time. This represents a temporal context, in constrast to PFH's location context. Not surprisingly, adding extra files to the search results greatly improved Recall measurements for Connections. However, precision was only slightly improved. The strength of Connections' approach was in deriving relationships from temporal context. One disadvantage of doing this is time. Like TaskTracer [36], another monitoring system, Connections needed to monitor user activity for some time (half a year in their experiments) in order to build its relationship graph. Initially, results would be no different to a content-only tool. Given that Connections has more overhead than a regular content-only tool, users are likely to use it only if they are satisfied with a promise that results may improve with time.

PFH has several advantages over Connections and TaskTracer: it works immediately and as intended since there is no monitoring; it can be used where monitoring is impractical or impossible; and it is both lightweight and flexible, making it usable on small devices (e.g. mobiles) and for other hierarchical corpora where monitoring is inapplicable (e.g. books). PFH does not need to replace the task-based tools, but may complement them. For instance, it can be used in preference to Connections and TaskTracer during their initial monitoring phases, and it can be used thereafter for searches that do not produce high confidence scores under their probabilistic models.

A trial experiment conducted by the authors of [126] used folders to derive contextual relationships, extending regular content-only matches with other files from the corresponding folders of the top results. They concluded that folders cannot be used for context because their experiment performed poorly. However, the likely reason for this was that they used an overly-optimistic approach: when the query specified a file extension, all files in a folder with that extension were assigned the combined weight of any content matches, and when no extension was given, all files within a folder were assigned

the score of the best-scoring file in the folder. This is a recall-centric scoring approach that demands flawless organizational effort on behalf of the user because it adds many new files to the search results and assigns them very high scores even if they do not match the query. So it is not surprising that their experiment performed poorly. Such a recall-centric approach is similar to running PFH with α=0, i.e. relying purely on structure as this is when PFH would give the same score to files in the same folder. The experiments in Chapter 2 show that such a low α is indeed a poor choice.

PFH differs to [126]'s folder experiment in three ways. First, PFH does not add noise to the results because it does not affect recall. Second, PFH maintains intra-folder relative ordering, whereas [126]'s experiment gave files equal scores and blurred the line between genuinely relevant and irrelevant results. Finally, rather than only "in the current folder", PFH's notion of location context is files and folders "in the neighborhood", for which PFH uses a distance function to determine proximity.

Connectivity analysis

The above notion of distance is related to web connectivity analysis, where the hyperlink graph of the web is analyzed by algorithms such as HITS to determine hubs, authorities, communities and topical hotspots. PFH uses a similar methodology as HITS.

Miller et al. [91] suggested using arbitrary path length to avoid counter-intuitive results from HITS in weakly connected graphs. Accordingly, PFH treats the hierarchy as a strongly connected graph and thus considers paths of any length. This fits well with the concept of 'neighborhoods' as it allows for an influence of nodes that are nearby but not directly connected. Other considerations in how PFH is formulated relate to the scoring model. While PFH uses Lucene and its VSM implementation, Li et al. [79] argued that there is little significant difference between several popular similarity measures for this type of search task. Dean & Henzinger [34] and Borodin et al. [17] suggested averaging hub weights by outlink cardinality, and PFH performs a similar calculation. XRank [52] performed a ranked query search over XML documents, which also has a hierarchical

structure. It used a connectivity analysis algorithm to make use of the XML structure, although it had the luxury of explicit x-reference XML links that are domain-specific and do not apply to the desktop. Perhaps because of the existence of these links, XRank based its analysis on PageRank instead of HITS. However, Section 2.2.2 argues why HITS is more suitable for file hierarchies.

Within the connectivity analysis literature, the closest works to PFH are the connectivity analysis algorithms themselves. Because of their strong relevance, more details are given in Section 2.2.

1.3 Web navigation on constrained devices

Having addressed the problem of known-item search in a traditional hierarchical environment, the thesis advances to a more modern personal space: viewing a page on a mobile phone browser. Chapter 3 is dedicated to this personal space and presents an approach for on-demand link clustering of the page's hyperlinks. The goal of the approach is to algorithmically determine which links belong with which other links. Similar to searching in a file hierarchy, we can exploit the webpage structure to assist with the clustering and even rearrange parts of the page to improve navigation.

The elicitation of the link clusters is useful for three practical problems related to mobile web browsing:

- improved accessibility and navigation for vision-impaired users.
- assisting web authors with creating mobile portal pages or reorganizing their existing page layout.
- section-relevant advertising for mobile news sites.

The presented approach involves labeling the links with features, exploit the page structure using a weight propagation scheme to make adjacent links share features, and then to cluster the links based on the features. The propagation step leverages the *Document Object Model* (DOM) structure to distribute weight between neighbor links as a sort

of rubber-band effect that keeps links together if they cannot be clustered elsewhere. Although the final procedure regarding what to do with the link groups once they have been identified will differ between the above three applications, each application shares the same initial link clustering step.

The remainder of this section describes why the above applications are important and useful, and outlines the scope and related work for search and navigation in this personal space.

Accessibility for vision-impaired users

The first application, accessibility for the vision-impaired, relates to how blind users navigate web pages. Sighted users are able to visually scan webpages in parallel and instantly ignore large portions of content, but vision-impaired users typically use screen reader software that serially reads page content as audio. This is a slow navigation process and is made unnecessarily slower if the user is looking for something in particular, such as a hyperlink. The use of the link clustering step is to rearrange the links on the webpage by presenting a small set of clusters at the top of the page. Extracting and rearranging the links can enable visually-impaired users to skip portions of the page content and locate a desired link quicker than with a linear scan.

The closest works to this application are HearSay and SADIe. HearSay [18, 112] partitioned the webpage DOM into labeled segments of related HTML elements for which the labels served as shortcuts. Visually-impaired users could use these shortcut labels to jump to elsewhere on the page. Although useful for browsing of familiar pages, one drawback is that an unfamiliar page may lead to disorientation and blind leaps—the user will not know where they are jumping to. Apart from different goals, HearSay differs to the presented approach in that it does not rearrange content. The presented approach can rearrange the page by extracting the links and placing the clusters where they can be quickly read by a screen reader. Furthermore, HearSay used manually crafted ontologies for labeling whereas the presented approach is domain-independent and automatic.

SADIe [57, 80] used an ontology to determine if elements on a page were considered 'removable' or 'important'. These two attributes could be used to improve navigation for visually-impaired users by removing visual fluff and extraneous page content. Elements determined to be 'important' could also be shifted to the top of the page and be scanned sooner. SADIe's goal was similar to that of the presented approach, but one difference is that its ontologies were manually crafted and domain-specific. Another difference is that, while SADIe could rearrange parts of a page, it did not partition or group them. Yet it is this grouping that allows visually-impaired users to skip unwanted links—with SADIe the links would still be read in serial.

Link organization

The second application, link organization and portals, refers to modifications that web authors can make to their pages after seeing the output of the link clusters. Chapter 3 groups links to determine which links belong with which other links. This can suggest to a web author if some links in their layout should perhaps be moved to be closer to their related links. It can also help web authors create portal pages by eliciting the key topics and segments of their webpage.

A related work in this case is LinkSelector [39], which used server access logs and the hyperlink connections between pages in a university domain to select a small subset of pages from the pool of possible pages that could serve as the university portal, i.e. the key navigation links on the home page. By choosing frequently accessed pages, a well-chosen set of navigation links can minimize the average path length that users will take to click on links and reach their destination. To determine which links provided best coverage and domain representation, LinkSelector repeatedly merged similar links until only a small number remained. Assuming that access logs are available, Perkowitz & Etzioni [106], Mobasher et al. [92] and Anderson et al. [6] have also proposed mining clickthroughs to guide improvements in the navigation and organization of pages.

The presented approach is fundamentally different to LinkSelector in several ways,

making the two difficult to compare. This is because LinkSelector outputs *some* of the links from a *domain*, whereas the presented approach outputs *groupings* of *all* links from one *page*. LinkSelector is also a server-side process, whereas the presented approach is intended to be run on the client-side, even on limited devices such as mobiles. Nevertheless, LinkSelector is still relevant as it is one of few works to directly manipulate links in regards to portals.

Advertising on news pages

The third application, sectioned advertising for mobile news portals, refers to a particular manner of advertising: placing topically relevant ads in appropriate positions in highly dynamic pages such as online news sites. Mobile news portals—a typical example of which is shown in Fig 3.1—are stripped down to the bare minimum of content. Their layout is a vertical column of hyperlinks, some being headlines and some being navigational links. This layout style allows several assumptions to be made that simplify the calculations and make the approach work on the client-side, with or without an HTML representation. The large number of topics discussed on a news page, in combination with their highly dynamic nature, make them poor candidates for standard contextual advertising. Additionally, different parts of the page discuss different topics, therefore it is better to show ads in specific positions on the page and not in others.

Online aggregators that cluster news articles from multiple sources, such as Google News[7], compare the full articles. This luxury is unaffordable on a mobile client, where we have no choice but to use only the headlines. Due to the extreme sparseness of keywords in the headlines, a typical clustering algorithm will struggle to create any meaningful clusters. In this case, it is the propagation step that helps maintain existing layout cohesion (due to the rubber-banding) when meaningful clusters cannot be made. As such, the cluster step may result in very few separations of links away from their neighbors because few

[7] http://news.google.com

links share any features in common. This makes the cluster step sometimes behave more like segmentation step that determines where one section ends and the next begins.

Webpage segmentation is a popular research area for mobile browsing because sites tend to have large dimensions while mobile screens are small. As such, there is an interest in being able to determine which segments of a complex page are genuinely important (and which are not), or ways of thumbnailing parts of a page so that the whole display can fit on a mobile screen. Yin et al. [144] extracted important parts of webpages to improve visibility and remove clutter, but their approach may not be useful for mobile news pages because they are already heavily stripped down and every element is important. Other works on segmentation, such as [28, 53, 74, 137], have focused on thumbnailing. Thumbnailing may not be useful for news portals because they are already non-complex, mobile-friendly pages with only a single column of items.

In terms of webpage segmentation, the closest approach in the literature is that of Chakrabarti et al. [25], who used a learning framework to combine DOM elements to determine segments in a page. A learning model can allow finding which HTML elements signal a section break. Although useful for template-based sites, their approach needs to be re-run whenever small changes are made to the template. Additionally, their approach is unable to rearrange links, which can be useful or even required if section markers are not present (e.g. if headlines are presented as a feed or a stream).

In general, works in web segmentation use the DOM and look for visual or structural cues to determine partitions. Chapter 3 focuses only on news portals and can therefore make some assumptions about the nature of the content, such as it being a single flat column of links. If we make such an assumption, then effectively there is no depth among the links and the DOM is no longer needed—the links can be assumed to be in a linear ordered sequence. Thus, apart from being able to reposition links, one advantage of the presented approach is that it can segment both template-based sites and non-HTML plain text presentations.

Once the sections are determined, each section can be classified into one or more

topics. The dominant topic(s) of the section may then be used to choose an ad of the same topic, a step that requires a classifier. More specifically, a classifier for very short text segments such as news headlines.

Text classification is a well-studied area. Algorithms that have been used for text classification include k-NN [85, 139], SVM [65], neural nets [93], Naive Bayes [7, 45, 70], entropy [107] and decision trees [110] (refer to Yang & Liu [141] for a review and comparison). Despite the first few methods being slightly more accurate [140], they are unsuitable because they are too expensive to run on a mobile. Given the extreme brevity of the input, namely a small set of anchor texts representing one or more headlines in a section, then the most time and space efficient method is Naive Bayes. The feature vectors for the evaluation step can be implemented efficiently as a hashmap or trie.

Apart from the hardware limitations, a second consideration for classifying news headlines is the topics (or classes) that will be used. The classes should somehow represent a marriage of news and advertising topics, such that a match between one and the other can occur. Previous work on classifying news articles have used standard pre-existing newswire categories, such as those of Dow Jones [85] or Reuters [7, 65, 70, 78, 140]. However, these categories are domain-specific to news and are suboptimal for matching ads. For example, the DJN codes[8] focus heavily on finance topics with very few markers for non-finance topics, whereas the 90 topics in the Reuters corpus [78] are fewer in number, likewise news-oriented and even outdated compared to today's Internet-age subject matter.

More recent works on news classification have used too few classes (e.g. only 3 in [149], 8 in [134] and 13 in [35]), which may result in less effective advertising due to very broad classes. Ideally, what we need is a classification domain that suits both news and ad topics and is also fine-grain. Constructing such a dual-domain classification may be difficult, and using Wikipedia may be an option. Because Wikipedia is a universal knowledge base, it is by definition an umbrella domain that subsumes all news and advertising topics.

[8] http://online.wsj.com/article/BT-CO-20090609-717517.html

Some work on Wikipedia categorization has already been done, e.g. Phan et al. [107] recently published feature vectors for 200 salient topics. The topics are unnamed because they were determined backwards via topic analysis, by distilling the feature vectors for 'hidden' topics rather than starting out with pre-defined topics and learning their features from a labeling training set. Nevertheless, the topic labels are not needed because they are not seen or used by users.

Consequent to the above considerations, the presented approach uses an NB classifier with 200 Wikipedia classes and their feature vectors. The evaluation step is space- and time- efficient, making it a suitable classifier for implementation on a mobile device. To our knowledge, the presented approach is unique in that it classifies only headlines (as opposed to full articles), does so using unnamed topics, and does so in an aggregated manner.

Headline aggregation is important for avoiding misclassifications that arise from the extreme sparseness of the keywords in the anchor text. Consider two headlines, one mentioning "java" and the other mentioning "python". Individually these headlines may be classified into groups representing "indonesia" and "reptiles". However, if taken together, the aggregate headline "java python" will be classified as "programming" instead. Enabling this kind of aggregation is precisely the purpose of the link clustering step that is initially run to determine the sections and link groupings. If the headlines in a group really do talk about different topics, then the dominant topic of the group should still be relevant to the group as a whole. The dominant topic will usually be the most commonly shared topic among multiple links in the group (such as "programming" being a common topic for "java" and "python") or will be a strongly confident topic for one of the individual links (such as a topic "music" for a headline about "mozart").

Clustering with keyword features

Each of the three applications described above involves the same initial link clustering step. Clustering in general is an important topic in data mining because it is useful to

group items together to identify patterns or relationships in data. Clustering also helps organize disorganized data. Using a given measure of distance, clustering algorithms aim to minimize the distance between items from the same group and maximize the distance between items from different groups. Algorithms are usually run on numerical or spatial data, with common distance metrics being functions such as Euclidean distance, Manhattan distance or statistical correlations such as Pearson, Spearman and Kendall.

In the domain of webpages, the items are textual instead of numerical. To define a distance between two texts (or portions of text), documents are typically considered as n-dimensional vectors where each word is a dimension. Representing text in such an algebraic form allows finding the prominent feature vectors that ultimately determine the clusters. Some algorithms that have been used for text clustering include the popular k-means [63] and SVD [148], as well as suffix trees [145, 146] and NMF [76].

Traditional clustering occasionally suffers from being unable to assign a short and meaningful label to a cluster. In other words, the clusters are created and the items are known to be related, yet it is non-trivial to assign a single label to each cluster that is short, descriptive and all-encompassing. One can, of course, label the clusters using the feature dimensions that most strongly define the cluster, but this label will not always be succinct or readable. As such, the presented approach creates the clusters but does not label them. Labeling is unnecessary for two of the three applications and is out of scope of the chapter. However, if labeling is required then an algorithm different to k-means should be used. For example, works in the Lingo project [97] and Zeng et al. [147] have specifically focused on first finding appropriate labels and only forming clusters that are known to be possible to label well. Unfortunately, these approaches increase processing complexity and may be unsuitable for use on the client-side of a mobile device.

Real-time labeled clustering also has a presence on the web. Clustering of search engine results has been previously discussed by Kummamuru et al. [73], Zamir & Etzioni [145] and Zeng et al. [147]. It has also been provided by commercial search engines such

as Vivisimo, Clusty, and more recently by Google's 'Wonder Wheel' visualizer. Clusty[9] is perhaps the best known of these examples; it is a metasearch engine that relays its input query to other engines, collates their results and clusters the retrieved titles and snippets. Clusty is related to Chapter 3 in that it performs live clustering, but direct comparisons are not possible: apart from different domains, different input and different output, other differences between Clusty and Chapter 3 include the lack of an accuracy yardstick (i.e. no user search query) and workload restriction (i.e. server vs. mobile client).

Another related work from the domain of search is XRank [52], which performed a query search over semi-structured XML documents. XRank worked at the granularity level of XML tag elements and returned nested elements rather than whole documents. It defined a distance metric for keyword proximity in 2-dimensions (likewise, the presented approach defines a distance metric for link proximity in 2-dimensions) and to exploit XML structure it used a modified PageRank [99] (likewise, the presented approach defines a hierarchical propagation using the DOM, albeit done in an efficient one-pass manner). Apart from these two related ideas, the works are otherwise quite different.

The work on mobile web browsing in Chapter 3 also combines several ideas from the other chapters: structural propagation from Chapter 2, tag labels from Chapter 4 and advertising from Chapter 5.

1.4 Searching in web2.0 tagging systems

In the previous personal space of web navigation on mobiles, the proposed solution required links to be labeled with keywords to assist in their accurate clustering. An obvious question remained about how to best determine a good set of keyword labels for a hyperlink or for a webpage. This question is directly related to web2.0 tagging systems, wherein communities of online users collectively annotate various web objects.

Such systems have recently become very popular and now provide large amounts of metadata for many web objects, including webpages. To investigate how to obtain a good

[9] http://www.clusty.com

set of labels for a webpage, the thesis temporarily leaves the mobile domain and visits another even more-modern personal search space: online tagging systems.

Chapter 4 is dedicated to exploring search and navigation in Delicious[10], one of the largest such systems. In this system users save, retrieve and discover web bookmarks by using keyword tags. In particular, the chapter proposes a novel scoring function, TagScore, that enriches the keywords to make them more expressive and more effective as labels. The properties, behavior and potential uses of TagScore are described, as well as experiments on real data to show how the enrichments improve search and navigation-based tasks.

The web2.0 movement has introduced many online systems based on the 'social tagging' model. In this model a system keeps track of community-shared web objects that are uploaded or created by interested users who annotate them with *tags*. Tags allow users to later recover or discover objects via a simple keyword search. The increasing popularity of these systems in the past few years has prompted considerable interest from the research community.

Brooks & Montanez [23] argued that any extensions to tagging systems should retain the ease of use of the tagging model such that consumers and creators of content—both novices and experts—must be able to understand and apply the tagging concept with minimal effort. This requirement for simplicity is one reason why TagScore does not aim to change the tags externally, but instead aims to rate them with a score to influence how they are used internally. The idea of tag scoring is not specific to individual systems or only to search, and can be used in other similar web2.0 systems or for other tag-based tasks beyond search.

Tagging is a fairly recent research area and there have been a number of works concerned with improving certain functions in certain online tagging systems. These works and their relation to TagScore are discussed below, divided into three sub-areas of research: search, suggestion and behavioral patterns.

[10] http://www.delicious.com

Tag search

Search refers to internal algorithms that help users discover new objects or recover familiar objects. In some systems, users can upload their own content or keep a list of favorites, and search through these lists. However, since these lists tends to be quite small, it is not vital for the search results to be ranked by relevance. Instead, we are interested in improving the accuracy of the algorithms used to navigate or browse the repository, as well as the algorithms that the system uses internally to match or recommend objects. TagScore is naturally suited for these purposes because its tag ratings can be used for many search-based tasks.

Michlmayr and Cayzer [88] looked at tag search with the intention of making it personalized. They created a profile based on a user's tag history to describe their interests, while also decaying the importance of unused tags with time to reveal shifts in interest. They proposed that profiles should be used for browsing, such as favoring search results that overlap part of the profile's tag set. TagScore's ratings and confidence scores can be directly used for such purposes as well. For instance, tags with a higher weight can contribute more strongly to the user profile, and bookmarks whose tag set have a high confidence score can signal a stronger user interest in their content.

Marchetti et al. [81] addressed word sense disambiguation, one of the criticisms of tagging, by building a semantic tagging system that was able to disambiguate polysemy using WordNet and Wikipedia. For tags with multiple meanings, users were allowed to select the particular meaning of the tag when they were selecting it. This would improve precision at the expense of some fluidity in how users use the system and how easily and quickly they are able to tag items. Such a system can complement TagScore's scoring method by boosting the value of tags for which the user disambiguates a word because the user putting effort into disambiguating a particular tag should indicate a higher degree of confidence for it.

Tag suggestion

Suggestion refers to support provided by the system to aid users in choosing tags. TagScore is naturally suited for this task because it assigns ratings to tags. A low-rated tag should be a poor suggestion (i.e. not so relevant) while a high-rated tag should be a good suggestion. In some cases the top rated tag may not be the optimal suggestion since it may be obvious to the user, and the suggestion slot could thus be 'wasted' on something too obvious. Hence, it may be better to suggest a mix of mid- and high-scoring tags. TagScore has an advantage over existing frequency-based suggestion methods in that it can be used for all repository items regardless of their popularity or stability of frequency counts.

Xu et al. [138] looked into a suggestion algorithm for Yahoo's MyWeb2.0, deciding to suggest popular and cooccurrent tags while minimizing overlap between the offered concepts. The idea was to make a small number of suggestions that had larger coverage. To avoid suggesting poor tags they introduced 'user reputation' scores. Reputation implies an appropriate use of tags, which requires a notion of what is an appropriate tag. Their proposal for measuring an appropriate tag was to sum the existing reputation scores. A reputation sum is similar to a simple frequency count[11], but should be slightly more accurate because a less-reputable user counts as less than a reputable user.

However, Chapter 4 suggests that this by itself may be insufficient because the majority of objects in any large-scale tagging repository lie in the long tail of the distribution: they will not be popular and their statistics will be very coarse, inaccurate or unstable. In a common scenario, only 1–2 users may have tagged the object and all of its tags may receive the same reputation value. This would make a reputation system unable to differentiate which are appropriate tags and which are not. The low frequency counts make it difficult to use unpopular objects for readjusting reputation scores because they can feed misleading and unstable values back into the scoring loop. As such, reputation methods may only be suitable for popular items.

[11] E.g. the frequency count of 24 users choosing a tag is 24 and the reputation sum of 24 highly-reputable users choosing a tag is $\approx$24.

A few other works have looked into reputation scores [71, 95] for the purpose of combatting spam, deciding that user reliability should be based on how often they agreed with the majority and how early they identified interesting objects. TagScore may complement such methods because a user's average TagScore confidence can be considered as part of their reputation. One advantage of using TagScore for this is that the scoring process can be user-independent (because TagScore itself is page-independent and user-independent) and thus updatable in real-time. Furthermore, using TagScore as a measure of reputation does not need to be done for each bookmark; rather, users may be screened at random, or at particular thresholds, or whenever the server has the necessary data available to validate the user's tags.

Chirita et al.'s [30] approach to tag suggestion was to generate words aligned to the user's personal interests. This was done via keyword extraction, taking the top terms from the viewed webpage, using them as a Desktop Search query to find relevant files on the computer, and then extracting the top terms from the retrieved files. In some cases this may not be effective, such as if no files are retrieved or if the computer does not belong to the user. In other situations the technique may be unsuitable, such as if there is no search application installed on the computer or if performance is important. In these cases TagScore can be used as a fallback method of making suggestions. It should be noted that personalized tags may be useful to the individual but not the community, and therefore systems that support reputation scores may require special treatment for personal tags. At present, the quality of Delicious tags is reasonably good, with about 7% of tags being irrelevant and 5% being too subjective [60]. In other words, at least 88% of tags are both relevant and objective.

Brooks & Montanez [23] examined blogs and reported that keyword extraction of the content was a stronger descriptor than the author's chosen blog tags. This conclusion was reached by comparing the clustering quality of blogs generated by two clustering methods. Not very surprisingly, tags produced less-focused clusters than TFIDF extraction. However, the authors reported that extracted tags are not necessarily what users will

search with and should not replace author tags. We encounter a similar case in Section 4.3.1.1 where titles are measured to be stronger content descriptors than tags, but we make a similar recommendation that they should be used for tag suggestion. Fortunately, TagScore can also rate titles and meta-keywords, so it can help decide whether the title or meta-keyword can also make good or bad tag suggestions.

There have also been a number of efforts toward visualization of tags, including Michlmayr and Cayzer [88], the online services *Revealicious* and *Extispicious*, and studies on tagcloud variants [55, 68, 117]. Visualizations can be used for suggestion by reminding users of their tag choices. If the tags were also rated with TagScore then users could display the clouds in different ways, e.g. larger words representing stronger suggestions.

Tag patterns

Marlow et al. [84] elicited the architecture of tagging systems and their dimensions, such as who tags what, how tags are aggregated and how the system supports tag choice. Both [84] and Ames & Naaman [5] discussed user incentives in tagging systems, shedding some light on why such systems are popular and what users gain from contributing to the repository.

Several works have investigated trends or behavior in how tagging systems are used. One early work, Golder & Huberman [43], looked at the dynamics of Delicious and found that a page's tags tend to stabilize over time as more users bookmark it. They cited tag imitation as the reason, arguing that initial tags influence future tags. A scoring function such as TagScore should therefore be useful by guiding unstable tags into a better equilibrium via better tag suggestion.

Heymann & Garcia-Molina [59] and Mika [89] considered building a tag taxonomy. This can help determine tag relationships and improve search, tag suggestion and browsing, each of which is susceptible to benefits from TagScore. Instead of a taxonomy, Schmitz [125] described a faceted ontology for Flickr[12]. Facets are one way of organizing tags into

[12] A faceted classification for Delicious was available circa 2005–06 at http://fac.etio.us.

a structure, and another way is clustering. Begelman et al. [11] suggested clustering tags by cooccurrence to group related tags. Some tagging systems do this, e.g. Flickr. TagScore can help with tag clustering by making the cooccurrence metric favor tags that not only cooccur but are also both rated highly.

Clustering the objects rather than their tags can also be useful for browsing as it groups related objects together. Ramage et al. [111] experimented with clustering Delicious data using a combination of tags, page content and anchor text. Unfortunately, it is unclear how many clusters should be made or how they will be labeled: too few clusters would be too vague and need broad labels while too many would be too specific, and neither of these is conducive to effective browsing. Furthermore, the problem of automated labeling of partitioned clusters for pure browsing tasks is known to be non-trivial because labels need to be short, accurate and readable. To our knowledge this has only been successfully applied to small sets of documents such as the top few hundred results returned by a search engine query, and not to collections anywhere nearly as large as Delicious. In contrast, Brooks & Montanez [23] opted for agglomerative clustering and built a tree of objects. Such a structure is perhaps easier to navigate because it already provides some explicit navigational links. Labels for the tree nodes do not need to be high quality to be useful because the explicit hierarchy already provides some context (such as sub-topics and super-topics) that partitioned clusters lack. In particular, [23] used pairwise cosine similarity for a partition experiment and expressed concern at its lack of intuitive meaning: it was difficult to tell if a particular score indicated a good or bad cluster construction. While TagScore is closer to a confidence than a similarity score, it can be used to determine if two documents are similar by swapping their tags and recalculating their confidences. For example, saying "*pages x,y had confidences of 0.7 before and 0.6 after switching their tags*" is perhaps more intuitive than saying "*cosine(x,y) is 0.1*"—we know where 0.65 stands in terms of TagScore, but we cannot make the same guess about a 0.1 cosine until other clusters are evaluated. As such, it may be possible to use a 'pairwise TagScore' as a replacement for pairwise cosine in clustering and have a more intuitive meaning behind

the values. Small-scale experiments showed surprisingly good results, but unfortunately the process is not much faster than using cosine because the full content needs to be used for each comparison to re-score a page with its new tags.

The few works that have rated tags in some manner have done so in different ways or for different purposes to TagScore. For instance, Xu et al. [138] summed reputation scores to determine tag ratings. In contrast, TagScore uses a combination of features (Section 4.2.1.5) that already exist, are applicable to all repository items and are calculated in a page-independent and user-independent manner—none of which are attributes of methods that use reputation scores. Shenkel et al. [124] combined content-based search in social networks with user-user friendships to personalize search results in favor of objects that a user's friends were more likely to be interested in. For tag ratings they used TFIDF but observed only small improvements in accuracy. In contrast, TFIDF is a partial component of TagScore's rating method and we observed significant improvements. Nevertheless, the focus in [124]'s work was query efficiency instead of accuracy. Finally, Heymann et al. [61] looked at deciding whether a given Delicious tag was suitable for a given page. Unfortunately their experiments focused on the top ≈2% of best-tagged pages. This makes it unclear if their approach, which relied on association rules to determine a tag's suitability using existing tags, would be as effective for the majority of the repository where pages are unpopular and have few tags. Their goal was also different to TagScore: they aimed to add new tags that were not there, whereas TagScore aims to rate what is there without changing it.

An investigation into tagging behavior by Noll & Meinel [96] argued that web2.0 has heralded a shift in describing web content from author tags to reader tags. They compared author titles and meta-keywords to reader tags against the PageRanks of the bookmarks to find correlations, and found that author tags were stable across all pages while user tags were not. Partially inspired by this metadata inequality between authors and readers, TagScore was made to utilize both author and reader metadata as a way of improving the system's use of tagging on the whole.

Tagging systems have certainly attracted interest from the research community in recent years. The contribution of a scoring function has practical uses for many tag-based functions and complements many of the existing works. The ideas in Chapter 4 for the personal space of online bookmarking help answer some retrospective questions about web navigation on mobiles, but are also useful for the fourth and final search space in this thesis.

1.5 Sponsoring mobile content

Having explored how web objects can be labeled with small sets of keywords, the thesis returns to the mobile domain to investigate a new and emerging personal space: sponsoring content on mobile devices.

Chapter 5 is dedicated to this search space, where a framework for offline, client-side mobile advertising is proposed. The presented approach involves placing much of the "intelligence" of content targeted advertising on the device. In particular, the chapter describes and evaluates a lightweight ad-selection process suitable for deployment under the strict constraints of small devices. Its offline nature is attractive because it enables other small devices, such as iPods and Netbooks, to have sponsored content, free downloads and possibly other subsidized costs for services and content.

Modern mobile devices are very popular and mobile users are beginning to use more and more content-based services. These services include texting, web browsing, video, music, email, tv, games and various other types of content. Advertisers are also looking for ways to better reach the large mobile market. Unfortunately, the current mobile marketing strategies for targeted advertising suffer from many shortcomings, such as low coverage of use cases, poor timeliness, poor relevance and repeated transmission cost. The proposed framework addresses all of these issues.

Mobile marketing is a rapidly growing business. Mobile-specific advertising frameworks have only recently begun to emerge, many of which are still in patent form and

without an implementation. Several companies have recognized the advantages of storing a database of ads locally on the user's device, which has obvious parallels to prefetch technologies such as Google Gears and Ajax. For example, Samsung [122] described a process for a device manufacturer, ad-broker and advertiser to take part in a revenue-sharing deal and place ads locally on televisions so that movie-on-demand search listings on the device could emphasize sponsored items. Western Digital [135] described a system specifically for mobile phones, where image ads were prefetched on the device and later shown in ad-reserved blank spaces in participating websites whenever the site was viewed on the mobile browser. The websites needed to specify which ad was to be shown, so the system's purpose was primarily to conserve bandwidth rather than repeatedly download the same image. This system is different to the proposed framework in that no "intelligence" is performed by the device.

SanDisk [123] proposed for ads to be stored on a phone and to be displayed based on predetermined phone events. Such events were defined as call events (incoming, outgoing, alarm, etc.) and readings from physical sensors (e.g. time, temperature, perspiration, etc.). Unfortunately, such physical triggers are not intuitive for advertisers to specify and their correlation to relevance is unclear. Scale will also be a problem because the triggers need to be specified manually and yet are intuitive, making the process somewhat slow. However, the system was a step up from previous systems because some intelligence was placed on the device, with exemplary uses being a flower shop ad shown whenever an incoming call from the 'home' address entry was received, and a pizza ad being shown when a call was made at night. A similar system was proposed earlier by Outland Research [98], where ads were shown alongside mobile web search results in response to physical sensor triggers such as weather conditions, temperature, time, heart rate, blood pressure, etc. The selection processes in both of these systems is rigid and non-intuitive, which may lead to low advertiser participation and weak user experience. In contrast, the proposed framework and selection algorithm are based on keywords, which online advertisers are familiar with and may even have already created campaigns for in the past. Additionally,

the proposed framework chooses ads dynamically for (essentially infinitely) many input cases, as opposed to predetermined triggers. Because the ads are targeted to what content is currently being viewed on the phone, the ad's relevance is also likely to be much higher.

The state-of-the-art in targeted advertising is now online. Much of the automation of online ad-serving relies on ads being labeled, indexed and searched using keywords. However, there are several differences to the proposed framework. In online Sponsored Search, multiple data sources are used to determine ad relevance. By typing a search query the user immediately reveals useful information to a search engine, which then uses the query and various historical data associated with it to display ads [46]. In contrast, on a mobile there are many types of content that a user does not access by typing into a search box, therefore many query-based features that search engines use are not for many of the mobile content types that we wish to target. The other major form of web advertising performed by search engines is Contextual Advertising, in which there is no query. Instead, ads are chosen to suit web page content. However, one of the reasons that the three biggest online ad-brokers are also the three major search engines is that they use web-specific data to choose the ads—data that they have very large quantities of. These include positional effects, click-fraud counter-measures, search query logs, historical clickthrough rates (CTR), link analysis, URL and domain analysis, anchor text, and content merging of related documents [50, 133, 142]. These sources apply to web content viewed through a mobile browser, but not to general (non-browser) mobile content. The lack of these data sources for general content removes what makes advertising by a search engine attractive in the first place. An additional difference is that online ad databases are shared, whereas on a mobile the user can have their own local, personalized database. A very lightweight approach for updates and selection is needed to make ad-serving possible on the client-side, but light methods have generally ignored by the web IR community because of the assumption that server farms will do the work.

Ribeiro-Neto et al. [115] provide something close to what is needed: they matched ads directly against web page content. Several of their match strategies used only the

ad data and the web data, without external sources. Nevertheless, we do not directly compare with their work because both the content domain and application are different. While [115] matches ads to content, their content is specifically webpages, which are long documents with many words and measures such as cosine similarity produce reasonable results. On mobiles, however, a typical document is very short and very sparse. It may be an SMS message, or a song title, or a description of a video clip. Cosine similarity does not work for such short data, and query expansion—which they regard as one of their superior strategies—is needed straight away on a mobile. Furthermore, their strategies specifically target HTML and the hyperlinked web, with their best performing strategy not only using hyperlink data, but also expanding the already-long webpage content to even longer by augmenting other related webpages to it. Such strategies are not applicable to mobile content and therefore cannot be compared. Regardless, their strategies relate only to the expansion and scoring steps of the proposed framework, which are only two steps in the procedure.

The works of Bollegala et al. [15], Sahami et al. [119] and Yih & Meek [143] are also relevant because they specifically focused on computing similarity between two short segments of text. Unfortunately, each of their solutions relied on submitting the short text segment as a query to a search engine and comparing the retrieved results instead of the original segments. This is too expensive for client-side processing on a mobile and does not satisfy the requirement of working offline.

Aside from similarity, the literature on web advertising heavily focuses on the complex game theory of bid optimization (e.g. [16, 38, 87] for some recent works). Chapter 5 focuses on timing and accuracy, thus largely ignores billing because it is out of scope. The distributed and offline nature of offline advertising complicates the revenue optimization problem due to delayed reporting. Because of this delay, it may be more suitable for advertisers to specify a weekly (or monthly) budget, as opposed to the daily budget that is usually specified for online advertising. The bids and unspent budget amount can be directly encoded within the tag weights, which directly influence the ads' chances of

selection. Hence, as a budget begins to run out, reducing the tag weights via periodic updates will help ensure that an ad is not shown more times than what the advertiser is willing to pay for.

Apart from billing, the chapter mentions several components for personalizing the selection method. Namely, these components record user feedback and history to learn a profile of the user's short-term and long-term interests, as well as using a geographic function to determine how appropriate or inappropriate an ad is based on its location. Because the aim of the thesis is to improve core searching methods instead of simply adding layers of personalization, these components are described but are only a minor focus. The purpose of the chapter is to demonstrate that we can place much of the intelligence of content targeted ad-serving on a small device and achieve good results. This by itself is a novel research problem for which the proposed ad-selection approach may serve as a baseline for improved methods in the near future.

Chapter 2

Searching in a hierarchy

2.1 Introduction

Users encounter content with a hierarchical structure on a daily basis, with obvious examples including files on a computer (files/folders), technical documentation, and books (chapters/sections). The main difference in content between these examples is that computer files have a wide variety of non-text formats whereas the other two cases are usually textual. However, many non-text formats can be described (or summarized) using text, therefore we can consider the domains to be similar for the purposes of this chapter.

Prior work in IR has focused on the personal desktop domain, called Desktop Search (or DS), which refers to searching for files on a computer. Certain file types are known to be related, such as a .tex file and a .bib file, or a .html file and a .css file—and indeed there are hundreds of such relationships—but rather than discuss the minutiae of file types, this chapter focuses only on how the files are organized. For simplicity the content is assumed to be textual. Although the experiments use a file hierarchy, ignoring the domain-specific relationships between file types means that the approach can be used on other types of structured hierarchies.

Desktop Search, the search through such a file hierarchy, is similar to keyword searching in a book in the sense that keywords are being used to retrieve content that is somehow

organized in a hierarchical structure. Existing DS tools do not analyze structure and instead perform a simple keyword search. The work in this chapter aims to improve on this unsophisticated way of searching by leveraging information about the structure.

Apart from structure, there are two important differences between DS and web search. In web search, millions of online users issue millions of queries per day, allowing search engines to use query logs, clickthroughs and the web hyperlink graph to determine relevant results. These techniques do not work for the desktop. Additionally, in web search the corpus is very large and impersonal, unlike the personal space of the desktop. The distinction is important because it means that traditional measures of precision and recall are not the most suitable; even if a retrieved result is query-relevant according to some metric, the user will ignore it unless it is what the were specifically looking for.

A more suitable metric is Mean Reciprocal Rank (MRR) because it uses the ranks of wanted results, i.e. known-item search. For instance, a user may search this thesis for "HITS", seeking sections or chapters that discuss the algorithm. The keyword may be mentioned many times, but the user is likely interested in navigating to the most important or relevant section. Consider a case where several sections mention "HITS" but do so only once. A typical keyword search will either rank them as equal or attempt to normalize by the section length. Alternatively, we can exploit the structure to find out which sections are closer together—perhaps even in the same chapter—and which are isolated. As another second example, consider the Java API documentation. A user may search for a query "write to file" and seek a class that they can use for disk IO. However, these keywords are very common and match hundreds of files. Yet, only a few of them are results that the use will be satisfied with. The goal of leveraging the structure is to determine which results are more likely to be what the user is after and to push the less-likely, noisy results lower in the ranking.

On the web, known-item search queries can be addressed by connectivity analysis algorithms such as HITS [69] and PageRank [99]. These algorithms analyze the inlinks (and in the case of HITS, also outlinks) of webpages to determine websites that exhibit

some degree of popularity or authority. Authoritativeness allows, for example, the DBLP website to be the highest ranked result for the query "dblp" even though millions of pages contain that keyword.

Unfortunately, files on the desktop, books and technical documentation lack the hyperlinks that enables such analysis. In some cases there exist some domain-specific links, such as sections in a book referencing other sections, or classes in an API referencing other classes, or symbolic links in a filesystem. In general, however, these are few and domain-specific. In order to address the general problem of content hierarchies, such domain-specific links cannot be used. Consequently, this chapter makes the assumption that the objects are organized in a tree hierarchy but have no internal references. Because there are no hyperlinks, another means of determining authoritativeness is needed.

As a solution to this problem, this chapter adapts the connectivity analysis paradigm to help harness the hierarchical structure for known-item searches. A lightweight rearrangement algorithm, Parameterized Filesystem HITS (or PFH) is presented. PFH rearranges the search results of an existing content-only retrieval tool in a way that favors less-isolated files[1]. The experiments use a file/folder corpus, but the concept generalizes to similar tree-like hierarchical arrengements. When comparing against five modern DS tools, the experiments show that some consideration of structure significantly improves MRR (i.e. known-item search) while keeping other IR metrics steady. The optimal consideration, as indicated by an input α parameter, may differ between separate domains, but the experiments show that there exists such an α that favorably improves MRR. The optimum value of α should be determinable for any corpus given a set of representative queries and relevances.

[1]This notion of isolation will become clearer later in the chapter; essentially, PFH biases in favor of query-matching files that are close to other query-matching files. This results in a bias against files that are very far (i.e. isolated) from any other query-matching results.

Overview of PFH

At present there are many commercial DS tools, including offerings from Google, Microsoft and Yahoo. The tools are applications that the user downloads and installs on their computer. They are content-focused and perform pure content analysis, making them unsophisticated in comparison to how web search engines answer queries. PFH fits somewhere in between web search and desktop search in that it uses inspiration from successful web approaches for the un-hyperlinked desktop environment.

Some recent research results have explored ways of extending traditional content-only search on the desktop, such as by monitoring the user and system activity in order to discover file relationships [36, 126]. The assumption is that users work on a single task at a time, so files used within a task 'window' may be related. PFH also fits somewhere in between the content-only and task-based models in that a certain amount of latent human judgment is taken into account (i.e. the hierarchy) but there is no monitoring phase or cold-start problem. As such, PFH can address situations where the task-focused approaches are less effective. For example, during the initial monitoring phase the task-based methods need to monitor user and system activity in order to construct their models (a process that takes many months [126]). A second situation where task-focused approaches are less effective is for searches that are not aided by the task-based relationship model. For example, the monitoring tool may be given a query about a task that has not yet been encountered during monitoring. In these cases, a non-task-based method like PFH will be useful.

Unlike the aforementioned applications, PFH is not a standalone DS tool. It is a rearrangement algorithm that works on top of the results given by an existing DS tool. The underlying tool may be sophisticated or unsophisticated, and in practice any tool can be used as long as it reports scores for its results ranking. Commercial DS tools hide their ranking scores from users, so the experiments use Apache Lucene [3] as the underlying retrieval engine. Specifically, given a set of content-only search results from Lucene, PFH determines a 'structure' score for the results and then uses the combination of structural

score and content score to rearrange the rankings. The input α parameter influences the balance between structure and content, allowing us to see how much consideration to give to one or the other.

Penev et al. [100] suggested that the organizational structure reveals hyperlink-like relationships in files. This chapter extends that initial idea and explores how much weight to assign to structure by introducing PFH. The key is to favor relevant content nodes from relevant node neighborhoods in the hierarchy, which is similar to how web search engines favor relevant authoritative websites with many inlinks from other relevant websites. Consider the simple hierarchy in Fig 2.1, consisting of several nodes:

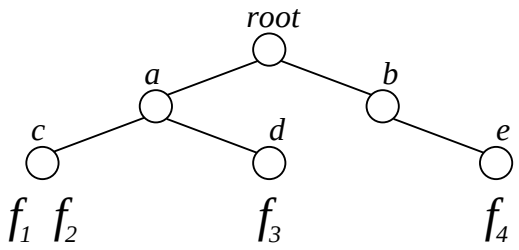

Figure 2.1: A simple hierarchy

For familiarity, suppose that the leaf nodes $f_{1,2,3,4}$ are files on a computer and that the structural nodes are their enveloping folders (i.e. directories). Suppose that a DS search tool returns these four files for some query with scores 0.25, 0.26, 0.33 and 0.6, respectively. In what order should they be ranked? The obvious order is by score, with f_4 on top. However, f_4 is somewhat isolated whereas the other files are close together. Suppose f_4 actually scored 0.34 instead of 0.6 and was only marginally ahead of f_3. When the difference is so small, can we justify placing f_3 on top because it is a more 'relevant neighborhood'? Because users group related files together, we know that isolated matches are often accidental matches. With an average DS query length of less than two keywords [37], there may be many such accidental matches and many of them will be isolated. Therefore intuition suggests that we can sometimes justify placing f_3 ahead if the scores are close.

In Fig 2.1, it only takes a small consideration of structure to place f_3 ahead of f_4

if its score was 0.34. If its score was the original 0.6 then it would take a much larger consideration. Of course, if the user really wanted f_4 after all, then the file would still rank highly. In this case the MRR measure will gauge if PFH has overall made a beneficial rearrangement. How much emphasis PFH places on content over structure, as indicated by the α parameter, affects when (or if) f_4 will be displaced from the top position.

Intuitively, one would expect a consideration of both structure and content to produce better results than either structure or content alone. This is the idea investigated in this chapter, which is organized as follows. First, some background on Desktop Search and connectivity analysis is given in Section 2.2. Section 2.3 then describes the methodology and construction of PFH, which is evaluated in Section 2.4. Section 2.5 gives a summary and some conclusions.

2.2 Background

This section gives a quick overview of DS and connectivity analysis, as well as some common terminology:

Content analysis, the examination of the textual composition of a communication, aims to determine the communication's objective meaning and is used to algorithmically judge its relevance against a given keyword query. On the web, an **authority** is a web document perceived to contain relevant information on a topic and a **hub** is a document linking to authorities. A hub may itself also be an authority. In our interpretation, authorities are hierarchy nodes with relevant content and hubs are hierarchy nodes containing authorities.

2.2.1 Desktop Search

DS tools are applications that index and search files on personal computers. Modern tools hide in the background and silently update their index. Today's tools are faster and can index more file types than the seminal tools of the 1980s, but the search techniques

have not changed. As hard disks and storage becomes bigger, however, the search results become larger.

If a user is after a specific file on her computer, she will type a keyword query to view a list of matches. She can sort the list by metadata (e.g. by size, by date, alphabetically) to help pinpoint the target file. If she is unsure or has forgotten the metadata, she may prefer to rank by relevance. Both *by date* and *by relevance* are popular sort criteria for desktop search [37].

This chapter is only interested in the rank-by-relevance function. This is because, presumably, if the user already remembers metadata about her target file, such as its size, date or name, then she would not need the files to be deeply analyzed in the first place and can simply locate her target using the metadata she remembers. This chapter proposes the use of PFH to rearrange the ranked-by-relevance desktop search results of the underlying retrieval engine.

In practice, using PFH for desktop search would require a few extra considerations, which are out of scope of this thesis. The first consideration is that certain file types on the desktop are not suitable for plain-text indexing and searching. For example, an email .mbox file is a file that concatenates many emails. Such a file is not a useful search result because it may represent thousands of emails. Handling this case would require the underlying engine to understand that the .mbox format needs special handling. The experiments assume that all items are plain text units.

A second consideration is dump folders, where different kinds of unrelated files are temporarily saved. Such folders are specific to filesystems and do not exist in some other hierarchies, such as books. Their existence in desktop search can negatively affect PFH's performance because they represent meaningless and random organization. PFH should therefore be aware of dump folders and treat them differently. Since how dump folders are named and used differs from person to person, the simplest solution is to ask the user to specify their dump folders (typically 'My Documents' or 'Desktop'). The search results from such folders can be presented separately in the search results interface, providing

two different ranked-by-relevance lists: one listing the top files from the dump folders and the other listing files from the rest of the file system.

Apart from such domain-specific considerations, the common denominator between files on a computer, text in a book and technical documentation is that the content is presented in a hierarchy of nodes called a tree. Being a tree, it is also a graph.

2.2.2 Connectivity Analysis

Connectivity Analysis measures how well-connected a node is in a graph. The World Wide Web is a connected graph of webpages that is regularly subjected to this kind of analysis using algorithms typically based on the core graph-theoretic principles of PageRank and HITS [26, 27, 32, 72, 77, 83]. Both algorithms have similar goals—to find the top authorities for a query—but HITS's principles are the more suitable for the hierarchical domain. This is because PageRank models the behavior of browsing, in particular the 'random walk' where a web surfer follows hyperlinks from page to page but suddenly jumps to a completely different page. Such a model makes sense for the web but not for a desktop because the user is not aimlessly browsing and does not get 'bored' and jump. Unlike PageRank's modeling of behavior, HITS models only structure. It finds hubs as well as authorities and its interpretations of what these two entities represent have elegant mappings to folders and files. These qualities make it the more suitable algorithm to investigate. The original HITS is shown below.

HITS. calculates hubs/auths on the web.
Inputs: webpages V and hyperlinks E.
Outputs: hub scores $\vec{H}$, authority scores $\vec{A}$.
1. $\forall v \in V$, initialize $H_0[v] := A_0[v] := 1$.
2. **for** iteration k **until** convergence:
3. $\quad \forall v \in V$, **set** $H_k[v] := \sum_{o:(v \to o) \in E} A_{k-1}[o]$
4. $\quad \forall v \in V$, **set** $A_k[v] := \sum_{i:(i \to v) \in E} H_{k-1}[i]$
5. $\quad$ **normalize**$_{L_1}(\vec{H_k})$ and **normalize**$_{L_1}(\vec{A_k})$.
6. **end for**
7. **return** $\vec{H}$ and $\vec{A}$

The algorithm posits that every web page serves two purposes: to be a hub and/or to be

an authority. Hubs may still be worthwhile search results if they link to good authorities. While PageRank computes only authority scores, HITS uses a node's outlinks to calculate its hub score. Hubs and authorities are in a mutually-reinforcing relationship in which a good hub should link to good authorities and a good authority should be linked to by good hubs.

First, a small *root set* of pages is retrieved from a search engine. This query-focused subgraph may be highly disconnected, so a *base set* is formed by adding a quota of nodes ± 1 hop away. Relevant pages that do not include the keywords can still be ranked for authority since they are part of this neighborhood. The new pages provide extra edges and induce an adjacency matrix M on the base set digraph.

Nodes are assigned a non-zero hub and authority score and HITS proceeds to iteratively "shake" the graph to distribute weight. Iteration stops when an equilibrium is reached. Under the Perron-Frobenius theorem [44], the algorithm quickly converges [12, 69] to the dominant eigenvectors of $M^T M$ and MM^T, returning them as the hub and authority scores.

Combined with content analysis, this produces better results than pure content analysis for linked document spaces. Examples of linked document spaces are the body of academic research, where edges are citations to research papers, and the world wide web, where edges are hyperlinks between webpages.

2.3 Approach

This section details the derivation of PFH, Parameterized Filesystem HITS, based on the principles of hubs and authorities in HITS.

PFH is an iterative algorithm that shakes a link graph of files and folders. An $0 \leq \alpha \leq 1$ parameter is introduced to linearly vary the reliance between structure and content, where 0 represents structure-only, 1 represents content-only and 0.5 is an equal consideration of both. At α=1, PFH leaves Lucene's results unchanged.

2.3.1 PFH Algorithm

The output of PFH is hub ($\vec{H}$) and authority ($\vec{A}$) scores with similar interpretation as HITS. However, hubs and authorities in a hierarchy such as a filesystem prevents a node from acting as both—a file is not a folder and a folder does not contain semantic content—therefore the output $\vec{H}$ represents the folders and $\vec{A}$ represents the files without overlap. The $\vec{A}$ scores are used to rank the final search results.

To obtain a root set of files F, PFH retrieves the top-250 results from Lucene. Lucene's content analysis score for a file f is referred to as C_f. While Lucene's results should already be focused on the query, they will be run through PFH and rearranged into a new ordering.

Do we need include any additional files in F? On the web, HITS considers webpages ± 1 hop away from the root set so that it can find authorities that do not contain the keywords. This does not apply to the desktop because users are expected to be recalling information and will at the very least be expected to use a keyword that appears in their target file. Therefore F does not need to be augmented with extra files, particularly not files that do not match the query words.

Because hubs and authorities are in a reinforcing relationship, a set of folders D (i.e. 'directories') is needed to be ranked for hubness. Since we are not interested in the files outside of F, only the folders forming a spanning tree on F need to be included in D. They provide implicit relationships such as 'containment' and 'sibling', so $F \cup D$ closely corresponds to the *base set* in the original HITS.

Augmenting D onto F to form the base set introduces implicit hierarchical connections between *all* nodes. Some links are direct, but many are formed by a path along the folder tree. In agreement with [82, 91], the influence of one node on another is decayed by their distance. The length of a link path δ_{d_1,d_2} between two folders[2] d_1 and d_2 can be deduced from their paths by first finding the deepest ancestor-or-self node λ shared by both and

[2] A file is considered be 0 distance away from its parent folder, so δ is the same if the file or the folder is used in the notation.

then, using $|n|$ to refer to n's depth in the tree, setting:

$$\delta_{d_1,d_2} = |d_1| - |\lambda| \quad + \quad |d_2| - |\lambda|$$

In Fig 2.1's example, $\delta_{e,f_4} = 0$, $\delta_{f_4,c} = 4$ and $\delta_{c,d} = 2$.

For a decay function, candidates considered were reciprocal factorials [91], parameter exponents [82] and the Inverse-Square Law from physics. In practice all three gave similar results and PFH was defined with the simplest one, $1/(1+\delta)^2$.

So far, we have a set of files F and their content scores $\vec{C}$, a set of folders D, and a decaying function that fades influence with distance. We can now formulate the 'structure' and 'content' components for files (authorities) and folders (hubs). The individual components will be first described, below, and the α parameter will be introduced at the end.

Suppose we want a 'content' component for a hub, d. This is considered as an aggregation of scores of the files d contains, because a relevant directory should contain relevant files. Formally, at iteration k:

$$d_{content} = \sum_{f \in F : \delta_{d,f} = 0} A_{k-1}[f]$$

To avoid having a hub with many weak authorities surpassing one with few but strong ones, its score is multiplied by a ratio of $\frac{nr}{nf}$, where nr denotes the number of relevant files in d and nf denotes the total number of files in d. In this case nr is trivially obtained from F. Additionally, a log factor can be used so that when two hubs have similar ratios, ones with more files are favored:

$$d_{content} = \frac{nr \ log_{10}(1+nr)}{1+nf} \sum_{f \in F : \delta_{d,f} = 0} A_{k-1}[f] \tag{2.1}$$

Next, we seek a 'structure' component for a hub. This is an aggregation of the objects

that d links to. Since folders contain both files and other folders, d is connected to all nodes in $F \cup D$ with distance decays.

With PFH's parameterized model, d's hubness influence over the authoritativeness of the files in F would be multiplied by α and over other folders in D by $(1-\alpha)$. However, as shown above, files are already aggregated under their parent folder. Therefore it is reasonable to consider the sum over F as a second sum over D. The two factors α and $(1-\alpha)$ cancel out, leaving:

$$d_{structure} = \sum_{d' \in D} \frac{H_{k-1}[d']}{(1+\delta_{d,d'})^2} \tag{2.2}$$

Hence, a hub's score takes the form (2.1) + (2.2). Note that if a folder has no files, it will have a zero content component but a non-zero structural component. This allows scores to propagate past folders that have only subfolders and no files.

Next, we want a 'content' component for an authority, f. This score, between 0 and 1, is already reported by the underlying retrieval engine:

$$f_{content} = C_f \tag{2.3}$$

Finally, we want a 'structure' component for an authority. This is an aggregation of objects that link to f. A file is contained only within a folder, so the sum is over D. This will later use an α factor, but for now:

$$f_{structure} = \sum_{d \in D} \frac{H_{k-1}[d]}{(1+\delta_{f,d})^2} \tag{2.4}$$

Hence, an authority's score takes the form (2.3) + (2.4). We can now piece together the individual components and introduce α. The PFH algorithm, shown overleaf, has three inputs. By construction $F \cup D$ is the base set, but they are passed in as separate arguments for convenience. The third input is a content analysis score $0 \leq C_f \leq 1$ for all $f \in F$, as reported by Lucene.

PFH: calculates hubs/auths in a filesystem.
Inputs: dirs D, files F and their content analyses $\vec{C}$.
Outputs: hub scores $\vec{H}$ for D, auth scores $\vec{A}$ for F.
1. **initialize** $\vec{H_0}$ and $\vec{A_0}$ to all 1's.
2. **for** iteration k **from** 1 **to** K:
3. **for** $d \in D$:
4. **let** $dc[d] := d_{content}$, as in (2.1)
5. **let** $ds[d] := d_{structure}$, as in (2.2)
6. **normalize**$_{L_\infty}(dc)$ and **normalize**$_{L_\infty}(ds)$
7. **for** $d \in D$:
8. **set** $H_k[d] := \alpha\ dc[d] \quad + \quad \underbrace{(\alpha + (1-\alpha))}_{cancels,\ as\ in\ (2.2)}\ ds[d]$
9. **for** $f \in F$:
10. **let** $fc[f] := f_{content}$, as in (2.3)
11. **let** $fs[f] := f_{structure}$, as in (2.4)
12. **normalize**$_{L_\infty}(fs)$
13. **for** $f \in F$:
14. **set** $A_k[f] := \alpha\ fc[f] \quad + \quad (1-\alpha)\ fs[f]$
15. **normalize**$_{L_1}(\vec{H_k})$ and **normalize**$_{L_1}(\vec{A_k})$.
16. **end for**
17. **return** $\vec{H}$ and $\vec{A}$.

PFH iteratively 'shakes' the node graph to distribute weight along the edges (i.e. folder paths). Intermediate results for $\vec{H}$ and $\vec{A}$ at iteration k are labeled $\vec{H_k}$ and $\vec{A_k}$. The algorithm converges to an equilibrium state very quickly, requiring only about 10 iterations to stabilize the topmost results [69]. We are working with a smaller result set—only the top-250—and therefore require even fewer iterations, but in the experiments we use K=20 to ensure that the positions of even the low-ranked results are stabilized.

Note that is it necessary to normalize the tentative components before using them. This is because the C_f content scores are normalized to [0,1], but the $\vec{H_k}$ and $\vec{A_k}$ scores are normalized to sum to 1. To ensure that the halfway split between structure versus content occurs precisely at α=0.5, the components must use the same scale. In the notation, **normalize**$_{L_p}(\vec{v})$ means dividing $\vec{v}$ by its L_p norm.

2.3.2 Observations

Boundary cases. It is easy to visualize α's extremities. When α=1, hub scores are ignored, $\vec{A}$ will inherit $\vec{C}$ and PFH returns Lucene's results unchanged. This is a content-only reliance. When α=0, $\vec{C}$ is ignored and $\vec{A}$ will depend on structural influences from hubs only. All files within a folder will receive identical scores. This is a structure-only reliance, and the top hubs will be those that are overall closest to the epicenters of 'relevant neighborhoods' in the base set. In other words, a folder will benefit if it has more files representing it among the top-250 Lucene results.

Simple filesystem example. Recall Fig 2.1. When α=1, PFH reports the authorities in the same order as Lucene: f_4, f_3, f_2, f_1. In terms of hubs, f_4's high content score promotes e to be the second best hub after c. The *root* and b receive the lowest hub scores. Although it has a lower content score, f_2 will rank above f_3 when $\alpha < 0.82$ because c is a stronger hub. It takes a large consideration of structure (0.55) to rank f_2 above f_4.

Complexity. PFH runs in $O(|D| \times max(|F|, |D|))$ time. The experiments fix K=20 and limit[3] $|F|$ to 250. The only unknown is D, the set of folders needed to form a spanning tree on F. Since D depends on F, it does not necessarily grow with the corpus. It will be small if the top-250 files are close together, but may be larger than $|F|$ if F is sparse. To speed up the computation, a $|D| \times |F \cup D|$ matrix of lookup δ values was constructed for PFH to use. This matrix contained the distance values between nodes, calculated just once rather than repeatedly. In the implementation this was computed naively in $O(n^2)$, but it can be optimized with dynamic programming. In the experiments the cumulative time for the matrix build and subsequent K=20 iterations of PFH averaged just 0.06s per query, so there was no need to optimize it. As mentioned earlier, PFH is a rearrangement algorithm that uses the content-only results of an underlying retrieval

[3]Kleinberg [69] limits the root set to 200 and expands the base set to typically 1000–5000. This is a tiny fraction of the web but is too much for the smaller, hierarchical corpora that PFH is suited for.

engine. Consequently, PFH's 'index' is subject to the same performance measures as that of the underlying engine.

Other observations. If the file hierarchy is flattened such that all files are in the same folder, all files receive the same hub influences and therefore C_f is the only discriminator. In other words, Lucene's results will be unchanged. If the root set is sparsely distributed and files are distant, hub influences will be weak because of large δ^2 decays. While the hub scores will be normalized, they will reflect the quality of the hub's own files and will incorporate little about the neighborhood. In this case, folders with many high-scoring files will be the top hubs, which still coincides with the notion of 'relevant folders'. If neither applies and the top-250 files are not extremely sparse or close, then the search problem is a typical DS search query. An experiment on such a case is performed in Section 2.4. If files are poorly organized (e.g. random), there will not be any meaningful organization to exploit. It is difficult to predict PFH's behavior in such atypical circumstances because files are almost never organized meaninglessly. Nevertheless, an experiment on such a case is performed in Section 2.4 to seek a 'safe' value of α for which PFH will perform not-worse than the underlying engine when the organization is meaningless.

2.4 Evaluation

Three experiments are performed: queries on an organized corpus, queries on a meaninglessly organized corpus, and file categorization.

In all experiments, PFH retrieved 250 results for its root set from Lucene v1.9, with Porter stemming enabled [108]. For experimental measurements requiring human relevance judgments, the union set of the top-50 ranked files from PFH and from each of the DS tools used as competition were judged in a blind test. The other DS tools are described below.

Competition and configuration

Four popular desktop search applications [94] were trialled: Copernic's, Google's, Microsoft's and Yahoo's. These are referred to as CDS, GDS, MDS and YDS. Lucene is included as the baseline.

For obtaining results from these tools, GDS's rank-by-relevance option was chosen; for MDS we selected 'Document' results and the rank-by-relevance option. CDS and YDS did not have relevance rankings and so their default lexical ordering was used. Each tool was configured to index only the corpus being experimented on.

The implementation of the commercial tools is a black-box. However, it is evident that they all use conjunctive matching and index stopwords. By default Lucene is disjunctive and ignores stopwords. Its developers likely considered this to be the most suitable setting for general use and these settings were left unchanged in the experiments. One can also notice that at least CDS and MDS stem, although weakly, and that all tools except Lucene match keywords in the file path (but only MDS uses it to influence the ranking). To avoid having the other DS tools using file path or timestamp information, which Lucene does not use, the names of the files and folders in the corpus were renamed to meaningless numeric names and their creation and last-modified timestamps were reset. This forced all tools to rely only on the content when differentiating files.

Note that this chapter is not concerned with the general exercise of indexing domain-

specific file formats, such as emails, music or photos. Therefore all files in the experiments were plain ASCII text that was sure to be easily indexed by each tool without problems. However, it is important to note that PFH obtains its root set from its underlying engine, therefore using an engine capable of retrieving and scoring emails or music or photos would also enable PFH to work on such files; PFH is only responsible for the content versus structure rearrangement, not for the indexing or retrieval.

2.4.1 Exp A – A structured corpus

This experiment aimed at evaluating PFH under MRR. Traditional metrics of Precision and Mean Average Precision are also reported.

Previous work in DS and intelligent search agents has performed evaluations on test subjects' personal desktops. A disadvantage of this is that results are irreproducible and that queries and results are not disclosed (for privacy reasons). Unfortunately there is not yet a standard DS corpus, although [29] are showing initiative in constructing one. In order to present reproducible and transparent results, the experiment used a well-known file hierarchy as its corpus: the JDK 1.5.0.

This corpus is a 10000 file tree containing source code for Java's library packages. It is an accessible and familiar corpus with a varied structure. Classes are richly documented with comments and these comments serve as the content. Javadoc was run on all files to strip away the source code and leave behind the developer comments in the API page format, which was then stripped of markup to leave behind only plain-text. The files were then tokenized so that no tool would have trouble indexing them. The corpus had 674 folders. On average there were 14 files per folder (med 5, max 321) and 930 words per file (med 460, max 38k). The average depth was 4.5 (med 4, max 9).

Although structured as a package tree, from the perspective of keyword queries there is much disorder in the hierarchy because there are many packages that deal with certain topics. Often it is a case of the implementation and interface being separate classes, or the comments reflecting similar content. Other times it is a case of many classes dealing

with similar issues. For example, if we combined the top-50 results from each of the DS tools for the query "connect remote server", we would have 242 unique results. This suggests that even the DS tools cannot come to an agreement on the most relevant files. Those 242 files are found in 45 different folders that lie in five separate superpackages, thus probing the entire hierarchy. As another example, the first query in Table 2.1 has four targets, but these files lie in different superpackages and have only the corpus root folder in common. Another property of the corpus is that many queries match a huge number of files. For example, the last query in Table 2.1 matches 94% of files disjuctively and 7% conjunctively, yet only 2 target files are rewarded. This behavior is typical of navigational search and is thus representative of DS queries.

	Query	$\lvert R\rvert$	$\bar{\delta}$	Target documents (full paths omitted for clarity)
	play sound	1%	4.7	AudioClip, AudioPlayer, Sequencer, JavaSoundAudioClip
	database	1%	5.1	Connection, DriverManager, DataSource, Statement, ResultSet
S	today date	3%	6.4	Date, Calendar, GregorianCalendar, System
F	display hierarchy	11%	5.3	JTree, JFileChooser, FileDialog
	display image	11%	5.4	ImageIcon, Image
	convert ip address	18%	5.7	InetAddress, Inet4Address, Inet6Address
	window tabs	7%	6.5	JTabbedPane
	schedule thread	8%	6.5	Timer, TimerTask
S	display html page	12%	7.4	JEditorPane, JTextPane, HTMLEditorKit, AppletContext
U	make beep sound	12%	6.7	Toolkit
	execute external program	14%	7.4	Runtime, ProcessBuilder
	connect remote server	14%	6.8	Socket, URLConnection, DatagramSocket
	select file	24%	6.0	JFileChooser, FileDialog
G	format numbers	31%	6.2	PrintWriter, Formatter, NumberFormat, Format, DecimalFormat
F	write data file	41%	6.5	FileWriter, FileOutputStream, PrintStream
	list supported fonts	46%	5.2	Toolkit, GraphicsEnvironment
	convert strings numbers	93%	2.9	Integer, Double, Long, Short, Float, Byte
	linked list	27%	7.2	LinkedList, List, Vector, ArrayList
	call garbage collector	30%	7.0	System
	generate random integer	36%	6.8	Random
G	file append data	38%	7.6	FileOutputStream, RandomAccessFile
U	read file data	43%	6.9	FileReader, FileInputStream, RandomAccessFile
	extract string pattern	91%	7.5	Matcher, Pattern, StreamTokenizer, StringTokenizer
	write object stream	94%	7.0	Serializable, ObjectOutputStream

Table 2.1: Test queries, grouped by $\lvert R\rvert$ (percent of the corpus retrieved by Lucene) and $\bar{\delta}$ (average pairwise δ of the top-250 files F).

Queries

A total of 24 queries were sourced from online Java FAQs that exemplify real-world user questions about some programming task, as shown in Table 2.1. These are considered to be navigational queries because they seek certain files. The targets files were the classes mentioned in the FAQ's answer. The queries—that is, the FAQ questions—were aligned to typical search practices by being condensed and expressed using between 2 and 3 keywords [37, 64, 127].

The queries were also categorized into four groups based on two splitting criteria: Specific–General (S/G) and Focused–Unfocused (F/U). Specific were those for which Lucene disjunctively retrieved 0–20% of the corpus. General queries used more-common keywords and disjunctively matched many more files. Focused were those for which F was not sparsely distributed, i.e. the average pairwise δ was below the median[4].

Results

For the precision-based metrics, relevance was judged in two passes. The first selected files that had anything at all to do with the query. The second removed the weakest selections to condense the first pass to 20 files maximum per query. A pooled assessment of the union set of top-50 results was used, requiring some 203 judgments per query. Results for MAP, P@10 and P@3 are shown in Figs 2.2-2.4.

Fig 2.2 shows Mean Average Precision (MAP), a stable metric across query set size [24]. MAP is not a comprehensive indicator for known-item search as it rewards potentially-wanted items. Nevertheless, PFH was within ±2.5% of the baseline when $\alpha > 0.69$. The difference was not statistically significant for most α (Wilcoxon $p >> 0.3$ when $\alpha > 0.5$).

Fig 2.3 shows P@10. PFH reached the baseline at α=0.54. The difference was not significant for $\alpha > 0.4$.

Fig 2.4 shows P@3, qualifying [66]'s observation that users often look at the top-3 results in search engines. PFH compared favorably for most α values, with the difference

[4]The query set and corpus medians were 6.5 and 7.0, so the queries were structurally representitive.

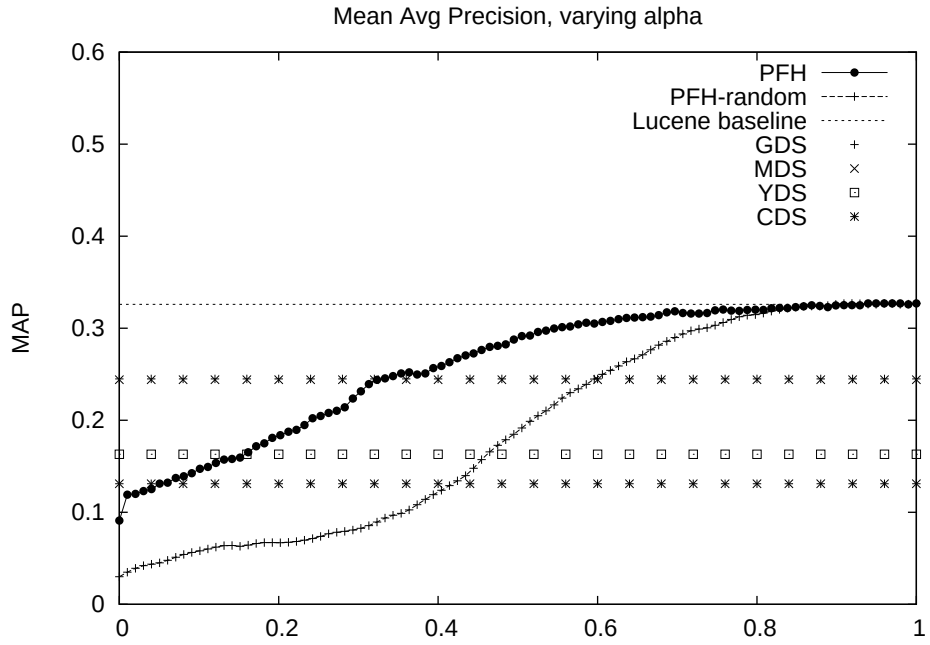

Figure 2.2: MAP for Table 2.1 queries in Exp A. Also shows Exp B's randomized bad-case scenario.

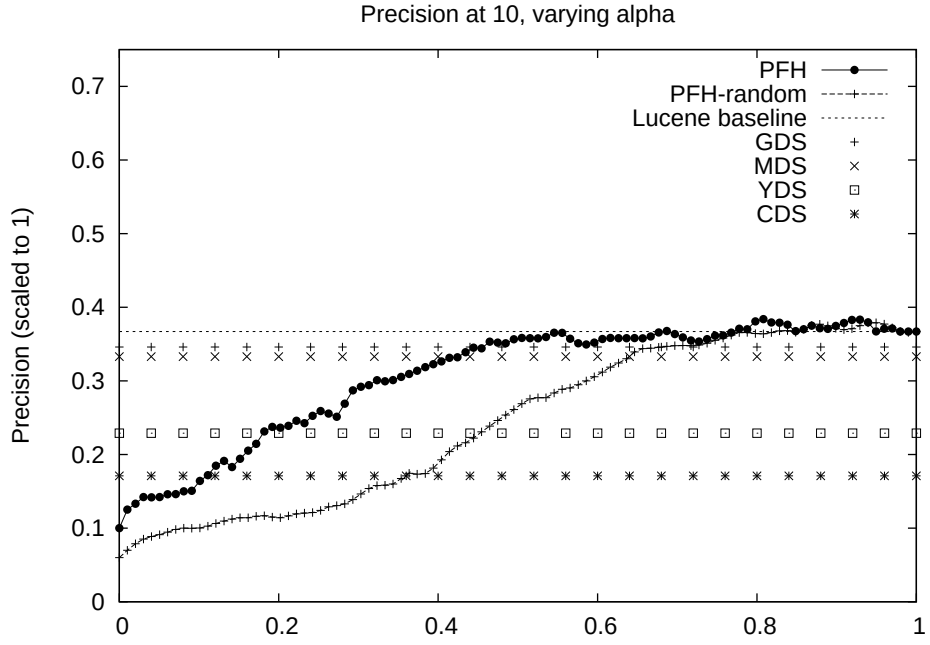

Figure 2.3: P@10 for Table 2.1 queries in Exp A. Also shows Exp B's randomized bad-case scenario.

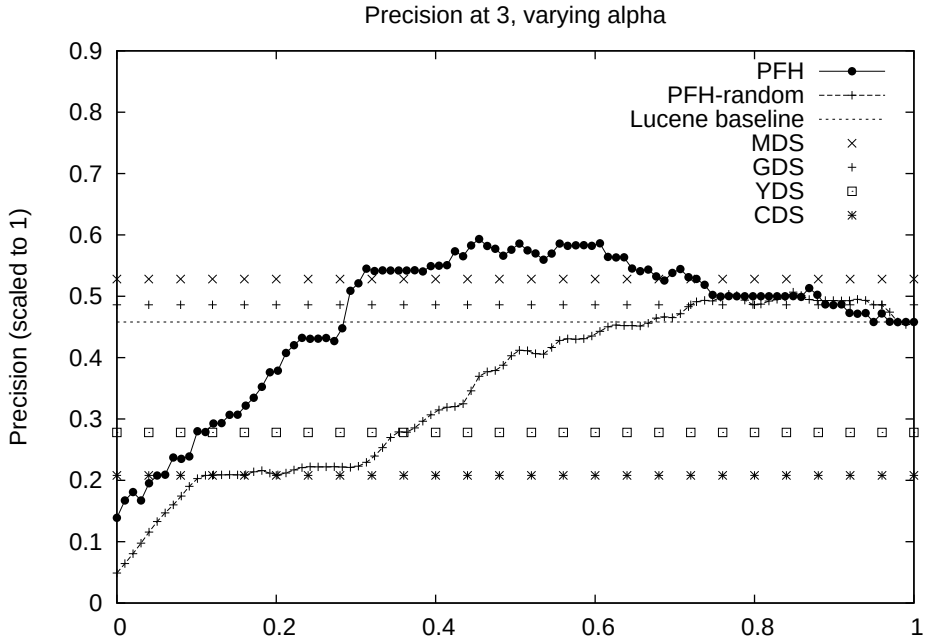

Figure 2.4: P@3 for Exp A. PFH had good precision for $\alpha > 0.6$ even under Exp B's randomized bad-case scenario.

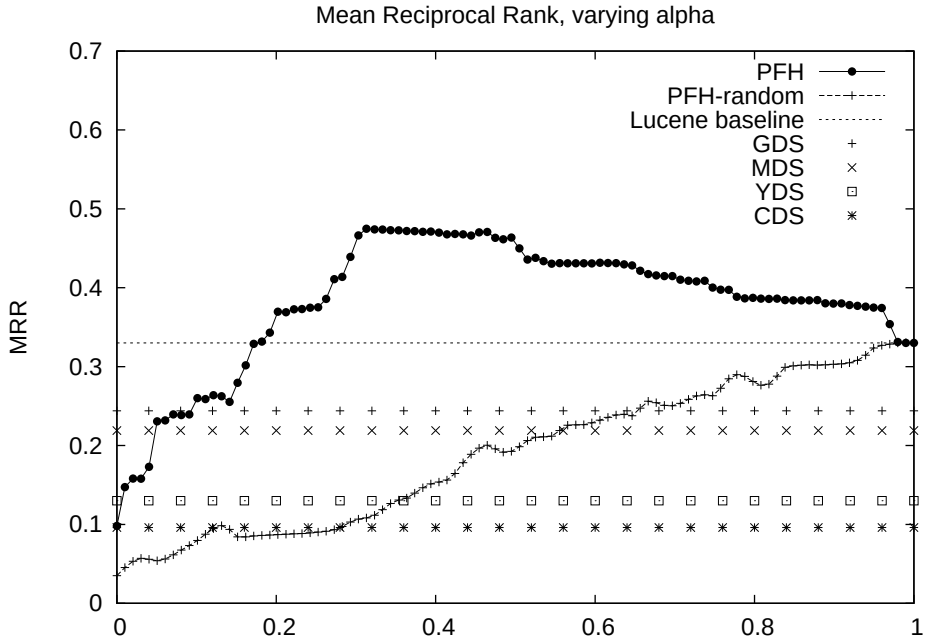

Figure 2.5: MRR for Exp A. Shows a significant accuracy improvement for known-item search, with the target items being ranked higher.

being significant ($p < 0.05$) in the middle. Comparing P@3 and P@10, it seems that PFH makes beneficial swaps in the top ranks but leaves the overall quality of the top-10 intact.

Fig 2.5 shows MRR, a useful metric for known-item search. PFH performed well when $\alpha > 0.17$, and the difference was significant for most α values ($p < 0.05$ when $0.27 < \alpha < 0.97$). The maximum was at 0.30–0.47, a strong bias towards structure that reflects a meaningful organization of files. However, this α-range produced suboptimal P@10 and MAP. Consolidating all four metrics revealed that α=0.75 was the best-performing value: compared to the baseline, it improved MRR and kept the other three metrics equal.

2.4.2 Exp B – Poor file organization

Exp A suggested that a 25% bias towards structure was good for a structured corpus. This experiment is interested in how this α value performed when the organization of files was poor, therefore 2 bad-case scenarios were created[5]. The first bad-case scenario used the normal Exp A queries on a randomized Exp A corpus, simulating poor organization with random file placement. The second bad-case scenario used auto-generated queries on the normal corpus, simulating poor organization by using vague or non-topical queries.

The first scenario is a smoke test to verify that $\text{PFH}_{\alpha=0.75}$ performed sanely when files were meaninglessly organized. In this case Lucene's results would be unaffected because Lucene is content-only. The second case is analoguous to the case where a user makes a poor query and expects to get the right result. In this case Lucene's results will be adversely affected and PFH will inherit poor results from Lucene. Overall, the combination of both of these bad-case scenarios can be used to reveal 'safe' values of α for which PFH does not worsen Lucene's results.

[5]Creating a worst-case scenario would require much surgery because results need to be returned in reverse. Such a hierarchy will not occur in practice as it will need more folders than files in order to isolate the relevant files on purpose. Even so, it would still be 'worst' for only one query.

Randomized corpus

To create a meaningless organization of files yet maintain the existing structural properties, random pairs of files were chosen and their textual contents were swapped. Enough swaps were made for each file's content to have jumped four times.

The results are presented in Figs 2.2-2.5 as the line 'PFH-random'. By construction, PFH-random still converges to Lucene at α=1 when the structure component is ignored.

The P@3 and P@10 graphs show that α=0.75 is still good for precision, while MAP suggests that 0.83 is safer. MRR visibly worsened. This was not unexpected: by randomly swapping files while keeping structure unchanged, dense folders acquired a large proportion of content-relevant files from the swapping. If the few target files did not end up near such hotspots, their ranks may have worsened. This explains why only MRR was worse than the baseline at α=0.75, although it was still higher than other DS tools.

This experiment showed that PFH performed adequately for $\alpha > 0.83$ even if organization was random. As such a disorganized corpus is unlikely in real life, it can be argued that 0.75-0.83 is a good α range.

Randomized queries

A 5% random sample of files was taken from the corpus and keyword queries were automatically created from the files' content by selecting words from the files. Each file was therefore guaranteed to be matched and retrieved by Lucene because its query directly matched its content; however, each file was not guaranteed to be ranked near the top. Term frequency values were ignored so that all unique words inside a file had an equal chance of being chosen to compose the query. Queries were either 1 or 2 words long in order to conform with Dumais et al.'s observation [37].

Instead of a single query, each file had 3 queries generated using the keywords inside the file. The keywords used had to lie in three different DF brackets of increasing rarity: the 2-19%, 10-27% and 19-35% brackets. Each bracket represented a pool of 2500 possible search terms. From the initial random sample, 354 files succeeded in having a query

generated using terms from the file that lied in each bracket. This meant that these 354 files had 3 queries of increasing keyword rarity that would be sure to select them using a keyword search. Intuitively, we would expect that the query with the rarer 19-35% keywords would be more strongly selective than the query with the 2-19% keywords.

Although the keywords were required to fit into a particular bracket, they were still randomly selected from the eligible keywords in the corresponding file. Therefore the generated query was not necessarily representative of the file's content. For example, the rare 19-35 bracket contained words such as 'think', 'smallest' and 'easily' which, if chosen, would not be representative of their file's topic. They are highly-selective words, but not topical words. As such, PFH would not be able to exploit the organization of files because files would be meaninglessly organized *in respect to the query*. An analogy would be a user having folders about 'school', 'home' and 'job' but wanting files about 'big'.

Relevance judgments were not collected for these queries and this experiment can only present MRR. Each query was generated from one file so there was only one target per query. As the average number of retrieved results was over PFH's quota of 250, it was more than enough for PFH's weight distribution to rearrange Lucene's results. The experiment is interested in finding which alpha ranges did not significantly worsen the accuracy compared to Lucene's baseline figure.

Fig 2.6 shows the results. As expected, rare 19-35 words produced higher MRR than common 2-19 words. Unlike Exp A, there is no improvement in MRR because PFH could not take advantage of any meaningful hierarchical organization. However, the graph shows that PFH was no worse than the baseline for $\alpha > 0.74$, consistent with the prior observations.

2.4.3 Exp C – Predictive save dialogs

A possible negative side-effect of improving DS search is that users may become lazy. They may pay less attention to how they organize their files because they know they can retrieve them quickly. A similar behavior already occurs online where instead of

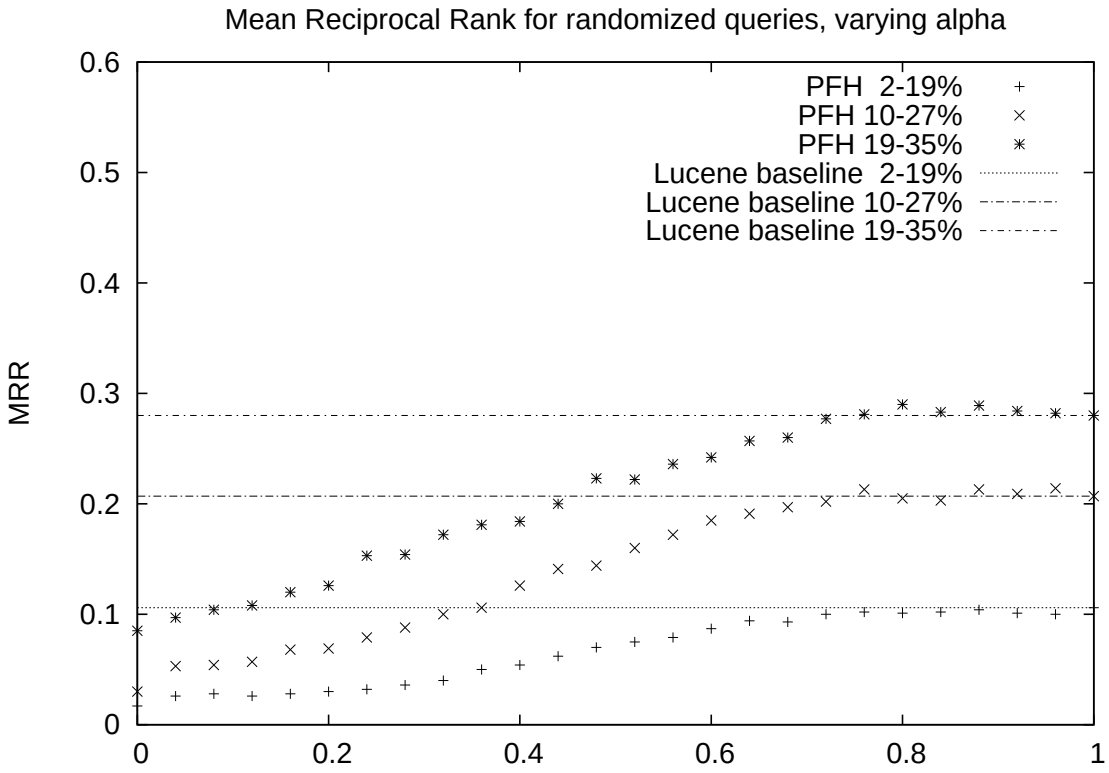

Figure 2.6: MRR for the three brackets in Exp B's randomized queries case (Section 2.4.2). Shows that $\alpha > 0.74$ are reasonably 'safe' values.

boomarking a site users often type a navigational query directly into the URL bar and let the browser redirect to the first search engine result.

While this behavior is not a problem online, it is undesirable for DS users because the number of files on a personal computer is growing with hard disk space. It is also undesirable for PFH because Exp B shows that it loses its advantage when files are randomly organized. Therefore it is still in the users' interests not to corrupt their established organizational practices.

One approach to help users maintain a modicum of diligence is to suggest suitable save locations for new files. A scenario could be that a user has just chosen to save a file from her web browser, and the browser immediately issues a query that represents the file to the resident DS tool. The results, a short ranked list of *folders* are displayed as part of the Save dialog. This all happens before the user has reacted to the dialog appearing.

Such a save-assistant would help users in two ways. First, clicking on a suggested

folder is a convenient shortcut to reach the desired location, or at least somewhere nearby. Second, it can serve as a cognitive aid. Recent operating systems such as Vista and OSX come packaged with a resident DS tool, making such an assistant a possible standard feature in the future.

This idea has been considered before. FolderPredictor [9], a component of TaskTracer [36], modified the Windows registry to change the default destinations in the open/save dialog. At the granularity of a "task", it would determine the suitable folders for a task-related file based on the user's past history of file access for that task. FolderPredictor has similar disadvantages as TaskTracer, which were described in Section 1.2.

PFH can categorize files by outputing folders as the results. It already computes the necessary information during regular execution so there is no extra work—the hub scores were simply discarded before. An advantage of using PFH for this is that it avoids the cold-start problem faced by monitoring approaches such as TaskTracer, and also removes the need for the user to specify when they are switching tasks.

To compare PFH against the other DS tools, they are required to return folders instead of files. A simple transformation is to consider a file result as a result for its parent folder. Ignoring repeated sightings produces a ranked list. This transformation was used for Lucene, MDS and GDS (CDS and YDS produced poor, alphabetically sorted results and were not tested).

Obtaining the top folders from PFH could be done in three simple ways. The first uses the above transformation, the second uses the hub scores directly, and the last is a hybrid (combination) approach.

The combination approach used was to set a directory d's score to $d_{combined} = \alpha\vec{A}[f] + (1-\alpha)\vec{H}[d]$, where f was the highest ranked file from d (0 if none). Sorting by this score gives a ranked list. Applying the transformation to the other DS tools necessarily outputs folders with at least one file inside, so any empty hubs in PFH's results were ignored.

Two cases were experimented on: the Exp A settings and Exp B's randomized queries. The target folder was considered to be the folder(s) of the query's original target file(s).

Fig 2.7 shows the results for the first case. PFH had a consistently higher MRR for categorizing the file and guessing the most suitable directory with all three approaches. The hub approach was steady; since it does not multiply structure by $1 - \alpha$ it does not converge to Lucene. The transformation approach peaked at α=0.47, likely due to the high P@3 and MRR at that point. Peaking at 0.66-0.92, the combined approach was best in the 0.75-0.83 region suggested by Exp A and Exp B.

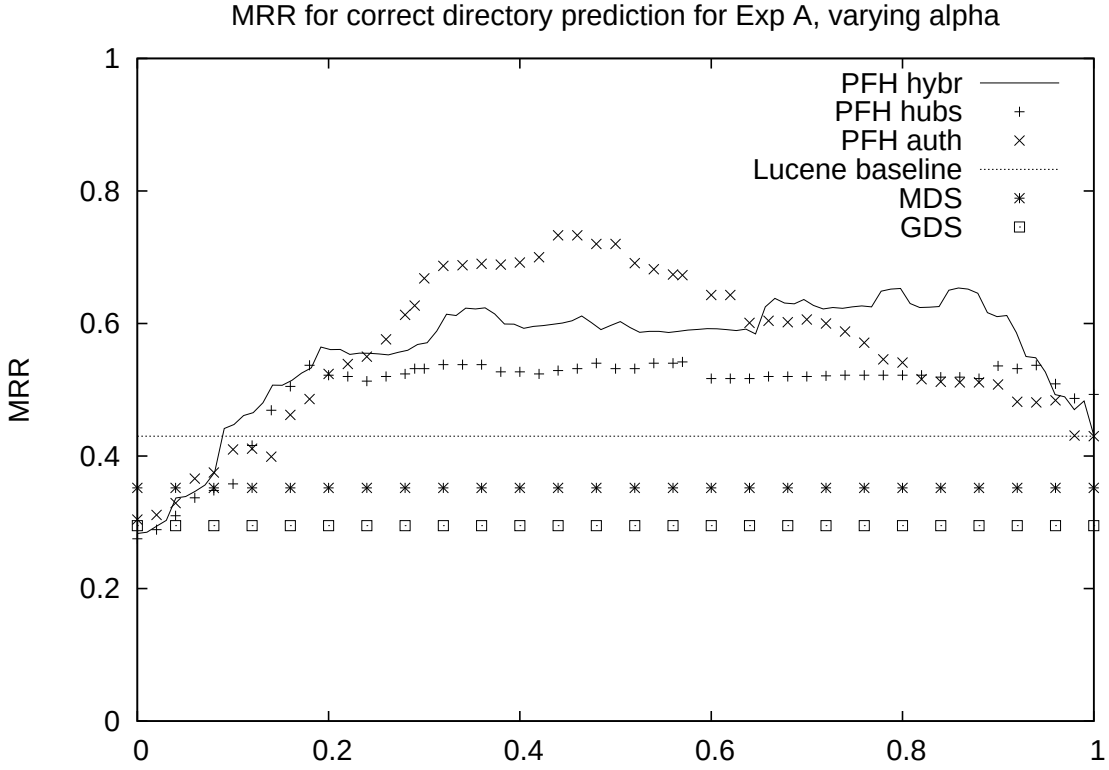

Figure 2.7: MRR for Exp C's categorization. The PFH approaches predicted the correct folders for files more often than the naive approaches.

Fig 2.8 shows the results for the second case, with graphs for each query bracket. The hub approach yielded poor results because of the meaningless organization. However, the hybrid approach performed well and peaked at $\alpha \approx 0.84$.

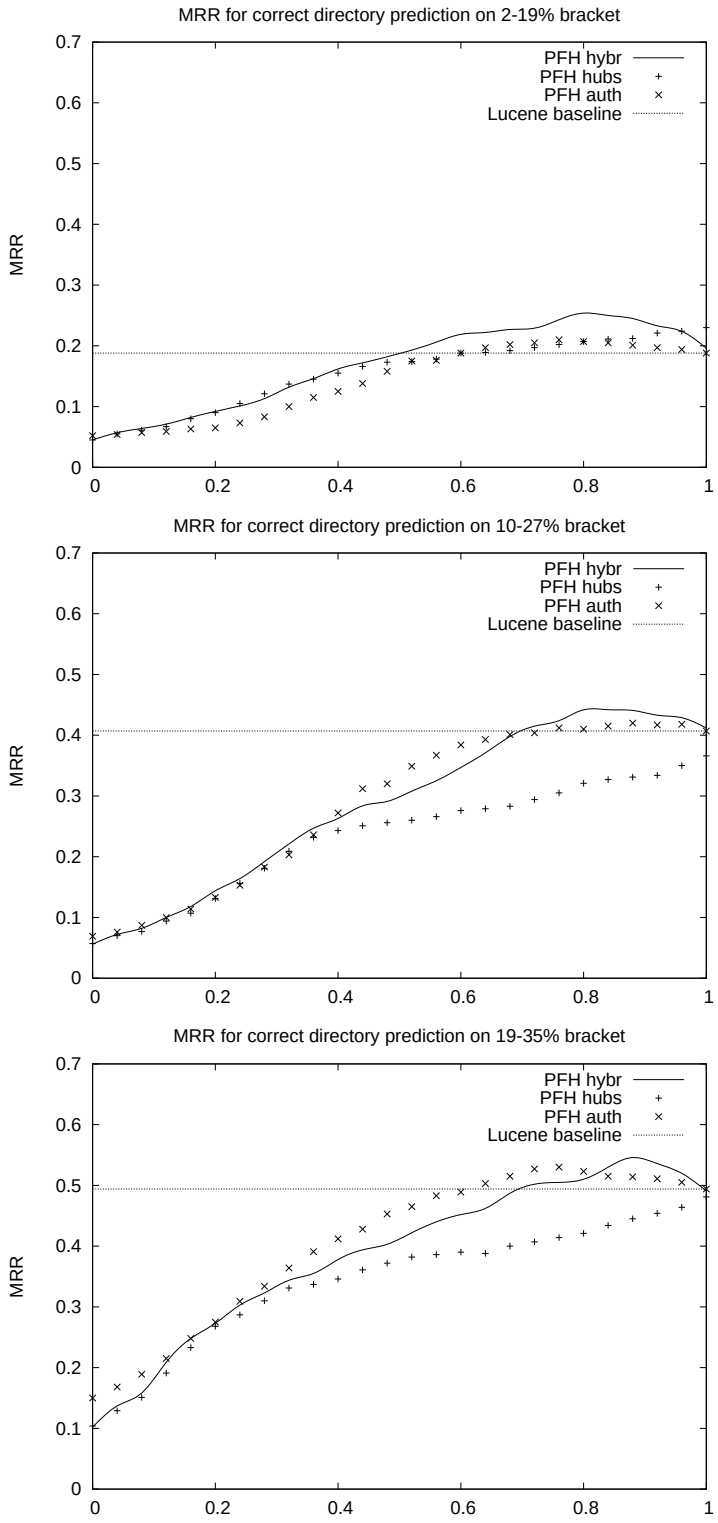

Figure 2.8: MRR for Exp C's file categorization using the three query brackets from Exp B (Section 2.4.2). Although this was a bad-case scenario designed to simulate a poor organization of files, the PFH hybrid approach performed above the baseline for each bracket when α=0.84 $\pm$ 0.15.

2.5 Summary

In the personal space of the desktop, search intent is often of a navigational nature (known-item search). Given that hierarchical content is usually organized with some 'meaning' behind the structure, we can leverage this information to improve known-item search. The chapter presents a novel algorithm, PFH, for rearranging content-only search results from a hierarchical corpus. PFH calculates a structural score for the search results and then rearranges them using their structure and content scores. How much consideration of structure to take is guided by an input α parameter, which is normalized such that α=0.5 represents a half-way split and equal consideration of structure and content. When $\alpha < 1$, the structure score begins to influence the results and PFH will favor items that are in relevant neighborhoods while penalizing items that are isolated. This idea is useful for hierarchies with a meaningful organization, such as files on a desktop, books and technical documentation.

PFH is formulated from the same simple principles as the well-known HITS algorithm, and used the α parameter to logically separate "structure" and "content" components to observe how varying the balance affected MRR, Precision and MAP. Three experiments on search in a structured corpus, search in a randomly organized corpus and categorizing files into folders showed that a small sway towards structure ($\approx$20%, with $\approx$80% content) performed best: PFH was as-good-as the content-only baseline in terms of P@n and MAP, but was significantly better under MRR. This indicates an improvement in known-item search accuracy.

The optimal α value is likely to be domain-specific. For some domains it may be close to 1, while for others it may be lower. However, we argue that an optimal α should be empirically determinable for any corpus if there exists a set of representative queries and relevance judgments. A sophisticated search system should be able to learn to vary α based on the nature and selectivity of the query, whereas some static corpora—such as web-based APIs—can have a single α determined empirically beforehand.

Overall, this chapter shows that a small consideration of structure is useful for hier-

archical corpora. The PFH algorithm provides this facility in a flexible manner because it is time and space efficient, and can be used on top of the search results of existing DS tools.

Rearranging the order of searched items can be useful in other domains. Chapter 3 proposes a rearrangement method for on-demand hyperlink clustering in another personal space: a web page currently being viewed in a mobile browser. In relation to this chapter, Chapter 3 also employs a structural propagation step, using the DOM, to leverage some organizational information about the hierarchy.

Chapter 3

Web navigation on constrained devices

3.1 Introduction

Hyperlinks, also known as anchor elements or anchor tags, are the primary mode of navigation on the web and are used by all web users. The average number of links on a webpage is 10 [109], although this figure is for the whole web and includes low level pages. The figure is likely higher for top-level domain, portal and index pages. To use some familiar examples, the Google homepage has 30 links, Microsoft Research has 70, DBLP has 90, Yahoo has 170, CNN has 170 and Digg has 250.

Most websites focus on presentation as opposed to meeting accessibility requirements, but there is an increasing awareness among web authors to create websites suitable for viewing on multiple browsers. Ease of use is important for the mobile web because of the limited display area. There are also groups of users such as the vision-impaired who have high accessibility needs.

This chapter presents an on-demand hyperlink clustering method for webpages viewed in a mobile browser. 'On-demand' means that the procedure is to be invoked by the user, when required, or by the web browser itself, when appropriate. The approach is intended

to be suitable for client-side implementation on a mobile browser. Several applications of the approach, each of which relies on being able to group links on a page into logical clusters, are identified: improved accessibility and navigation for vision-impaired users, assisting web authors with creating portal pages for mobiles (or simply making suggestions on how they may reorganize an existing page layout), and section-relevant advertising on mobile-friendly news portals. These three applications are described next.

Accessibility for vision-impaired

Vision-impaired users typically use screen readers to navigate webpages. Screen readers are software that parse pages serially and speak out the content using audio. When visiting a webpage, a web user is generally interested in either seeing the content on the present page or on finding a link to continue to another page. Vision-impaired users have a more difficult time with both cases, but this chapter is interested in the latter case. Consider the scenario of a blind user looking for a link. If she is unfamiliar with the page layout, or if the page content has a dynamic nature, then she can locate the link by serially scanning through each link in sequence and hearing the audio of their anchor text. This is a time consuming task, especially for high-level index pages with hundreds of links. Clustering the links into related groups can aid her in finding her desired link: rather than go through them in sequence, she can listen to the link clusters and skip entire clusters of unwanted links.

The concept of navigating via clusters is similar to *Interactive Voice Response* (IVR), the automatic telephone operator that presents a caller with a menu of choices and detects the user dialing to navigate this menu. Rather than have all options presented in a list, IVR presents a tree of options. Likewise, a small set of link clusters gives the user a more compact representation of the links on the page. If a cluster of links is not what a user is looking for, they can quickly ignore it and inspect the next cluster.

An additional problem for visually-impaired users is websites that use pop-up menus, which can confuse screen readers. The link clustering method can be beneficial here as

well because it will first extract all the links, cluster them, and is then able to rearrange the page such that the clusters are placed on the very top and read by the screen reader immediately. If the clusters are labeled with a description, the user can navigate them similar to IVR.

Layout organization and portals

Link clustering creates groups that are perceived to contain related links. This means that two links may be placed into the same group even if they are far apart in the visual layout. A web author can use the proposed method to view the groupings, which may give them suggestions on how to improve their layout.

Mobile portals refer to clean interfaces that replace the cluttered desktop versions of pages, providing a small set of key links[1]. Clustering can assist web authors in creating such pages by helping them decide which links from the cluttered desktop version should be represented on the mobile-friendly portal. For instance, a complex university homepage may have 100 links, a few of which are about undergraduate study. If these particular links are grouped together, the web author may decide that the portal should contain a link "undergraduates" that points to a separate page with just the undergraduate links. The creation of this intermediate page reduces the complexity of the portal while also organizing the links in a logical manner.

News advertising

Much of the web is funded by advertising. Common ad forms include pop-ups, pop-unders, banners, skyscraper ads, text ads, video-embedded ads and interstitials. This chapter considers "section ads" for mobile news sites. The concept refers to ads that will be placed alongside a "section" of the page, such that the ad and the section are somehow related. A typical mobile news page, such as Fig 3.1, maybe have several "sections" and therefore several ads.

[1]E.g. m.cnn.com, m.facebook.com, m.flickr.com, m.nytimes.com, m.youtube.com, etc.

Section ads are useful because the user will be reading the page content and have a targeted ad relevant to that particular part of the content visible on the screen. As such, the ad may be relevant to only a small section of the page and yet not relevant for any other sections. Mobile news portals cover many topics and thus are susceptible to clustering or segmentation.

A typical news site may already have pre-defined sections, such as “world news”, “local news”, “business”, and so on. However, these topics are quite broad and not necessarily useful for matching ads. For instance, the section name “local news” is not useful for matching against an ad. Instead, the headlines themselves need to be checked. Because news headlines are dynamic and change many times per day, news pages are somewhat poor candidates for online contextual advertising (a process that involves categorizing a page).

The motivation of using the link clustering method for news is twofold. First, we observe that a mobile news portal page has many links (headlines) that cover many topics, and therefore certain parts of the page are relevant to a certain topic while other page of the page are not relevant to that topic. Second, we observe that news portals have a highly dynamic nature and therefore an automatic way of determining the sections will enable such advertising. The link clustering method, combined with topic classifier, is suitable for this purpose.

Link clustering

The three applications identified above are only loosely related in terms of their use cases, but they share the same initial labeling, propagation and clustering steps to identify the link groups. This chapter provides sample experiments of the link groups on several well known websites, and is organized as follows. First, Section 3.2 describes the details of the approach, such as how the links are labeled and weighed, how these weights are distributed by exploiting the document structure, and how they are clustered. The section also gives some background on news advertising, which requires an extra step (Section

3.2.5). Section 3.3 then shows the effectiveness of the method on a number of well-known websites. Section 3.4 gives a summary and some conclusions.

3.2 Approach

There are two types of anchor elements. The first type is a regular hyperlink with a *href* attribute that causes the browser to load a page when it is clicked. The second type does not link to a page and is either a section marker with a *name* attribute or an arbitrary javascript action. Because we are interested in navigation, links without a *href* are ignored. The chapter also uses the terms 'anchor' and 'link' interchangeably to follow conventional expressions (e.g. anchor text), although both describe the same element.

The approach begins by first extracting a page's links and labeling them with keywords intended to describe them (Section 3.2.1). Keyword influence is subsequently propagated to nearby links as a means of preserving structural context (Section 3.2.2). The vectors are then clustered (Section 3.2.3).

3.2.1 Anchor representation

Anchor text

The text inside the markup tag, called the anchor text, is the most important descriptor of a link. This is because it is the text which the user sees and clicks. Modern search engines place significant weight on this text for link analysis purposes [19]. For example, a tag <*a href=...>register new account</a*> describes itself as a link to where the user can sign up and create a login.

An anchor element generally has a single child node in the DOM tree, that of the text node containing its anchor text. Sometimes other nested child nodes may be included, such as presentation markup like a <*br*> element or a <*span*> element. These rare cases are handled by letting the anchor text be any text extracted from the nested children. Images also require special handling because an <img> tag may entirely replace the text

node. In this case, if the image has an *alt* attribute then the value of that attribute is used as the anchor text.

Destination's URL

The URL that the link points to is another source of keyword information. For example, a link *...* appears to describe a page about new login accounts. The name of the destination page and any subfolder structure provides some extra keywords to describe the link with. Likewise, if the anchor holds an *<img>* tag, the *src* attribute serves as a second URL.

Destination's content

The content of the page being pointed to is a rich source of information to describe its link. For example, the anchor text '*register new account*' coupled with the destination '*newaccount.php*' give a reasonable idea of what to expect, but inspecting the destination page itself would provide a better summary. Unfortunately the content of the destination is not readily available in most cases and is prohibitively expensive to download and summarize in real-time. One way to obtain a description of the destination is for the web author to provide a few keywords as part of the *title* attribute in the anchor element[2]. In general anchor text is short, so having these extra keywords to make the link description longer is very helpful. Two other possibilities for describing the destination content are to use the anchor text window or DOM cues. The first refers to non-anchor words surrounding the anchor. The second refers to moving up the DOM tree and looking for elements known to be descriptive of their descendent nodes, such as an *<h2>* heading or a *<table>* "summary" attribute. The experiments in Section 3.3 assume that anchors have a "title" attribute that contains the top thirty TFIDF words from the destination.

[2]This attribute is part of W3C's HTML 4 spec.

Label weights

After labeling an anchor with its set of keywords extracted from the HTML, the words are weighted. Weights are necessary so that links can influence each other in the propagation step. The weights are initially set to 1.0 unit for any terms extracted from the anchor text and 0.5 for any other terms. Terms in the title attribute that describe the destination page are assigned their normalized TF as their weight. All term weights as then multiplied by IDF from a global web corpus unrelated to the page being inspected. Finally, the weights are L_∞-normalized such that the top label for any link has a weight of 1 unit.

3.2.2 Propagation step

The keywords describe a link's perceived *content*, but there is an additional source of information on the page: the DOM tree's *structure*. The purpose of a link can be made clearer by its neighborhood context. For example, a "forgotten your password?" link is often next to a "register new account" link; in cases where these two pages have completely different content, a structural consideration will allow them to be considered as related to each other.

Websites are authored by people and read by people, so their links are commonly arranged in some meaningful manner. Links that are closer together tend to be more related than links that are far apart. In many ways this is helpful for determining the clusters because sometimes the links are already grouped together, but there is still a need to somehow extract this information in an automated matter. A consideration of structure preserves some of the existing organization. It is achieved using a propagation step that propagates the influence of a link's keywords onto every other link in the DOM. A decay function is used so that distant links influence each other weakly. For instance, the "register new account" link may have labels such as {register, new, account} and it will borrow {forgotten, your, password} from its neighbor link with a reasonably high weight. Other links will also borrow the same keywords for their own description, but with a lower weight depending on the decay function.

The propagation precedure is a simple one-pass loop: for each pair of links (a,b), add each keyword from a's feature vector to b's feature vector, but multiply by the value of the distance decay from a to b, and then do the reverse with b and a. At the end of the propagation loop, the new more-populated feature vectors will replace the original sparse vectors. The process gives each link a $|K|$-dimensional feature vector, where $|K|$ is the number of keywords used to label all the links.

The decay function used is $\frac{1}{(1+\delta)^2}$. There are other possible choices for a decay function, but the inverse-square law is reasonable as it harshly penalizes large distances. The value of δ is the distance between two links. Supposing that link positions in the DOM tree are represented using Dewey encoding [128], then δ is calculated as follows:

$\delta(a, b)$: distance between links a and b.
1. **if** $b == a$ **then return** 0.
2. **let** aid be a's Dewey stack, of depth $|aid|$.
3. **let** bid be b's Dewey stack, of depth $|bid|$.
4. $last := \min(|aid|, |bid|) - 1$
5. **for** i from 0 to $last$ **loop**
6. **break if** $aid[i] \neq bid[i]$
7. **end loop**
8. **return** $2(|aid| - 1 - i) \; + \; \sqrt{|aid[i] - bid[i]|} \; + \; (|bid| - 1 - i)$

This simple function finds at which level in the tree a and b's branches deviate and returns a sum of three components: the number of levels 'up' from a to this position, an 'across' component of how many sibling nodes to jump over to reach b's branch, and the number of levels 'down' that branch to reach b.

Note that 'influence' is related to the concept of inheritance. However, inheritance is a parent-child unidirectional relationship that should occur from a higher tree level to a lower tree level. This is clearly not useful in this case because links are already leaf nodes in the tree and are not children of each other. Hence, for anchors to influence one another we must allow traveling 'up' the tree. This is contrary to traditional inheritance and therefore line 8 makes it more expensive (i.e. more distance) to travel up the tree. The 'across' component comes into affect when a and b's branches are part of sequential list of elements. In such cases there will be many links with equal 'up' and 'down' components

that give each other the same influence, therefore an 'across' component can differentiate between closer siblings and farther siblings. In practice this is actually very common, with the most obvious example being links embedded within paragraphs of text. The 'across' component allows us to decay influence by a larger amount when there are more sibling branches separating the branches of a and b (typically this indicates a larger physical distance in the page layout). For example, the distance between two anchors with Dewey IDs 1.2.1.3.1.4 and 1.2.1.7.2 would be 7. Similarly, $\delta(1.1, 1.50)$ and $\delta(1.3.1.1, 1.4.1.1)$ are also 7. Note that δ is not a symmetric function due to the 'up' component being doubly penalized.

3.2.3 Cluster step

Suppose that K is the set of all keywords used to label all anchors A on a page. The propagation step ensures that there is a $|K|$ dimensional, non-zero vector for each $a \in A$. The dimensions with the highest values will be the keywords most specific to a, but other dimensions may receive modest boosts due to the structural locality.

The clustering step itself uses an unweighted k-means, plugging in these feature vectors as input. The parameters used for k-means were to use the cluster's mean as its centroid and to use cosine similarity to determine which links were similar.

K-means requires the number of desired output clusters as input. Creating too few or too many clusters would not be useful and therefore we set the desired number of clusters for a set of anchors A to $\min\left(\lceil\frac{|A|}{10}\rceil, 12\right)$. This choice is largely arbitrary but its justification is that an average of 10 links per cluster is suitable and that there should not be more than a dozen clusters due to cognitive overload. Input pages that produce a very large cluster can have such clusters broken down by running the algorithm a second time using only the large cluster, although the experiments do not do this.

Because k-means seeds the clusters randomly, the output depends on the initial configuration and results may differ between runs. For this reason it is desirable to run the clustering step several times and to choose the best output (the experiments run it 20

times). In an automatic setting and without supervision, deciding which k-means run was the 'best' run is simply a matter of saving the run with the smallest mean intra-cluster distance.

3.2.4 Expected cost of navigation

While it is simple and fast, one drawback of k-means is that sometimes it is difficult to label a cluster accurately. In two of the three identified applications of the proposed method, a label is actually not needed. However, visually-impaired users who use the method for web navigation would prefer to have labels.

We can now analyze the expected cost of locating a link, under the assumption that the position of the wanted link is uniformly distributed in the page. Serial scanning over n links has an expected search cost of $\frac{n}{2}$, i.e. the user will on average need to listen to half the number of links on the page. With link clustering, the expected cost is lower both with and without labels: if n links are split into c clusters (where typically $c \leq \frac{n}{10}$, as in Section 3.2.3) then the expected cost of going through labeled clusters is $\frac{c}{2} + \frac{n}{2c}$. That is, the total cost is the combined cost of first hearing the labels being read out to find the right cluster and then finding the link within the cluster. If n=100 and c=10, this equates to 5 labels and 5 anchors, or 10 audio phrases. If the clusters are unlabeled, and if we assume that the user must hear three links in a cluster to decide if the cluster sounds relevant or not, then the expected cost is $3 \times \frac{c}{2} + \frac{n}{2c}$, or a total of 20 audio phrases. Both link clustering methods have a lower cost than the $\frac{n}{2}$=50 for scanning in serial with the screen reader. Ordering the clusters such that the clusters with the more prominent links are positions first will further reduce the cost[3].

3.2.5 News advertising

Online ads are typically HTTP objects that are downloaded separately by the browser and rendered together with the page. Ads are often targeted to the content, e.g. a sports

[3]Likewise, in IVR the menu choices that the caller is most likely to choose are spoken first.

site is more likely to show an ad for gym equipment than an ad for books. The dynamic nature of news sites makes this difficult because there is no fixed or clear category for the page and the headlines keep changing. Consider the mobile version of the New York Times site in Fig 3.1. It has 70 links, the first 40 being headlines and the other 30 being template links to other parts of the NYT domain. The layout is a column of links, and there are also a few images and sentences.

Now consider Fig 3.2, the desktop browser version of the third headline in Fig 3.1. The page is more cluttered as it designed for large screens. There are four Google's AdSense ads shown on the page, but they are not very relevant to the article; the article

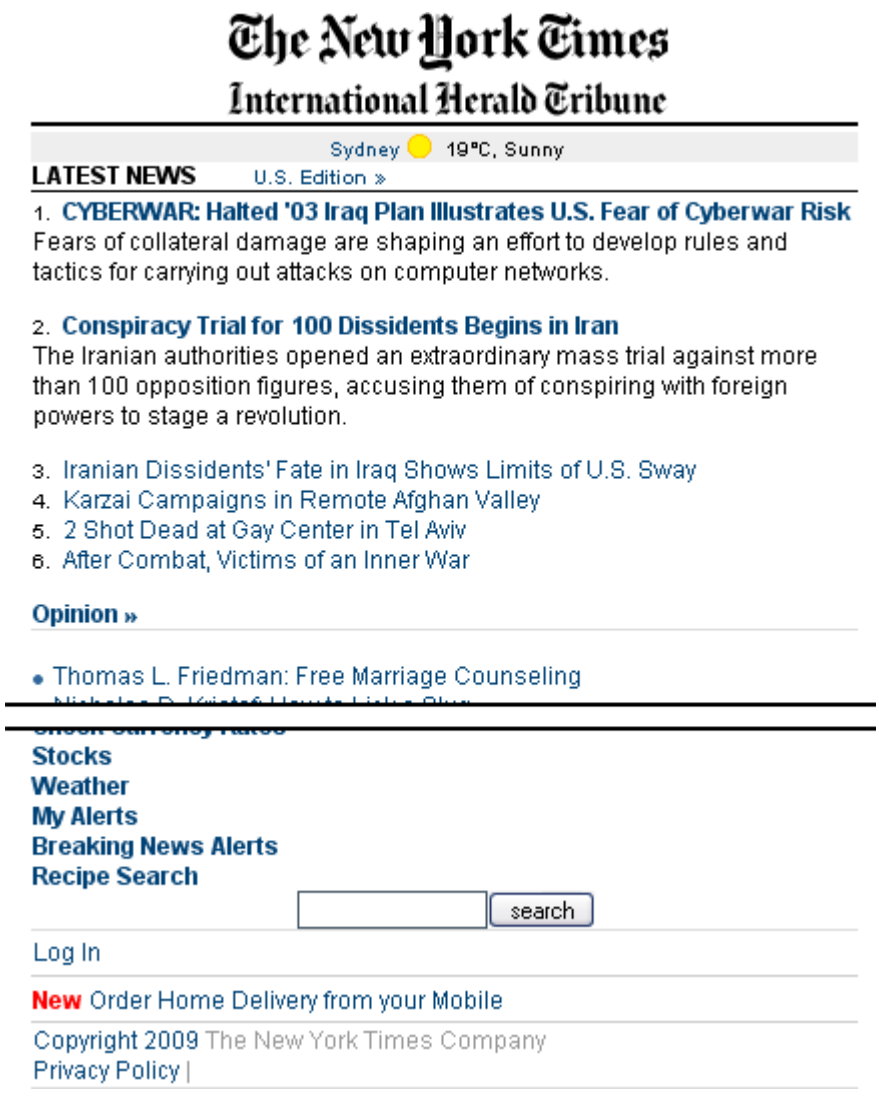

Figure 3.1: The New York Times site (mobile version), partially cut to fit.

is about US–Middle East politics while the ads are about solar panels, basketball videos, a bookshop and information about the coffee industry. This chapter is interested in advertising on the mobile index page (Fig 3.1).

Mobile news portals have some common features that allow a few assumptions to be made to simplify and speed up the processing pipeline. First, the vertical column-like layout means that there are few horizontal divisions such that if the text is left-justified,

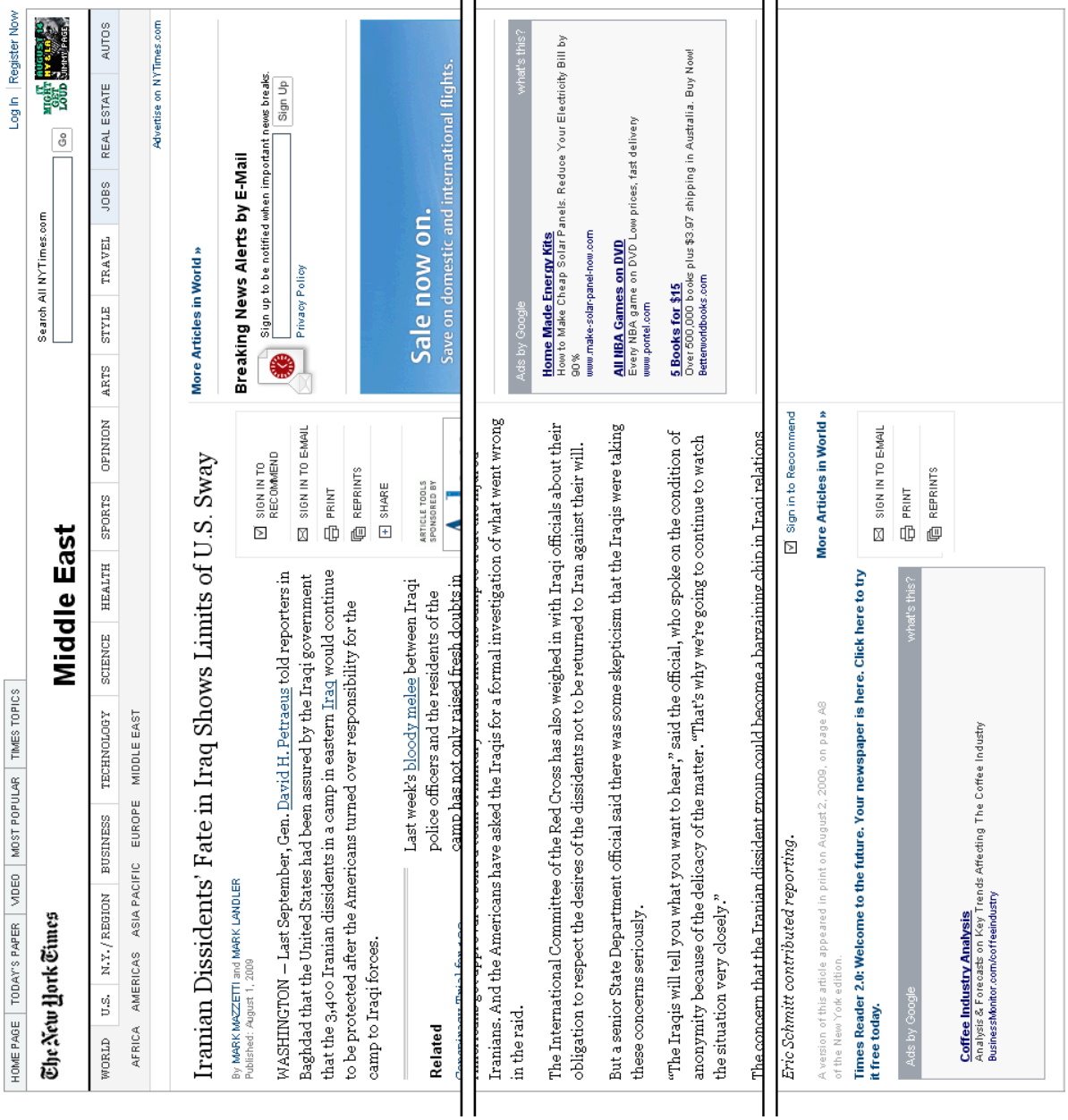

Figure 3.2: An article on the desktop version of the NYT site. Two of the slices show AdSense ads (middle-right, bottom-left).

then some space may be available to float an ad image near it. Second, the sequential presentation of headline anchors means that the link depth in the DOM is very shallow. If we assume that the tree is flat, then the propagation step can be simplified because Dewey IDs no longer need to be assigned or compared[4]. The distance calculation then becomes trivial. Third, most of the visible HTML elements on the page are links. Albeit sparse, their anchor texts are fairly descriptive. Therefore extra keyword information about the destination page is not required, and instead we can perform all the link clustering work using only the anchor text and line numbers.

The process of determining which ads to show is as follows. First, the column layout is segmented to determine appropriate section boundaries. The link clustering approach can be used for this purpose. Although the headline anchor texts are short and sparse and the clustering step would have difficulty in forming logical clusters, it is the propagation step that can maintain cohesive groupings. If desired, however, we can optionally rearrange any links that are considered to be in the wrong cluster. Once the sections are determined, a classifier is used to classify the section into one or more dominant topics. The classification categories of the news headlines would then be matched against a database of ads that have also been similarly classified into the same categories, and an ad from the appropriate category can be chosen by whatever means or features are appropriate (e.g. billing, size, shape, relevance, etc.). The ad selection is out of scope of this chapter, and we are only concerned with determining the topics that the ad should match.

Headline classifier

A simple Bayes classifier was built to classify the news headlines. The categories are those from Phan et al. [107], who used a latent topic analysis model [13] to extract 200 salient topics from Wikipedia. A few of the topics are partially shown in Table 3.1. A learning step is not needed for this classifier because the features are already known. Only an evaluation step is needed, and it is shown below.

[4]A flat list of links would be interpreted has having Dewey IDs of $\{1 \ \ldots \ n\}$.

nbc(W): classify headlines
Input: W, set of anchor text words.
1. tokenize and stem W.
2. remove tokens from W that do not describe any known class.
3. **for** each class c:
4. $p_c := \text{nmissed} := 0$
5. **for** each word w in W:
6. **if** $P(c|w) > 0$ **then** $p_c := p_c - \log P(c|w)$
7. **else** $nmissed{+}{+}$; $p_c := p_c - \log .0001$
8. **endif**
9. **endfor**
10. make note of p_c if $nmissed < |W|$.
11. **endfor**
12. **return** $\langle c, e^{-p_c/|W|} \rangle$ for the highest seen p_c.

The Naive Bayes classifier above is suitable for short input such as news headlines. It returns the most dominant class c and its score. The 'naive' assumption in the Bayes model is word independence, thus the probabilities of the words decribing the class are multiplied together. In this case, logarithms are used because the true probabilities are small and floating point precision is not always able to represent the value. As is typical of the Bayes model, each class is assigned an initial probability of 1.0 and, for each input word, the probability is reduced by a small amount if the word describes the class and by a large amount if it does not. A small non-zero quantity (0.0001) is used for words that do not describe the class, otherwise the product would multiply by zero.

For this particular problem it is sensible to divide the p_c value by the number of input words rather than the number of classes so that short inputs can be classified with high confidence. For example, the input $W = \{nasdaq\}$ will produce an output of *topic107* with a p_c score of 1.0, suggesting a virtual certainty that W belongs to topic107 (which happens to be the only class with "nasdaq" as a feature). In advertising it is sometimes desirable not to show any ad [21], therefore returning the score can be used to decide how confident the classification is. Additionally, the top-k classes can be returned instead of only the top-1, and the relative differences in scores can help make the choice of ad more accurate. The experiments in the next section show the top k=3 topics for each link cluster. The reason for aggregating the headlines into clusters and classifying

the whole cluster rather than each headline is described in Section 1.3; it is to reduce misclassifications and to support ambiguous headlines with multiple class instances.

Class	Features
topic006	israel israeli arab palestine war gaza strip peace government conflict military iraq refugee political ...
topic007	people park tourism public travel hotel activities visitors home outdoor events tickets entertainment...
topic008	population rate people income living poverty growth world countries developed housing social household...
topic009	century modern history period ancient developed medieval classical influence renaissance empire latin culture...
topic020	party election vote candidate majority political democratic opposition coalition government conservative...
topic023	building construction wall architecture house site design style architect roof interior garden temple...
topic033	insurance debt risk loss loan policy payment cost financial period mortgage premium coverage assets...
topic056	women sex male female gender homosexual behavior feminist orientation intercourse identity relationships...
topic060	radio television tv stations broadcast channel news network media program fm commercial digital coverage...
topic063	political movement social rights democracy liberal freedom society economic power conservative ideology...
topic066	american award actor career academy school actress film star magazine fame director movie relationship...
topic068	iraq attack united suicide terrorism government afghanistan security military world saddam bombing...
topic073	internet online users site com content community web website information software media personal...
topic074	disorder mental therapy depression treatment people symptoms social anxiety health personality stress...
topic094	security mail address email computer information spam user identity privacy email secure protection...
topic104	software windows file microsoft operating version user files os applications linux mac code interface...
topic107	market price stock value exchange trading sell options buy spread index risk futures asset financial...
topic124	media news magazine press newspaper journal publication article review editor paper editorial information...
topic125	fire safety emergency disaster accident risk cause protection rescue accidents crash warning life help...
topic129	vitamin diet food body fat diabetes weight blood health disease dietary protein cholesterol loss liver...
topic137	bank money account credit financial reserve balance savings deposit loans transactions business branch...
topic151	mobile phone service call network wireless line internet technology user office standard broadband modem...
topic159	league team game season baseball football players record stadium field sports coach championship fans...
topic161	hitler jews nazi war germany holocaust camps world fascism propaganda prisoners political death regime...
topic168	city town street capital airport home university museum park public development cultural industrial...
topic172	dragon elves creatures battle king quest magic dwarves elf mountains warriors demons monsters base...
topic177	report intelligence investigation official alleged announced statement revealed conspiracy charges...
topic182	science fiction star doctor space alien technology fan worlds fictional story novel galaxy television...
topic184	law court legal civil judge party defendant rule trial action criminal justice damages supreme act...
topic196	company products market business sales sold advertising brand industry consumer customers inc logo...
topic198	president united bush congress american war national administration campaign washington senate clinton...

Table 3.1: Partial feature vectors of the classes referenced in the experiments (Fig 3.6, Fig 3.7). These classes are the topics assigned by the Section 3.2.5 classifier. The full vectors have 200 dimensions each.

3.3 Evaluation

This section show the results for a number of well-known webpages that typify the described applications.

3.3.1 Navigation and organization

Centrelink homepage

Centrelink[5] is a statutory agency that delivers various Human Services to the Australian public on behalf of the government. The agency's site has four main sections. Fig 3.3 shows the "About Payments" tab with its menu extended. The page has 111 links in total at an average DOM tree depth of 12.8 (med. 13, $\sigma = 1.4$). The user navigation of this page relies heavily on the 3-tier pop-up menu.

A total of $\lceil \frac{111}{10} \rceil = 12$ clusters were create with the proposed method. Fig 3.3 shows some of these clusters in a color-coded manner: pink links about the agency (top), yellow links about the agency's website (bottom), blue image-links for the four main tab sections, green links for a "children"-related cluster in the menu, and a further 8 clusters are hidden within the pop-up menu. The full breakdown is in Table 3.2.

If a vision-impaired user was looking for welfare allowance about "having a baby", then they can intuitively find this link in the children cluster. The results appear quite useful because the approach is able to unite logically related links from a complex page layout in a fully automatic manner. The most notable problem with the cluster output in Table 3.2 is that Cluster 10 is much too large and mixes several topics that perhaps should be further subdivided. Setting the number of output clusters to 13 or 14 can achieve this, but it is difficult to know the optimal number of clusters beforehand.

[5] http://www.centrelink.gov.au

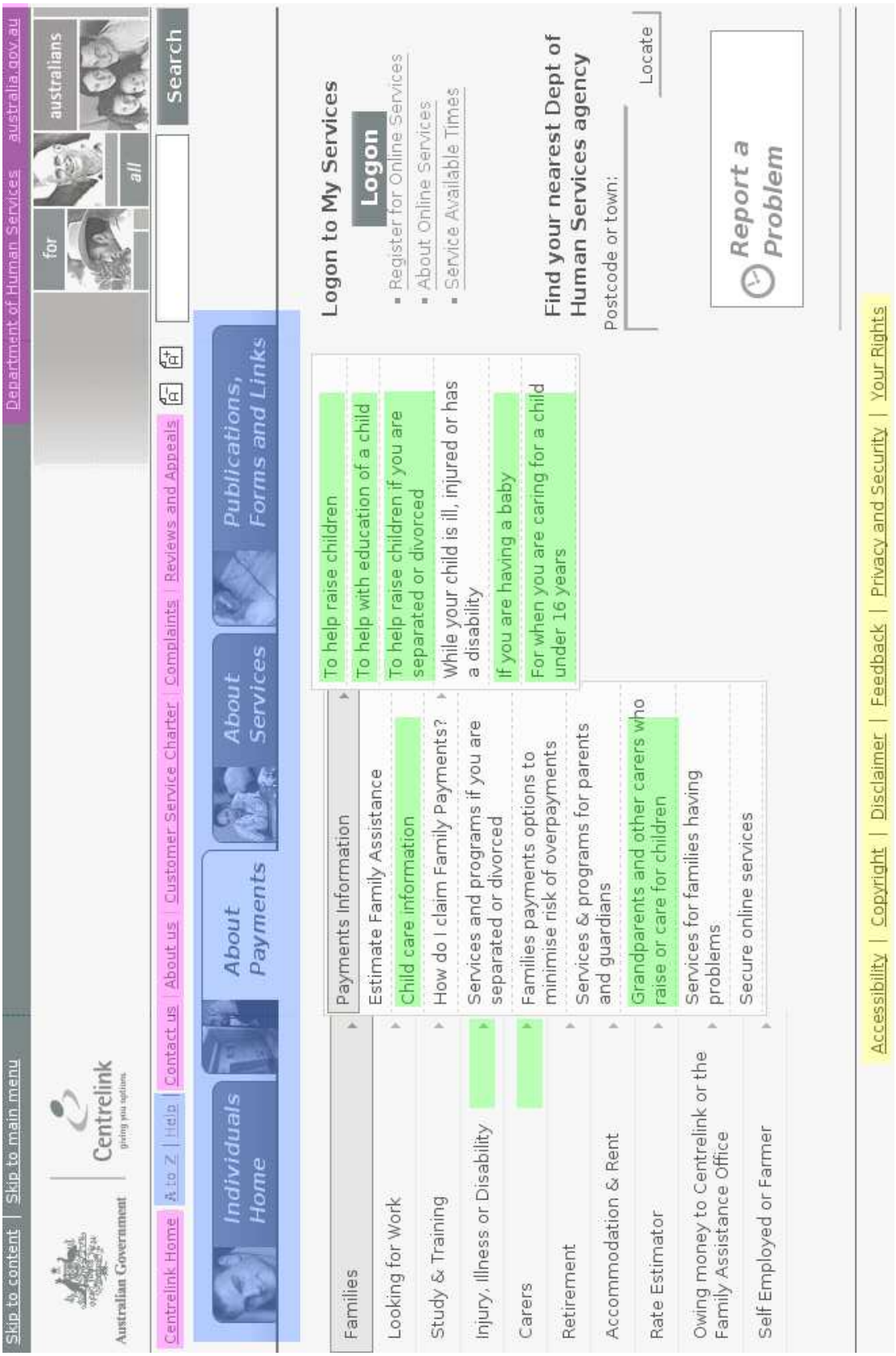

Figure 3.3: The Centrelink site, which has 111 links and a complex pop-up menu. The color-coding shows a few of the clusters (other clusters are hidden in the pop-up menu).

Anchors' Dewey IDs	Anchors' text
Cluster 1: 8 links about the Centrelink agency	
1.1.2.3.1.1.1.1.7.1.[1-2].1 1.1.2.3.1.1.1.2.1.[1,4-8].1	"*australia.gov.au*", "*department of human services*", "*centrelink home*", "*about us*", "*customer service charter*", "*contact us*", "*complaints*", "*reviews and appeals*"
Cluster 2: 6 links about the Centrelink website	
1.1.2.3.1.1.3.1.1.[2-7].1	"*accessibility*", "*copyright*", "*disclaimer*", "*feedback*", "*privacy and security*", "*your rights*"
Cluster 3: 6 links leading to some of the main sections of the site	
1.1.2.3.1.1.1.2.1.[2-3].1 1.1.2.3.1.1.1.3.2.[1-4].1.1	"*a to z*", "*help*", "*individuals home*", "*about payments*", "*about services*", "*publications, forms and links*"
Cluster 4: 7 links about types of welfare allowances	
1.1.2.3.1.1.2.4.2.2.2.[1-7].1	"*sickness allowance*", "*newstart allowance*", "*youth allowance*", "*youth disability supplement*", "*mobility allowance*", "*pensioner education supplement*", "*disability support pension*"
Cluster 5: 7 links about carers	
1.1.2.3.1.1.2.3.1.1.2.4.2.6.1 1.1.2.3.1.1.2.3.1.1.2.8.1 1.1.2.3.1.1.2.3.1.2.2.6.2.4.1 1.1.2.3.1.1.2.3.1.5.2.[5-8].1	"*carer allowance*", "*grandparents and other carers who raise or care for children*", "*returning to work when you stop being a carer*", "*information and advice for carers*", "*grandparents or other carers who raise or care for children*", "*services and programs for carers*", "*returning to work when you stop being a carer*"
Cluster 6: 10 links about raising children	
1.1.2.3.1.1.2.3.1.1.2.1.2.[1,3,5].1 1.1.2.3.1.1.2.3.1.1.2.[5,7].1 1.1.2.3.1.1.2.3.1.1.2.1.2.[2,6].1 1.1.2.3.1.1.2.3.1.[1,5].2.3.1 1.1.2.3.1.1.2.3.1.1.2.4.2.2.1	"*to help raise children*", "*to help raise children if you are separated or divorced*", "*if you are having a baby*", "*services and programs if you are separated or divorced*", "*services and programs for parents and guardians*", "*to help with education of a child*", "*for when you are caring for a child under 16 years*", "*child care information*", "*child care benefit*", "*payments for when you are caring for a child*"
Cluster 7: 9 links about family	
1.1.2.3.1.1.2.3.1.1.2.[2,6,9].1 1.1.2.3.1.1.2.3.1.1.2.4.2.1.1 1.1.2.3.1.1.2.3.1.8.2.1.1 1.1.2.3.1.1.2.3.1.9.2.[1-4].1	"*estimate family assistance*", "*family tax benefit*", "*families payments options to minimise risk of overpayments*", "*services for families having problems*", "*centrelink/family assistance rate estimator*", "*owing money to centrelink or the family assistance office*", "*compensation recovery teams*", "*centrepay*", "*more choice for families*"
Cluster 8: 13 links about study and training	
1.1.2.3.1.1.2.3.1.2.2.5.2.[1-4].1 1.1.2.3.1.1.2.3.1.2.2.6.2.2.1 1.1.2.3.1.1.2.3.1.3.2.[1-8].1	"*training and development courses*", "*training and development courses*", "*courses to help you develop your business*", "*courses to help you develop your business*", "*services and programs for studying and training*", "*services and programs for studying and training*", "*moving from study or training to work*", "*moving from study or training to work*", "*overcoming barriers to study or working*", "*overcoming barriers to study and work*", "*payments while you are studying and training*", "*claim forms while you are studying and training*", "*information for school leavers*"
Cluster 9: 10 links about health problems	
1.1.2.3.1.1.2.3.1.1.2.1.2.4.1 1.1.2.3.1.1.2.3.1.2.2.6.2.3.1 1.1.2.3.1.1.2.3.1.4.2.[1-2,4-7].1 1.1.2.3.1.1.2.3.1.5.2.[1,4].1	"*working with an illness, injury or disability*", "*working with an injury, illness or disability*", "*payments if you are ill, injured or have a disability*", "*claim forms if you are ill, injured or have a disability*", "*services and programs for people with an illness, injury or disability*", "*while your child is ill, injured or has a disability*", "*payments while your child is ill, injured or has a disability*", "*payments while your child is ill, injured or has a disability*", "*someone to deal with centrelink for you*", "*claim forms if you are caring for frail, aged, ill or disabled persons*"
Cluster 10: 22 links about maternity, job search, retirement	
1.1.2.3.1.1.2.3.1.1.2.4.1 1.1.2.3.1.1.2.3.1.1.2.4.2.[3-5,7].1 1.1.2.3.1.1.2.3.1.2.2.[1-4,7-8].1 1.1.2.3.1.1.2.3.1.2.2.6.2.1.1 1.1.2.3.1.1.2.3.1.5.2.2.1 1.1.2.3.1.1.2.3.1.6.2.[1-6].1 1.1.2.3.1.1.2.3.1.7.2.[1-3].1	"*maternity payment*", "*maternity immunisation allowance*", "*how do i claim family payments?*", "*parenting payment*", "*payments while you are looking for work*", "*services and programs when looking for work*", "*claim forms while you are looking for work*", "*search for jobs with australian job search*", "*information for school leavers*", "*working at centrelink*" "*payments for your retirement*", "*services and programs for retirement*", "*payments when you are caring for an adult*", "*advice on being retrenched*", "*advice on being retrenched*", "*information about postponing your retirement*", "*how do i claim age pension?*", "*double orphan pension*", "*accommodation and renting issues*", "*rent deduction scheme*", "*services for homeless people*", "*concession and health care cards*"
Cluster 11: 6 links about farm help	
1.1.2.3.1.1.2.3.1.10.2.[1,3].1 1.1.2.3.1.1.2.3.1.10.2.2.2.[1-3].1 1.1.2.3.1.1.2.3.1.4.2.3.1	"*farm help*", "*payments if you are self employed or responsible for a farm*", "*dairy type grant*", "*drought assistance*", "*payments to assist with tasmanian freight costs*", "*payments if you are in crisis or needing special help*"
Cluster 12: 7 links about online security and fraud	
1.1.2.3.1.1.2.3.1.1.2.10.1 1.1.2.3.1.1.2.3.1.[2-3].2.9.1 1.1.2.3.1.1.2.3.1.9.2.[5-7].1 1.1.2.3.1.1.3.1.1.1.1	"*privacy notice*", "*secure online services*", "*secure online services*", "*secure online services*", "*secure online services*", "*information about fraud and reporting a suspected fraud*", "*report a suspected fraud online*"

Table 3.2: Output clusters of the Centrelink site, showing Dewey IDs and anchor text. The link clusters are color-coded in the Fig 3.3 screenshot.

DBLP homepage

DBLP[6] is a bibliography of computer science research. The site has 93 links, although several are duplicates. Fig 3.4 shows a color-coding of the 86 unique links using $\lceil \frac{86}{10} \rceil = 9$ output clusters, which are detailed in Table 3.3. The page uses simpler HTML and simpler visual layout compared to the Centrelink page. Although there are many links, a large number of them are already coherently organized and the proposed link clustering method managed to preserve their cohesion.

Of note is that the large blue cluster, which can be said to represent the DBLP site and dataset. It unites links from various positions in the layout: 5 links about the site ("*Home*", "*FAQ*", "*Welcome*"), 7 links about mirror pages and 2 links for downloading the dataset. If a web author was interested in reorganizing the layout of the page, they can use these cluster suggestions to place the two dataset links ("*XML*", "*DTD*") closer to the other blue links, such as near the three mirror sites on top.

Perhaps the only improvement on this output is to create a new cluster for "search"-related links, which are currently mixed in Clusters 1 and 2. A separate cluster would be ideal here but it is again difficult to know the optimal number of clusters beforehand.

[6]http://www.informatik.uni-trier.de/~ley/db

Figure 3.4: The DBLP site, which has 93 links and a simple layout. The color-coding shows all of the clusters.

Anchor text	Anchor URL	Description	Anchor text	Anchor URL	Description
Cluster 1: 10 links mostly about the site					
(an image)	uni-trier.de	Trier logo	*Author*	./indices/a-tree	Author search
ley@uni-trier.de	mailto:ley@uni-trier.de	Maintainer	*Author*	./indices/a-tree	Author search
Michael Ley	/~ley/addr.html	Maintainer	*Title*	./indices/t-form.html	Title search
Michael Ley	/~ley/addr.html	Maintainer	*Title*	./indices/t-form.html	Title search
copyright	./copyright.html	About	*Advanced*	./indices/query.html	Adv. search
Cluster 2: 14 links about the dataset					
(an image)	/	Home	*Home*	/	Home
(an image)	dblp.uni-trier.de	Home	*FAQ*	about/faq.html	About
ACM SIGMOD	acm.org/. . . /dblp/db	Mirror 1	*Welcome*	./welcome.html	About
VLDB Endow.	vldb.org/dblp/db	Mirror 2	*Faceted search*	dblp.l3s.de	Search/mirror
SunSITE . . .	sunsite. . . /dblp/db	Mirror 3	*L3S Research . . .*	l3s.de/growbag	Creator of above
dblp.uni-trier.de/xml	/xml	DBLP as XML	*CompleteSearch*	dblp.mpi-inf. . . mirror	Search/mirror
A simple DTD	./about/dblp.dtd	XML spec	*Holger Bast . . .*	mpi-inf.mpg.de/~bast	Creator of above
Cluster 3: 11 links about journals					
Journals	./journals	Journal list	*CACM*	./journals/cacm	Journal
Journals	./journals	Journal list	*TOIS*	./journals/tois	Journal
TOPLAS	./journals/toplas	Journal	*DKE*	./journals/dke	Journal
VLDB J.	./journals/vldb	Journal	*Inf. Systems*	./journals/is	Journal
TPLP	./journals/tplp	Journal	*TCS*	./journals/tcs	Journal
TODS	./journals/tods	Journal			
Cluster 4: 9 links about conferences					
Conferences	./conf/indexa.html	Conf. list	*SIGMOD*	./conf/sigmod	Conference
Conferences	./conf/indexa.html	Conf. list	*PODS*	./conf/pods	Conference
ER	./conf/er	Conference	*EDBT*	./conf/edbt	Conference
ICDE	./conf/icde	Conference	*POPL*	./conf/popl	Conference
VLDB	./conf/vldb	Conference			
Cluster 5: 10 links about other publications					
Series	./series	Index of books	*By Subject*	./subjects.html	Topics
IFIP	./series/ifip	Springer books	*Logic Prog.*	./conf/indexl.html	A topic
LNCS/LNAI	./journals/lncs.html	Springer books	*Database Systems*	./conf	A topic
DB textbooks	./books/dbtext	DB books	*IR*	./ir.html	A topic
Collections	./books/collections	DB/DM books	*Reference*	./books/reference	DB books
Cluster 6: 6 links about recent Turing Award news					
Turing Award 2007	acm.org/. . . award-07	ACM Award	*Joseph Sifakis*	./indices/. . . /sifakis. . .	2007 recipient
Edmund M. Clarke	./indices/. . . /clarke. . .	2007 recipient	*E. Allen Emerson*	./indices/. . . /emerson. . .	2007 recipient
Tribute to Jim Gray	eecs.berkeley. . . tribute	1998 recipient	*Klaus R. Dittrich. . .*	ifi.uzh.ch/krdsym	Tribute
Cluster 7: 17 links about related services					
Related Services	./links.html	Links	*Google Scholar*	scholar.google.com	Papers
CiteSeer	citeseer.ist.psu.edu	Papers	*CiteseerX*	citeseerx.ist.psu.edu	Papers
Live Academic	academic.live.com	Papers	*CS BibTeX*	liinwww.ira.uka.de/. . .	Papers
HAL	hal.archives-ouvertes.fr	Papers	*io-port.net*	io-port.net	Papers
CoRR	arxiv.org/corr/home	Papers	*NZ-DL*	nzdl.org	Papers
Zentralblatt MATH	emis.de/zmath	Papers	*MathSciNet*	ams.org/. . .	Papers
Erdös Number Proj.	oakland.edu/enp	Citations	*Math Genealogy . . .*	gene. . . nodak.edu	Researchers
BibSonomy	bibsonomy.org	Tag papers	*CiteULike*	citeulike.org	Tag papers
Libra	libra.msra.cn	Papers			
Cluster 8: 9 links about orgs					
Comp. Sci. Orgs	../organizations.html	Org list	*ACM*	acm.org	A major org
IEEE Comp. Soc.	computer.org	A major org	*DL*	acm.org/dl	Digital Library
DL	computer.org/. . . /dlib	IEEE DL	*SIGMOD*	acm.org/sigmod	ACM group
IEEE Xplore	ieeexplore.ieee.org	IEEE DL	*SIGIR*	acm.org/sigir	ACM group
IFIP	ifip.or.at	A major org			
Cluster 9: 7 links about the SIGMOD Anthology					
ACM Sigmod Anth.	./anthology.html	Digital library	*Universität Trier*	uni-trier.de	Anth. sponsor
ACM Sigmod Anth.	./anthology.html	Digital library	*ACM SIGMOD*	acm.org/sigmod	Anth. sponsor
VLDB Endowment	vldb.org	Anth. sponsor	*EDBT Endowment*	edbt.org	Anth. sponsor
Microsoft Research	research.microsoft.com	Anth. sponsor			

Table 3.3: Output clusters of the DBLP site, showing URLs and anchor text. The link clusters are color-coded in Fig 3.4.

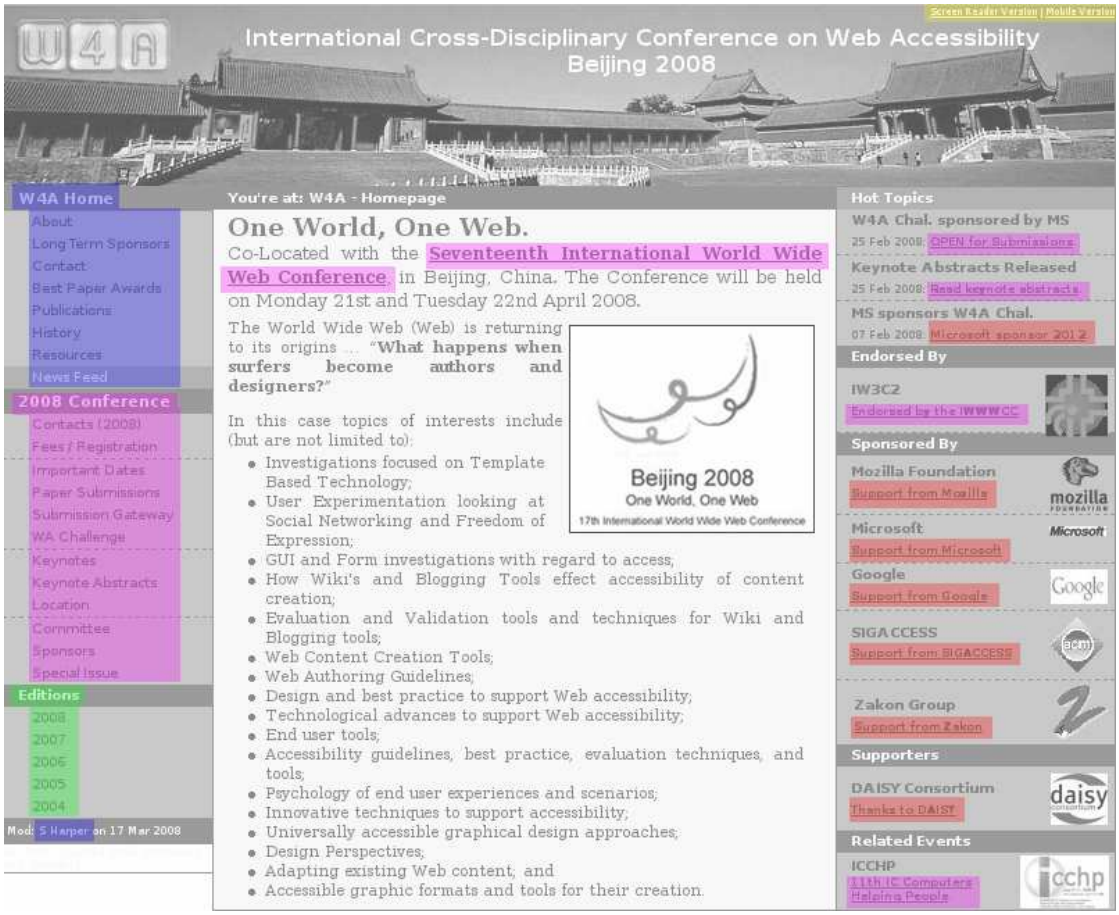

Figure 3.5: The W4A site, which has 43 links and a simple layout.

W4A homepage

W4A[7] is an annual conference for web technology and accessibility. Fig 3.5 shows a color-coding of the homepage using $\lceil \frac{43}{10} \rceil = 5$ clusters. The blue cluster merges the links about the background of the conference, including a stray link (bottom left) to the homepage of the co-founder; the green cluster contains links to homepages from previous years; the large pink cluster are links specific to the 2008 venue, such as the accepted papers, keynotes and registration. Also part of the pink cluster are links to three related conference bodies (IW3C2, WWW and ICCHP, in the middle, center and bottom right); the small yellow cluster on top has two links leading to screen reader and mobile-friendly versions of the homepage; finally, the red cluster collects the sponsors together.

[7] http://www.w4a.info

Although there are no errors in this output, one potential improvement is to split the large pink cluster into two smaller clusters.

3.3.2 Pipeline

The previous experiment used the following pipeline:

- parsing step, involving anchor extraction and Dewey labeling: 0.10s
- pre-processing step, involving URL canonization, tokenizing, keyword extraction, stop filter and stemming: 0.50s
- propagation step, involving feature vector normalization, δ calculation and weight propagation: 0.48s
- clustering step, involving k=20 runs of k-means: 2.1s

The majority of time was spent in making separate runs of k-means to improve the cluster quality. Depending on the application, fewer or more runs may be performed. For assisting web authors with reorganizing layout, time is not important and a higher k can be used. For screen reader navigation a smaller k can be used.

A small k can also be used for the next experiment on section ads. The pre-processing and propagation steps can also be sped up based on the assumptions (Section 3.2.5) made about the page layout of news sites. Namely, Dewey IDs are not assigned, the pre-processing step does not consider depth and the propagation step has a much smaller set of K keywords to propagate due to the sparsity of keywords in the headlines.

3.3.3 Sectioned news advertising

New York Times

Fig 3.6 shows a color-coding of the output clusters for the New York Times site from Fig 3.1. In line the assumptions from Section 3.2.5, link depth was flattened and both the cluster and classification steps used only the anchor text to describe the links. Because only one source of keywords was used, the link features are sparse and few links share

keywords in common. K-means would normally produce inconsistent and poor clusters in this case, but the rubber-band effect of the propagation step is able to retain much of the layout cohesion.

The output sections in Fig 3.6 are mostly right, but not quite perfect. For example, the first link in the fourth cluster would appear to semantically and logically belong in the third cluster. Nevertheless, the overall sectioning appears useful because the clusters are large and logical, hence suitable for advertising.

As mentioned previously, it is also possible to partition the page using heuristics, such as detecting specific HTML elements that are known to be section markers on the

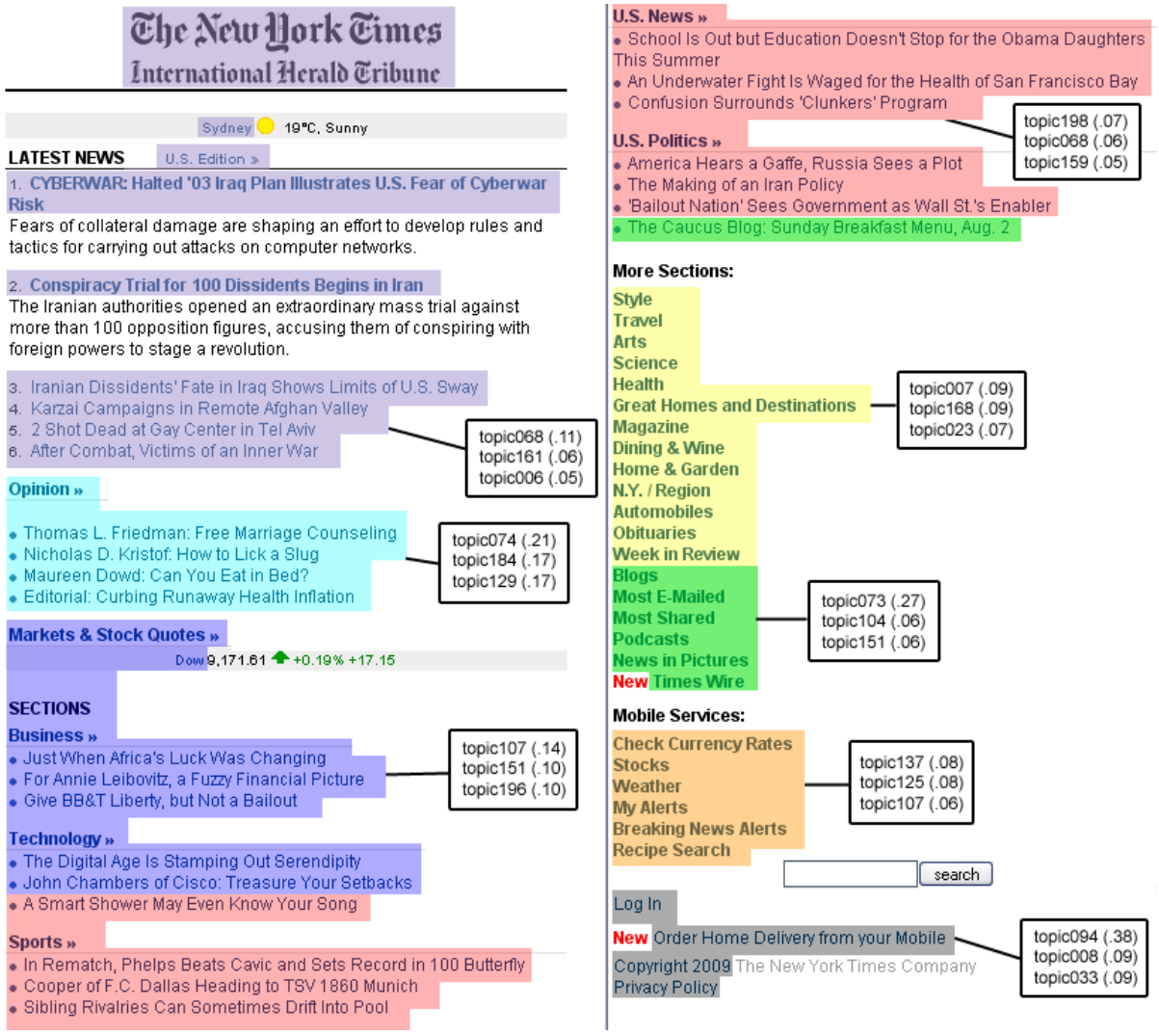

Figure 3.6: The New York Times mobile site, with color-coded clusters and their dominant classes. The classes are described in Table 3.1.

NYT domain. However, the benefit of using the link clustering method is that it is not domain-specific and that it can group also group together links that are not in a sequential run. For instance, the approach detected that the link "*The Caucus Blog*" belonged to the green cluster, which also contains the "*Blogs*", "*Most e-mailed*" and "*Most shared*". These links are highly related and belong together, yet this grouping cannot be found by merely detecting special HTML elements.

Fig 3.6 also shows the top-3 dominant topics for each cluster using the Section 3.2.5 classifier. The accuracy of the top-3 topics is 75% and each top topics is relevant to its respective cluster (i.e. $P@3 = 0.75$ and $P@1 = 1$). Selecting an ad that belongs to one or more of the top-3 topics and placing it in that particular position in the page would constitute a section-relevant ad. The numeric scores in Fig 3.6 are the p_c scores for the classifier, which helps gauge the relative confidence between the top-3 topics. Although the topics are not named, their subject becomes evident once we eyeball their features in Table 3.1.

Consider the dominant class for the first cluster of "latest news" headlines: *topic068*. This class had twice the score of *topic161* and *topic006*. Looking at Table 3.1, it appears to refer to US political and military conflicts, which is highly relevant to the cluster's headlines. Consider also the third cluster, which mixes hyperlinks about stocks, business and technology. Its dominant classes were determined to be *topic107*, *topic151* and *topic196*, and looking at Table 3.1 reveals these classes to represent financial markets, online technology and business. Thus all three dominant topics are highly relevant despite the cluster mixing several concepts.

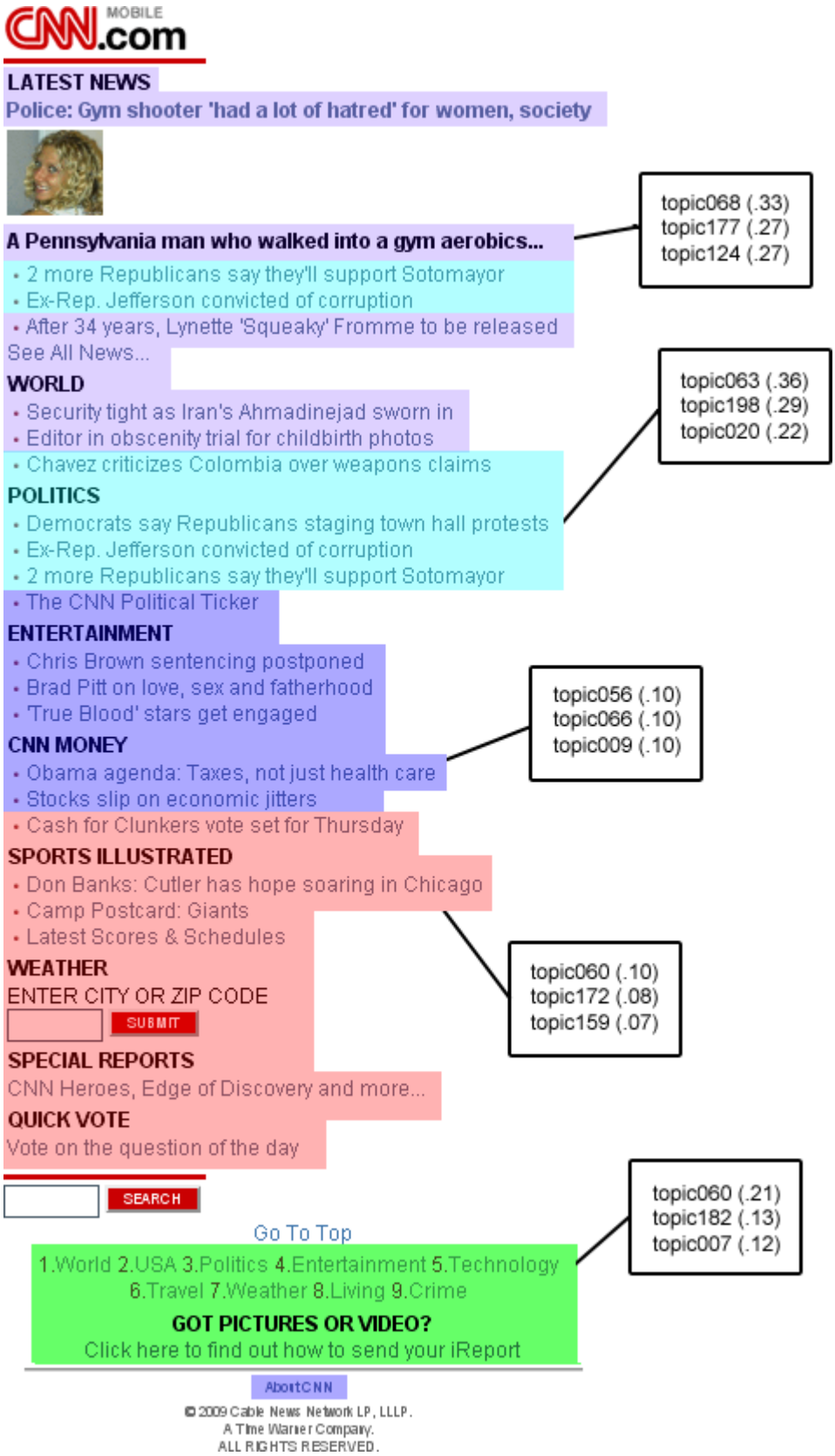

Figure 3.7: The CNN mobile site, with color-coded clusters and their dominant classes. The classes are described in Table 3.1.

Cable News Network

Fig 3.7 shows the output for the CNN site. There are fewer links on the CNN site compared to the NYT site, and hence only $\lceil \frac{45}{10} \rceil = 5$ output clusters. As with the NYT site, the classification topics are quite accurate: $P@1 = 0.8$ and $P@3 = 0.67$.

The most notable error in the output is that a couple of links appear to logically belong in other clusters. For example, "*Cash for clunkers vote set on Thursday*" should be in the third cluster but was placed in the fourth cluster because two other links had the keyword "vote". Note that since the user will not be shown the color-coding, they will not notice if the link is misclassified.

Although it is possible to classify each headline individually, this is undesirable because the classifier still incurs a processing cost and it is preferable to run it 5 times rather than 50 times. Nevertheless, one possible use for classifying links in an individual manner is interstitial advertising. This is a style of advertising that shows in an ad as an in-between page after a link is clicked and before the destination is shown. In this case the classifier needs to be run only once for the link that was clicked. If a database of pre-classified ads is stored locally on the client then the selection can be performed on the client-side. This may be a possible way for online news sites to monetize their content.

3.4 Summary

The chapter presents an approach for on-demand hyperlink clustering on a currently viewed webpage, using a hierarchical weight propagation step and cluster step. The propagation is a one-pass weight distribution intended to maintain link cohesion in the cluster step. Three applications of the method are identified: improving navigation for vision-impaired users, helping web authors organize or reorganize their hyperlink layout on a page, and showing topical ads in particular positions in highly dynamic pages such as online news sites.

As described in the chapter, in an ideal case the hyperlinks on a page should be assigned

a set of descriptive keywords, or feature vector, to improve the clustering accuracy. The feature keywords may come from many places, such as the anchor text, URL, an external folksonomy (e.g. Delicious) or even the destination page's content. The more information that is available to build a feature vector, the better the classification accuracy. The next chapter further investigates this concept of labeling webpages with descriptive words. In particular, Chapter 4 explores another personal space—a web2.0 tagging system—and proposes a way of enriching the tags such that they become more expressive and more useful to the system's internal search and navigation algorithms.

Chapter 4

Searching in web2.0 tagging systems

4.1 Introduction

The online web2.0 movement has introduced many web-based tagging systems. In these systems, a community of web users use keyword *tags* to annotate, reference, organize and search for objects in a shared repository. The tagging process is sometimes referred to as social or collaborative because users can often see the tags of other users and in some systems can even add their own tags to others' objects.

The increasing popularity of tagging systems represents a paradigm shift from traditional author-provided metadata to user-provided metadata, and systems exist for a variety of web objects and content such as bookmarks (Delicious, Furl), photos (Flickr, Shutterstock), academic and scientific articles (CiteULike, Bibsonomy, Connotea), music (Last.fm) and videos (YouTube, Vimeo).

Because every user in the community is a potential contributor of new objects, the repository size can grow quite large. This means that the tags themselves are the primary mechanism for not only annotating, but also for searching, browsing and navigating. The search process is similar to issuing a keyword query to a web search engine. However, unlike search engines there is not always an explicit link structure in a tagged repository. In many of today's tagging systems the search functionality is somewhat unsophisticated,

operating on the level of tag alone and using them as simple boolean filters. This practice becomes increasingly inadequate as the repositories grow larger.

This chapter presents an approach for improving the usefulness of the tags within a system in a way that is invisible to users but beneficial to the system's internal processes. In particular, tags are made more expressive using a scoring function to assign them hidden ratings. The ratings allow the system to use tags more effectively behind-the-scenes. The scoring function, TagScore, is introduced and a scoring scheme appropriate for the web is described.

It may be impossible to define what constitutes an 'optimal' tag for an object, and it is still difficult to define what constitutes a 'good' tag, but we can at least reason that an 'appropriate' tag is one that the tag-assigner is more likely to choose for an object and that the tag-searcher is more likely to search with for that same object. These two conditions necessarily require the tag to be descriptive of the object, otherwise the two parties will not agree on their choice of tag. In many tagging systems the authors who contribute the objects are also themselves users of the system who search the repository, therefore there is some overlap in understanding of both parties' needs and behavior.

The hypothesis proved in this chapter is that accuracy improves if we make tags more expressive. While not surprising, this chapter contains what we believe is the first work that aims to improve the correlation of tag-only comparisons to full document comparisons. The chapter provides the implementation details of TagScore and several experiments using Delicious data to demonstrate its appropriateness. Additionally, the effectiveness of several external types of metadata (titles, meta keywords, synonyms and cooccurrences) is examined to determine how they useful each is for the scoring process. Some hypotheses in the literature are verified and supported by the experiments, and TagScore allows us to quantify the relative differences of the measurements.

Tags

Conceptually, a tag is any descriptor that a user will interpret to carry meaning about some object. In the future we may see complex tag forms such as audio or visual tags, but the simplest form in use today is plain-text keywords. Keywords are easy for users to create, recall and associate, and are also easy for a system to support. Today's tagging systems equate a tag to be a short plain-text description of one or a few words in length. This design choice lets users search the repository in a way which they are already familiar with from having used web search engines.

Unfortunately, tags are ad hoc and ad lib and have been criticized for carrying less information than formal systems (e.g. Dublin Core) and taxonomies, which results in some problems when using them as keyword filters. In contrast to taxonomies, tagging systems rarely impose limitations on the choice of words and therefore issues such as the Vocabulary Problem [40] and polysemy ambiguities are problematic. Additionally, there are many idiosyncratic tags such are subjective choices (e.g. 'todo', 'jobsearch') and syntax issues such as spelling errors, foreign words, use of punctuation, browser footprints ('safari_export', 'imported') and spliced compounds ('climatechange', 'mcescher'). Further discussion on criticisms may be found in [56, 86, 130].

While it can be debated whether tags are the best way to obtain metadata on a large scale, it is undeniable that tagging systems are becoming more popular and currently outpace our understanding of how to best support their socially-driven annotation model for effective search and navigation. As such, this chapter is concerned with how to make better use of tags *as they exist today* and makes an assumption that tags are text keywords.

Tagging Systems

Social tagging is an activity wherein a community of web users assigns labels to web objects. The resulting annotations are a bottom-up, distributed classification system in which the labels reflect the attitudes of the community. It is 'social' because the objects are shared and their tags are visible to others. Incentives to participate in such systems include

discovery (search), recovery (bookmarking, classification) and self-publishing (advertising one's works) [5, 84].

Today's tagging systems are built around the three major axes of *users*, *tags* and *objects* (sometimes called *resources*). Each axis may be flat or may have subdivisions. For instance, Flickr subdivides each axis: users create and join user groups, photo tags are clustered based on their usage, and the photos are collected into photo pools. These kinds of logical groupings may be helpful in improving the accuracy of search-based tasks, but to address the general problem we must assume that such domain-specific groupings are not present. Therefore the chapter assumes that each axis is flat.

Online tagging systems use the tag model in a social environment in which users can look at other users' objects, see other users' tags and even subscribe to other users. Marlow et al. [84] highlighted the fundamental differences between the different types of tagging systems, but they share one important feature in common: that users share *both* the objects and their tags. The simple provision that tags are visible to others enables search. Consequently, improving the expressiveness of tags should improve the accuracy and effectiveness of all system functions that rely on search.

The experiments in this chapter focus on one of the most popular tagging systems, Delicious[1]. Its users are content consumers who bookmark internet webpages. Delicious is 'social' because users can see the tags chosen for a URL by the community as a whole or individually, and it is often called a 'collaborative' system because an object's tags come from multiple users.

Making tags more useful

In terms of search, tags act as filters. A single tag selects a subset of all objects and a multi-tag (boolean AND) query selects a smaller subset. The wealth of information in the repository will not be as useful as it could be if search is difficult. This can happen if the results are noisy, inaccurate, or there is a lack of system functionality to support different

[1] http://delicious.com, owned by Yahoo.

kinds of search. Delicious's repository now has over 200 million URLs. The system already provides some alternate methods of exploration, such as browsing by 'most popular' and 'most recent'. However, other exploration methods are possible.

A 'find similar' function is one such method that Delicious does not yet implement. Consider that a user has just bookmarked a page and wants to find similar or related pages from the repository. Using the tags as simple boolean filters will not work so well because a conjunctive query will require results to overlap on all tags (which fails for pages with a reasonable number of tags) and even simple disjunctive queries in the large repository return hundreds of thousands of results. The alternative is for the user to guess which tags to include or exclude from their query. Unfortunately, tags do not support intuitive query refinement and, as the repository continues to grow, this simplistic retrieval strategy of boolean filters is insufficient. The user would prefer that the system automate the process and retrieve results ranked by relevance. To do this effectively the system will need the tags to be more expressive so that it can decide which tags are more important than others for matching.

TagScore provides this ability. It enriches the tags to numerically quantify their appropriateness for an object. Knowing when a page is poorly described by its tags should reduce noise. Inversely, knowing when a tag is appropriate should improve accuracy. A simple way to determine the goodness of a tag is to count its frequency: how many users chose the tag for the bookmark. But this does not work well for pages with few bookmarks and Section 4.1.1 suggests that 67% of Delicious pages have fewer than a dozen bookmarks. This means that frequencies for most objects are quite low and inaccurate or unstable. Yet there is no reason why a less-popular object cannot be the 'most similar' to the query. Hence, simple frequency counts are only a half-solution.

Fig 4.1 shows what we expect to get from using TagScore on an object: a numeric rating (or 'contribution') for each tag as well as an overall score. The contributions will be used for a similarity measure to determine how similar two objects are, whereas the overall score—which is dependent on the contributions—can be considered as a confidence that

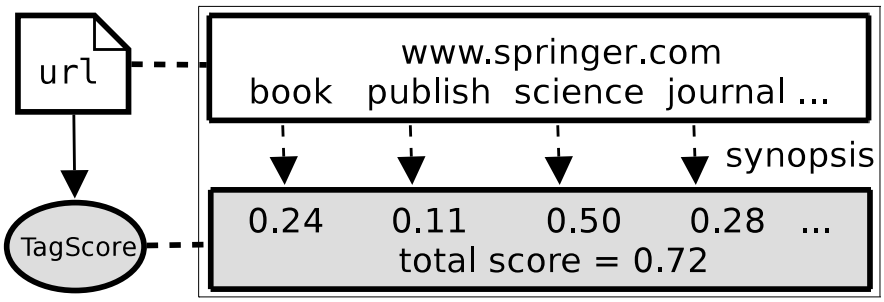

Figure 4.1: The tags act as a webpage's summary, or synopsis. TagScore enriches the synopsis by assigning tag ratings and an overall confidence score.

the set of tags adequately and exhaustively describe the object. As one of several efficiency shortcuts trialled in the experiments, this confidence score can be used to prune less-confident comparisons from broad searches. Such a confidence score is useful because it allows the system to focus on the more promising results.

The tags, their contributions and the overall confidence score give us an extremely succinct synopsis for a document. Comparing synopses is a fast operation, so the goal is to maximize the correlation of the synopses approximated comparisons against the full comparison while minimizing the time and space overhead needed to enrich the tags.

Because they are very lossy, comparing synopses is a way of approximating the similarity of the documents. Approximate similarity is a well-studied problem in IR. It can be scaled to very sparse data with high dimensionality such as text (e.g. Houle and Sakuma [62]). However, tagging systems have several key differences that require a different solution. The goal of approximate similarity is to determine a useful way to reduce the dimensionality of the data in a way that loses as little information as possible under acceptable size constraints. In contrast, systems such as Delicious are already dimensionally reduced because the data (i.e. bookmarks) is already summarized to a minimum using only a handful of tags. Furthermore, answering a kANN query involves building a new and efficient index to approximate the k-Nearest Neighbors using some given distance metric. In contrast, systems such as Delicious already provide the efficient index—an inverted list over tags—and instead need to specify the distance metric itself. Because

of these fundamental differences, this chapter focuses on tags and tagging systems rather than kNN methodology.

If we consider tags to be a succinct summary of a document, then searching the repository—either using a direct tag query or using a target document whose nearest neighbors are sought—is equivalent to a multi-tag search where tag overlap is maximized. The basic Delicious synopsis comprises a document's tags, their individual frequency counts, and the number of users who bookmarked the document. Fig 4.1 shows how TagScore enriches this synopsis by introducing contribution ratings and an overall confidence score. There is an obvious space versus accuracy trade-off regarding how much new information should be added and TagScore uses a minimal amount (a few bytes per tag, as all the extra information is numeric). Much like the basic synopses, the enriched synopses are still quite lossy. However, it is the most appropriate kind of data that should be used for search functions in tagging systems because it is very compact and efficient.

Experiments between the search approximation using the basic synopses and enriched synopses is given in Section 4.3.4, where both are compared against the full comparison. We assume that the full comparison is an oracle, i.e. gives the correct and desired answer.

This chapter extends Penev & Wong [101, 103] with additional experiments and analysis of the metadata sources. It is organized as follows: the dataset and the TagScore scoring function are described in Section 4.2; experiments in Section 4.3 examine TagScore's goodness as a scoring function, compare the effectiveness of the synopsis enrichments for search-based tasks, and investigate the relative usefulness of using several metadata sources. Section 4.4 gives a summary and some conclusions.

4.1.1 Dataset

The experiments section uses Delicious bookmarks from DMOZ100k06 [96], a random sample of 100,000 web pages from the Open Directory. A total of 4992 pages from this sample had entries in Delicious. Ignoring inaccessible or unsuitable pages (e.g. body $<$ 15 words, flash, etc.), 4278 of these were downloaded for content analysis. The average

page was 32KB, down to 12KB after stripping HTML. Each page was then prepared for processing by case-folding, tokenizing, removing stopwords and stemming.

Each page is considered to have three components: a title, a meta and a body. The title and body elements appear in virtually all pages. Meta refers to the terms listed in the meta-keywords element, which 62% of the dataset had. Discarding extreme values, the average number of words in the three components was 5.2, 38 and 480, respectively. On average a page had only 5.0 tags, with a range[2] of 1-25, median 3 and s.d. 6.

An assumption is made that this dataset has a similar power law distribution of tags, similar ratio of idiosyncratic tags, and a similar number of average tags per page as the full Delicious repository. The random sampling suggests that this should be so.

4.2 Approach

4.2.1 TagScore

TagScore is a keyword-oriented scoring function for rating the tags T against a document D. Scoring functions return values with a total ordering, but their range can usually be expressed in the unit interval and so TagScore returns values between 0 and 1. Another design decision is that TagScore should describe the existing state of social bookmarking, i.e. a flat tagspace of keywords.

For rating a tag, it is desirable to alleviate some of the criticisms of tags (Section 4.1). While polysemy impedes accuracy, it can be mitigated by the use of cooccurrence statistics and a thesaurus. Idiosyncratic tags such as spelling errors are inevitable in practice, but their impact can be reduced by taking into account the collective annotation of multiple users together. The Section 4.1.1 dataset has a median of 6 users per entry, meaning that the tags were expected to come from six different people; this is a low enough for idiosyncratic tags to appear among the 'top tags' for the entry but is high enough for them have a lower frequency among the users.

[2]The DMOZ100k06 dataset was constructed to include only the top-25 tags for an entry.

With these desiderata in mind, the rating of a tag can take into account several factors: the popularity of the tag amongst users, how the tag matches the content (both directly or indirectly), and how the tag matches any metadata provided by the author. Additionally, some external lookups such as rarity and synonym tables can be used to reason about language. The rest of the section describes these elements in more detail, starting with the tagging information that already exists in the dataset.

4.2.1.1 Tag popularity and bagweight

Delicious counts how often a tag is applied to a bookmark. The Section 4.1.1 dataset honors this notion of popularity by attaching a 1–5 integer rating per tag. These ratings roughly represent the tag frequencies (binned to five discrete levels) and reflect the relative importance between tags according to the community. The dataset normalizes the values so that each page has at least one tag with a 5-rating.

For use in TagScore, the ratings are normalized again such that a tag t in a bag of tags T is given a weight $w(t, T)$ of one-fifth its popularity rating, i.e. the top tag is worth 1 unit. This scheme is coarse-grain due to having only five levels of accuracy. Additionally, tags with the full popularity rating are not necessarily the best possible tags for the page but are merely the most relevant *relative to the other* chosen tags[3]. While both of these factors are likely to worsen the performance of TagScore in the experiments, they are adequate for this chapter as a proof-of-concept and Section 4.3 will show that TagScore still achieves a statistically significant improvement.

The sum of the weights, $\sum w(t_i, T)$, will be referred to as the *bagweight*. This is a more meaningful number than the number of unique tags. Consider the case where T comprises one popular and five unpopular tags. It would be misleading to call the popular tag a 'one in six' because it accounts for half of T's usage. In this case the bagweight will be $1 + 0.2 \times 5 = 2$, which gives the desired ratio. An interpretation of bagweight is "T's

[3]This incompleteness is a limitation of the dataset and exists in all tagging systems.

total weight can be emulated by *bagweight*-number of equipopular tags". In general, a higher bagweight indicates a larger number of highly-popular tags.

4.2.1.2 Document structure

Given the structured nature of web pages and the different information conveyed within, matching a tag or a concept in some parts can be considered more important than in others. For instance, a page's title and meta-keywords are the author's description of the body's content. They have a different purpose to the body in that they act as a summary of the content. Thus they merit a higher TF weighting that words in the body. Because the body has many more words it is balanced by giving terms in the body a TF of $\frac{1}{3}$ ($\approx \sqrt{\frac{5+38}{480}}$ as in Section 4.1.1). For example, if a term t appears once in the title and five times in the body, its weighted TF for the document, $w(t, D)$, would be 2.67. The document length norm is set to $\sqrt{|D|}$, which becomes $\sqrt{\sum_{d \in D} w(d, D)}$. These values, which consider the author's view of the content, are used in Section 4.2.1.5 for matching against the users' view of the content to calculate the tag ratings.

4.2.1.3 External sources

Analyzing text requires an external source of information to reason about language. TagScore uses a stoplist, DF table, cooccurrence table and WordNet. The first of these is used during pre-processing. The others were pre-built offline from the dataset and are used by TagScore during its online calculations. All four lookups can be considered as global language statistics because they require few updates.

Document Frequency. A simple lookup to return DF_t, the number of documents a term t appears in. The dataset had a 235k vocabulary (199k post-stemming). Most of these words formed a long-tail (rare words, spelling errors, etc.) that were ignored to leave behind 76k usable word stems.

Cooccurrences and WordNet. The dataset suggests that the average number of tags per page is quite low—only 5.0—and therefore simple string matching of tags will

miss many words or concepts related in meaning. An page-independent way to check that 'football' (say) is a good tag for a page talking about 'soccer' would be to measure the strength of any relationship between the two words. Such relationships can be easily computed using cooccurrence or a lexicon. Cooccurrence is calculated using cosine,

$$CO(x, y) = \frac{DF_{x \wedge y}}{\sqrt{DF_x \cdot DF_y}}$$

The more often two words cooccur, the closer their relatedness value to 1. For example, in the experiments and dataset CO(computer,fridge) was 0.03, indicating that the words cooccurred rarely; a stronger relationship was CO(web,search) at 0.52 and even stronger was CO(sigmod,sigkdd), which shared all pages in common and had a relatedness value of 1. As one of the pre-built lookups, the top-50 cooccurrent terms for each entry in the DF table vocabulary was determined. The top three terms in the 'football' example were ⟨soccer nfl sport⟩, achieving the goal of finding the association.

Associated words can also be found using a lexicon such as WordNet [90]. Another pre-built lookup was constructed using WordNet's "synonyms" and "overview" outputs. The top-25 TF terms from these two outputs were extracted for each entry in the DF table. The relatedness value was set to the normalized TF, e.g. for "internet" WordNet output "network" the most often and so it is assigned a value of 1, while the next most common word, "computer", occurred half as often and is assigned 0.5.

The relatedness values are somewhat crude, but both of these lookups are used in a lossy manner such that exactness is not critical. They are used to query expand the relatively small tag bag into a larger set of related words, i.e. from an average of 5 tags to several hundred.

4.2.1.4 Expansion Tags

Most pages have very few tags and we would reasonably expect that their tag bag T is not a comprehensive description of the page. Therefore TagScore expands the bag by retrieving up to 100 related terms (per tag) from the cooccurrence and WordNet lookups.

The aim is to obtain a set of related terms, each with a relatedness score in (0,1]. The process is shown below:

---

expand(T): expand tags T to get related terms E.
Output: mappings E of ($word \mapsto relatedness\ score$).
1. $E := \{,\}$
2. **for each** $t \in T$:
3. **for each** $w_{w \neq t}$ in $\langle t$'s top-50 cooccurrent terms$\rangle$:
4. $E[w]$ += $CO(w, t)$
5. **for each** $w_{w \neq t}$ in $WN := \langle t$'s top-25 synonym terms$\rangle$:
6. $E[w] \mathrel{+}= \frac{1}{2} \frac{TF_w}{\max TF_{WN}}$
7. redo lines 5–6 for t's top-25 overview terms.
8. **end for**
9. if the max relatedness score in $E > 1$, divide all E by it.
10. **return** E

The WN scores are multiplied by half because the WN lookup is used twice and many terms are repeated. On average E is 73 times larger that T. It is less than 100 because the CO and WN lookups have some overlap and because sometimes WordNet's output is brief and fails to fill the 25 term quota.

As will be discussed in Section 4.2.1.6, whether each tag is expanded to 100 or 50 or even just 25 terms will not significantly affect the final outcome because these new terms are compacted into a single abstract tag rating and they are not part of the synopsis. The number of expansion terms only needs to be large enough to improve the match rate because the average number of user tags is generally much too low (only 5.0). The effect of not using the expansion terms at all—that is, when E is empty—is shown in Table 4.1.

4.2.1.5 Tag contributions

Despite its simplicity, traditional TFIDF has been shown to work well for tag-based tasks [23, 30] and is very efficient. TagScore calculates the tags' contribution ratings using their importance in the bag (w_T, Section 4.2.1.1), their importance in the page (w_D, Section 4.2.1.2) and their importance globally (IDF), as shown overleaf in the **rateTags** function.

Once the tags' contribution ratings are calculated, a confidence score can then be determined for the entire synopsis. To minimize computational overhead this should be

rateTags(T, D): calculate contributions of tags.
Inputs: bag of tags T, document D.
Output: mappings of $(tag \mapsto rating)$.
1. $bagweight := \sum_{t \in T} w(t, T)$
2. $docnorm := \sqrt{\sum_{d \in D} w(d, D)}$
3. **initialize** $contrib[t_i] := 0$
4. **for each** $t \in T$:
5. **continue if** $DF_t < 1$
6. $w_T := w(t, T)$ / $bagweight$
7. $w_D := w(t, D)$ / $docnorm$
8. $contrib[t] := w_T \cdot w_D \cdot \log \frac{|ndocs|}{DF_t}$
9. **end for**
10. **return** $contrib$

performed in a page-independent manner so that previously scored pages do not need to be used as frames of reference. In other words, only the page itself and the values calculated from the page must be used because normalization using other pages cannot be done. However, summing the contribution ratings directly does not produce values in [0,1], and the low number of tags—sometimes only one—makes any kind of vector normalization unsuitable. A trick around this is to use a DCG-like geometric sum:

conf(C): sum contributions.
1. **let** $S := \text{sort}(C)$, descending.
2. **return** $\sum_{i=0} 2^{-i} \min(\frac{1}{2}, S_i)$

There are several beneficial reasons for using such a calculation. Since the addends are sorted in reverse and their individual value is capped at $\frac{1}{2}$, the sum will not exceed $2S_0$. Additionally, a page with only one tag cannot score too highly, which is what we would expect. The sum has several desirable properties:

- it allows arbitrarily many terms but sums to [0,1).
- the highest terms dominate, insisting on quality over quantity.
- it is strictly increasing.

Having the highest addends dominate allows a page with few-but-good tags to outscore a page with many-but-poor tags. The third point is a theoretical advantage but does not hold in the current TagScore implementation because the tag weights are normalized by

the bagweight. For example, suppose there are several good tags that dominate the total. As more tags are added, the tag weights are being divided by a larger bagweight and, at some point, the contribution ratings of the top tags will fall below the $\frac{1}{2}$ cap in the above sum and the confidence score may decrease. It can be argued that this is how a scoring function *should* behave, e.g. Xu et al. [138] argue that having fewer tags is preferable to having too many tags. This supports the idea that confidence should drop if the tag bag is oversaturated.

In practice, very popular Delicious pages have hundreds of tags and the above case of diminishing returns may occur. Fortunately there are some simple countermeasures: for systems where tags have a histogram with popularity levels, either the true frequencies (i.e. not binned to discrete levels) can be used or only the top-k tags can be used, and for systems where tags are used once and have no frequency distinction, either take the top-k tags ordered by their global selectivity/rarity or take the first-k. The dataset used for the experiments imposes a limit of k=25, which is a reasonable bound for Delicious since 85% of Delicious pages have 10 of fewer tags and this limit of k is not exceeded often.

4.2.1.6 Putting it together

So far this section has showed how TagScore assembles related query expansion terms, how it determines a tag rating and also defined a way to sum ratings for a confidence. These pieces are combined below.

TagScore(T, D): returns a confidence score in $[0, 1)$.
Inputs: bag of tags T, document D.
1. $E :=$ **expand**(T)
2. $T_r :=$ **rateTags**(T, D)
3. $E_r :=$ **rateETags**(E, D)
4. $E_{total} := \frac{1}{2}$ **qty_conf**$(\text{range}(E_r)) + \frac{1}{2}$ **conf**$(\text{range}(E_r))$
5. **return conf**$(\text{range}(T_r) \cup \{E_{total}\})$

The contribution ratings in Fig 4.1 come from line 2 and the overall confidence score from line 5. The expansion set of related terms, E, is computed temporarily and is treated as a single albeit abstract tag. The set E is not recorded as part of the synopsis, but its

match is incorporated into the overall confidence score by adding its abstract confidence, E_{total}, to the normal tag contributions. In the implementation, E_{total} is calculated as the average of E's quantity[4]) and quality (the normal **conf** sum)

Note that E_{total} cannot negatively impact the overall confidence and only serves to boost it, which is useful for pages with very few but good tags. Also note that the code for **rateETags** is omitted. It is the same as **rateTags** except with the bagweight summing over only the terms that match; this is because the expansion terms should not incur a denominator penalty if they are not matched since, after all, they are synthetic and not part of the synopsis.

4.2.2 Feasibility

The results produced by TagScore are described in Section 4.3, but we can briefly consider TagScore's feasibility for implementation in practice. Note that TagScore needs to access the webpage content to rate the tags. However, Delicious only keeps pointers to the objects (in the form of bookmarks) rather than save the objects themselves. This is specific to Delicious and is not the case with many tagging systems, but because the experiments focus on Delicious we can list some methods that the content can be accessed.

The first method is to download the page as needed. Delicious is a Yahoo property and may have privileged access to the search engine's cache, which contains billions of pages. Downloading pages from Archive.org may also be faster and more stable than trying to fetch the pages from their original web hosts.

The second method is for the Delicious plugin in the user's web browser to push the content to the server. At present, when a user bookmarks a page, the plugin obtains the page title from the browser window and shows a pop-up asking the user to input some tags. At the same time the plugin asks the server for tag suggestions, and after a brief moment it updates the pop-up with the retrieved suggestions. While returning these suggestions the server may request for the plugin to send the page content along with the

[4]I.e. **qty_conf**$(\cdot)$, which is implemented as a capped sum of caps: $\min[1, \sum \min(x_i, \frac{1}{2})]$.

user's tag choices. Since the user is already viewing the page in the browser, the plugin can easily summarize its content. Even a simple frequency tally of the words will suffice for scoring purposes. The scoring cannot be done on the client side because the plugin will not have the necessary lookups available, but it is possible for the plugin to forward the page summary to the server. An experiment in Section 4.3.2 will show that using just the first 2.5KB of content gives similar results as using the whole page, so the plugin can use such a shortcut.

The TagScore calculation itself is fast, but because tags change over time we need to consider how often scores should be recalculated. Assuming the content is not kept persistently on the server, how often does the server need to obtain it from outside? The work of [43] helps answer this question by showing that the tags for a bookmark stabilize over time as it attracts more users. Hence a recalculation may not be necessary after a page has reached a certain level of popularity and its tags have stabilized. Scoring should instead focus on the earlier stages when the tags are still unstable. The content may be obtained only several times during the entire lifetime of a bookmark, and suitable thresholds for accessing it or having the server request it from the plugin could be when it reaches 5, 10, 50 and 100 bookmarks.

4.3 Evaluation

This section begins with several experiments to analyze the behavior of TagScore as a scoring function. These are followed by experiments to test the importance of the lookups and metadata sources. Finally, another experiment tests the accuracy of the enriched synopses for two search tasks.

4.3.1 Behavior

It is desirable for a scoring function to produce a smooth and unbiased distribution. Fig 4.2 shows the TagScore distribution of overall confidence scores using three different

Label source	Samples	Avg/Median TagScore	$E = \emptyset$	$\neg\, CO$	$\neg\, WN$
$T = D$'s meta	2664	0.415 / 0.39 (σ=0.22)	0.361 (σ=0.23) (-13%)	0.395 (-5%)	0.417 (+0%)
$T = D$'s title	4275	0.555 / 0.59 (σ=0.22)	0.510 (σ=0.23) (-8%)	0.543 (-2%)	0.554 (-0%)
T = User tags	4278	0.460 / 0.53 (σ=0.24)	0.388 (σ=0.24) (-16%)	0.450 (-2%)	0.454 (-1%)

Table 4.1: Summary of **TagScore**$(\mathbf{T}, \mathbf{D})$ using three sources of keywords as the page's tags. Also shows the effect of not using the expansion terms E (where $E = CO \cup WN$), not using CO and not using WN, as detailed in Section 4.3.3. The drops in scores reflect how much matching related terms helps—16% for Delicious tags.

sources for a page's labels: its title, its tags and its meta-keywords. The graph shows that TagScore is reasonably balanced and unbiased, with a 0.46 mean and 0.53 median for tags. Table 4.1 provides more details about the graph.

Fig 4.2's smoothness also indicates a lack of ties and the linear slope suggests a uniform-like distribution. These properties enable the pruning of large searches to desired confidence percentiles using simple calculations. For example, to discard the 30% least confident synopses from a calculation to speed it up, the cutoff would be set to roughly 0.30. Note that the TagScore calculation is page independent. This has obvious benefits in practice, but it also means that the distribution will not be the ideal $y{=}x$ straight line unless external frames of reference are used during the scoring.

Fig 4.3 represents a smoothed curve for the ratio of high-scoring (top quartile) pages with a certain bagweight. It shows that a similar portion of pages received a high score as the portion that existed with that particular bagweight, which means that TagScore is not biased against having too few or too many tags and a page can score highly regardless of its tag count. The slight bias against pages with very high bagweight is due to diminishing returns and dilution of having too many highly-popular tags, as discussed in Section 4.2.1.5.

4.3.1.1 Title vs Tags vs Meta

The labels rated by TagScore do not have to be tags. They may be any kind of keywords and may optionally be weighted based on some weight scheme, e.g. on user popularity, position in text, IDF, etc. Fig 4.2 shows the TagScore behavior under the author's

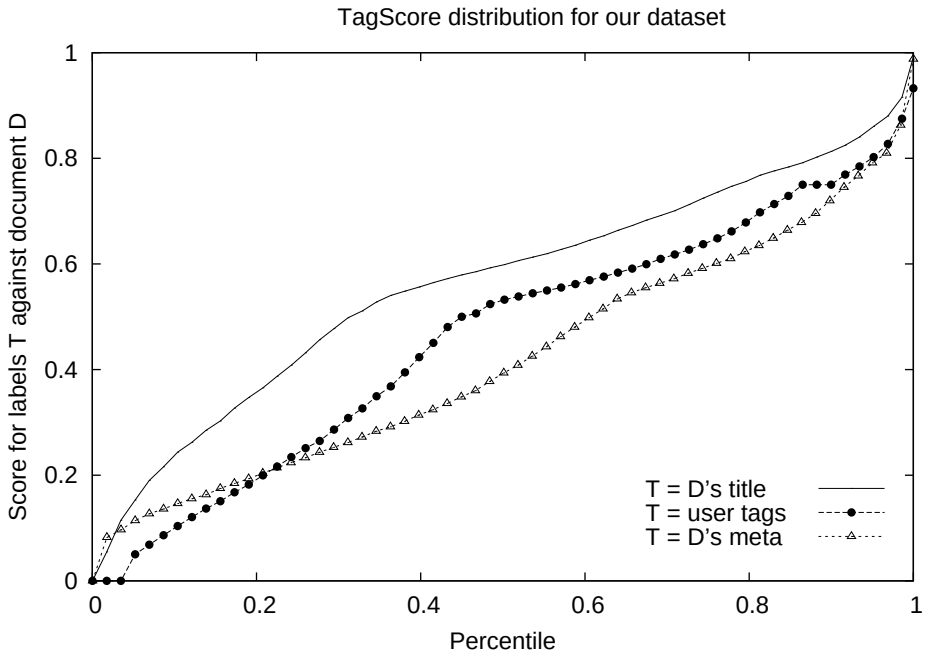

Figure 4.2: TagScore confidence distribution for the dataset using three different sources for a page's labels: title, user tags and meta-keywords. The title and meta are given by the webpage authors. The graph shows that the title is a slightly better descriptor of content compared to user's tags, and both are superior to the meta terms.

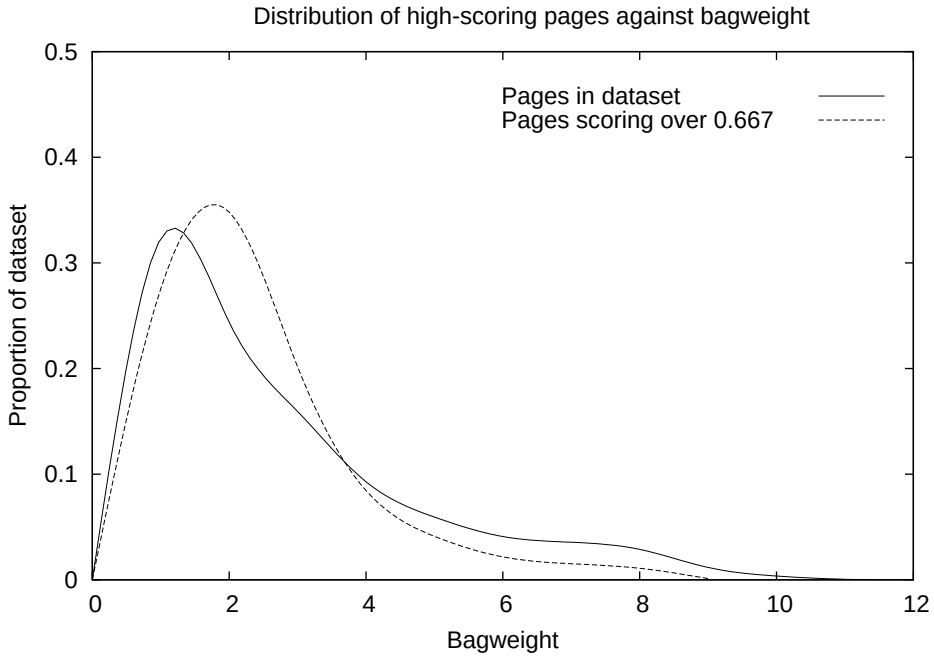

Figure 4.3: Best-fit for the distribution of *bagweight* (Section 4.2.1.1) of all pages vs high-scoring pages. Shows that the confidence scores are evenly distributed among pages with few tags and pages with many tags.

description[5] of the page in the form of the title and meta. Title appears to be a stronger content descriptor than user tags and meta, which is the outcome most web developers would expect to see. For example, a news article often has a descriptive title such as "*Yahoo buys social bookmarking service Delicious*", while its tags are likely to be less descriptive (e.g. "*delicious yahoo web*") and its meta even less so (e.g. "*internet news business technology*"). TagScore appears to capture this sentiment and can quantify it. Fig 4.2 reveals a few other behaviors about TagScore:

- the slightly denser middle region forms because of the way the confidence score is calculated in a page-independent manner. Notably, pages with one tag can score close to the average if that tag receives a high rating (Section 4.2.1.5). The title and user tags have medians of only 3–4 words, thus many pages with only one good tag receive an average confidence and result in a bulge around the middle. Meta-keywords has more words on average and produces a more uniform spread.

- the 4% of pages tied on 0.75 all had a single user tag that was rated very highly. This is the maximum confidence score a page with only one tag can reach under the geometric sum: 0.5 by itself and 0.25 for its abstract expansion tag.

- the user tags start below both the title and meta because the latter are provided by the author and rarely mismatch the content, whereas user tag sometimes mismatch completely (about 4% of pages scored zero).

4.3.2 Ignoring title, meta and body

TagScore inspects the title, meta and the body of a page. This section investigates the effect on scores if these sections are ignored or abridged. While this does not directly evaluate TagScore (because results are presented in terms of TagScore), it is precisely that TagScore enables such comparisons in the first place that makes them interesting and improves our understanding of webpage metadata.

[5]These labels are part of the page source. When rating them, the corresponding section of the source is ignored to avoid the obvious self-match.

Ignoring meta-keywords

In the dataset, 62% of pages had a meta-keywords list. The number of terms in this list ranged from 0–1300, which has a high variance. Table 4.2 shows the effect on TagScores if only a fixed quota of the meta terms was used. The original list was replaced with the corresponding subset list and TagScores were recalculated for all pages with meta.

Substitution	Avg TagScore		τ
full list (default)	0.504	-	-
top-25 TF terms	0.505	(+0%)	0.97
first-25 terms	0.495	(–2%)	0.92
first-10 terms	0.480	(–5%)	0.85
empty list	0.429	(–15%)	0.74

Table 4.2: Decreases in confidence and τ when using fewer meta terms.

The data in Table 4.2 shows that ignoring the meta-keywords decreased the average confidence by a significant amount (15%). The correlation with the original result was quite strong (Kendall $\tau = 0.74$), but may not justify ignoring all of the meta element because it is inexpensive to use and has reasonably good match accuracy. The table shows that taking only the first 25 meta terms is suitable in practice because it minimally affects TagScores and correlation.

Ignoring title

Ignoring the title element during matching resulted in a mean confidence of 0.436, which is comparable to Table 4.2's result of 0.429 for ignoring meta-keywords. Correlation between the no-title and no-meta results was strong ($\tau = 0.76$), suggesting that title and meta are roughly interchangeable for being matched against by user tags. However, title achieved this accuracy using much fewer terms on average, and titles are both more visible on a webpage and more reliable to be included [49]. Thus, this experiment suggests that title is slightly more useful than meta.

Shortening the body

What the body shows strongly affects what tags user will label the page with, so the body should not be ignored. But the body's content can be very long and it would be useful to have a known quota of how much content should be downloaded and parsed. Noll et al. [96] recommended using the first 2.5KB of the body and showed that it decreased the match rate from 49% to 42%. This value, which is in terms of percentage match for tags-to-body string comparisons, represents a 14% drop.

To investigate how such a shortcut affected TagScore confidences, we performed a similar experiment by replacing the body of each page with its first 2500 non-markup text characters. Leaving the title and meta-keywords (if any) intact and using them in the TagScore calculations, the new mean, median and σ were 0.47, 0.52 and 0.24. These values were almost identical to the originals in Table 4.1, where the entire body was used. The very strong correlation ($\tau = 0.77$) with the original list indicates that using the first 2.5KB is useful in practice under TagScore because it provides a fixed quota of content to parse yet minimally affects the result. At an average page size of 12KB, abridging the body to 2.5KB will reduce the parsing work by 80%.

The above experiment left the author-provided metadata intact and only tested shortening the body. To verify the recommendation in [96], the experiment was repeated but this time matching the tags only to the body (i.e. pretend that the meta and title do not exist). The mean TagScore for match the tags to the full body was 0.48 and after applying the 2.5KB shortcut it decreased to 0.43. This represents a 10% drop. While correlation was still reasonably strong ($\tau = 0.61$), the preceding experiment—in which the shortcut was applied but the author's opinions were still used for matching—resulted in less disorder and should be preferred in practice.

The closeness in drops (49%–42% in [96]'s match rate versus 0.48–0.43 in TagScore) is coincidence because the two measures are not comparable (e.g. a bag of tags may have a 100% match rate but still achieve a near-zero TagScore if none of the tags are rated highly). Nevertheless, this experiment supports the recommendation to use only the first

2.5KB of the body in practice, and also indicates that the page title and first-25 meta terms should also be used.

4.3.3 External lookups

Section 4.2.1.3 described two optional lookups that were used to assemble the expansion terms. These lookups are denoted in Table 4.1 as CO for cooccurrences and WN for WordNet. Their purpose is to increase the confidence of synopses whose tags do not directly match, and to smoothen the TagScore distribution by reducing the number of ties. Ties are undesirable because they make it more difficult to prune a set of synopses at a particular cutoff. Additionally, the expansion terms enable a synopsis with a single tag to score highly, which supports a preference of quality over quantity.

Although the lookups are useful, they are the costliest part of TagScore's performance (Section 4.3.5). An experiment was conducted to test the effect of removing them. Disconnecting a lookup means having less related words available for use, which is expected to increase the number of ties and decrease the average confidence.

The results are in the right-hand side of Table 4.1. Scores fell only marginally when removing either lookup, but significantly (16%) when removing both. The reason for the larger combined drop is that the CO terms and WN terms share some overlap and many terms are still available for matching if either lookup is removed, but disappear when both are removed.

Nevertheless, it is a good idea to use at least one lookup because the average page has only 5 tags, which results in many tied confidence scores. The τ correlations for disconnecting WN, CO and both were 0.93, 0.80 and 0.76. Disconnecting only WordNet caused the least disorder, a finding that supports Chirita et al.'s [30] observation that cooccurrences are more useful than a thesaurus for keyword-based tasks. One factor likely to have understated WordNet's performance for this domain is that it matched only 10% of the dataset's vocabulary, whereas cooccurrences can be calculated for any term.

4.3.4 Approximate similarity

Section 4.1 described how tags act as a synopsis and how TagScore enriched it. This section reports the results of three experiments to test the enriched synopses against the naive (non-enriched) synopses and against the full page comparisons.

The first experiment took the synopsis of a randomly chosen page from the dataset and made a one-to-many comparison by looking for similar synopses among the other pages. The top-25 results were taken and their correlation against the cosine similarity ranking of the corresponding pages was recorded. Only the top-25 most similar results were considered because Delicious shows 25 results on a page of search results, and users are mostly interested to know that the very top results are accurate [51]. The second experiment repeated the first experiment but checked the correlation of all results retrieved. This means that lower-ranked results, which tends to receive similar score in the long tail, are also considered. The third experiment took a large portion of pages and performed a many-to-many comparison to find the most similar pairs. Additionally, to verify that comparing between 'confident' synopses improves accuracy and that TagScore's notion of confidence is sensible, the experiments were repeated at different TagScore cutoffs to prune low-scoring synopses.

In each experiment, the full comparison was done using cosine similarity on the content of the webpages and is assumed to be the correct answer oracle. In the three experiments, Kendall τ correlations were calculated between the full content cosine comparisons and the tag-only synopsis comparisons (both enriched and naive) that attempted to approximate cosine.

Kendall's τ is a correlation measure often used to determine the similarity of two ranked lists. Its interpretation is that for taking two items at random, $\tau = P(right\ order) - P(wrong\ order)$. It is the gap between probabilities that the relative order of two random items in one list is the same in the other. A correlation of τ=-1 means the lists are inverted, τ=0 means they are independent (a neutral relationship) and τ=1 means they are identical. A positive value of 0.5 means $P(right\ order) = 0.75$, which is considered a

reasonably strong correlation. Reaching 1 is impossible due to the lossiness of representing webpages by their tags. Rather, the aim is to maximize τ while using minimal space and memory overheads for the enrichments.

There are many possible combinations of the synopsis variables that are able to produce a ranked list of near neighbors; the ones used in this work are given in Section 4.3.4.1. From the combinations mentioned below, those that somehow used the TagScore tag ratings were considered as 'enriched approximations' and those that used information already available in Delicious were considered as 'naive approximations'.

4.3.4.1 Finding similar documents

This experiment picked a random page d_x and searched for similar pages by comparing synopses. Similar pages were ranked by similarity, and similarity was determined with a greedy approach where a 'something' was accrued for each shared tag between the query and the object. While this means that an object would not be retrieved unless it had at least one tag from the query, it is a necessary step due to practical considerations: the aim of TagScore is to enrich existing tags rather than add or remove tags and therefore the repository search engine will not be able to retrieve any objects with TagScore that it could not already retrieve without it. Since the retrieval is implemented with inverted lists over tags, at least 1 tag must be shared for an object to appear in the search results.

Because a greedy approach maximizes tag overlap, the highest-ranked results are likely to overlap on more than one tag. It is possible to optionally take into account IDF here so that rare tags are given greater preference than common tags. For example, for the query "latex reference" the user would prefer a page tagged with only "latex" than a page with only "reference". Including a consideration of IDF in the overlap maximization is therefore expected to benefit the naive approximations because TagScore's enrichments already take IDF into account. This experiment will show that the IDF-improved naive approximations are still worse than the TagScore enriched approximations. A description of the experimental setup is given next.

findSimilar(d_x): get documents like d_x with tags T_x.
1. $D := \bigcup_{t \in T_x} inverted\_lookup(t) \; - \; \{d_x\}$
2. optionally, prune low-scoring docs from D at some cutoff.
3. **for each** $d_y \in D$ with tags T_y:
4. **for each** tag $t \in T_x \cap T_y$:
5. sim_{d_x,d_y} += something. (optionally $\times IDF_t$)
6. **return** D ordered by their sim.

A similarity measure for the 'something' (line 5) is needed to make d_y more similar to the query T_x when more tags overlap. This 'something' must consist of data available in the synopsis, which contains the URL, the tags, their popularities and the total bookmark count. An enriched synopsis, such as Fig 4.1, also contains the overall confidence score and tag contribution ratings. However, the confidence score is suitable for pruning weak results as opposed to judging relevance or similarity. Hence, the 'something' should be formed using the bookmark count, popularities or contribution ratings. Labeling these three data as b, p and c, 11 variables were constructed: c, p, $h(p)$, $h(c)$, pc, $h(pc)$, $h(p)h(c)$, cpb, $h(cpb)$, pb and $h(pb)$. In this notation, $h(\cdot)$ denotes harmonic mean and individual letters symbolize the product of the two corresponding values, i.e. p means to multiply the two popularities of the tag and cpb would be a product of six values. In the implementation, b was dampened by taking its square root and p had already been pre-discretized, as in Section 4.2.1.1.

All of the variables considered are either products or harmonic means, which perform better than sums or averages because they give zero when either value is zero. Several other combinations of the above variables exist, but they result in near-zero correlation overall and are omitted. Additionally, two more special variables were added to the above list: the integer 1 and the tag frequency in the second page only ($T_y.p \times T_y.b$). The former trivially counts the number of tags that *overlap* and the latter favors the more popular tags from the more popular results in the search.

The 13 variables above were used for the 'something' in **findSimilar** to determine the sort order of the search results. All were reasonable choices and expected to produce positive correlations with cosine. However, some of the variables produce very many ties.

Ties are undesirable because they can give misleading correlations, especially if many results are tied such as the case of using the *overlap* variable. Therefore any ties must be broken and each object needs a unique rank. To break ties that resulted using the first of the 13 variables as sort criterion, one of the other 12 variables was used as a second criterion ($13 \times 12 = 156$ combinations in total). Any remaining ties were then broken lexically. For the cosine results, any ties were broken reverse-lexically to avoid benefitting from sorting a run of tied results alphabetically and matching the oracle. In all there were 156 ways of generating a ranked list for synopses using **findSimilar**, of which 126 used the enrichment variable c in some way and the other 30 were considered to be naive approaches.

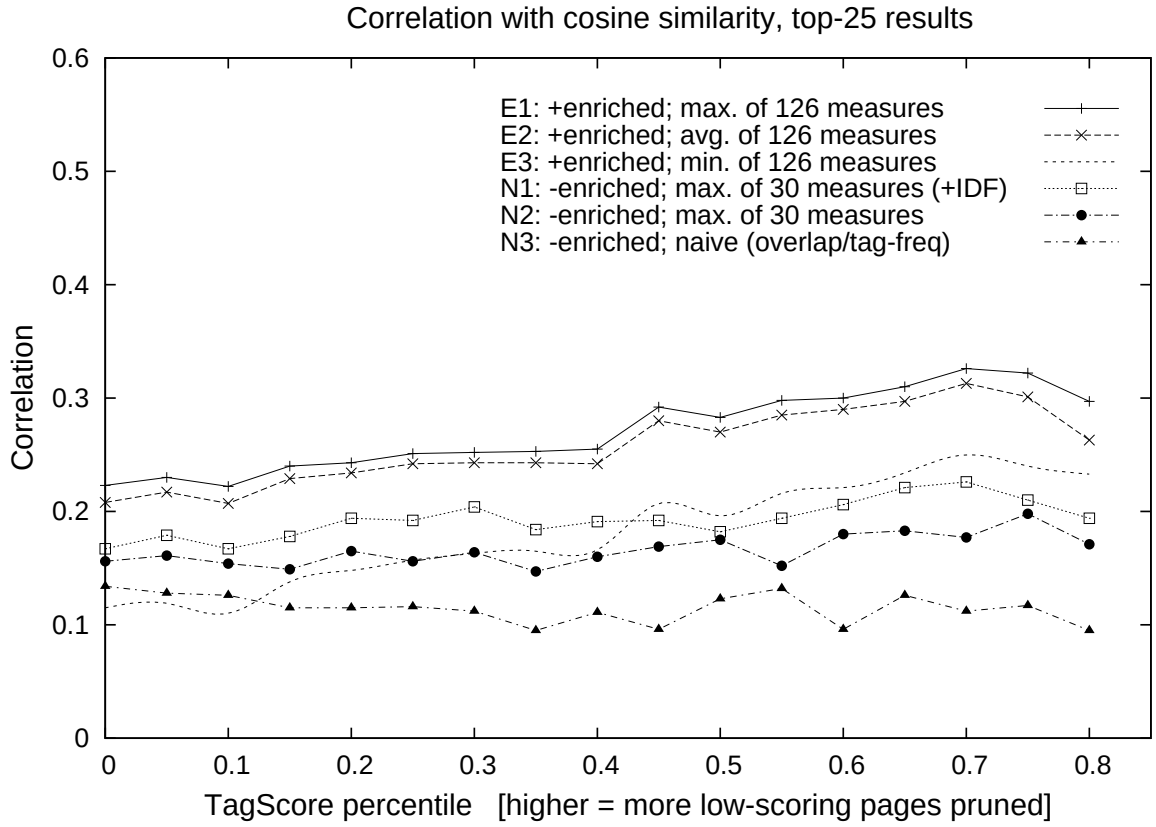

Figure 4.4: τ-correlation of synopsis approximations compared to cosine similarity. Shows that enriched synopses are more accurate (Section 4.3.4.1). The x-axis is TagScore percentile, showing that correlation improves when less-confident synopses are pruned.

Fig 4.4 shows a summary of the entire experiment using 6 curves. At the bottom, N3 represents perhaps the most obvious naive approach of using *overlap* and breaking ties

with frequency. In other words, a bookmark was considered a closer match if it had many tags in common and these tags were popular. Since overlap is a good sort criterion for a similarity ranking, the correlation was positive. N3 is the only curve representing a single measure.

The curve N2 plots the *best* instances (averaged over 500 runs) among the 30 naive approximations. N1 is similar to N2, but included the optional IDF multiplier. This was expected to boost the correlation of the naive similarity and appeared to do so.

The curve R3 plots the *worst* instances (averaged over 500 runs) among the 126 enriched approximations. It is comparable to the best among the non-enriched. Similarly, R2 and R1 were the average and the best instances. R2 and R1 are close because all measures using c performed similarly well. Multiplying by IDF had little effect in these case and the curve is omitted.

The results are clear: even the worst of the enriched approximations outperform the best of the naive approximations and are closer to cosine. It is also evident that τ increased as the pool of synopses was cleansed of low-scoring pages using a confidence cutoff. This suggests that, under TagScore, lower-scoring pages are indeed less-confident and poorly represented by their tags.

4.3.4.2 Correlation using the full set of results

The previous experiment cut the results list at 25 before computing cosine and τ. This experiment repeated the previous experiment but verified the correlation using all returned results. Fig 4.5 shows that there is no notable difference and that the correlation among the top results was similar to the correlation of all results. In addition, the average of the Mean Reciprocal Ranks for the top-3 most 'correct' oracle results was 0.30, which was 15% higher than the naive measures at each percentile. This indicates that the desired results were consistently ranked higher.

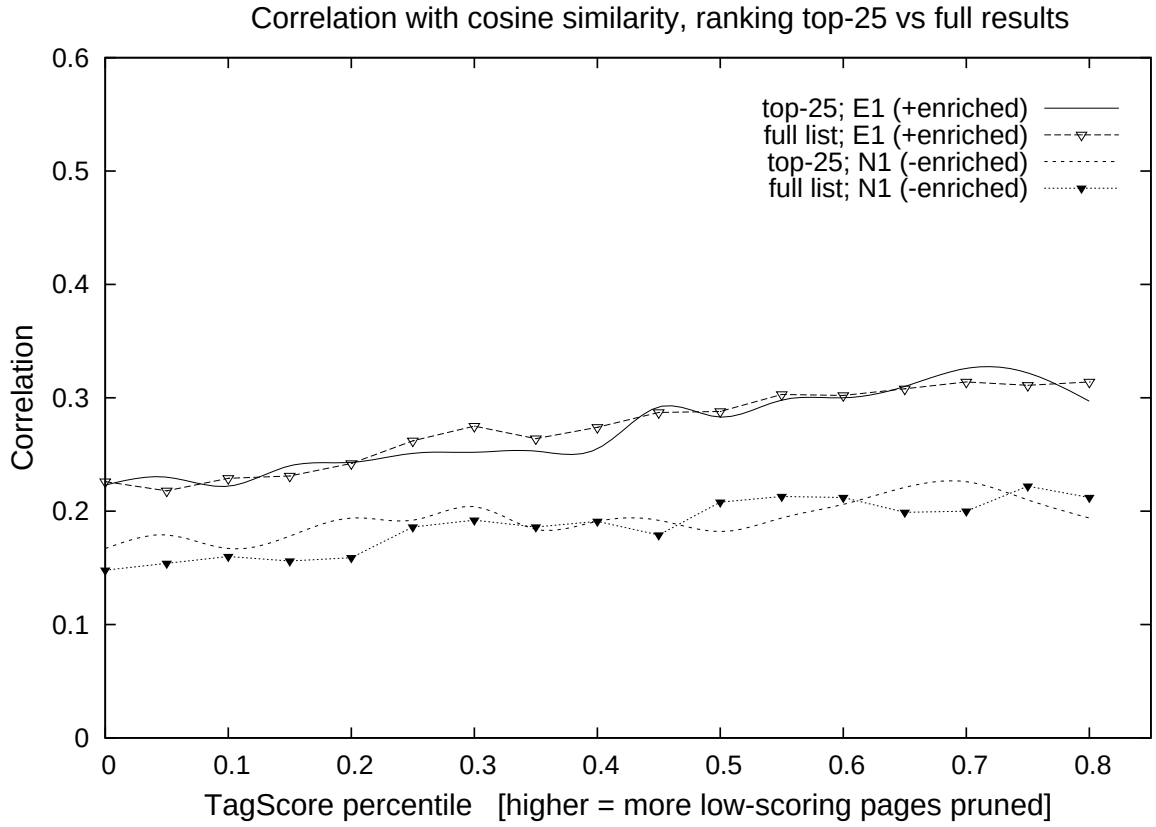

Figure 4.5: τ-correlation of synopsis approximation compared to cosine similarity. Shows τ is roughly equal for the first page of the top search results as compared to the full results (Section 4.3.4.2). The x-axis is TagScore percentile, showing that correlation improves when less-confident synopses are pruned.

4.3.4.3 Finding similar pairs in a set

This experiment took a large portion of pages, compared synopses in a pairwise manner and ranked the most similar pairs on top. The enriched approximation used was *cbp*, which performed well in the previous two experiments. The results are shown in Fig 4.6. Also shown is the best of the naive approximations, which favored synopsis pairs (d_x, d_y) where the overlapping tags had high popularity in both d_x and d_y. The result was again clear: correlation is good and increases with TagScore confidence percentile.

Such a pairwise calculation is computationally expensive but has many potential uses, especially in smaller sized sets where a cross-product is manageable. For instance, finding similar pages within a user's bookmarks will allow the system to recommend tags that are meaningful and personal to the user; finding pages among a user's subscriptions that

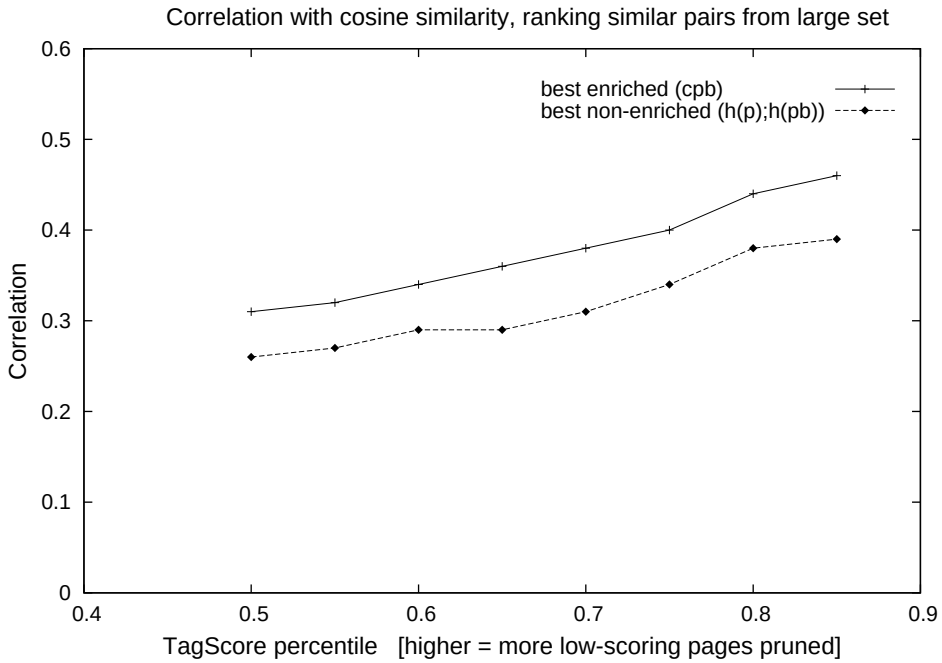

Figure 4.6: τ-correlation of approximations compared to cosine similarity. Shows enriched synopses can find similar pairings with good accuracy (Section 4.3.4.3).

are similar to the user's own bookmarks will allow the system to recommend links; an automated search that finds similar lists of bookmarks will allow the system to recommend subscriptions; and an automated search that finds similar pages (as in the above experiment) will allow the system to add certain tags by itself.

4.3.5 Performance

Experiments were run using Perl on a 3GHz Pentium 4 standard desktop machine with 7200rpm SATA disk running Ubuntu Linux.

TagScores for the dataset were computed in 25.5s, or a rate of 170 docs/sec. About 31% of the execution time was spent in necessary disk IO. Another 40% was spent in optional disk IO to assemble the expansion terms from the cooccurrence and WordNet lookups. Thus, keeping the lookups in memory would offer a large gain. Disconnecting the lookups, the side-effects of which are given in Section 4.3.3, would offer a similar gain.

Taking two random synopsis, comparing their similarity and throwing away the result ran at 220k cmp/sec, or 940 times faster than cosine similarity under the same conditions.

These comparisons did not necessarily have any tags in common. Comparing synopsis pairs that had at least one tag in common ran at 70k cmp/sec, or 490 times faster. This was slower on a per-comparison basis but also reduced the total work done because zero-scoring pairs were skipped. Reading in every synopsis from disk for *every* comparison ran at 20k cmp/sec, or 140 times faster than cosine.

In each case the approximation ran between two and three orders of magnitude faster than the full comparison that it was approximating and had a correlation several times higher than the naive approximations. The cost of the enrichment is quite small: comparing enriched synopses is about 4% slower than comparing naive synopses (because of extra arithmetic) and the space overhead for recording the enrichments amounts to a few bytes per page[6].

4.4 Summary

The chapter provided a detailed description of TagScore, a scoring function to enrich the tags in a social tagging system. The rapid growth of tagging systems demands attention and the goal of researchers answering this call has been either to understand the properties of such systems or to better support how they are currently used. The contributions of this chapter relate to both a better understanding and a better use. TagScore enriches tagging systems by rating the goodness of the tags and assigning an overall confidence score to their document. The enrichments are purely numeric and require minimal storage overhead. The ratings allow tags to be more expressive and allows the system to better use them internally, while the confidence score allows the system to automatically prune weak documents in large searches.

Using real Delicious data, the experiments showed that TagScore is an unbiased scoring function and produced a suitable distribution. In terms of search accuracy, the experiments showed a 200% improvement over the naive approach in τ correlation between

[6]With an average of 5.0 tags, TagScore would introduce 5 numeric ratings and 1 confidence score to the average page. Depending on implementation choices, this needs as little as 6–24 bytes of storage.

the faster tag-tag comparisons and the slower page-page comparisons. This means that similar pages can be found efficiently and with higher accuracy than before. Additionally, experiments are provided to gauge the usefulness of several sources of metadata for the tag enrichment task, and show that titles are stronger content descriptors than both user tags and meta-keywords, and that cooccurrent words are superior to thesauri. Some observations agree with intuition, but TagScore allows us to quantify the relative differences for the first time. Also provided are several experiments on how much content and metadata can be used during scoring, which results in identifying several useful shortcuts for how a TagScore-like function may be implemented in practice.

The chapter focused on enriching tags in the personal space of web2.0 tagging systems, and the presented approach is useful for all objects in the repository (as opposed to only popular ones). The following chapter progresses to a modern, emerging personal space—content viewed on a mobile phone—and uses some of the ideas discussed in this chapter. In particular, Chapter 5 proposes an algorithm for offline targeted advertising on small devices.

Chapter 5

Sponsoring mobile content

This chapter builds on some ideas encountered during the previous chapters to describe an offline, client-side advertising framework for heterogeneous content being viewed on small portable devices such as mobile phones. In particular, it presents an ad-serving agent for this personal space and evaluates an ad search and selection method that is suitable for such constrained devices.

5.1 Introduction

Following the advent of large-scale wireless infrastructure some three decades ago, mobile phones have become the fastest growing technology in history and represent one of the largest known consumer markets. The devices have evolved from bulky brick-phones to slim flip-phones and now touch-sensitive smartphones. Today there are some 2.5bn mobile users with a collective 3.3bn subscriptions among them [114].

Use is widespread in developed countries with the US having 86% penetration and most of Western Europe boasting over 100% penetration (due to multiple subscriptions). But the market has become saturated because those who want a phone already own a phone. This means that service providers compete for a share of existing users. They attract customers by offering a more attractive handset or a more attractive price, and sustained revenue comes from subscriptions. A big money-maker is *Value-Added Services*

(VAS), an industry term encompassing messaging (SMS, EMS, MMS, MIM), voicemail, email, web browsing, mobile-tv, downloadable content (games, apps, video, music) and all services beyond end-to-end calls that "add value" to use.

VAS make mobiles more functional, convenient and entertaining, but often incur a separate payment. Their price affects the subscription rate and 3 in 4 users cite price as the critical factor [4]. Thus, lowering VAS prices should be a priority for service providers who want to increase market share. One way to lower prices is to subsidize costs with advertising, but ads will need to be timely and accurate to be effective. Timeliness refers to ads being shown at opportune moments. Accuracy refers to ads being contextually relevant to the user activity, geographically relevant to all users in an area or personally relevant to the individual.

Existing mobile marketing approaches struggle with the timeliness and accuracy requirements, whereas the described framework aims to address both. Shown in Fig 5.1, it consists of an ad-database and ad-selector software on the device.

The ad-database is anticipated to store tens or hundreds of thousands of ads of various formats (text, image, video, interactive). Each ad will be labeled with keyword tags to facilitate efficient indexing and retrieval, as well as relevance scoring. The ad-database can and should be localized (i.e. be relevant to a region) and be synched to the ad-broker via periodic updates.

The ad-selector's role is to filter, score and rank ads for some query. This 'query' is a piece of arbitrary text understood to represent the current user activity or currently displayed content. Note that such a textual representation is possible for most types of content, especially VAS content. The ad-selector can also use the tags to track the users' perceived interests and bias its future selections to their tastes. It can also make location-sensitive choices if it has GPS and if the ads have location components.

Essentially, the framework is concerned with placing much of the intelligence of ad-serving onto a small device, which is non-trivial due to the hardware constraints. This chapter presents a framework and selection algorithm suitable for this domain, and is

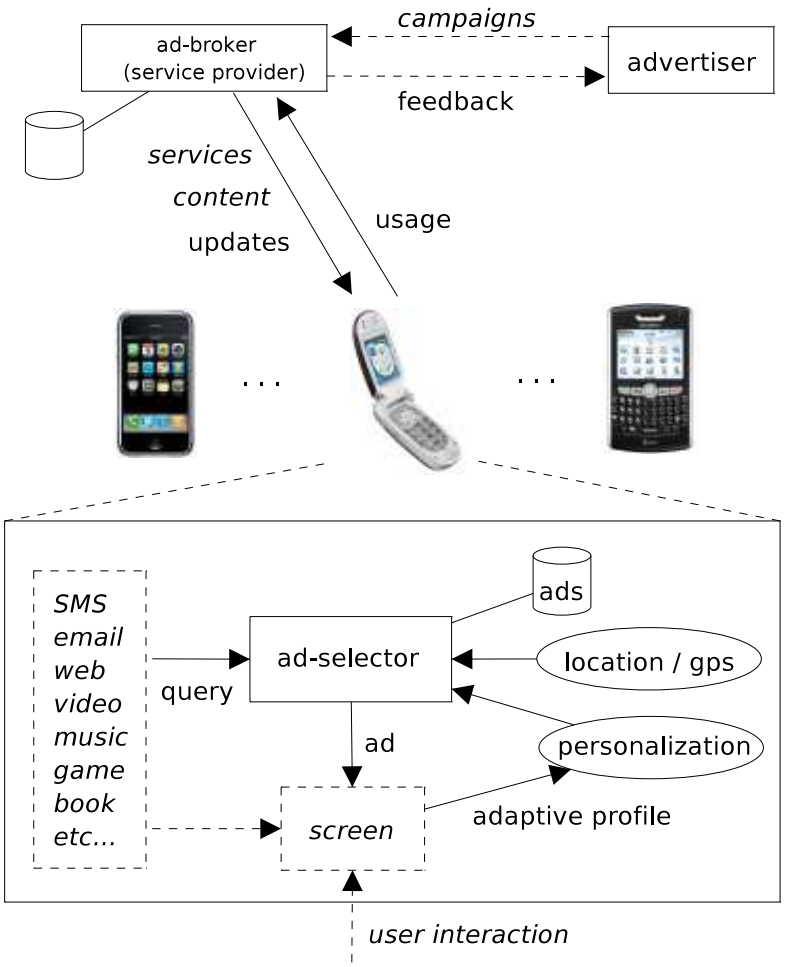

Figure 5.1: Overview of proposed framework and business domain. This chapter specifies the ad-selector component.

organized as follows. Section 5.2 first gives some background on mobiles and advertising. Section 5.3 expounds the ad-database and ad-selector components. A prototype ad-selector is constructed in 5.3 and then evaluated in Section 5.4. Section 5.5 gives a summary and some conclusions.

5.2 Background

Mobile devices

For some a mobile is strictly a communication tool, but for many it is an important everyday accessory kept within reach at all times. The power of the devices is improving,

with Apple's iPhone, Palm's Pre, Samsung's Omnia and Sony Ericsson's Xperia all boasting 400–624MHz processors. The hardware allows mobiles to perform computationally intensive tasks, such as playing video games. In terms of capacity, modern phones have multiple gigabytes of storage as built-in or via an SD card.

Value-added services (VAS) add value to mobile usage. Used by 3 out of 4 owners, SMS is the most popular with an estimated 2 trillion messages transmitted in 2008. Other VAS are growing their own customer bases, such as the 800m users who browsed the mobile web in 2007. The large customer base creates a lucrative market for advertisers.

Online advertising

The advertising model is responsible for the livelihood of many traditional media, such as broadcast television, newspapers, magazines and radio. Modern examples of its success are online keyword-targeted advertising systems such as Google AdWords. Targeted advertising has a long media history in magazines and television, and keyword advertising itself predates web search engines by a full century[1].

The era of web advertising began in 1993–94 with GNN and HotWired, both selling click banners. In 1996 OpenText improved the return-on-investment by placing specific ads alongside specific searches on a search engine. GoTo was more successful with a similar idea in 1998. Google launched AdWords in 2000 to monetize its search and later turned it into a self-service portal to give advertisers high levels of control.

The state-of-the-art in targeted advertising is now online. The three major search engines are also the three largest online ad agencies, reaching 80% of the world [48]. All three provide Sponsored Search and Contextual Advertising. In the former, ads are shown alongside search results and can be highly relevant because the user immediately reveals an information need with their query. In the latter, small ads are inserted into generic webpages based on multiple relevance factors (Section 1.5). Sponsored search is more effective with conversions [48], but contextual advertising helps monetize much

[1]The first 'Yellow Pages' was published in the 1880s.

of the web today. Despite persistent challenges from ad-blocking and click-fraud, online advertising is a big business (e.g. it forms 97% of Google's revenue [47]).

Mobile advertising

Although a few years younger than its online counterpart, mobile advertising is growing rapidly and Strategy Analytics projects expenditure to boom 900% over the next 3 years. The two main types of mobile ads today are advertiser-to-user messages and content-embedded ads.

Message ads are prominently SMS, EMS and MMS. Each is a form of telemarketing that is more intrusive than embedded ads because messages can not be ignored—phones flag their arrival with an icon or pop-up and users must dismiss the notification to return the phone to its prior state. Users must also manually delete the ad even if they do not open it. But message ads suffer from bigger problems: poor timing, poor accuracy and cost of repeated transmission. They are not timely because they usually arrive when the phone is on standby. Phones that beep to notify the arrival of a message may even interrupt the user from another task.

Message ads are often targeted using demographics and focus groups. Individuals may also be targeted if the advertiser knows their interests. This is typically done using a questionnaire, but unfortunately the practice is invariably opt-in and suffers from low participation, thereby failing to make optimal use of the customer base. Even when user-targeted, message ads do not address the problem of poor timing.

Embedded ads are much more timely. They are typically injections, prepended clips or web ads. Injections occur when a message passes through some intercepting server that parses it, inserts an ad and forwards it to the recipient. Prepended clips are short audio-video segments played before a normal clip. Web-embedded ads are online ads viewed on the mobile browser.

While they primarily generate revenue for the website instead of the mobile service provider, web ads are still highly interesting to both service providers and users. From

the provider's point of view, a user click means more bandwidth and more revenue. From the user's point of view, web ads often relate to the page content, may be relevant to their interest, and are always up to date. One disadvantage of web ads is that they handle only web browsing, which is only one of the many types of VAS. Another is that they cost users money (for bandwidth) even if they are ignored. This somewhat limits web ads to text because mobile users who are not interested in ads will not want to spend their bandwidth on image or video ads. In this sense, the described framework complements web ads by showing ads of similar timeliness and relevance for almost any VAS content. Users who normally ignore web ads (i.e. hyperlinks to websites) may be interested in this system because it can show various other kinds of content-relevant ads (such as videos or interactives) and may even reduce the user's phone bill by subsidizing service cost.

It is also worth highlighting the increased use of secondary communication channels such as Wi-Fi and Bluetooth, where users are not always downloading content from the service provider or from the web—they may download it from their laptop, a kiosk, another phone, and so on. The framework may heighten user experience in these cases if it can show ads relevant to the user's viewed content. In particular, users may enjoy the system because it can show entertaining ads such as images, movies and interactives [10] that are already stored on the device.

The first MVNOs to substantially subsidize mobile costs are recent. In 2007, Blyk.co.uk launched a free service where users were required to view ads in exchange for mobile credit. Despite being UK-only, age restricted and invitation-based, the service achieved reasonable success by reaching its 100k customer target in half the expected timeframe. In 2008, ComTel.com.au announced a similar service where users would receive 5 SMS ads per day. Both companies rely on SMS for delivery. According to Empowered [31], 40% of surveyed users say they "pay too much" for mobile use and 50% would consider receiving SMS ads if ads reduced costs and were relevant to their general interests. If the market accepts SMS ads tailored to general interest then it should be more welcoming of timely ads tailored to immediate or specific interest.

Figure 5.2: Ads can now be timely and relevant, being shown in response to user interactions. Here an SMS exchange invokes the ad-selector, which finds a potentially useful ad of a movie trailer. A teaser is temporarily overlaid (e.g. as a banner along the top, in free space) to invite the user to click it and see the ad content.

Improving mobile advertising?

The imbalance of reach and effectiveness between online and mobile methods may lead us to think that perhaps a better way to advertise on mobiles is to simply 'dial a search engine'. Indeed, many types of displayed content have a text representation and engines excel at analyzing text. But this idea raises several questions. What content does the phone send? Isn't it private? And how often should it contact the engine? For every user action? Who will pay for this bandwidth? And what about latency? Is money lost if the user has switched activities by the time the ad arrives?

The proposed framework places a small search engine within the phone, thus resolving the above questions. In particular, it provides a filtering and selection method that can work under the heavy processing constraints imposed by small devices. The experiments shows that it can filter through many ads with fast response time, achieves good accuracy and is scalable.

The system has multiple advantages over current mobile marketing since it addresses timeliness, relevance, localization and personalization for various content:

- ads can be shown for heterogeneous activity, including SMS, e-books, music and video downloads, web browsing, etc.

- ads are more relevant because they are content-targeted to what is currently being viewed.

- ads are timely because they relate to the current activity. For instance, if reading (or writing) a message '*want to see a movie?*' to a friend, the user is likely thinking about her friend and about movies. A very basic query would be simply 'movie'. Retrieved ads may vary in presentation and format—a box office trailer, a TV schedule, a link to IMDB, a cinema ticket voucher—but all would relate to her thoughts. As is it sometimes desirable not to show an ad [21], the ad-selector can refrain using a back-off function or minimum confidence threshold.

- ads can be localized because the ad-database is region specific, e.g. a Palo Alto pizzeria would not have its ad on phones in New York.

- ads can be personalized by adaptively learning a user profile (on the phone) and using it to influence future selections and future updates to the ad-database.

- ads can save users money by subsidizing service costs.

- the ad-database is exchangeable and allows domain-specific ads to be pre-packaged with content, e.g. a 'digital tourism guide' that a user can download to their phone at a Bluetooth kiosk at an airport may come packaged with tourism ads.

These properties lead to a cheaper service price, richer user experience, more effective means of reaching the market for advertisers and more customers for the service provider.

5.3 Approach

5.3.1 Ad-database

The ad-database consists of ad records with an ID, title and set of tags. The tags are used for indexing, retrieval, relevance scoring and personalization.

Fig 5.3 shows an ad entry. `ID` is an ad's machine reference, delegated by the ad-broker, which allows simple internal referencing and updates. `Title` is how humans reference the ad and is included for two reasons. One is that the small teaser banners, such as Fig 5.2,

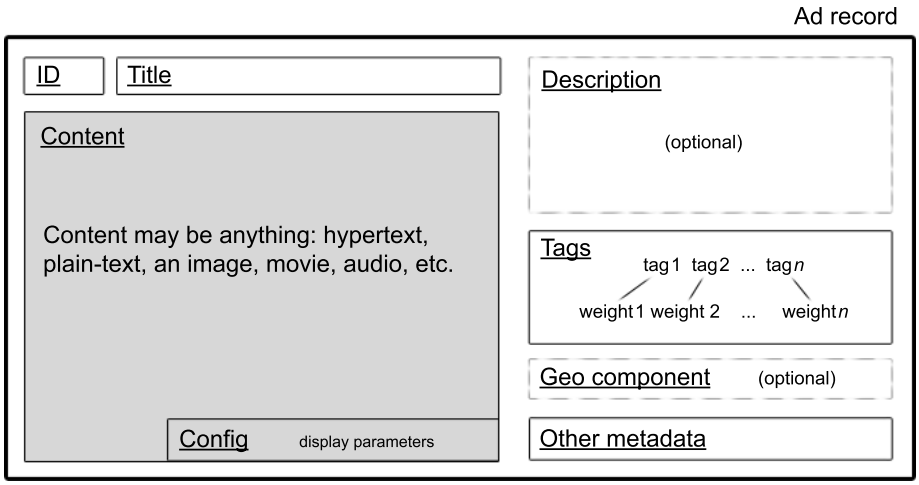

Figure 5.3: Abstract ad representation.

will usually need a text label. The other is that titles are a useful additional supporting feature for matching [115]. The `Description` is an optional text forming a second such feature.

`Content` is the ad itself and `Config` are its parameters such as compression and encoding. The **Metadata** are extra fields tracked for updating and bookkeeping purposes. Useful metadata fields include checksums, base ratings, expiry, timestamps (latest update, view, click) and total views and clicks.

`Geo` is an optional component defining the ad's relevance area, which will be used to boost or penalize candidate ads that are near or far from the user. A basic representation is a set of ⟨latitude, longitude, radius⟩.

The `Tags` are text annotations that help index and retrieve ads efficiently. They also allow a fast relevance approximation, scoring and personalization (Sections 5.3.2.2, 5.3.2.3, 5.3.2.6). A tag may be a keyword or phrase that describes the ad itself, such as "iphone". This is something that many advertisers are already familiar with from their experiences with online advertising. A tag may be also be a special token representing a classification, such as "category:electronics". This provides an extra feature for matching and has been shown to improve accuracy [22, 75]. Finally, a tag may also be a special syntax string of no particular semantic meaning that is added to ads and content that are authored by the same institution. This allows a company to have its own ads shown for its own content.

Each tag will have a weight attached to represent its *strength*. Advertisers desire high levels of control over tag choice and billing, therefore the weight may incorporate the tag's relevance as well as its bid and budget health. However, this chapter focuses on timing and accuracy rather than billing. The financial details are out of scope and therefore the experiments treat the tag weights as content descriptors. In practice the other two types of tag will no doubt be useful and the categorical tags can be built in a way similar to Section 3.2.5's classifier for news headlines.

Updates and administration

The ad-broker's role involves contracting with advertisers and provision of updates. The updates may be divided into region-specific patches so that the ad-databases are localized. For convenience and privileged network access, the broker may be the MNO. In terms of size, the broker's master database of ads will be much larger than the phones' ad-databases. This is because the broker may want to track new ads, old ads, and ads for multiple regions, whereas individual devices only have a small database for the user. Ideally the ad-database should support as many ads as possible provided the phone can search them within reasonable time. Giuffrida et al. [42] report that a mobile marketing company inducts 20–30 ads per day, so supporting 30×365=11k ads would be a start.

The phone will need to return some feedback to the broker so that advertisers can receive statistics on their campaigns. Advertisers will at least want to see CTR and individual tag performance, meaning that clicks and their triggering tags may be part of bookkeeping. However, other personal data can remain on the device.

The remainder of this section constructs a prototype ad-selector to be evaluated in Section 5.4.

5.3.2 Ad-selector

The ad-selector filters suitable ads for some input text query. In a commercial system the ad-selector would need to obtain its input automatically. For instance, to receive the SMS

exchange in Fig 5.2 it needs to be invoked by the phone's OS, be part of the OS, or be a registered listener to message events. Even after an ad is selected the agent must decide where and how to display it. However, these are out of scope because the chapter is only concerned with search and selection. Hence, the remainder of this section will assume that the ad-selector's role is to convert a piece of text into a set of ads with assigned probabilities of their selection. The selection steps are:

1. **Transforming** the input text into weighted tags.
2. **Query expansion** to add some extra tags.
3. **Retrieval** of candidate ads.
4. **Prefiltering** to discard weak candidates based on their approximated relevance.
5. **Scoring** the passed candidates with a more in-depth manner.
6. **Postfiltering** to discard weak candidates after scoring.

The final set of candidates enter a draw ('lottery') where selection is randomized using the probabilities defined by the ad's relevance scores. The implementation of these steps in the prototype are described next.

5.3.2.1 Transformation and expansion

While not strictly part of the selection process, accepting arbitrary text as input makes the ad-selector more versatile. An ideal input query would be weighted, human-assigned tags that are assigned in a similar way as how advertisers would have assigned the ads' tags (viz. an ESPGame [132] between users and advertisers), but unfortunately most content does not have human tags—or any tags at all—and so this information needs to be approximated algorithmically.

The transformation is relatively simple, extracting salient TFIDF words and growing the top terms to obtain a tag-form query. This transformation process involves case-folding, tokenizing, stopword removal, stemming, TFIDF and query expansion. The calculations require a stemmer, stoplist, IDF table and word relationship table to be kept on the device. The experiments section uses a Porter stemmer, 600-token stoplist, 90k-

token IDF table and 60k-token relationship table, all of which total 0.1% of an iPhone's capacity. In particular, the relationship table is the same as in Sections 4.2.1.3–4.2.1.4.

The top ten TFIDF terms in the input are query expanded by adding at most 50 related terms each from the relationship table. This expansion typically increases the tag-form query by twelve times, as shown in Table 5.2. The new tags' weights are set to $\frac{1}{4}$ of the product of the tag strength of the tag they were retrieved for and the relatedness strength from the relationship table. Consequently the terms are of a lower weight and lower importance than the original tags. For example, for the input "*want to see a movie tonight?*" the tags are simply "movie" and "tonight". They would be expanded to create a tag-form query of 34 terms that includes "film", "television", "actor", "night", "evening", and so on, with a low weight.

5.3.2.2 Retrieval and prefiltering

Query expansion substantially increases recall but many of the matched candidates ads will be weak. For performance reasons they must be filtered so that only promising ads progress to the scoring step. Differentiating the promising from the weak requires approximating their relevance, and this step can be implemented within the retrieval step.

Retrieval: candidates are ads from the inverted index that match at least one tag in the expanded tag-form query. To approximate their relevance, sum the products of their IDF, weight in the query and in the ad.

Prefilter: weak candidates are discarded to keep only the top-k, as ordered by their approximated score. Determining k may use a score-based or size-based cutoff. Since the prefilter is needed for performance reasons it is better to use size-based. In general k can be a function of the database size, and the experiments set k=50. Note that because the ad-selector is concerned with choosing a single ad, it does not need to maximize recall.

5.3.2.3 Scoring

The previous two steps provided a weighted tag-form query and a set of promising candidates. An ad A can now be scored more thoroughly for the query Q using its tags, location and personalization:

$$s_{A,Q} = base(A) + rel_{A,Q}\,(1 + \alpha\ g(A))\,(1 + \beta\ p(A))$$

where $base(\cdot)$ is a base rating, rel is a relevance score (Section 5.3.2.4), $g(\cdot)$ is a function returning a penalty or boost in [-1,1] for A's location (Section 5.3.2.5) and $p(\cdot)$ is a personalization factor in $[0, 1]$ (Section 5.3.2.6).

The ad will receive a low score if its rel is low or if it specifies a location and the location is inappropriate. It will receive a high score if its rel is high and even higher if it is aligned to the user's interests or has an appropriate location. The parameters α and β are tunable but from preliminary testing α=1 and β=4 are suitable. Note that $g(\cdot)$ has a positive, neutral or negative effect while $p(\cdot)$ is only positive or neutral. This is because it is possible to adaptively learn a person's likes but not so easily their dislikes.

The base score is useful for ads that always have a positive relevance and qualify as candidates for any query. This may include service provider bulletins, but it would be more suitable for temporarily boosting ads that the user has historically shown strong interest in and has not yet clicked since their most recent update. The magnitude of the rating affects the increase in selection probability. A typical ad that may suit this is the movie session times at the cinema, which are usually updated weekly.

5.3.2.4 Relevance

The relevance between the ad tags and query tags is implemented as a greedy sum:

$$rel_{A,Q} = \sum_{t \in (A.tags \cap Q.tags)} w(t, A.tags) \cdot w(t, Q.tags) \cdot IDF_t$$

where $w(t, \cdot)$ is a weight in [0,1] and IDF measures rarity. An obvious improvement is to use the Fig 5.3 title and description as supporting matches, but these were not used in Section 5.4 to show the minimalistic result of using tags exclusively.

5.3.2.5 Location

The ad-selector can make location-sensitive choices if it knows its location. Aside from GPS, a mobile can obtain its location by other means, such as by letting the user place a marker on a digital map, prompting the user for a zipcode or triangulating a position using signal strength from known call towers.

Section 5.3.1 specified location as optional because many ads are location-independent. For example, the Coca-Cola logo and many websites are globally relevant. But business such as restaurants and shops have fixed addresses and can benefit from location sensitivity. Location sensitivity is somewhat out of scope because the ads in the experiments section are location-independent (they are web ads), but the aim of this section is to show that it is possible to add location functionality to the ad-selector in a way that is fast to compute and produces useful values.

The geography function $g(\cdot)$ acts as a boost or penalty. It does not need to be symmetric or continuous but it needs to be clamped to $[-1, 1]$ to control its effect. Desiderata for $g(\cdot)$ are that ads with no location are unaffected, being close to a focused ad with small radius is better than being close to an unfocused ad, being within radius should never incur a penalty, and that we need some leeway at the radius edge because it is not always intuitive for advertisers to specify. Below is a piecewise example that satisfies these:

$$g|_r^d = \begin{cases} 0 & \text{if unspecified} \\ 1/\sqrt{\log d \cdot \log r} & \text{if } d \leq r \\ (g|_r^r + 1)\frac{3}{4}^{\log((\frac{1+d}{1+r})-1)} - 1 & \text{if } d > r \end{cases}$$

where r is radius, d is user distance and log is a non-negative $\log_2(2 + x)$. In the third case $g(\cdot)$ takes the value at the edge and reduces it by a fixed amount each time the user's distance effectively doubles. The maximum boost is when the ad is focused and the user is there while the maximum penalty is when it is focused and the user is infinitely far.

Suppose a small pizzeria targets customers within a 3 mile radius. By not giving

its location the pizzeria's ad will compete against other ads, such as a national pizza chain, based on tag calculations only. By giving a location it can gain a boost when the situation is economically favorable, i.e. when the user is close. If a user is 1 mile from the pizzeria, its ad score will be boosted by $g|_3^1 = 52\%$. A user 10 miles away will not have it boosted and any users farther away will have it penalized. At 50 miles the ad would be penalized by 40%, which favors the national chain and may result in the pizzeria's ad being discarded by the postfilter.

5.3.2.6 Personalization

People naturally have both persistent and transient interests, which we refer to as long-term (LT) and short-term (ST). The ad-selector can model these using tags, build a simple user profile and align its selections with the profile.

Passive observation has several advantages over the 'questionnaire approach': it requires no user effort, it is self-updating, it can glean fine-grained interests and it can track ST interest. LT interest is modeled as repeated confirmations of ST interests and ST interests as ad clicks. For example, the transient interests for a computer engineer may be {*movie, wolverine, jackman*}, perhaps after viewing the Wolverine trailer ad of actor Hugh Jackman. Her persistent interests may be broader tags such as *technology*, *internet*, *movie* and *software*. Meanwhile, a novelist's persistent interests may include *book*, *movie*, *news*, *music*, etc. Both users share a movie interest, but their other unique tastes can influence the ad-selector into choosing different ads for these users given the same database and input. The two interests are modeled using tags as follows:

long term: the most frequent tags from historical ad clicks. Their interest score is assigned by normalizing the frequencies.

short term: tags from recently clicked ads are added to the top of the ST list and gradually pushed down. Their interest score is initially 1 and the whole list is decayed by a factor $0 < d < 1$ whenever a new ad is clicked, when n ads in a row are shown but

ignored, or every h hours. A tag is dropped from the list when its score falls below a s_{min} threshold.

From preliminary testing suitable values are d=0.8, n=8 and $s_{min} < 0.1$. Suppose the lists are labeled LT and ST, each mapping tags to interest scores in (0,1]. A basic personalization function for ad A with tags $t_{1..n}$ would be to return 1 if any t_i is in either list and 0 otherwise. Using the interest scores and t_i's weight would give a more fine-grain scoring and we can further stipulate that it is better for A to match both lists. This gives a simple $p(\cdot)$:

$$p(A) = \frac{1}{2} \max_{t_1,t_2} \left(w_{t_1} ST[t_1] + w_{t_2} LT[t_2] \right)$$

5.3.2.7 Postfilter and final selection ('lottery')

Aside from location and personalization factors, the selection process so far produces the same result each time. If the device were to always choose the top candidate then there would be little variety in selections, especially if the same query is run moments later. It is therefore desirable to randomize the final selection and provide some variety. The candidate ads' scores have already been calculated and can now be used to determine their lottery probabilities. But before making the choice, a postfilter is needed to again discard weak ads. Despite passing the prefilter, these ads may have been weakened by underperforming in the scoring comparative to the other candidates (e.g. penalized for location). Unlike the prefilter, which is used for performance, the postfilter decides the final candidates and is critical to the final accuracy.

The filter can be either size-based (quantity) or score-based (quality). The advantage of size-based is it provides guarantee that there will be some variety in selections, whereas with score-based only the most promising ads would be considered even if there are very few of them. In either case, a filter that is too strong will pass too few ads and a filter that is too soft will pass too many weak ads. Some experiments are conducted in Section 5.4.2.2 to find a suitable postfilter.

The selection process for the sample ad-selector constructed in this section combines the steps from Sections 5.3.2.1-5.3.2.7:

---

ad-selector : set ad probabilities for transformed input Q.

1. $E := Q.tags \cup \bigcup_{t \in Q.tags}$ queryexpand(t)
2. $C := \{cand \in \text{ad-db} \mid cand.tags \cap E \neq \emptyset\}$
3. apply prefilter and take top k candidates
4. add ads with positive base rating

   //score candidates using more features
5. **for each** $c \in C$:
6. $\quad rel_{c,Q} := \sum_{t \in c.tags \cap E} w(t, c.tags) \cdot w(t, E) \cdot IDF_t$
7. $\quad s_{c,Q} := base(c) + rel_{c,Q}(1 + g(c))(1 + 4p(c))$
8. **end for**
9. apply postfilter and discard weak candidates
10. set probabilities using L_1-norm

The following section provides timing and accuracy experiments using this selection algorithm. The location-sensitivity and personalization calculations are disabled in the accuracy experiments because they alter the results in a user-sensitive and hard-to-reproduce manner.

5.4 Evaluation

The implementation of the Section 5.3 ad-selector being evaluated runs as a prototype on Java Wireless Toolkit 2.5.2, a mobile handset simulator. Two ad-databases, G-db and D-db, we compiled for testing as described next.

G-db: 430 real web ads from Google's search engine. The 114 generative queries used to fetch these ads came from Google's top searches[2]. The framework requires ads to be tagged with weighted tags, so keyphrase recommendations were generated using an automatic labeler [2]. From the recommendations, the 10 most recommended keywords (unigrams) are used as the tags (in some cases fewer than ten

[2]http://www.google.com/intl/en/press/zeitgeist2008/

were recommended). The tag weights were set as the normalized TFs among the recommendations such that 'best' tag received a unit weight. For the cases where the initial query's keywords were in the top ten or were not recommended, the query keywords were added as tags with a medium weight. (The query was added because it represented the only label that was virtually guaranteed to have been used by the advertiser for the ad to have been retrieved.)

D-db: 4278 Delicious items, as described in Chapter 4. These items are web bookmarks instead of ads, but we are only interested in their tagged nature and not their content. The tags were assigned by Delicious users and naturally have a weight represented by their frequency. The more popular the tag with users, the closer its weight to 1.

Note that D-db contains non-ads tagged by humans while G-db contains real ads tagged and rated mechanically.

5.4.1 Dataset independence

A first experiment is to show that timing is largely dataset independent. Three test ad-databases of equal sizes but different natures were constructed:

- 430 real ads from G-db; avg. 9.9 tags per ad, σ=1.5.
- 430 random ads from D-db; avg. 6.1 tags, σ=5.9.
- 430 carefully chosen D-db ads that resulted in the same tag distribution as G-db; avg. 9.9 tags, σ=1.5.

Fifty queries were executed on each of these ad-databases. The queries consisted of tags randomly chosen from the tagspace intersection such that each query would produce a result for each database and trigger all selection steps. Execution time is shown in Fig 5.4. Compared to G-db, the closeness of the middle column indicates that changing the

distribution's mean and variance has no noticeable effect while the third column indicates that there is little timing difference between human-assigned and machine-assigned tags.

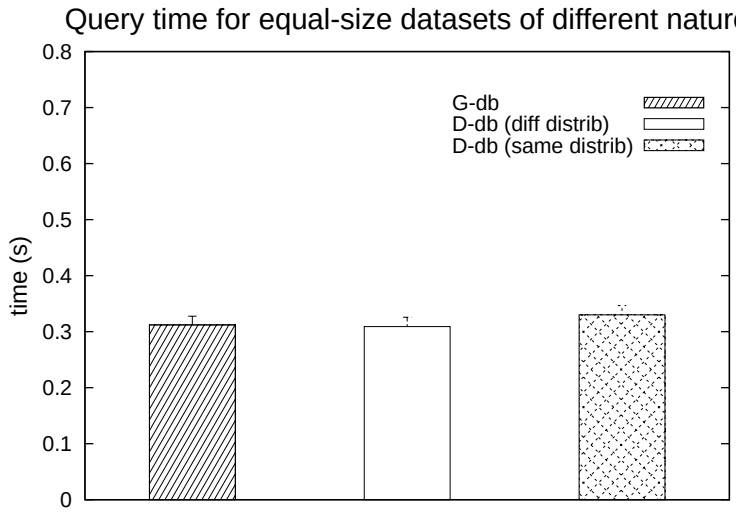

Figure 5.4: Mean processing time for ad-databases of differing tag sources and tag distributions.

5.4.2 Accuracy

Showing ads in response to user interactions makes them timely and Fig 5.4 shows that retrieval can be sufficiently fast, but to be effective ads also need to be accurate. Assessing accuracy requires a set of ads, queries and corresponding relevance judgments. The G-db collection of real ads was used for the following experiments.

5.4.2.1 Queries and relevances

Three variants were formed from a subset of the 114 generative queries used to compile the G-db:

- **Direct queries**: 35 of the 114 queries, randomly chosen. Examples: *software, coldplay, paint, java.*
- **Near queries**: 35 topically aligned queries, one for each direct query, that used different keywords. The near query was created from the direct query via refinement:

a human judge choose three new keywords from Google's related queries shown below the search results. The near queries for the above examples were: *computer hardware ipod, band tour lyrics, picture color interior, plugin flash microsoft.*

- **Mix queries**: the direct and near query as one.

Ads in G-db were then judged for relevance relative to the 35 direct queries only (i.e. not to the near query). Both 'strict' and 'kind' judgments were taken. Under strict judging ads needed to be "very relevant" while under kind judging ads needed to be "somewhat relevant". For example, for the query *travel*, strict judgments included ads for flights and accommodation, kind judgments included ads for Visas and Google Maps, but ads for outdoor camping or gift shopping were not considered relevant.

Since the direct queries were popular search queries, they generally consisted of 1–2 words and some were quite broad, therefore even strict judging accepted many ads: a median of 10 per query ranging from 4 for *ice cream* to 35 for *shopping*. Kind judging shows a median of 21 and range of 8–74.

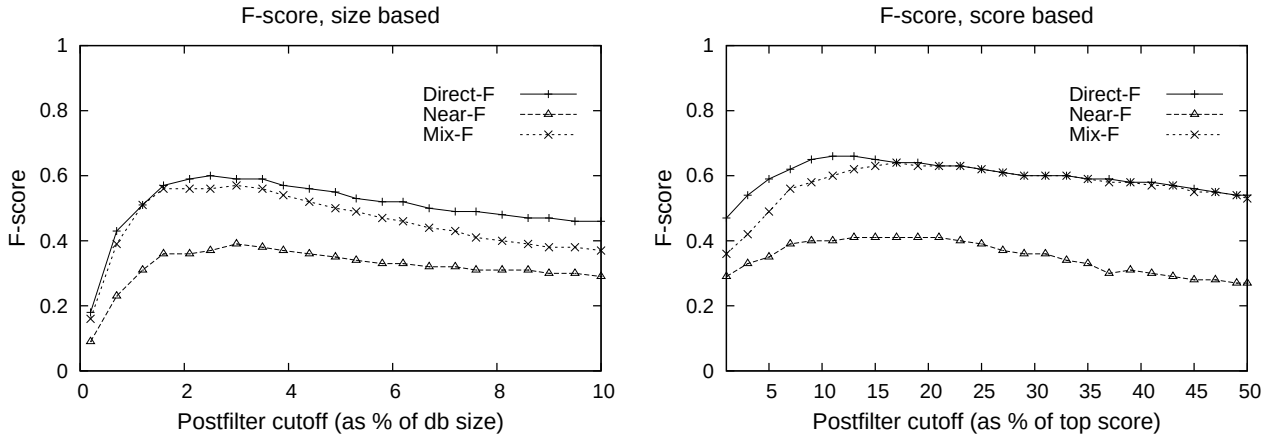

Figure 5.5: F_1-score for a size-based vs. score-based postfilter, using strict relevance judgments. The three query variations consolidate to peaks at 3.0% for size-based and 18% for score-based.

5.4.2.2 Finding a postfilter cutoff

The final step in selection before making a choice is a postfilter (Section 5.3.2.7) to remove weak ads. The candidates that pass the filter enter the lottery, but a question remains of how many ads to discard in a way that ensures some variety in selection but does not permit too many weak ads. The filter could use either a quality (score-based) or quantity (size-based) cutoff. We would expect that using too strong a cutoff will increase precision but decrease recall and may lower the user experience by not having enough variety in selections for the same query. On the other hand, too weak a cutoff would improve recall and selection variety, but decrease precision. A compromise can be found by calculating the F_1 measure [116] under both types of postfilters. The results in Fig 5.5 consolidate to maximum averages at 3.0% for size-based and 18% for score-based. The F_1 score for the direct queries at these points is 0.57 and 0.62, respectively. The difference is small but it appears that a score-based filter is a better choice. The next experiment will confirm this.

5.4.2.3 Precision and MAP

The previous experiment found suitable cutoffs for the two choices of postfilter. Their Precision and MAP is shown in Fig 5.6. At the optimized cutoffs, MAP is ≈0.5 for both but precision is much higher for the score-based filter.

MAP is a widely-used measure and gives some insight into both precision and recall. However, recall is somewhat misleading in this case because the ad-selector applies two filters to purposely discard ads for performance and user experience reasons. Applying these cutoffs naturally reduces recall and renders its final value meaningless. Instead, its *relative* is maximized as part of tuning F_1 and we can now focus only on precision.

Table 5.1 shows the P@(1,5,10,$|Lottery|$) at the two F_1 score maximizing cutoffs for the 3 query variants, using both postfilters, and under both strict and kind relevance judgments. The results confirm that score-based is the better postfilter because it has far better lottery precision.

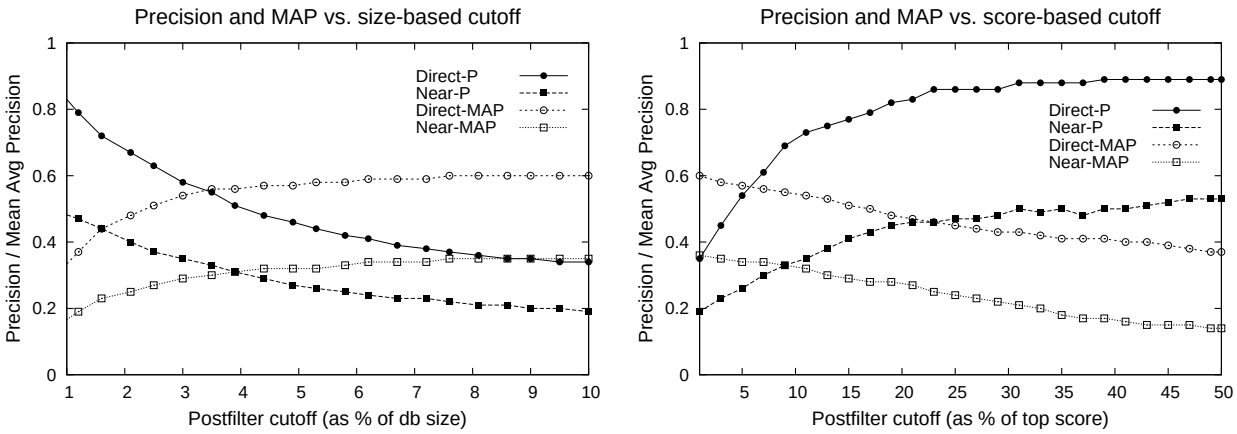

Figure 5.6: Precision and MAP for the lottery, using strict relevance judgments. The score-based cutoff (at its optimal 18%) outperformed size-based (at 3.0%) and was the better postfilter.

Judg.	Query/postf	P@1	P@5	P@10	P@|L|	Acc%	G
Strict	Direct score	.91	.75	.57	**.80**	**85**	(89)
Strict	Direct size	.91	.78	.62	.55	79	
Strict	Near score	.49	.46	.36	**.44**	**47**	(-)
Strict	Near size	.49	.47	.39	.35	45	
Strict	Mix score	.83	.74	.55	**.70**	**76**	(-)
Strict	Mix size	.83	.76	.58	.52	70	
Kind	Direct score	.97	.85	.69	**.94**	**96**	(99)
Kind	Direct size	.97	.91	.78	.73	92	
Kind	Near score	.74	.63	.50	**.63**	**67**	(-)
Kind	Near size	.74	.66	.56	.52	64	
Kind	Mix score	.94	.87	.70	**.87**	**91**	(-)
Kind	Mix size	.94	.90	.78	.67	84	

Table 5.1: Precision using both strict and kind judgments for three query variants under two possible postfilters. The ad-selector's final accuracy, as well as Google's for the same dataset, is on the right.

While P@n metrics reveal the goodness of a ranking, the ad-selector's choice is based not on the ranking but on the scores that defined the ranking order, i.e. the probabilities of the lottery candidates. Keeping in mind that the lottery has already been F_1-optimized, the more important measure of accuracy is the percentage of occasions where the final choice would be one of the query-relevant ads. This figure is given in the Acc% column in Table 5.1: the final accuracy of the prototype was 85% under strict judgment for direct

queries. For near queries, which used different keywords but looked for the same ads, it was nearly 50%.

Search engines represent the state-of-the-art in targeted advertising and, because the G-db was compiled from an engine's results, the same ads, queries and relevances could be used (retrospectively) to assess the engine's accuracy. This value is given in the last column of Table 5.1. The accuracy is very high for both systems, which can be explained. In the case of the engine, the queries were the engine's own top searches, which provide much historical CTR data for choosing ads that the users are most likely to click on. In our case it is likely a dual factor of the G-db being small and that the ads were labeled using the engine's keyword labeling assistant [2], which can assign intelligent labels using various data collected by the engine. A comparison with the engine is useful because the ad-selector is concerned with filtering ads that *are* labeled, not with how those labels were derived.

Note that the near and mix queries in Table 5.1 are included as reference, but the search engine's accuracy is not assessed for them because it returns completely different ads for those queries.

5.4.3 Scale

Determining scalability requires knowing the cost of each step. As this experiment concerns timing instead of relevance, the larger D-db can be used. Both the D-db and a 50% random sample of it were used to show the effects of doubling the number of ads.

Three sets of realistic queries were constructed (with 30 instances) and grouped according to their difficulty. Here difficulty refers to workload and selectivity instead of semantics and disambiguation. The query sets are described in Table 5.2.

Easy queries were single tags randomly chosen from D-db. Being indexed meant they would produce a match and trigger all selection steps. Being randomly chosen meant they would generally lie in the long-tail of rare tags, hence match few items. **Medium** queries were popular Delicious tags, which are very common in the D-db and produced

Set	Description of query set	|Input|	|Qry exp|	|Candidates|	Net time
Easy	Random tags from ad-db index, e.g. *nederlands*	1	12.3 tags	172 (σ=198)	0.26s (σ=.07)
Medium	Delicious top-30 popular tags, e.g. *science*	1	37.3 tags	878 (σ=330)	0.48s (σ=.07)
Complex	Online news stories, e.g. 'Sampras tips Federer to win Wimbledon. *Tennis great Pete Sa...* [320 words]'	280	119 tags	1386 (σ=165)	0.85s (σ=.10)

Table 5.2: Summary of three sets of queries on D-db that increase the number of candidates prior to prefilter. The step costs for these queries is shown in Fig 5.7.

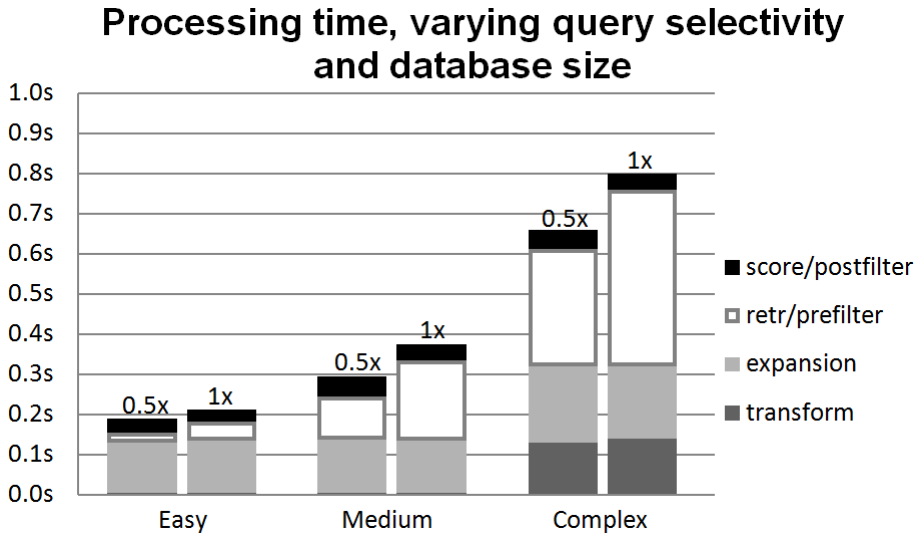

Figure 5.7: Time breakdown for Table 5.2 queries showing step cost. Also shows change in cost when doubling the database size.

many candidates for every query. **Complex** queries were several paragraphs of text from online news articles, randomly chosen from the same news site and on the same day. For the complex queries the input consisted of long passages and required more substantial use of the transformation step. The larger number of input terms resulted in larger query expansions and even more candidates (sometimes over 1/3 of the ad-database) that required prefiltering.

The step cost breakdown is in Fig 5.7 and shown as stacks. The bottom stack, the transformation step, used 0.05s per 100 input words and was invisible for short queries. Query expansion averaged 0.13s for the single tag queries and just 0.2s for complex. The time was kept low because only ten terms are expanded (Section 5.3.2.1). The retrieval

and prefilter, a dual-purpose step that repeatedly probes the index, was the slowest part. However, it is also the step that can be most improved because the prototype tested used a $O(\log n)$ index rather than $O(1)$. Finally the top stack, representing the scoring, location, personalization and postfiltering together, was equally fast because the prefilter controlled how many candidates reach the scoring step.

Note that some mobile activities, such as typing messages and watching videos, take longer than 1s and sometimes the ad-selector may have multiple seconds of time to respond. Nevertheless, comparing the 50% and 100% stacks shows that doubling the size does not double execution time. Extrapolating to 2x, the prototype should handle the Complex queries in 1s on a dataset of 9k. In particular, only the retrieval and prefilter step cost changes and this is the step for which performance can be improved the most.

Overall, the results are encouraging and suggest that we can place much of the intelligence of ad-serving and ad-selection onto a small device, fitting in a large number of ads and being able to perform sub-second searches with accuracy not far below that of the online search engines themselves.

5.5 Summary

The chapter describes a framework for mobile advertising that addresses the timing and accuracy problems of current mobile marketing approaches. The framework borrows ideas from the state-of-the-art in targeted advertising—that of web search engines—to present an efficient selection method for generic content. Experiments are provided to show that the ad-selector is reasonably fast, scalable and that given well-specified data has surprisingly good accuracy, yet can achieve these results purely on the client-side of a small device such a mobile. The contributions of the chapter include an introduction of the "phone as a search engine" problem and the selection method, which runs under the heavy constraints imposed by the mobile hardware and can serve as a baseline for future algorithms.

Although the chapter focuses on mobile service providers and mobile content, the concept has wider applications. Consider that some devices like Wi-Fi Netbooks and iPods do not persistently communicate with a service provider but still display various types of content. A similar framework and search algorithm will enable offline content-targeted advertising on such devices. The presented approach is also related to prefetch technologies such as Google Gears and Ajax in the sense that they aim to heighten user experience by prefetching some data onto the client-side and making the content feel more responsive. For these reasons the chapter may be of interest to advertisers and device manufacturers alike.

Search in the personal space of mobile devices is still quite a recent topic and various search problems are yet to be explored. However, it seems certain that the recent trend of "everything is going mobile" will continue into the near future such that more and more content will be available on mobiles. This means that users will be using small devices for more and more types of multimedia content, and therefore research on enabling search functionality in this personal space is both timely and valuable.

Chapter 6

Conclusions

6.1 Summary and Future Work

Chapter 1 made a case for the importance of search and made a distinction between search in impersonal and personal search spaces. It argued that users encounter personal spaces very frequently, which means that research results in the area are valuable. After defining the general topic area, Chapter 1 introduced each of the chapters and described their scope and related work. Those chapters are now summarized below.

Chapter 2 proposed a lightweight rearrangement algorithm for improving known-item search in hierarchical corpora by exploiting the structure of the hierarchy. The algorithm, PFH, was inspired by the authority and hubs model used in HITS. PFH was constructed to take in an α parameter to allow investigating how search performance over several metrics changed when the reliance between structure and content was varied. The algorithm computed a structural score for traditional content-only search results and distilled hubs (top folders) and authorities (top files). Based on the premise that hierarchical organization is meaningful and useful instead of random and useless, PFH favored results in structural hotspots and penalized the isolated matches that are often search noise. The experiments showed that α=0.8 resulted in a significant improvement in MRR while keeping other metrics steady. Although an optimal α may be corpus-

dependant, it can be estimated given a small set of representative queries and relevance judgments. Additionally the experiments showed that PFH can be used to predict correct folders for where to save a new file. A preliminary version of the work appeared in [100].

A future work for Chapter 2 is to analyze PFH's effectiveness and determine optimal α ranges for other hierarchical domains, such as books. As a simple example, the keyword 'HITS' appears 27 times in this thesis; the structure can be exploited to determine which are the most relevant hubs (chapter, section, subsection) and authorities (particular paragraphs) for this word. It is also worth investigating how to treat special agglomerative hubs that mix unrelated authorities together, such as the "Desktop" folder on a computer.

Chapter 3 leveraged the structure of the DOM for on-demand clustering of hyperlinks on a webpage. Links were first described with keyword labels, then the weights of the features were propagated through the DOM, and finally the features were clustered. The elastic property of the propagation step was able to keep links together when they did not share many features and could not be partitioned into logical groups. Several applications of the approach were identified: improving link finding for visually-impaired users, link organization suggestions for web authors and advertising for mobile news sites. The last of these applications also introduced a headline classifier. A preliminary version of the work appeared in [102].

Because Chapter 3 combines multiple research areas into one, there are several avenues for future work. First, it would be useful to expand the feature vectors with more words, such as anchor text windows or query expansion. Although the domain constraints limit how much processing can be devoted to the clustering step, it would also be useful to use these richer features for generating cluster labels. Another future work is to investigate heuristics that can help automatically determine the optimal number of output clusters for different sites. Furthermore, one of the applications required a classifier suitable for both news and ads, which may also be investigated further. For instance, fewer and more categories can be used to determine what is the optimal number of categories where the trade-off between too broad and too specific consolidates. An entirely different set of

categories may also be used, such as one that is optimized for the intersection of news topics and advertising topics. One way to determine such a category set could be to extract a large number of topics from a universal knowledge base (e.g. 1000 topics from Wikipedia) and to remove topics that describe too few items in either domain.

Chapter 4 focused on a problem that was identified in Chapter 3: how to effectively label a web link with keywords. The chapter investigated an online bookmarking system and proposed a scoring function, TagScore, to rate the goodness of user tags. TagScore's purpose was to calculate confidence scores for the web links and to enrich their tags so that they became more expressive, both of which allow the system to use the information more effectively behind-the-scenes for various search tasks. The experiments showed that TagScore was unbiased and produced a suitable distribution of confidence scores, which can be used for pruning. The enrichments were purely numeric and required minimal storage overhead while improving the correlation of the fast tag-tag comparisons against the slow page-page comparisons by two orders of magnitude. Several additional experiments tested the usefulness of a few types of metadata (page title, meta-keywords, synonyms and cooccurrences) for such a scoring function. The work is to appear in [105].

Because Chapter 4's method is applied on such a fundamental level in tagging systems, there are multiple future work avenues. The direct applications of TagScore include classical keyword search and approximate near neighbor search, but it may also be worth investigating its indirect use for other search-based tasks encountered in social networks and tagging systems, such as tag suggestion, user reputation scores, clustering, friend recommendation and item recommendation.

Chapter 5 returned to the mobile domain and proposed a framework for offline, content-targeted advertising on mobile devices. The chapter described a lightweight ad-selection agent suitable for implementation on a mobile and evaluated its execution in terms of time, scale and accuracy. The experiments showed promising results and suggested that it is possible to place most of the intelligence of ad-serving onto a phone. The offline property is attractive because it enables advertising on other small devices (e.g.

iPod) without a connection and without having advertisers know what content users are looking at. The work is to appear in [104].

Chapter 5 describes a practical problem with immediate benefits, although the commercial implementation of the system is more involved that what the chapter describes. There are several avenues for future work that deal with these extra considerations. For instance, the personalization and location-sensitivity components are briefly mentioned, and can be investigated further to best build user profiles and how to best determine location-suitability using GPS. The feedback of using the profile to adaptively personalize the database and selection method can also be further investigated. The distributed nature of the system complicates billing, so another future work is to handle bid optimization in a system with delayed reporting. Furthermore, in preference to a generic DBMS, a custom spatial database may be built that optimizes retrieval and update cost while supporting geographic queries (i.e. street-level relevant advertising). Additionally, it may also be worth investigating ways of presenting relevant ads on streaming mobile content such as mobile TV, which will require new considerations for timing and spacing of impressions.

6.2 Conclusion

With more information available at our fingertips than ever before, there is an even greater need to effectively organize and effectively search it. Search engines are continuously improving their ability to perform these tasks, with several breakthroughs in the past decade contributing to their improved effectiveness. However, the web represents an impersonal search space due to the user's lack of jurisdiction and familiarity over its content.

In this thesis I distinguish between the personal and impersonal, and highlight that users very often encounter personal search spaces, such as their own files, their bookmarks, their music, their emails, etc. I argue that the nature of queries in these spaces is typically

navigational and the the classical assumption of informational intent needs to be reworked. Each such personal space has its own constraints and domain-specific features, which must be considered individually when building search systems for each space.

In this thesis I investigate four such domains—searching for a file in a file hierarchy, website navigation on a mobile phone browser, tag search in an online bookmark system, and sponsoring content on mobiles—and present novel algorithms for each. The proposed methods are highly practical and applicable to real-life search problems. Overall each chapter proposes and evaluates a new method for search or navigation in a different personal space and, when taken together, this thesis advances the art and literature for search in such domains.

Web search engines have already started integrating vertical searches into their interfaces to synthesize webpage results with other types of content, such as images, videos, news, stocks, shopping, maps, restaurant reviews, quicklinks, timezones and more—all accessible from a single search box. Some search engines have also begun fusing the personal and the impersonal into one, via personalization (e.g. MyYahoo), collaboration (e.g. Google Wave) or combining offline and online search results into one (e.g. Google Desktop). Such trends of integration and amalgamation are likely to continue.

In the near future we may see technology advance to levels where we not only have a unified interface for the impersonal space, but also one for all of our personal spaces: a single search box to comb through the files on our laptops, the content on our phones, our email, bookmarks, photos and music, friends and contacts, social events, the video games on our consoles, the books and DVDs on our shelves, and maybe even the food in our fridge. The imagination can take us many places, but the same one-size-fits-all search models will not get us quite as far. It is therefore important to investigate all of these personal spaces individually and work within their constraints—that is how we can form the building blocks needed to construct such a unified search box.

Bibliography

[1] http://www.alexa.com/topsites.

[2] https://adwords.google.com/select/KeywordToolExternal.

[3] Apache Lucene. http://lucene.apache.org.

[4] *The Netsize Guide*. 2008. http://www.netsize.com.

[5] Ames, M., and Naaman, M. Why we tag: motivations for annotation in mobile and online media. In *CHI* (2007).

[6] Anderson, C., Domingos, P., and Weld, D. Adaptive web navigation for wireless devices. In *IJCAI* (2001).

[7] Baker, L. D., and McCallum, A. K. Distributional clustering of words for text classification. In *SIGIR* (1998).

[8] Balmin, A., Hristidis, V., and Papakonstantinou, Y. ObjectRank: Authority based keyword search in databases. In *VLDB* (2004).

[9] Bao, X., Herlocker, J., and Dietterich, T. Fewer clicks and less frustration: reducing the cost of reaching the right folder. In *IUI* (2006).

[10] Bauer, H., Barnes, S., Reichardt, T., and Neumann, M. Driving Consumer Acceptance of Mobile Marketing. *JECR 6*, 3 (2005).

[11] Begelman, G., Keller, P., and Smadja, F. Automated tag clustering: Improving search and exploration in the tag space. In *Collaborative Web Tagging Workshop, WWW* (2006).

[12] Bharat, K., and Henzinger, M. R. Improved algorithms for topic distillation in a hyperlinked environment. In *SIGIR* (1998).

[13] Blei, D. M., Ng, A. Y., and Jordan, M. I. Latent dirichlet allocation. *JMLR 3* (2003).

[14] Boardman, R., and Sasse, M. A. Stuff goes into the Computer and Doesn't Come Out: a Cross-tool Study of Personal Information Management. In *CHI* (2004).

[15] Bollegala, D., Matsuo, Y., and Ishizuka, M. Measuring semantic similarity between words using web search engines. In *WWW* (2007).

[16] Borgs, C., Chayes, J., Immorlica, N., Jain, K., Etesami, O., and Mahdian, M. Dynamics of bid optimization in online advertisement auctions. In *WWW* (2007).

[17] Borodin, A., Roberts, G., Rosenthal, J., and Tsaparas, P. Finding authorities and hubs from link structures on the World Wide Web. In *WWW* (2001).

[18] Borodin, Y., Mahmud, J., Ramakrishnan, I. V., and Stent, A. The HearSay non-visual web browser. In *W4A* (2007).

[19] Brin, S., and Page, L. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks 30*, 1-7 (1998).

[20] Broder, A. A taxonomy of web search. *SIGIR Forum 36*, 2 (2002).

[21] Broder, A., Ciaramita, M., Fontoura, M., Gabrilovich, E., Josifovski, V., Metzler, D., Murdock, V., and Plachouras, V. To swing or not to swing: learning when (not) to advertise. In *CIKM* (2008).

[22] Broder, A., Fontoura, M., Josifovski, V., and Riedel, L. A semantic approach to contextual advertising. In *SIGIR* (2007).

[23] Brooks, C. H., and Montanez, N. Improved annotation of the blogosphere via autotagging and hierarchical clustering. In *WWW* (2006).

[24] Buckley, C., and Voorhees, E. M. Evaluating evaluation measure stability. In *SIGIR* (2000).

[25] Chakrabarti, D., Kumar, R., and Punera, K. A graph-theoretic approach to webpage segmentation. In *WWW* (2008).

[26] Chakrabarti, S., Dom, B., Gibson, D., Kleinberg, J., Raghavan, P., and Rajagopalan, S. Automatic resource list compilation by analyzing hyperlink structure and associated text. In *WWW* (1998).

[27] Chakrabarti, S., Dom, B. E., Kumar, S. R., Raghavan, P., Rajagopalan, S., Tomkins, A., Gibson, D., and Kleinberg, J. Mining the Web's Link Structure. *Computer 32*, 8 (1999).

[28] Chen, Y., Ma, W.-Y., and Zhang, H.-J. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW* (2003).

[29] Chernov, S., Serdyukov, P., Chirita, P.-A., Demartini, G., and Nejdl, W. Building a desktop search test-bed. In *ECIR* (2007).

[30] Chirita, P.-A., Costache, S., Nejdl, W., and Handschuh, S. P-tag: large scale automatic generation of personalized annotation tags for the web. In *WWW* (2007).

[31] ComTel/Empowered Communications. Making mobile advertising a rewarding reality. Media briefing and market release, 25 Aug 2008.

[32] Crimmins, F. Connectivity Review. http://dev.panopticsearch.com/connectivity-review.html (accessible via archive.org), 2000.

[33] Dean, J., and Ghemawat, S. MapReduce: simplified data processing on large clusters. In *OSDI* (2004).

[34] Dean, J., and Henzinger, M. R. Finding related pages in the world wide web. *Computer Networks 31*, 11–16 (1999).

[35] Del Corso, G. M., Gullí, A., and Romani, F. Ranking a stream of news. In *WWW* (2005).

[36] Dragunov, A., Dietterich, T., Johnsrude, K., McLaughlin, M., Li, L., and Herlocker, J. TaskTracer: a desktop environment to support multi-tasking knowledge workers. In *IUI* (2005).

[37] Dumais, S., Cutrell, E., Cadiz, J., Jancke, G., Sarin, R., and Robbins, D. Stuff I've Seen: a system for personal information retrieval and re-use. In *SIGIR* (2003).

[38] Even Dar, E., Mirrokni, V. S., Muthukrishnan, S., Mansour, Y., and Nadav, U. Bid optimization for broad match ad auctions. In *WWW* (2009).

[39] Fang, X., and Sheng, O. R. L. Linkselector: A web mining approach to hyperlink selection for web portals. *TOIT 4*, 2 (2004).

[40] Furnas, G., Landauer, T., Gomez, L., and Dumais, S. The vocabulary problem in human-system communication. *Communications of the ACM 30*, 11 (1987).

[41] Ghemawat, S., Gobioff, H., and Leung, S.-T. The Google file system. In *SOSP* (2003).

[42] Giuffrida, G., Sismeiro, C., and Tribulato, G. Automatic content targeting on mobile phones. In *EDBT* (2008).

[43] Golder, S., and Huberman, B. The structure of collaborative tagging systems. Tech. rep., Hewlett-Packard Labs, August 2005.

[44] Golub, G., and Loan, C. V. *Matrix Computations*. 1989.

[45] Good, I. J. *The estimation of probabilities: an essay on modern Bayesian methods*. 1965.

[46] Google Inc. http://adwords.google.com/support/bin/answer.py?answer=10215.

[47] Google Inc. http://investor.google.com/fin_data.html.

[48] Google Inc. Google content network (2009). http://www.google.com/ads/-research/gcnwhitepaper/.

[49] Google Inc. Web authoring statistics (2006). http://code.google.com/webstats.

[50] Google Inc. Methods and apparatus for serving relevant advertisements. US2004/0059708, 2002.

[51] Granka, L. A., Joachims, T., and Gay, G. Eye-tracking analysis of user behavior in WWW search. In *SIGIR* (2004).

[52] Guo, L., Shao, F., Botev, C., and Shanmugasundaram, J. XRANK: ranked keyword search over XML documents. In *SIGMOD* (2003).

[53] Gupta, A., Kumar, A., Mayank, Tripathi, V. N., and Tapaswi, S. Mobile web: web manipulation for small displays using multi-level hierarchy page segmentation. In *MOBILITY* (2007).

[54] Hall, E. *The Hidden Dimension.* 1966.

[55] Halvey, M. J., and Keane, M. T. An assessment of tag presentation techniques. In *WWW* (2007).

[56] Hammond, T., Hannay, T., Lund, B., and Scott, J. Social Bookmarking Tools (I). *D-Lib Magazine 11*, 4 (2005).

[57] Harper, S., and Bechhofer, S. SADIe: Structural semantics for accessibility and device independence. *TOCHI 14*, 2 (2007).

[58] Henderson, S. Genre, task, topic and time: facets of personal digital document management. In *CHINZ* (2005).

[59] Heymann, P., and Garcia-Molina, H. Collaborative creation of communal hierarchical taxonomies in social tagging systems. Tech. Rep. 2006-10, Stanford University, April 2006.

[60] Heymann, P., Koutrika, G., and Garcia-Molina, H. Can social bookmarking improve web search? In *WSDM* (2008).

[61] Heymann, P., Ramage, D., and Garcia-Molina, H. Social tag prediction. In *SIGIR* (2008).

[62] Houle, M. E., and Sakuma, J. Fast approximate similarity search in extremely high-dimensional data sets. In *ICDE* (2005).

[63] Jain, A. K., Murty, M. N., and Flynn, P. J. Data clustering: a review. *ACM Computing Surveys 31*, 3 (1999).

[64] Jansen, B., Spink, A., Bateman, J., and Saracevic, T. Real life information retrieval: a study of user queries on the Web. *SIGIR Forum* (1998).

[65] Joachims, T. Text categorization with support vector machines: learning with many relevant features. In *ECML* (1998), no. 1398.

[66] Joachims, T., Granka, L., Pan, B., Hembrooke, H., and Gay, G. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR* (2005).

[67] Jones, W., Phuwanartnurak, A., Gill, R., and Bruce, H. Don't take my folders away!: organizing personal information to get things done. In *CHI* (2005).

[68] Kaser, O., and Lemire, D. Tag-cloud drawing: Algorithms for cloud visualization. In *Tagging and Metadata for Social Information Organization Workshop, WWW* (2007).

[69] Kleinberg, J. Authoritative sources in a hyperlinked environment. *Journal of the ACM 46*, 5 (1999).

[70] Koller, D., and Sahami, M. Hierarchically classifying documents using very few words. In *ICML* (1997).

[71] Koutrika, G., Effendi, F. A., Gyöngyi, Z., Heymann, P., and Garcia-Molina, H. Combating spam in tagging systems. In *AIRWeb* (2007).

[72] Kumar, R., Raghavan, P., Rajagopalan, S., and Tomkins, A. Trawling the Web for emerging Cyber-Communities. *Computer Networks* (1999).

[73] Kummamuru, K., Lotlikar, R., Roy, S., Singal, K., and Krishnapuram, R. A hierarchical monothetic document clustering algorithm for summarization and browsing search results. In *WWW* (2004).

[74] Lam, H., and Baudisch, P. Summary thumbnails: readable overviews for small screen web browsers. In *CHI* (2005).

[75] Le, D.-T., Nguyen, C.-T., Ha, Q.-T., Phan, X. H., and Horiguchi, S. Matching and ranking with hidden topics towards online contextual advertising. In *WI* (2008).

[76] Lee, D., and Seung, S. Learning the parts of objects by non-negative matrix factorization. *Nature*, 6755 (October 1999).

[77] Lempel, R., and Moran, S. SALSA: the stochastic approach for link-structure analysis. *TOIS* (2001).

[78] Lewis, D. D. *Representation and Learning in Information Retrieval*. PhD thesis, University of Massachusetts, 1992.

[79] Li, L., Shang, Y., and Zhang, W. Improvement of HITS-based algorithms on web documents. In *WWW* (2002).

[80] Lunn, D., Bechhofer, S., and Harper, S. The SADIe transcoding platform. In *W4A* (2008).

[81] Marchetti, A., Tesconi, M., Ronzano, F., Rosella, M., and Minutoli, S. Semkey: A semantic collaborative tagging system. In *Tagging and Metadata for Social Information Organization Workshop, WWW* (2007).

[82] Marchiori, M. The quest for correct information on the Web: Hyper search engines. *Computer Networks 29*, 8–13 (1997).

[83] Marendy, P. A review of world wide web searching techniques focusing on HITS and related algorithms that utilize the link topology of the world wide web to provide the basis for a structure based search technology. http://net.pku.edu.cn/~webg/papers/a-review-of-world.pdf (accessible via archive.org), 2001.

[84] Marlow, C., Naaman, M., Boyd, D., and Davis, M. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In *HYPERTEXT* (2006).

[85] Masand, B., Linoff, G., and Waltz, D. Classifying news stories using memory based reasoning. In *SIGIR* (1992).

[86] Mathes, A. Folksonomies - Cooperative Classification and Communication through Shared Metadata, 2004.

[87] Mehta, A., Saberi, A., Vazirani, U., and Vazirani, V. Adwords and generalized online matching. *JACM 54*, 5 (2007).

[88] Michlmayr, E., and Cayzer, S. Learning user profiles from tagging data and leveraging them for personal(ized) information access. In *Tagging and Metadata for Social Information Organization Workshop, WWW* (2007).

[89] Mika, P. Ontologies Are Us: A Unified Model of Social Networks and Semantics. In *ISWC* (2005), vol. 3729 of *LNCS*, Springer.

[90] Miller, G. A. WordNet: a lexical database for English. *CACM 38*, 11 (1995).

[91] Miller, J., Rae, G., Schaefer, F., Ward, L., LoFaro, T., and Farahat, A. Modifications of Kleinberg's HITS algorithm using matrix exponentiation and web log records. In *SIGIR* (2001).

[92] Mobasher, B., Dai, H., Luo, T., and Nakagawa, M. Effective personalization based on association rule discovery from web usage data. In *WIDM* (2001).

[93] Ng, H. T., Goh, W. B., and Low, K. L. Feature selection, perceptron learning, and a usability case study for text categorization. In *SIGIR* (1997).

[94] Noda, T., and Helwig, S. Benchmark Study of Desktop Search Tools. *University of Wisconsin-Madison E-Business Consortium* (2005).

[95] Noll, M. G., man Au Yeung, C., Gibbins, N., Meinel, C., and Shadbolt, N. Telling experts from spammers: expertise ranking in folksonomies. In *SIGIR* (2009).

[96] Noll, M. G., and Meinel, C. Authors vs. Readers - A Comparative Study of Document Metadata and Content in the WWW. In *ACM DocEng* (2007).

[97] Osiński, S., Stefanowski, J., and Weiss, D. Lingo: Search results clustering algorithm based on Singular Value Decomposition. In *IIS* (2004).

[98] Outland Research. Method and apparatus for improving the matching of relevant advertisements with particular users over the internet. US2006/0206379, 2006.

[99] Page, L., Brin, S., Motwani, R., and Winograd, T. The PageRank Citation Ranking: Bringing Order to the Web. Stanford Digital Library Technologies Project, 1998.

[100] Penev, A., Gebski, M., and Wong, R. K. Topic distillation in desktop search. In *DEXA* (2006).

[101] Penev, A., and Wong, R. K. Finding similar pages in a social tagging repository. In *WWW* (2008).

[102] Penev, A., and Wong, R. K. Grouping hyperlinks for improved voice/mobile accessibility. In *W4A* (2008).

[103] Penev, A., and Wong, R. K. Tagscore: Approximate similarity using tag synopses. In *WI* (2008).

[104] Penev, A., and Wong, R. K. Framework for timely and accurate ads on mobile devices. In *CIKM* (2009). To appear.

[105] Penev, A., and Wong, R. K. Using metadata for social tag search. *WIAS 7* (2009). To appear.

[106] Perkowitz, M., and Etzioni, O. Adaptive Web Sites: an AI Challenge. In *IJCAI* (1997).

[107] Phan, X.-H., Nguyen, L.-M., and Horiguchi, S. Learning to classify short and sparse text & web with hidden topics from large-scale data collections. In *WWW* (2008).

[108] Porter, M. An algorithm for suffix stripping. Readings in Information Retrieval, 1997.

[109] Qin, T., Liu, T.-Y., Zhang, X.-D., Chen, Z., and Ma, W.-Y. A study of relevance propagation for web search. In *SIGIR* (2005).

[110] Quinlan, J. R. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers, 1993.

[111] Ramage, D., Heymann, P., Manning, C. D., and Garcia-Molina, H. Clustering the Tagged Web. In *WSDM* (2009).

[112] Ramakrishnan, I. V., Stent, A., and Yang, G. HearSay: enabling audio browsing on hypertext content. In *WWW* (2004).

[113] Ravasio, P., Schär, S. G., and Krueger, H. In Pursuit of Desktop Evolution: User Problems and Practices With Modern Desktop Systems. *TOCHI 11*, 2 (2004).

[114] Reuters/Informa. 'Global cellphone penetration reaches 50 pct', November 29 2007.

[115] Ribeiro-Neto, B., Cristo, M., Golgher, P. B., and Silva de Moura, E. Impedance coupling in content-targeted advertising. In *SIGIR* (2005).

[116] Rijsbergen, C. J. v. *Information Retrieval*. Butterworths, London, 1979.

[117] Rivadeneira, A. W., Gruen, D. M., Muller, M. J., and Millen, D. R. Getting our head in the clouds: toward evaluation studies of tagclouds. In *CHI* (2007).

[118] Robertson, S., and Sparck-Jones, K. Relevance Weighting of Search Terms. *JASIST 27*, 3 (1976).

[119] Sahami, M., and Heilman, T. D. A web-based kernel function for measuring the similarity of short text snippets. In *WWW* (2006).

[120] Salton, G. *Automatic Text Processing: the Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[121] Salton, G., and McGill, M. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[122] Samsung. A method and system for providing sponsored content on an electronic device. EP1968001, 2008.

[123] SanDisk. Method for advertising on mobile devices. US2008/0108337, 2007.

[124] Schenkel, R., Crecelius, T., Kacimi, M., Michel, S., Neumann, T., Parreira, J. X., and Weikum, G. Efficient top-k querying over Social-Tagging Networks. In *SIGIR* (2008).

[125] Schmitz, P. Inducing ontology from flickr tags. In *Collaborative Web Tagging Workshop, WWW* (2006).

[126] Soules, C., and Ganger, G. Connections: using context to enhance File Search. In *SOSP* (2005).

[127] Spink, A., Wolfram, D., Jansen, B., and Saracevic, T. Searching the Web: The public and their queries. *JASIST 52*, 3 (2001).

[128] Tatarinov, I., Viglas, S. D., Beyer, K., Shanmugasundaram, J., Shekita, E., and Zhang, C. Storing and querying ordered XML using a Relational Database System. In *SIGMOD* (2002).

[129] The Beagle++ Project. http://beagle2.kbs.uni-hannover.de.

[130] Tonkin, E., and Guy, M. Folksonomies: Tidying Up Tags? *D-Lib Magazine 12*, 1 (2006).

[131] Turpin, A. Personal correspondence, November 2006.

[132] von Ahn, L., and Dabbish, L. Labeling images with a computer game. In *CHI* (2004).

[133] Wang, X., Broder, A., Fontoura, M., and Josifovski, V. A search-based method for forecasting ad impression in contextual advertising. In *WWW* (2009).

[134] Wermter, S., and Hung, C. Selforganizing classification on the reuters news corpus. In *COLING* (2002).

[135] Western Digital. Caching advertising information in a mobile terminal to enhance remote synchronization and wireless internet browsing. US6826614, 2001.

[136] Whittaker, S., Terveen, L., and Nardi, B. Let's stop pushing the envelope and start addressing it: a reference task agenda for HCI. *HCI 15* (2000).

[137] Xiao, X., Luo, Q., Hong, D., and Fu, H. Slicing*-tree based web page transformation for small displays. In *CIKM* (2005).

[138] Xu, Z., Fu, Y., Mao, J., and Su, D. Towards the semantic web: collaborative tag suggestions. In *WWW Workshop* (2006).

[139] Yang, Y. Expert network: effective and efficient learning from human decisions in text categorization and retrieval. In *SIGIR* (1994).

[140] Yang, Y. An evaluation of statistical approaches to text categorization. *Journal of IR 1* (1999).

[141] Yang, Y., and Liu, X. A re-examination of text categorization methods. In *SIGIR* (1999).

[142] Yih, W.-t., Goodman, J., and Carvalho, V. R. Finding advertising keywords on web pages. In *WWW* (2006).

[143] Yih, W.-t., and Meek, C. Improving similarity measures for short segments of text. In *AAAI* (2007).

[144] Yin, X., and Lee, W. S. Using link analysis to improve layout on mobile devices. In *WWW* (2004).

[145] Zamir, O., and Etzioni, O. Grouper: a dynamic clustering interface to Web search results. In *WWW* (1998).

[146] Zamir, O., and Etzioni, O. Web document clustering. In *SIGIR* (1998).

[147] Zeng, H.-J., He, Q.-C., Chen, Z., Ma, W.-Y., and Ma, J. Learning to cluster web search results. In *SIGIR* (2004).

[148] Zha, H., He, X., Ding, C., Simon, H., and Gu, M. Bipartite graph partitioning and data clustering. In *CIKM* (2001).

[149] Zhang, Y., Chu, C.-H., Ji, X., and Zha, H. Correlating summarization of multi-source news with k-way graph bi-clustering. *ACM SIGKDD Explorations Newsletter 6*, 2 (2004).