Using context to assist in personal file retrieval

CRAIG A. N. SOULES

August 25, 2006

CMU–CS–06-147

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh PA

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Thesis committee

Prof. Gregory R. Ganger, Chair
Prof. Jamie Callan
Dr. Jim Gemmell (Microsoft Research)
Prof. Christopher Olston

This work was funded in part by the past and present members and companies of the PDL Consortium (including APC, EMC, Equallogic, Hewlett-Packard, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate, Sun, and Symantec), and by USENIX. This material is based on research sponsored in part by the Air Force Research Laboratory, under agreement number F49620–01–1–0433.

To my father.

Abstract

Personal data is growing at ever increasing rates, fueled by a growing market for personal computing solutions and dramatic growth of available storage space on these platforms. Users, no longer limited in what they can store, are now faced with the problem of organizing their data such that they can find it again later. Unfortunately, as data sets grow the complexity of organizing these sets also grows. This problem has driven a sudden growth in search tools aimed at the personal computing space, designed to assist users in locating data within their disorganized file space.

Despite the sudden growth in this area, local file search tools are often inaccurate. These inaccuracies have been a long-standing problem for file data, as evidenced by the downfall of attribute-based naming systems that often relied on content analysis to provide meaningful attributes to files for automated organization.

While file search tools have lagged behind, search tools designed for the world wide web have found wide-spread acclaim. Interestingly, despite significant increases in non-textual data on the web (e.g., images, movies), web search tools continue to be effective. This is because the web contains key information that is currently unavailable within file systems: *context*. By capturing context information, e.g., the links describing how data on the web is inter-related, web search tools can significantly improve the quality of search over content analysis techniques alone.

This work describes Connections, a context-enhanced search tool that utilizes temporal locality among file accesses to provide inter-file relationships to the local file system. Once identified, these inter-file relationships provide context information, similar to that available in the world wide web. Connections leverages this context to improve the quality of file search results. Specifically, user studies with Connections see improvements in both precision and recall (i.e., fewer false-positives and false-negatives) over content-only search, and a live deployment found that users experienced reduced search time with Connections when compared to content-only search.

Acknowledgments

This work would not have been possible without the help and support of many individuals. As my adviser, Greg Ganger taught me how to read, write, and pursue research. His constant encouragement and near-miraculous ability to always find time for his students have made working with him a true pleasure. As members of my committee, Jamie Callan, Jim Gemmell, and Chris Olston helped to shape the direction of this work, filled in many of the gaps in my knowledge, and helped steer me toward solutions. Anind Dey was also invaluable in helping me design and execute the online user study for this work.

Throughout graduate school I've had the good fortune to work with a large number of excellent colleagues. When I first arrived, Garth Goodson, John Strunk, Mike Scheinholtz, and Jiri Schindler helped to get me on my feet. As the years progressed, my discussions and interactions with many students at CMU, particularly Mike Abd-el-Malek, John Griffin, David Petrou, Brandon Salmon, Steve Schlosser, Eno Thereska, and Jay Wylie, have always been both enjoyable and educational. I also thank M. Satyanarayanan for opening my eyes to research.

The Parallel Data Lab has been a great source of feedback from its many industry sponsors and has fostered a strong, collaborative environment for its many students. I thank Garth Gibson for founding the PDL and its current director, Bill Courtright, for continuing its excellent tradition. The PDL has also brought me into contact with a number of staff members, without whom my life would have been far more frustrating. Particularly, I thank Joan Digney, Karen Lindenfelser, and Linda Whipkey for their help throughout the years.

My life would not be the same without the many friends I have made here in Pittsburgh. My close friends Bryan Jacobs and Jeremy Richardson have kept my life both interesting and entertaining. The many other people of note include Sarah Aerni, Dan Arp, Erik Degelman, Court Heller, Reagan Heller, Maureen McGranahan, Dan Reed, Wil Paredes, Kevin Peterson, Rachel Schlosser, Reed and Erika Taylor, Shawn Wall, Marshal Warfield, and Ben Ziskind. I also thank the many instructors and students I've interacted with at Oom Yung Doe, and the members of the mechanical engineering softball team captained by Bryan Smith.

Finally, I thank my mother, Aline Soules, for her unwavering support throughout the years.

Contents

Figures

Tables

1 Introduction

In a perfect world, file systems would be omniscient, helping users organize, locate, and access their data so that they could always find what they want when they want it, acting almost like a personal assistant. Unfortunately, we are far from this ideal today. In fact, information management is one of the largest problems in computing today, as can be seen in the amount of time, money, and effort spent by both companies and individuals in addressing this problem.

And, the problem continues to grow. As computers become more pervasive, users generate and store more data, increasing the complexity of both organization and retrieval. Increases in network connectivity have improved both data accessibility and simplified data sharing, at the trade-off of users having to organize and retrieve more data across more locations.

1.1 The landscape today

Despite these increasing complexities, the interface of today's file systems are no different than they were thirty years ago. They provide only a mapping between a name and a piece of associated data (i.e., a file), leaving users unassisted with the task of organizing their data in such a way that they can find it again when it is needed. While this may have been sufficient in the past, recent increases in both the power and capacity of personal computers have drastically increased the amount of data generated by the average user, while simultaneously removing the need to delete old data. As our computers move from being bookshelves to being libraries, they must also move from being pieces of furniture to being librarians.

Before a file system can begin to assist the user in organizing or finding their data, it must first understand the data that it stores. What do a file's contents represent? How will the file be used? How and when will a user access the file? What is the file's relationship to both other files and user tasks? Without being able to answer these questions, the system will never be able to accurately determine or satisfy a user's information need.

One potential solution to this problem is attribute-based naming. Attribute-based naming systems allows users to classify each file with multiple attributes [35, 38, 67]. Once in place, these attributes provide the system with an understanding of a file's properties, giving it a chance to answer the questions posed above. Unfortunately, there remains the problem of generating useful and accurate attributes, as it is unrealistic and inappropriate to require users to proactively provide accurate and useful classifications. To make attribute-based naming systems viable, there must be methods to automatically classify files, and, in fact, this requirement has led most systems to employ search tools over hierarchical file systems rather than change their underlying methods of organization.

The most prevalent automated classification method today is content analysis: examining the contents and pathnames of files to determine attributes that describe them. Systems using attribute-based naming, such as the Semantic file system [35], use content analysis to automate attribute assignment. Search tools, such as Google Desktop [36], use content analysis to map user queries to a ranked list of files. However, a file's contents can only answer the first of the questions posed above, and then only for the files containing structured, indexable content (e.g., text). The problem then is finding additional sources of information that can provide information, automatically, to the system.

1.2 Context

Context is "the interrelated conditions in which something exists or occurs" [76]. Examples of a file's context include other concurrently accessed files, the user's current task, even the user's physical location — any actions or data that the user associates with the file's use. A recent study [71] showed that most users organize and search their data using context. For example, a user may group files related to a particular task into a single directory, or search for a file by remembering what other files they were accessing at the time. Such contextual relationships may be impossible to gather from a file's contents. Further, context information can begin to answer many of the outstanding questions that the system must answer to properly organize and search data. This work proposes to utilize context to help users retrieval their data from ever increasing data sets.

1.3 Thesis statement

This thesis will develop methods for automatically gathering attributes from context information and applying them to organization and search. Specifically, the thesis of this work is:

Automatically gathered context information can provide more attributes to more files than per-file content analysis alone. This increase in information will, in turn, improve the effectiveness of file indexing and search tools.

This thesis is validated with the following steps:

(1) I show that additional context-based file attributes, in the form of inter-file relationships, can be automatically identified from file system activity.

(2) I confirm the feasibility of context-enhanced search by developing a working search tool, Connections, that combines context and content information to provide file search results.

(3) I show that the added context attributes provided in Connections improve the quality of file search results through two user studies that compare context-enhanced search to traditional content-only search.

1.4 Problem breakdown

This thesis shows that context information that is automatically identified by the system can be utilized to improve the quality of file search tools. This requires: a source of context, methods for gathering information and identifying the context of individual files, a system that utilizes this context in file search, and a methodology for evaluating the quality of the system.

Specifically, this work uses temporal locality as its source of context. It describes Connections, a context-enhanced search tool that uses inter-file relationships identified from temporal locality to (1) identify additional relevant results for user queries (increasing recall) and (2) re-rank the results to place more relevant results earlier (increasing precision).

1.4.1 Temporal locality

Temporal locality is a way that many individuals use to recall events and items. "What was I doing when I last had my keys?" "I downloaded that file when I got back from my meeting." "I know I was working on my report when I read that paper." By connecting items that are accessed nearby in time, Connections can leverage these connections to behave in similar ways. Presented with the meeting or the report, Connections can locate related items such as the downloaded file or the related paper.

Despite the potential of temporal locality, it cannot work alone. In the examples above, Connections required a start point (such as the meeting or report) to locate the desired files. Connections identifies these starting points by leveraging existing content-only search tools, using the results provided by such a tool as starting points. Connections enhances these results by identifying other files that are closely related to these starting points. Furthermore, Connections uses the strength of these relationships to identify which results are most relevant to the current search. This is important because (1) content-only search is imperfect, and some of the starting points may be invalid, thus identifying the set of most strongly related results can help to eliminate some of the incorrect starting points, and (2) if many content-only results that appear less relevant, but are strongly related by context, they should likely be ranked higher in the results.

1.4.2 Connections

Connections gathers information about temporal locality using system level traces of the file system. It employs these traces to build a *relation graph* that describes the relationships among a user's files. At query time, Connections first runs a content-only search to identify starting points, and then accessed the relation graph to locate additional related files and re-rank results.

The key questions for building Connections become (1) how does Connections translate system-level information into user beliefs about how their data is temporally related, (2) how does Connections utilize these relationships to locate related data from a set of starting points, and (3) how does Connections re-rank the complete set of results. Additional, questions include the mechanics of how Connections gathers file system traces and stores the relation graph.

1.4.3 Evaluating utility

Unlike many performance metrics (e.g., run time, transactions per second, etc.), utility cannot be measured objectively. Because the results of a search are specific to the user's data and context, the quality of those results are subjective to that user.

This work evaluates Connections through two user studies. The first study looks at recall and precision, two metrics traditionally used to evaluate the quality of information retrieval systems. Recall measures the ratio of correct results returned to total correct results in the system. Precision measures the ratio of correct results returned to total number of results returned. The higher the recall, the more correct results returned to the user, and the higher the precision, the fewer incorrect results returned to the user. Comparing the recall and precision of two systems can show which more effectively reduces these false-negatives and false-positives. The second study examines questions about user experienced utility. Specifically, can Connections's context-enhanced search result in more successful, faster searches? By logging user behavior with both content-only and context-enhanced search tools, along with surveys regarding success and retry rates, this study can answer questions about user-perceived utility.

1.5 Contributions

In the course of proving the above thesis, this work provides the following key contributions:

(1) A set of algorithms for identifying temporal inter-file relationships and using them to extend and re-rank the results of traditional content-only search tools.

(2) A framework for implementing and evaluating techniques for adding context knowledge to traditional content-only search tools.

(3) Connections, a working context-enhanced search tool for both Linux and Windows that utilizes the above-mentioned algorithms and framework.

(4) An analysis of Connections using these two studies that shows the increased effectiveness of using context information in file search.

(5) A performance analysis of Connections showing that the computational and space-utilization overheads of context-enhanced search are small.

The key advantages of Connections's context-enhanced search are its abilities to (1) locate more relevant items for most searches, increasing average recall, (2) locate relevant items that contain little or no parsable contents (e.g., multimedia files), increasing both recall and precision significantly for non-text searches, and (3) its ability to distinguish the most relevant results from a set of textual results with closely ranked confidence values, providing slightly increased average precision for textual searches.

The key limitations of Connections's context-enhanced search are (1) it requires a large warm-up period (several months) to learn a user's behavior before it can be useful in search, (2) the large potential for false-positives means that the most successful algorithms only locate data that the user has created or modified, and (3) because it does not gather keywords from context, it relies soley on content-analysis to provide relevant starting points in the graph to perform its expansion and re-ranking, which in some environments may be difficult or impossible. Further discussion of these limitations, and possible ways to combat them in the future, can be found in Chapter 8.

1.6 Outline

The remainder of this dissertation is organized as follows. Chapter 2 describes the related research that has led to this work, as well as important ongoing work from other areas that both motivates this work, and has the potential to further enhance this work in the future. Chapter 3 outlines the framework for combining temporal context with content analysis, and describes the algorithms for identifying, searching, and ranking results with context. Chapter 4 describes the details of Connections design and implementation. Chapter 5 describes the two user studies's methodologies and uses them to evaluate the utility of context-enhanced search as experienced by users. Chapter 6 evaluates the performance and space-utilization overheads of Connections's context-enhanced search. Chapter 7 provides a discussion of the sensitivity of the algorithm parameter settings, and the qualitative effects of changing the underlying components within Connections (e.g., the source of content information). Chapter 8 concludes and describes potential areas of future work.

2 Background an related work

My thesis builds on research from file systems, information retrieval, and human computer interaction. This chapter describes systems and research that laid roadwork for this work, as well as related work that approaches the problem of data organization and retrieval (or similar problems) from different (and often complementary) angles.

The chapter is broken into three sections: organization, search, and context-aware computing. Within each, I describe key pieces of related work and then summarize their contributions and how they relate to context-enhanced search.

2.1 Organization: hierarchies and attributes

Shortly after the advent of persistent storage devices in the 1950s came the need for digital filing systems. Although the very first file systems provided only a flat namespace, this was quickly replaced with a hierarchical scheme, designed to mirror physical filing systems. Within such a system, collections of files are collated within a single named directory. These directories can be recursively collated into parent directories, forming a tree of named directories that eventually will lead the user to a particular file.

Today's file systems [18, 57, 66] are nearly identical to the first file systems of the 1960s: hierarchical, named directory trees. Unfortunately, as seen in library systems [7], as storage systems grow larger and larger, strict hierarchical filing schemes begin to fail. Although many hierarchical file systems provide some level of cross-tree linking that can augment the strict hierarchy, these links are difficult to maintain in the face of file updates and removals, making them an imperfect solution to this problem. Many believe that we are at a point with personal storage where the rapid growth in available storage (removing the need for file removal), combined with the increased uses for personal computing (resulting in more files being created), is creating a scenario in which current hierarchical file systems will no longer be sufficient.

This realization led to the idea of *attribute-based naming.* In attribute-based naming systems, files are tagged with particular keywords or tuples that identify categorical information about the file. For example, this thesis might be tagged with ⟨author:Craig Soules⟩ or perhaps the keyword ⟨context-enhanced search⟩. Once files are assigned attributes, the system can leverage these attributes to provide automated organization (e.g., a virtual directory hierarchy) and improved search mechanisms.

2.1.1 Attribute-based naming systems

One of the first systems to implement attribute-based naming was the Semantic file system [35]. The Semantic file system maintained both a traditional hierarchy and a set of [category:keyword] tuples with each file. Users could access files either through the traditional hierarchy or using virtual directories created for a particular category. For example, a user could create a virtual directory for the category "author," and within it would be created a directory for each existing keyword of that category (e.g., Craig Soules). Virtual directories could be created recursively, allowing the user to hone in on a particular file or set of files. Tuples in the Semantic file system were assigned either directly by users or through automated background tasks, called *transducers*, that examined the contents of files to identify accurate attributes.

The Semantic file system implements one way that a traditional hierarchy can be merged with attribute-based naming, and researchers have explored a wide variety of other mergings between these organizational schemes. For example, multi-structured naming uses Venn diagrams to describe the attributes that make up a file's namespace [67]. This flexible scheme allows the integration of both flat and hierarchical namespaces.

The HAC file system extends on the ideas of the Semantic file system by allowing users to associate arbitrary content queries with virtual directories, and then saving the specific results of the query with that directory so that it can be modified by the user manually [38]. This allows users to create directories on particular topics quickly, by starting with a search and then modifying the results to suit the desired organization.

The BeOS file system (BeFS) allowed users and applications to create external indexes of files using a particular attribute [34]. For example, an email application might create an index that associates each email file with the author of that email, allowing users to quickly search and sort these files for a particular author's emails.

Prospero provides per-user namespaces that are generated using a combination of hierarchies and user-programmed queries, called "filters" [60]. This allows users to place shared documents along different paths in their personal namespace, hopefully making it easier to locate data later.

The Presto document system uses an attribute tagging scheme similar to the Semantic file system, but instead of virtual directories, provides a variety of graphical user interfaces designed to assist the user in organizing and finding data using these attributes [26]. Attributes are also assigned on a per-user basis, allowing personalization of data organizations, combining the ideas of the Semantic file system and Prospero to a degree.

The pStore system proposes to combine many of the techniques of previous attribute-based naming systems with semantic information such as file relationships, dependencies, and access patterns [80]. Although they propose a design for storing and retrieving such data, the acquisition of semantic clues is left as an open question.

2.1.2 The world wide web

The world wide web is, perhaps, the largest distributed naming system in existence. To help manage this namespace, the web combines a hierarchical naming scheme with *hyperlinks* [12]. Hyperlinks provide a form of attribute-based naming by assigning keywords within one document to another document in the hierarchy. As users explore the web, they can locate the data they are looking for by following the attributes found within various hyperlinks. Hyperlinks also provide a secondary organization to the web, linking together documents by topic, rather than by hierarchical location. Still, although hyperlinks provide a very intuitive navigational structure, one of the key advances in the broad usability of the world wide web has been accurate search tools (a topic discussed further in Section 2.2.2).

Further extensions to the web, such as the Semantic Web [13], provide mechanisms for embedding self-describing attributes about the data within pages. These attributes resemble more closely the attribute-based naming systems from the realm of file systems, by storing attributes with the data they describe, rather than within the pointers to the data as is done with hyperlinks.

Similarly, Harvest [14] gathers attribute information from pages and stores them separately from the document contents, allowing search tools to access the attributes exclusively. This reduces the network overheads for indexing and allows page creators to add additional attributes not gathered directly from content.

2.1.3 Summary (Organization)

Although many proposals exist for attribute-based naming, very few commercial file systems have implemented any such features, and use of these features are almost non-existant. In the world wide web, several proposals exist for extending web pages with attribute information, but, although these proposals have been in place for almost 15 years, use of these extensions is not yet commonplace.

I believe the reason that attribute-based naming has been unable to gain traction is two-fold. First, implementation of attribute-based naming is more complex than traditional hierarchies (as evidenced by the extended time-to-market of projects such as WinFS [78]). Second, once in place, existing attribute-based naming systems rely on only two mechanisms as sources of attribute information: user input and content analysis (e.g., transducers). Unfortunately, user input is often time-consuming and unpleasant, which dissuades users from actually performing this task. As a result, much of the community has refocused its efforts on external search tools, using content analysis as the basis of such tools.

2.2 Search: content analysis techniques

Search tools take in attributes, often keywords, and display results that they believe are relevant to those keywords. This section outlines search tools from both file systems and the world wide web, and describes how context has the potential to improve the results of traditional content-only search systems.

2.2.1 File system search

One of the first file system search tools, still in use today, is the UNIX tool *grep*. Grep scans the contents of files for particular terms or expressions, reporting the line on which they were found. Other tools, such as *find*, were added to traverse the namespace and match terms or expressions against file pathnames. Although their functionality is indispensable, tools such as grep scan the entire contents of files on each invocation, leading to poor performance when searching through large data sets.

Glimpse addresses this problem by combining grep with an index of keywords [54]. When a user performs a search, Glimpse first consults the index, which indicates a subset of files that contain the keyword. It then runs grep over this (hopefully) smaller subset of files. By narrowing or widening the size of the subsets within the index (potentially even considering sections of large files), the user (or system administrator) can make a trade-off between index size and search performance. This kind of indexing for performance is now commonplace among search tools.

As search tools such as grep have grown into the modern information retrieval tools of current research, the content analysis techniques have grown both more accurate and more complex. Tools such as Lemur [49] and Terrier [73] combine several of the latest advancements in content analysis to provide accurate, ranked results. Often, these tools perform best when experts generate the queries using the full power of their expressive query interfaces [58]. Although the underlying techniques of these systems are not directly relevant to this work, the results they generate form the basis of my context-enhanced search. As a result, further improvements in these tools are complementary to my approach; as they become more accurate, so does context-enhanced search (this is discussed further in Chapter 7.2). Furthermore, finding ways to tie the content analysis tools more closely to context analysis could result in improvements in query accuracy (as discussed in Chapter 3.1.3), and this is an interesting area for future work.

Recently, several commercial tools have been released that harness modern information retrieval techniques, including Google Desktop [36], Yahoo! Desktop search [37], and Windows Desktop search [77]. The emergence of commercial tools indicates that the increases in user data sets are creating a demand for accurate search facilities on personal computers, underscoring the importance of this issue.

Although the focus of most information retrieval systems is specifically on plain-text analysis, some search tools (e.g., LXR [51] and CScope [70]) use an understanding of content structure (in these examples C code) to provide directed search for particular usages of terms (e.g., finding function calls vs. function definitions). Also, these kinds of structure-specific analysis techniques can often be integrated with existing plain-text analysis. For example, tools such as Apple's Spotlight [8] and Google Desktop allow external tools to parse specific document types (e.g., Postscript or instant messenger logs) and integrate their contents with the rest of the index.

2.2.2 Web search

Although the structure of the web has the potential to make browsing for information easy, there remains the problem of locating an appropriate starting point at which to begin browsing. As a result, some of the most visited websites today are web search engines.

The original web search engines, such as Lycos [52, 56], Yahoo! [82], and Altavista [5], applied the same techniques used today in file system search. They combined user submitted information with text analysis (e.g., word frequency, inverse document frequency) and structural analysis (e.g., heavier weights to words in titles, headers, etc.). However, as discussed above, the web contains another important source of attributes: hyperlinks.

One of the first systems to make use of hyperlinks was WebGlimpse [53]. WebGlimpse used hyperlinks to define neighborhoods: the set of pages within a particular link distance from the currently viewed page. Using this idea, WebGlimpse allowed users to narrow their search to within their current neighborhood. The key insight of WebGlimpse was that a neighborhood makes up part of a user's *context*, and, by leveraging that, the user may find what they are looking for more quickly.

Shortly after the advent of WebGlimpse, two techniques emerged that used the link structure of the entire web to combine content and link analysis in web search: HITS and PageRank.

The key insight of HITS is that the web primarily contains two kinds of pages: hubs and authorities. Hubs are pages that contain links outward to many authorities (e.g., a page with a list of auto manufacturers). Authorities are pages that are referred to by many hubs (e.g., the official Ford web site). This recursive definition is inherent in the design of the HITS algorithms [46]. When a user performs a search using HITS, it first locates a starting set of pages using traditional content analysis techniques. Using this as a starting set, HITS next gathers a super-set of pages that are within a single hop of the starting set. It then runs a recursive algorithm that assigns hub and authority rankings to each page in the super-set.

PageRank uses hyperlinks by treating the structure of the web as a Markov model [16]. It then calculates the stationary probabilities of the model and assigns the probabilities at each node as the PageRank, or "importance," of that particular page. This probability indicates the likelihood of that a user randomly browsing the web will land on the given page, thus pages with incoming links from other "important" pages will have a high importance. When a user performs a search, the system can first locate pages that match on content, and then can rank them using their PageRank.[1]

The Google web search engine combines PageRank with both traditional content analysis and two other novel techniques. The first is to use the text of hyperlinks as classifications for the linked page, thus making use of the linker-assigned attributes within hyperlinks. The second is to watch the behavior of users who access the results of their search to gather implicit user feedback. For example, if a user performs a search and then begins to browse the results, clicking on the first link, then the second, then the third, it is likely that after

[1]Although this is one likely use for PageRank, the actual merging of content analysis with PageRank in systems that use it, such as Google, has not been published.

clicking on that final link they found what they were looking for. Using this information, it may be possible to improve the ranking of pages that match the given query terms.

More recently, groups have begun exploring ways to further exploit the graph structure of the web to propagate information among related pages. Closely related to my work is the work on relevance score propagation and term frequency propagation [62, 68, 69], that propagate attributes of a given page to other pages through a given graph structure (such as the hyperlink structure or site-map). Study of this area has shown that attribute propagation can improve retrieval accuracy on the web. It is not surprising, then, that the similar algorithms used in this work also improve the retrieval accuracy in local file systems. The challenge in local file systems is identifying a graph structure that can accurately map the relationships between files.

2.2.3 Summary (Search)

Over the last 40 years, information retrieval has grown from simple expression matching to an entire field of study. In file systems, content analysis, and specifically text analysis, makes up the bulk of existing systems. However, in the world wide web, some of the biggest improvements in search quality haven't come directly from content analysis, but instead from user-provided classifications in the form of hyperlinks.

What hyperlinks provide, that content analysis cannot, is *context*. The Merriam-Webster dictionary defines context as "the parts of a discourse that surround a word or passage and can throw light on its meaning" [76]. On the web, the keywords in a hyperlink identify the context of the linked website by identifying (1) how the creator of the link thinks about the linked site, and (2) how a user clicking on the link will think about the linked site.

Unfortunately, current file systems do not include hyperlinks, making it difficult, if not impossible, to apply the search techniques used on the world wide web. My work tackles this problem (to an extent) by identifying inter-file relationships from user access patterns. Unlike hyperlinks, these relationships do not contain keywords, but they can be leveraged in similar ways.

In the next two sections, I will describe work from other areas of file systems that utilizes inter-file relationships and work from context-aware computing that uses various forms of context to achieve similar goals.

2.3 Inter-file relationships

Although no explicit linking between files exists within file systems today, work in both prefetching and cache hoarding indirectly identify inter-file relationships. File prefetching aims to hide storage latencies by predicting what the user will access next and reading it into the cache before they request it. Cache hoarding attempts to predict what the user will access next to hide network latencies (or provide access to network data during disconnected operation) and thus requires a similar prediction algorithm.

Most file system allocation policies attempt to improve locality among related files and their metadata by placing files in the same directory nearby one another on disk [30, 57].

However, researchers found that these groupings do not always match to the access patterns of users and applications. This is because directories do not always capture the context of users and applications.

One way to address this problem is through prediction schemes that use temporal locality, rather than spatial locality, to correlate common user access patterns with individual user contexts [6, 39, 47, 48]. Such systems keep a history of file access patterns using *successor models*: access models that predict the next access based on the most recent accesses. If a sequence of accesses matches one of the stored successor models, the system assumes that the user's context matches this model and prefetches the specified data.

My work explores using a variety of techniques similar to those used in prefetching and cache hoarding to enhance file organization and search. Further details on existing algorithms for generating successor models and their contributions to the algorithms used in context-enhanced search are described in Chapters 3.2.1 and 3.2.2.

2.4 Context-aware computing

Context-aware systems utilize context information to tailor services to particular users. Context within a computer system has many definitions [25], including information such as location (e.g., gps, relational distance), environmental conditions (e.g., weather, lighting), and time (e.g., time of day, relational accesses).

Much of the work in context-aware computing falls under the label of "ubiquitous computing" [19, 25]. Still, many research projects utilize context to aid organization and search. The potential benefits of this for personal file search were made clear by a recent study of user search behavior, which showed that individuals often use context to guide their search, rather than specific content keywords [71].

2.4.1 Data organization and retrieval

One approach to context-based data organization is to integrate attribute-based naming more closely with the system's user-interface. Projects such as Haystack [63] and MyLifeBits [32] provide interfaces designed to quickly group and tag files with attributes. These systems also provide mechanisms for integrating various forms of application assistance, some of which provide context similar to some of the ubiquitous computing work described above [33].

Although a file's context is generally separate from its contents, several researchers have leveraged content-similarity to identify contextual groupings among files. Scatter/Gather [23] uses content-similarity techniques to scatter a collection into clusters. A user can select particular clusters of interest, which are then gathered into a single sub-index, and then re-scattered into further refined clusters. This process is repeated recursively until the user's information need is satisfied. Grokker is a commercial search tool that utilizes a similar form of content-similarity [40]. It combines clustering with a novel drill-down interface that allows users to hone in on particular groupings quickly.

Another potential source of context information is the existing metadata within hierarchical file systems, such as access time and permissions. Several projects treat metadata as attributes, and use them to help organize and search a user's data. The Lifestreams search tool [29] ranks its search results using the latest access time of the file. This provides the user with a visual representation of the temporal context of the presented files. By seeing other files accessed nearby in time, users may be able to more quickly locate what they are searching for. Stuff I've Seen [27] uses context information already stored by the file system (e.g., access time, thumbnails, author, etc.) to enhance both the content search and the presentation of results. This work was the predecessor to the commercial Windows Desktop Search [77].

2.4.2 Personalized search

Another way to leverage context is to identify the user's current context, and use that as a clue toward retrieving information relevant to the task at hand [45]. Below is a sampling of some of the most relevant related work.

As mentioned earlier, much of the work in ubiquitous computing is focused on identifying a user's current context to assist with tasks and communication. One such project that was aimed more toward file search is the Rememberance Agent [65], which identifies the context of the user from the contents of recently accessed files. By feeding relevant keywords from these files into a content search engine, the Agent can supply the user with a set of files potentially related to their current activity. The Jimminy system [64] extended this idea by using information about the user's environment (e.g., physical location) to locate relevant notes and information to help with the user's current task.

The work of Czerwinski, et. al., uses the content of currently viewed web-pages to retrieve additional pages with similar contents [24]. Their tool combines this automated retrieval with an interface that provides thumbnails of related pages allowing users to quickly scan them for potentially useful items. Teevan, et. al., identified the user's context using a combination of their local data set (gathered from a content index of their data) and recent web activity (searches, visited URLs, etc.) [72]. Using this profile, they could direct searches toward particular topics of interest to the user.

The Lumiere project [42] identified a user's current context from their recent actions, and uses this data to provide the user with help information. By developing and training a Bayesian network for predicting common user tasks, the system could identify if a user may need help (e.g., if they appear to be attempting something unsuccessfully) and can suggest actions to assist the user with their task. A simplified version of this work was integrated with the Microsoft Office Assistant (i.e., Clippy). The drawback of this approach is the amount of training data required to successfully determine the user's context. This problem is exacerbated in the case of context-based search, since the interaction of the user with the file system is not restricted by a well-structured interface. It might be possible to exploit this kind of application-based context-prediction for common user applications to further enhance the context-based search described in this thesis, but this is left as an area of future work.

2.4.3 Data mining

Data mining is a broad field that covers most activities that attempt to identify behaviors or correlations within data sets. By some definitions, web search techniques such as PageRank, as well as my work on context-enhanced search, fall under the domain of data mining. Outside of link analysis, perhaps the most closely related work to context-enhanced search is the area of identifying *association rules* across time-series data [3, 4, 28, 43, 61, 74]. Much of this work focuses on finding sets of items with some *minimum support*, based on the number of transactions in the data that contain the set. Theoretically, the set of association rules between files could form a graph of inter-file relationships for use in context-enhanced search. The problem with this approach is that, just as in traditional successor models, this work aims to find the most prevalent relationships, which may not always be the most important for a given search. Although it might be possible to modify existing approaches to more broadly capture context (as I have done with successor models in this work), this is left as possible future work.

Perhaps a more fruitful application would be to use some of the clustering work from data mining [15, 31, 41] to collate search results before they are returned to the user, similar to systems such as Clusty [21] and Grokker [40]. One of the potential drawbacks of context-enhanced search is the increased number of results returned to the user. Although my work has found that proper ranking can reduce false-positives, such clustering techniques may further reduce the effect of false-positives by helping users quickly eliminate sets of closely related, irrelevant results.

2.4.4 Machine learning

Most machine learning work focuses on classifying items using attributes associated with the items identified from pre-classified training data [59]. The difficulty with applying machine learning to the problem of inter-file relationships is gathering accurate training data; most users are unwilling (and perhaps unable) to properly identify accurate relationships among a large set of files.

However, machine learning could play an important role in refining existing information about submitted queries. For example, just as on web systems, when a user submits a query, the interactions of the user with the results set can be traced to identify which results are most likely to be accurate. This data could then be used to train a system to help identify correct results in future queries.

2.4.5 Summary (Context-aware computing)

Context is powerful tool for more accurately identifying a user's information need, especially in the world wide web, where contextual clues are readily available. Unfortunately, in the local file system, the relational context available in the web does not exist. As a result, most researchers have instead focused on utilizing available content information (including content similarity) and file system attributes.

My work uses temporal locality to provide the relational context missing from local file systems. Once in place, these inter-file relationships are a complementary source of context that can be used to enhance most (if not all) of the work described above.

2.5 Wrap-up

As our need for improved data organization and search has increased in recent years, the focus of researchers has shifted toward techniques that extend the flexibility and personalization of local file storage and search. Context is a tool that many believe can significantly improve the quality of such systems, and recent studies back this idea strongly. My work combines techniques from various existing systems to introduce a new form of context, inter-file relationships, to local file search. Once in place, these contextual relationships have the potential to extend or replace many of the sources of context used in related systems.

3 Context-enhanced search

While context has many potential forms and uses, this dissertation focuses on using contextual relationships identified from user access patterns to enhance traditional content-only search. This chapter begins with a description of a specific architecture of context-enhanced search and discusses some of the trade-offs inherent in its design. It then describes each of the three algorithms required by this architecture.

3.1 Architectural overview

My architecture for context-enhanced search was designed with two goals in mind. The first goal is for the interface presented to the user to remain unchanged when moving from content-only search to context-enhanced search. This simplifies offline analysis, because the tools for querying and result analysis are the same for both systems. It also simplifies online studies, since the underlying search tool can be changed with no user-visible interface change.

The second goal is for content analysis and search to be decoupled from additional context analysis and search. There are four reasons for this. First, by keeping the two components separate, the benefits of contextual relationships are clearer. If they were intertwined, its possible that modifications made to the content analysis component could improve the results in some undetectable fashion, and thus any measured benefits could not be strictly attributed to context analysis. Second, this separation makes it easier to test the system with varied content analysis methods, since underlying mechanisms are agnostic to the surrounding context analysis techniques. Third, this decoupling simplifies analysis of the performance and space utilization overheads of context analysis by presenting a clear delineation between the two analysis techniques. Fourth, if benefits can be gained within a decoupled framework, it may be possible to use contextual relationships for other tasks without additional modifications.

The remainder of this section (1) overviews the design of existing content-only search tools, (2) describes the added components required for context-enhanced search that meet the requirements described above, and (3) discusses the trade-offs of our design choices.

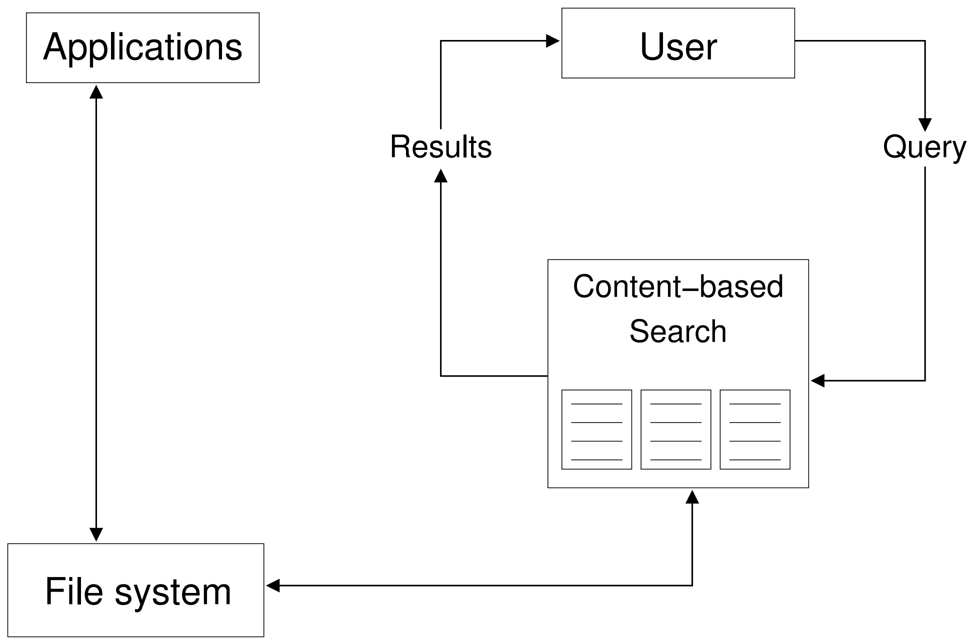


Figure 3.1: Architecture of content-only search

3.1.1 Content-only search

Figure 3.1 illustrates a traditional content-only search tool. Users interact with the search tool by submitting queries using some well-defined query language. The search tool then consults either the file contents or a pre-computed index derived from file contents, ranks the results, and returns this ranked list of results to the user.

In many search tools, the query language is a list of keywords. However, more advanced search tools provide extensive, descriptive query languages that, when exploited by query experts, have been shown to improve the results of user queries [17, 58]. Because one of the goals of context-enhanced search is to mimic the interface to content-only search tools, it is important that my system allow any form of query desired by the underlying content-only search tool.

For performance reasons, many content-only search tools perform content analysis offline, storing the results of the analysis in a pre-computed index that can be consulted at query time. For context-enhanced search to match the query speed of content-only search as closely as possible, it should also make judicious use of offline indexing.

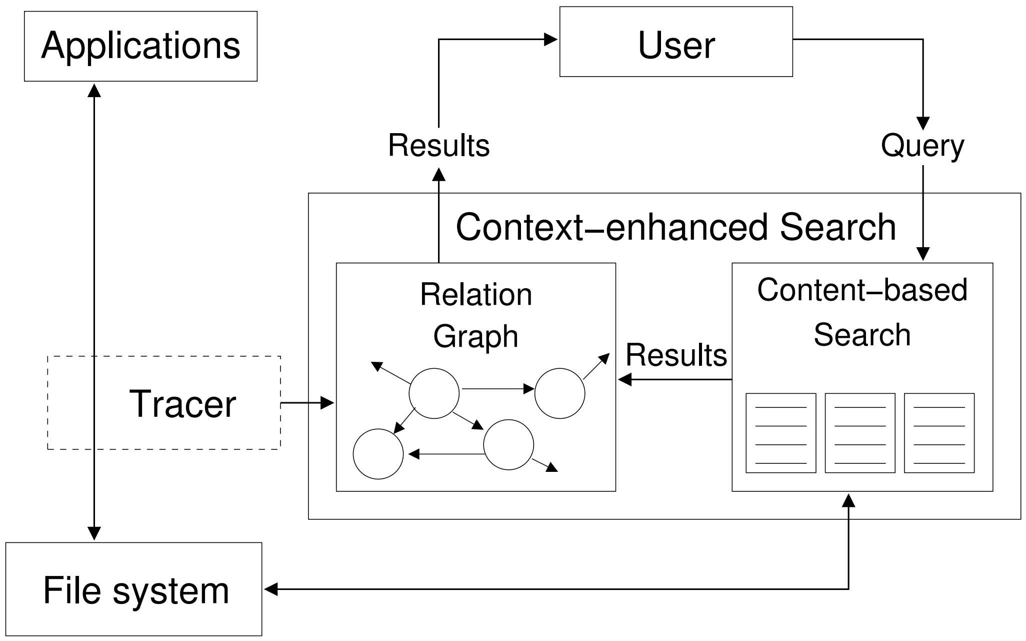


Figure 3.2: Architecture of context-enhanced search

3.1.2 Context-enhanced search

Figure 3.2 illustrates the changes required to provide context-enhanced search within the same framework as existing content-only search tools while keeping it separated from the internal workings of such tools. Just as in Figure 3.1, the user submits queries to the system using a well-defined query language. Queries are then submitted to the content-only search tool, which processes the request in the same fashion as before, returning a ranked list of results. These results are then fed into a *relation graph* that stores contextual relationships. These relationships are identified from user accesses gathered by a transparent tracing component that sits between applications and the file system. The relation graph uses the contextual relationships to extend the content-only results with additional contextually related results and to re-rank the entire set of results before returning them to the user.

This design meets the previously stated goals. Because the query language is interpreted by the content-only search tool, it remains unchanged, allowing context-enhanced search to support unmodified query languages. Because traces are gathered independently of the queries, they can be processed offline to create a query-optimized relation graph. This design also provides a strict decoupling between content and context analysis by performing each in serial.

The additional components required by context analysis, the tracer and the relation graph, introduce three required algorithms. The first is a method for identifying contextual relationships from traces. The second is a method for extending the provided content-only results with additional contextual results using these relationships. The third is a method for ranking results based on the provided contextual relationships. Sections 3.2, 3.3, and 3.4 provide details on these algorithms.

3.1.3 Alternate methods

I chose the requirements for context-enhanced search in this work on the basis that the goal of this thesis is not to provide the ultimate answer on combining context with search, but rather to explore the potential benefits of such a combination. The result is that, while my approach is both flexible and easy to measure, there are many other approaches to merging content and context, some of which might provide more accurate search results. This section describes a few such approaches.

The architecture described above extends the results of content-only search, but the system could instead extend the content index, using contextual links to propagate the content attributes of files to related neighbors. The success of work on weight propagation on the world-wide-web (described in Chapter 2.2.2) suggests that such an approach could be successful in personal file systems if given an accurate relationship structure. Furthermore, the query speed of the system is likely to be better, since the relation graph manipulation steps are all included in the indexing. The drawback of such an approach is that by integrating the relation graph with the content index, incremental updates are far more difficult. Without separating the two components, its difficult to distinguish which attributes were generated from content and which from context without reindexing all of the file contents from scratch. It could be possible to tag context attributes directly and remove them before re-indexing, but this would likely increase indexing time because attributes would have to be re-propagated after each re-indexing.

Another potential design is to integrate contextual relationships with the network models used in most content-only search tools [10]. In these systems, documents are connected to terms through probabilistic links. Issued queries result in a directed acyclic graph (DAG) that connect the query to the specific query terms and then to the relevant documents. Context could be used to add an additional layer to the DAG that connects documents to other documents.

By tying content and context more closely, it might be possible to extend existing query languages to incorporate context information more directly, potentially improving search accuracy. Furthermore, closer ties between content and context would likely improve query speed. However, this kind of tight coupling breaks the requirements of my system, as it would be difficult to isolate the advantages of the various changes to the system. Therefore, I am leaving this technique as future work.

3.2 Identifying inter-file relationships

The first algorithm required by the context-enhanced architecture takes the traces of user activity as input and use them to identify the contextual relationships among individual files. I propose using methods similar to those employed by successor models that identify a probabilistic graph of inter-file relationships from temporal locality.

3.2.1 Basic successor models

As mentioned in Chapter 2.3, prefetching and cache hoarding use successor models to identify common patterns from user actions. The belief is that these common patterns match to user tasks, and thus a user repeating a previously seen task will perform actions that match the identified pattern.

The most basic successor models try to identify the next file access using history information. Models such as *first-successor* or *last-successor* make a prediction based on a single previously seen sequence. In the first-successor model, the first time the system sees an access to file it will generate a permanent prefetching rule. For example, if the first access to file A is in the sequence ⟨A, B⟩, the system will create a rule that always prefetches B after an access to A. In the last-successor model, the system will update the prefetching rule based on the most recently seen sequence. For example, if a later access sequence of ⟨A, C⟩ is seen, a last-successor system would update its rule to prefetch C after A in the future. More complex models, such as *k-of-n*, create prefetching rules using cumulative history information, in this case choosing a successor that was seen at least k times out of the last m accesses to the file.

The key objective of these models is to identify the next access to be performed in the system. As a result, these models are built to capture short-term behavior of the system. Conversely, my work is interested in long-term behaviors that capture common user tasks.

3.2.2 Probability graphs

A more long-term approach, developed by Griffioen and Appleton [39], identifies a *probability graph* from traces of system activity. Each file in the system is represented as a node in the graph, and links indicate successive accesses seen in the trace. Probabilities of transitions are calculated as the weight of a given link divided by the total outgoing weight from a node.

Because the probability graph is used to identify data for prefetching, it is only interested in successive accesses. As a result, many of the choices made when generating the graph are directed toward this task. Their system creates a single graph for the entire system, because this most closely models the accesses seen by the shared storage subsystem. This system only examines accesses over very short periods of time (generally less than 100 milliseconds), because larger periods of time become less effective for prefetching. Because their system is focused on predicting for read-ahead, it only considers EXEC() and OPEN() calls, relying on the common application behavior of reading the entire contents of a file after an OPEN()

call. Finally, it only considers temporally ordered connections (i.e., file 'A' leads to file 'B'), since the reverse link is of little use for prefetching prediction.

3.2.3 Relation graphs

The differences in objectives between my work and file prefetching led me to consider a super-set of graph creation algorithms that may be better suited to identifying relationships that make up user tasks, rather than just successive accesses. To explore the space of such *relation graphs*, I developed a parametrized model that provides four knobs that affect graph creation: isolation level, window size, edge direction, and access filter.

The *isolation level* specifies attributes that make up separate threads of control. For example, the isolation level could be set on the user id, meaning that interleaved accesses between users would not create links between their files. Alternately, the isolation level could be set on the process id, avoiding interleaved accesses between separate processes. The isolation level can also specify attributes that result in the formation of separate graphs. For example, the system may want separate graphs for each user's files based on an inode's owner id.

The *window size* specifies how many unique file accesses to consider when creating links in the graph. The window size is defined by two variables: a count that limits the number of trace entries considered together and a time that limits the temporal distance of accesses in the trace. Files accessed within the window are connected through links in the graph as determined by the edge direction and access filter (described below). The window is maintained as a FIFO queue of files, with files re-accessed within a window being re-queued but not generating additional links. In traditional probability graphs, the window size is set to have a count of two (examining only successive accesses) and a time of 10-100 milliseconds. In relation graphs, both the count and the time are generally much larger to more accurately capture full user tasks.

The *edge direction* specifies whether relationships are strictly ordered. If edges are directed, then they maintain the temporal ordering found in probability graphs. If edges are undirected, then the causal nature of temporal ordering is lost (potentially allowing tasks to be captured more accurately). For example, a user reading in a piece of code, and then writing that code to another file indicates information flow forward in time, but without indicating that in the graph, the causality is lost.

The *access filter* specifies which system calls are of interest and how relationships are formed between them. Specifically, it categorizes the files associated with a system into two classes: input and output. When a file is classified as an input, it is placed into the window. When a file is classified as an output, a link is created from all of the inputs in the window to the output (if a link already exists, then the weight of the link is incremented). For example, in traditional probability graphs only EXEC() and OPEN() calls are considered and are classified as both input and output (i.e., they always form relationships). Another possibility might be to consider only READ() and WRITE() calls, with READ() calls classified as inputs and WRITE() calls classified as outputs (i.e., a causal relationship is assumed from READ() to WRITE()). This approach would result in a sequence like READ(A), READ(B),

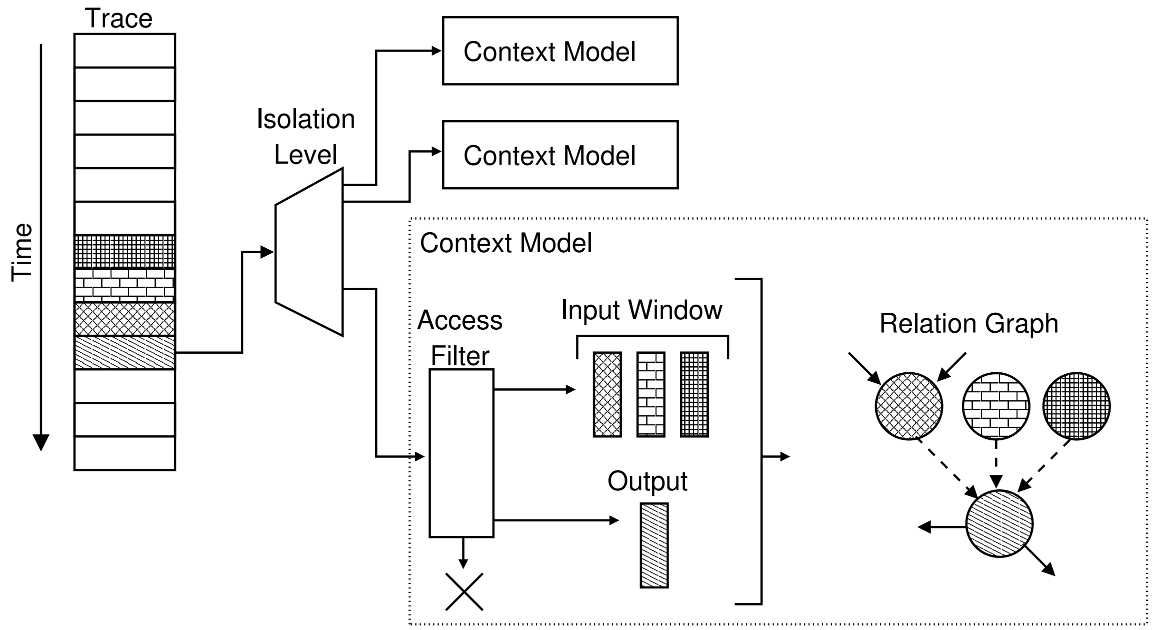


Figure 3.3: Generalized successor model

WRITE(C) forming links from A to C and B to C, but not A to B.

Figure 3.3 illustrates the framework for the modified successor models used in this work, showing how each of the four components described above fit into the system. On the left is the trace of accesses seen in the system, which is scanned in time order, and on the right is a set of relation graphs created from the traces.

Each entry is first passed through the *isolation level* de-multiplexer, which decides which context model the access belongs to. Within the context model, the entry is next passed through the access filter. The access filter marks each file in the entry as being an *input*, an *output*, both, or neither. These classifications allow the filter to specify the causal relationship of files in the trace. If the access filter does not classify a file as either an input or an output, it is dropped. If a file is classified as an input, it is placed into the *input window* (limited by the window size). If a file is an output, a link is added from each of the files in the input window to the output file (or if a link already exists, its weight is incremented), shown as dashed arrows in the relation graph. The distinction between input and output files is made to maintain temporal ordering in the case where directed links are used. Classifying files as both inputs and outputs removes the causal relationships between different access types (e.g., WRITE() calls could point to READ() calls), but not the temporal causality that is specified by the link direction.

It is interesting to note that by setting the parameters correctly, the original probability graphs can still be created. Specifically, they are created when there is a single context model, an input window size of 2, an access filter that considers EXEC() and OPEN() calls to be both inputs and outputs, and directed links.

3.2.4 Trade-offs

Although flexible in design, relation graphs still cannot capture certain kinds of relationships. This section outlines several of the draw-backs and trade-offs inherent in my relation graph model.

Behavioral knowledge

Because the relation graph is a summary of user actions, some of the semantic knowledge contained within the original sequence of user actions is lost. Specifically, flattening long sequences of actions into a single graph may result in a loss of some causal relationships, since the context of one sequence may get hidden by other, stronger contexts for overlapping sets of files. For example, a sequence of events such as $\langle$A,B,C,B,C,E$\rangle$ may be the common behavior of an application when the user is using files A and E, but when flattened in the graph, the connection between A and E may be diminished by the connection from A to B and C, particularly if B and C are heavily connected to many other files.

Potentially, choosing an appropriate window size may help to diminish the problems associated with this kind of application behavior, but it is insufficient alone. The search and ranking algorithms described in Sections 3.3 and 3.4 try to alleviate this problem, but a better understanding of particular application behavior could further reduce its effects. Potential areas of future work to automatically identify application behavior are briefly described in Chapter 8.

Hyperlinks

The design of the relation graph is similar to the graph of pages on the world wide web but, unlike the web, fails to capture keywords that are relevant to each of the links in the graph. While the information in the graph is still valuable, there could be added value by finding methods to convert these links into hyperlinks (i.e., links with associated keywords). Because of the breadth of potential techniques and its tangential nature to the goal of this thesis, examining how to capture, assign, and utilize such keywords is left as future work.

Hysteresis

Another problem with maintaining a relation graph is that it is very difficult to perform hysteresis, because the link weights are not attributed with any particular time.[1] This makes it impossible to decay weights on a per-link basis, and, if the entire graph were decayed during each update, infrequently accessed files would eventually lend no weight to the rankings, even when they should.

[1]Hysteresis is also a problem in traditional probability graphs and, because of the same difficulties, has never been fully addressed [39].

As a result, the approach discussed in this thesis does not eliminate links from the graph until search time. While this does not appear to pose problems for storage space (as discussed in Chapter 6.2), there is a potential problem that a user's context for a file may change over time. This shift in context can result in reduced accuracy of the relation graph and is discussed in Chapter 7.4.

The decision to avoid link weight normalization also has ramifications for the searching and ranking algorithms that are briefly discussed in Sections 3.3 and 3.4. In fact, some of the choices in the design of these algorithms were made to account for this potential problem.

Existing systems

Because the relation graph is built using trace data, there are two potential problems for existing systems. The first is that trace information may not be available in some systems, rendering this technique impossible. The second is that, even if traces can be gathered, there is currently no method to retro-fit relation information to an existing system. This means that files that are not accessed in the future cannot be located using contextual clues. Although it might be possible to attempt to retrofit identified behaviors from existing traces to untraced files, such predicted user behavior could result in inaccurate relationships.

3.2.5 Summary (Identifying inter-file relationships)

Despite some of the drawbacks discussed above, relation graphs provide a computationally simple, yet effective, method for identifying and storing inter-file relationships. In most cases, the design of the relation graph erred on the side of flexibility and simplicity in order to provide a computationally feasible implementation, while allowing room for exploration of a variety of techniques. Although there may be better techniques, starting with these provides intuition as to which areas are worth closer examination and will likely produce the most future gains.

3.3 Searching relationships

The second algorithm required by context-enhanced search is a method of extending the results of a content-only search with additional contextually related files. The goal of context-enhanced search is to find files that are contextually relevant to the keywords presented by the user. Conceptually, some, if not all, of the files located using content-only search should be relevant to the user's information need, as has been shown in previous file search work. As such, files contextually related to those results should also have relevance to the user's information need.

My approach to finding contextually related files is to take the results of a content-only search as input, and perform a *breadth-first expansion* on the relation graph, using the inputs as starting points in the graph. After the expansion is complete, the resulting nodes form a smaller *result graph*, which contains only the nodes found during the search and the links between them.

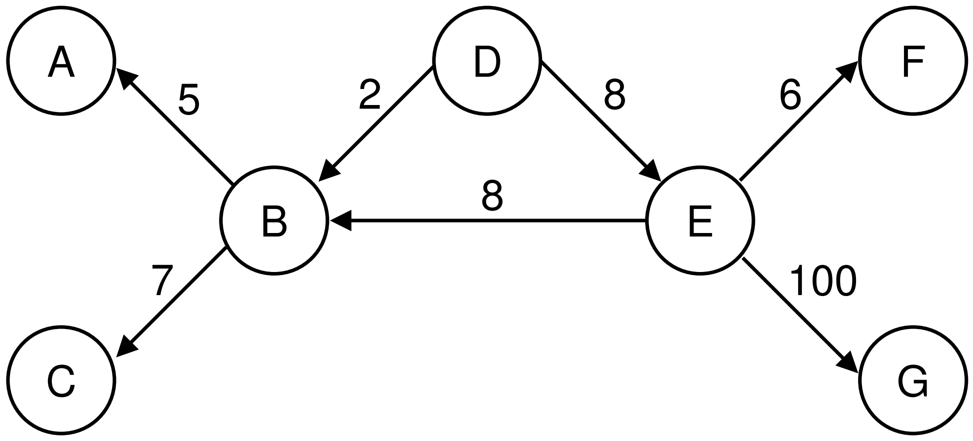


Figure 3.4: Relation graph example

My algorithm limits the number of files in the result graph using two parameters: a weight cutoff and a path length. Limiting the search serves two important functions. The first is to reduce the computational complexity of the ranking algorithm by limiting the number of files it must consider. The second is to reduce the number of false positives in the results that are returned to the user.

3.3.1 Weight cutoff

The goal for the weight cutoff is to avoid following incorrectly formed connections between files. Incorrect connections form from cases such as user context switches (i.e., the user starts working on an unrelated file for some reason), "launcher" applications (e.g., a shell launches an application which opens a file, but the shell is unrelated to the file), or configuration files (e.g., a text editor's configuration file is unrelated to the files being edited).

The weight cutoff deals primarily with user context switches, by ignoring transient, "lightweight" links. Initially, lightweight was defined to be a scalar value based on the number of times the link had been seen. However, because the relation graph's link weights are never normalized (see Section 3.2.4), a single number cannot capture the concept of lightweight across all nodes in the graph. For example, if a user accesses file A frequently and file B infrequently, then, over time, the invalid links created for file A may outweigh the valid links created for file B.

As a result, the weight cutoff instead uses link weight *ratios*, which act as an online normalization method. Each link is assigned two ratios: (1) its weight compared to the outgoing weight of its source node, and (2) its weight compared to the incoming weight of its sink node. These two ratios are then compared against a cutoff ratio. If both ratios fall beneath the cutoff, then the link is ignored during the search. This addresses the example above, because the total weight of file A would be significantly higher than the total weight of file B.

Figure 3.4 illustrates an example relation graph. Assuming that D is the starting point of a search, consider the link from D to B. In this case, both the outgoing and incoming ratios are 20%. Thus, if the weight cutoff is set above 20%, the link would be ignored during search.

The choice to only drop links for which both ratios beneath the cutoff was made to avoid cutting all incoming links to a given node. For example, consider the link from E to F in Figure 3.4. In this case, although the outgoing ratio is only 5%, the incoming ratio is 100%. Thus, even if the link's outgoing ratio is beneath the cutoff, it may be best to keep the link, since there would be no other way to locate that node during a search. This presents a trade-off: the decision to err on the side of keeping these links forces the system to rely more on the ranking algorithm to cull false-positives, since more incorrect nodes are likely to be found during a search.

3.3.2 Path length

The goal for the path length is to avoid including files from the relation graph that are are too far removed from the context of the original content-only search results. It is specified as the maximum number of steps to take from a content-only result node during the breadth-first expansion. Although the ranking algorithm will generally rank these "too-distant" files low in the results, omitting them from the ranking process can, in some cases, significantly reduce the computational overhead of the ranking algorithm.

3.3.3 Trade-offs

An alternative to these search parameters would be to always consider the entire relation graph during any search, and trust the ranking algorithm to provide the user with the most relevant data first. The advantage of such an approach is that it doesn't proactively remove potential paths of interest from the search space. Assuming that false-positives can be successfully pruned during ranking, such an approach might improve the quality of returned results.

The disadvantage of such an approach is that the query processing time of such a system would grow with the size of the relation graph. By introducing parameters on the search space, a system can limit the effect of graph size on query performance by tuning the weight cutoff, path length, and number of content-only results considered. This gives the system tunable knobs to make a direct trade-off between accuracy and performance.

3.4 Ranking the results

Once the search algorithm has identified the result graph, the ranking algorithm orders the files in the graph to present the user with the most relevant results first. Traditionally, content-only systems use methods such as term frequency and inverse document frequency to identify which files most accurately represent the specified keywords; however, with context-enhanced search, many of the resulting documents contain none of the keywords, making these techniques insufficient.

A solution to this problem is to combine the confidence provided with the content-only search results with the link structure of the result graph to identify the most relevant documents. I considered two basic approaches. The first is to use the followed path weights to propagate confidence values from the content-only results to contextually related files. The second is to use existing web ranking algorithms that utilize link structure to rank results.

3.4.1 Basic Breadth First Expansion (Basic-BFE)

Basic-BFE uses the rankings provided by content-search to guide the rankings of contextually related items. Close relations, and relations with multiple paths to them, will receive more weight than distant relations with few incoming paths. Intuitively, this matches the user's activity: if a file is rarely used in association with content-matched files, it will receive a low rank, and vice-versa.

Because link weights are assigned based on frequency, and different applications access files at different rates (even when performing similar tasks), Basic-BFE tries to limit this variability in its algorithms. Specifically, it uses an alpha parameter, described in the equations below, that is designed to reduce link weight variability.

Let N be the set of all nodes in the result graph, P be the path length used to generate the result graph, and α be a tunable parameter. Each $n \in N$ is assigned a weight w_{n_0} by the content-based search scheme. If a file is not ranked by content analysis, then $w_{n_0} = 0$. The following algorithm is then run for P iterations.

Let E_m be the set of all incoming edges to node m. Let $e_{nm} \in E_m$ be the percentage of the outgoing edge weight at n for a given edge from n to m. Assuming that this is the i^{th} iteration of the algorithm, then let:

$$w_{m_i} = \sum_{e_{nm} \in E_m} w_{n_{(i-1)}} \cdot [e_{nm} \cdot \alpha + (1 - \alpha)] \tag{3.1}$$

The value w_{m_i} represents all of the weight pushed to node m during iteration i of the algorithm, and α dictates how much to trust the specific weighting of an edge. After all runs of the algorithm, the total weight of each node is then:

$$w_n = \sum_{i=0}^{P} w_{n_i} \tag{3.2}$$

This sum, w_n, represents the contributions of all contextual relationship paths to node n plus the contribution of its original content ranking. The set of nodes in N are then sorted by weight from highest to lowest to create the final ranking.

As an example of how the algorithm works, assume that the graph in figure 3.4 is the result graph, the path length is 2, the weight cutoff is 10%, $\alpha = 0.25$, and the content-based search returns $w_{D_0} = 4$ and $w_{B_0} = 2$. Consider w_B. On the first pass of the algorithm, w_{B_1} is updated based on w_{D_0} and w_{E_0}. In this case, $w_{E_0} = 0$, so only w_{D_0} affects its value:

$$w_{B_1} = 4 \cdot [(2/10) \cdot 0.25 + 0.75] = 3.2 \tag{3.3}$$

On the second pass, $w_{E_1} = 3.8$ (using the formula from above) and $w_{D_1} = 0$, thus:

$$w_{B_2} = 3.8 \cdot [(8/113) \cdot 0.25 + 0.75] = 2.92 \tag{3.4}$$

The final weight w_B is then the sum of these weights, $2 + 3.2 + 2.92 = 8.12$.

3.4.2 Web ranking algorithms

This dissertation considers two web ranking algorithms: PageRank and HITS. Although other algorithms could be considered, the objective of this work is not to make an exhaustive study of the available ranking algorithms, but to show the value of context-enhanced search and to understand which classes of algorithms are likely to succeed based on the structure of the relation graph.

PageRank

PageRank is the ranking algorithm used by the Google web search engine [16]. It takes the graph of hyperlinks in the web and calculates the principal eigenvector of a stochastic transition matrix describing this graph. This eigenvector describes the probabilities for reaching a particular node on a random walk of the graph. This probability is referred to as a page's *PageRank*.

This dissertation uses an implementation of the Power method described in [44] to calculate the PageRank of each file in the relation graph. Unfortunately, Google's method of merging content search results with PageRank is not clearly documented[2], thus I implemented three possible uses for a file's PageRank.

The first implementation, PR-only, ignores the content rankings, and uses only PageRank to rank the files within the result graph. The second implementation, PR-before, applies a file's PageRank to its content-based ranking (i.e., take the product of the original ranking and the PageRank as the new ranking) and then runs the Basic-BFE algorithm. The third implementation, PR-after, runs the Basic-BFE algorithm and then applies the PageRank to the final results.

HITS

The HITS algorithm attempts to locate *authority* and *hub* nodes within a graph. Authority nodes are those with incoming links from many hub nodes, while hub nodes are those with outgoing links to many authority nodes. In the web, authorities are analogous to pages linked many times for a particular topic (e.g., the official CNN web site), while hubs are

[2] "Finally, the IR score is combined with PageRank to give a final rank to the document." [16]

analogous to pages with lists of links to authorities (e.g., a page with links to all news websites).

HITS identifies authorities and hubs using three steps. First, it runs a content-based search to locate an initial set of nodes. Second, it creates a sub-graph of the relation graph by locating all nodes with incoming/outgoing links from/to the starting nodes. Third, it runs a recursive algorithm to locate the "principal eigenvectors of a pair of matrices M_{auth} and M_{hub} derived from the link structure" [46]. These eigenvectors indicate the authority and hub probabilities for each node.

This dissertation considers two separate implementations of HITS. The first implementation, HITS-original, runs an unmodified version of HITS using the relation graph. The second implementation, HITS-new, begins with the result graph derived in Section 3.3, and then runs only the third part of the HITS algorithm. The evaluation in Chapter 7.1 examines both the hub and authority rankings of each implementation.

3.4.3 Super-nodes

Due to the behavior and interaction of many applications, the relation graphs generated by the algorithms in Section 3.2.3 often contain nodes that are strongly connected but have little relevance to the user. For example, the *bash* login shell maintains a history file that tracks recently executed commands. Because it is updated every time a user performs an action, it becomes strongly connected to many files over time.

These *super-nodes* can mislead graph-based ranking algorithms into believing that they and their neighbors are more relevant than they should be. For example, most web ranking algorithms are designed to find *authority* pages that are heavily referenced by other pages on the web, which can match closely with the behavior exhibited by super-nodes.[3] To try to reduce the misleading nature of super-nodes, I examined four ways of modifying the Basic-BFE ranking algorithm to detect them and weaken their relevance: PR-detect, clique-detect, stddev-detect, and percentile-detect.

The *PR-detect* algorithm uses the PageRank algorithm to try to detect super-nodes. As mentioned above, web-ranking algorithms tend to rank super-nodes highly, because super-nodes mimic some of the behaviors of authority nodes on the web. PR-detect is similar to the PR-after scheme described in Section 3.4.2, except that instead of taking the product of the original rank and the PageRank, PR-detect takes the quotient.

The *clique-detect* algorithm uses the "clustering coefficient" [75] of the graph formed by a node and its neighbors to try to detect super-nodes. The clustering coefficient of this graph is a measure of how close the node and its neighbors are to forming a clique.[4] Equation 3.5 gives the formal definition of the clustering coeffcient C_n of the graph formed by node n and its neighbors. Let M_n be the set of neighbors of node n.

[3]Full discussion of the adverse affects of super-nodes can be found in Chapter 7.1.

[4]For simplicity purposes, links in the relation graph are treated as undirected when creating the graph of a node and its neighbors.

$$C_n = \frac{|M_n| + \sum_{m \in M_n} |M_n \bigcap M_m|}{(|M_n|)^2 - |M_n|} \tag{3.5}$$

A node's clustering coefficient is thus defined as the ratio of the number of links between it and its neighbors to the number of links that would exist if it and its neighbors formed a clique. The intuition behind this approach is that super-nodes will form very poor cliques with their neighbors because many of their neighbors will be unrelated and, thus, not connected to each other. Ideally, taking the product of a node's clustering coefficient and its original rank should reduce the relevance of super-nodes.

The *stddev-detect* algorithm uses the number of links connected to a node to try to detect super-nodes. In this approach, the system calculates both the average number of links going to and from each node in the relation graph and the standard deviation. It then penalizes links coming from or going to nodes that exceed the average by one or more standard deviations, with the number of deviations for node n being notated as D_{out_n} and D_{in_n}, respectively. Specifically, it applies a penalty β for each deviation above the average, modifying equation 3.1 to be equation 3.6:

$$w_{m_i} = \sum_{e_{nm} \in E_m} w_{n_{(i-1)}} \cdot \beta^{D_{out_n}} \cdot \beta^{D_{in_m}} \cdot [e_{nm} \cdot \alpha + (1 - \alpha)] \tag{3.6}$$

This reduces the effect of super-nodes by reducing both their relevance from incoming links ($\beta^{D_{in_m}}$) and their effect on the relevance of their neighbors ($\beta^{D_{out_n}}$). The potential drawback of this approach is that the use of standard deviation relies on the assumption that the distribution of link counts is Gaussian.

The *percentile-detect* algorithm, like stddev-detect, also uses the number of links connected to a node to try to detect super-nodes, but instead penalizes only those nodes with link counts that are part of the 95th percentile and above. Specifically, it applies a penalty β as shown in equation 3.7:

$$w_{m_i} = \sum_{e_{nm} \in E_m} w_{n_{(i-1)}} \cdot \beta_{out_n} \cdot \beta_{in_m} \cdot [e_{nm} \cdot \alpha + (1 - \alpha)] \tag{3.7}$$

In the normal case, $\beta = 1$, thus there is no penalty to most nodes. Nodes above the 95th percentile for outgoing or incoming links are penalized by $\beta = \beta_{95}$ while nodes above the 99th percentile are penalized by $\beta = \beta_{99}$.

While exploring algorithms for reducing the effect of super-nodes on relevance, I chose to only modify the Basic-BFE algorithm. The main reason for this was that the Basic-BFE algorithm outperformed the other ranking schemes during my evaluation; I believed that, by modifying the best performing scheme, I would likely end up with the best possible end result. Of the algorithms that I examined, the last three, clique-detect, stddev-detect, and percentile-detect, could be applied to the web-ranking algorithms. The clique-detect algorithm could be applied unmodified, while the stddev-detect and percentile-detect could be used to modify link probabilities before running PageRank or HITS. Examination of these applications is left as future work.

3.4.4 Trade-offs

Despite the breadth of ranking algorithms described above, the collection explored here is far from exhaustive. As the exploration of context-enhanced search continues, the distinction in behavior that differentiates web search from file system search may become clearer, helping to identify how existing web techniques can be more successfully applied to file system search. Furthermore, as related web ranking techniques such as relevance propagation are studied more, it may be useful to integrate these techniques more closely with the algorithms I have described above.

3.5 Wrap-up

This chapter describes an architecture and set of algorithms for implementing context-enhanced search, with the goal of being able to explore the effects of applying context on both the utility and performance of personal file system search. Each set of algorithms also provides various parameters that enable more thorough exploration of the space of context-enhanced search, with the goal of understanding which techniques are most effective in satisfying user queries. The next chapter describes two implementations of this architecture, describing details of the components that gather and store the data required by context-enhanced search.

4 Connections

To evaluate context-enhanced search, I built Connections, an implementation of the context-enhanced search architecture described in Chapter 3. The key components of the implementation are the tracer that gathers system call traces from the system, the database for storing and querying the relation graph, and the content analysis system.

4.1 Tracing

To run Connections under both Linux and Microsoft Windows requires a separate tracing infrastructure for each system. In both cases, the tracing component logs file system calls to a file that can later be read by the indexing component of Connections; however, the specific details of each implementation differ.

4.1.1 Linux

The Linux implementation of Connections was developed to explore the parameter space of the algorithms described in Chapter 3. As such, the tracing component was designed to capture as much information as possible, providing the widest array of information.

Architecture

Figure 4.1 illustrates the Linux tracing infrastructure. The Linux tracer is implemented as a kernel module that intercepts calls between the application and the file system by interposing wrapper calls on the kernel's system call table.[1] It runs under both 2.4 and 2.6 versions of Linux, although a kernel patch is required under 2.6 kernels to export the system call table to loaded modules.

My tracing module is similar in design to other Linux tracing tools, such as the Linux Trace Toolkit [81]. When a call is first seen, a timestamp is generated and space is reserved in the trace buffer for that call. When the call is completed, missing arguments (such as the return value) are filled in and stored in the buffer. Once a certain threshold of data has been gathered, data is flushed from the buffer to a trace file stored in the file system. A virtual device driver provides applications with an interface to control the tracer: deciding

[1]To allow for file descriptor reconstruction, the tracer also watches some process management calls such as FORK() and EXEC().

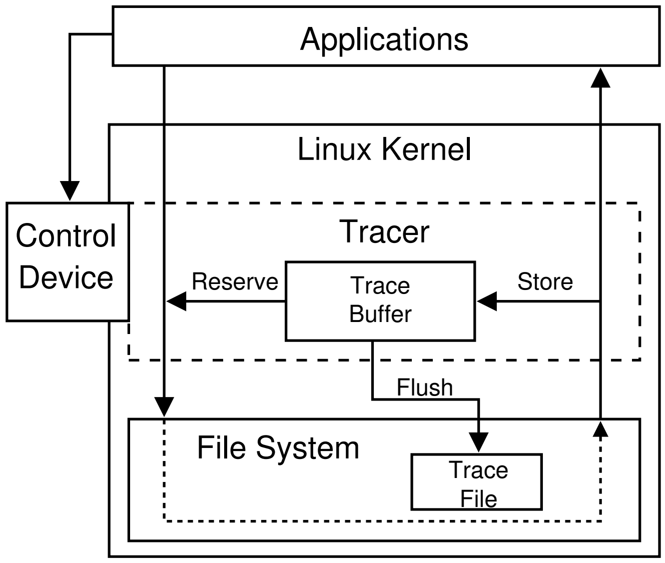


Figure 4.1: Linux tracer components

which calls should be traced, setting the size of the trace buffer, and starting/stopping the tracing.

The key difference between my module and most other tracing tools for Linux is that my module will block applications to avoid missing trace entries, rather than discarding entries. Although this can reduce system performance, I chose to err on the side of having complete information. Furthermore, using a trace buffer of 1 MB made such blocking extremely infrequent. As such, the overheads of my Linux tracer are similar to those of other tracing systems (less than 2.5% when measuring system call events) [81].

State reconstruction

The watched calls are listed in Table 4.1. Because Linux's file system interface follows POSIX guidelines, all data accesses are made through file descriptors. As such, Connections must recreate the system's process state, when scanning the trace, to match file descriptors to file names.

The traced "Process Management" and "File Descriptor Management" calls are used for this purpose. While scanning a trace, Connections maintains the set of active processes as well as the file descriptors and working directory associated with each. Using the "Process Management" calls, Connections tracks file descriptors passed to child processes through FORK() calls, implicit CLOSE() calls caused by process EXECVE() and EXIT(), and working directory updates through CHDIR() and CHROOT() calls. Using the "File Descriptor Management" calls, Connections tracks the file name of newly created file descriptors by combining

System Call	**Description**
File descriptor management	
OPEN(), CREAT()	Opens a file, returning a file descriptor
CLOSE()	Closes a given file descriptor
DUP(), DUP2()	Copies a file descriptor
Data access	
READ(), READV(), PREAD()	Reads data from a file through a file descriptor
WRITE(), WRITEV(), PWRITE()	Writes data to a file through a file descriptor
TRUNCATE(), FTRUNCATE()	Removes data from a file
Directory management	
LINK()	Creates a link to a file in a directory
MKDIR()	Creates a new direcotry and links it
MKNOD()	Creates a link to a device interface
RENAME()	Relinks a file
RMDIR()	Unlinks a directory
SYMLINK()	Creates a soft link to a file
UNLINK()	Unlinks a file
GETDENTS(), READDIR()	Reads entries from a directory
READLINK()	Reads the contents of a soft link
Metadata management	
ACCESS(), CHMOD(), FCHMOD()	Read/write the permission bits of a file
CHOWN(), LCHOWN(), FCHOWN()	Write the ownership of a file
UTIME(), UTIMES()	Read/write timestamps of a file
STAT(), LSTAT(), FSTAT()	Read all of a file's metadata
Process management	
FORK(), VFORK(), CLONE()	Fork a process
EXIT()	Exit a process
EXECVE()	Execute a file's contents in this process
CHDIR(), FCHDIR()	Change the working directory of a process
CHROOT()	Change the root directory of a process

Table 4.1: Traced calls in Linux

the working directory of the process with the name passed to OPEN() and CREAT() calls, and uses DUP() and CLOSE() calls to track the active file descriptors for each process.

Filtering

The access filter described in Chapter 3.2.3 must choose how it filters based on the available system calls. As such, the Linux version of Connections implements three access filters: *open*, *read/write*, and *all-calls*. The open filter treats executed files of EXECVE() and files accessed by OPEN() calls as both inputs and outputs, similarly to traditional probability graphs. The read/write filter treats files accessed by READ() calls as inputs and files accessed by WRITE() calls as outputs. The all-calls filter uses all of the data access, directory management, and metadata management calls, treating calls that read information as inputs and calls that write information as outputs. Specifically, files accessed by READ(), ACCESS(), UTIME(), and STAT calls, as well as the starting file of LINK(), RENAME, and SYMLINK calls, are treated as inputs; files accessed by WRITE(), TRUNCATE(), CHMOD(), CHOWN(), UTIMES(), as well as the newly created file of LINK(), RENAME, and SYMLINK calls, are treated as outputs.

4.1.2 Microsoft Windows

Unlike the Linux implementation, the Microsoft Windows implementation of Connections was designed specifically for the online study described in Chapter 5. As such, its tracing infrastructure is designed to minimize overheads and to target the specific information required for the parameter settings used in the online study.

Architecture

To trace Microsoft Windows machines, Connections uses VTrace, a publicly available tracing tool developed by Jacob Lorch that runs under Windows 2000, Windows XP, and Windows 2003 Server [50]. The general architecture of VTrace is similar to the Linux tracer, but VTrace has more flexibility in calls that it can trace and can also trace events passed from user-space applications.

To reduce the overheads of tracing and the size of trace files, Connections uses a modified version of VTrace that removes all calls not associated with the file system. Further, due to the frequency at which Windows accesses system files, Connections's modified VTrace also provides an option to filter file system calls at the tracing layer based on a user-defined set of directories.

State reconstruction

The watched calls are listed in Table 4.2. Data accesses in Microsoft Windows use file handles, similar in behavior to POSIX file descriptors. Unlike the Linux tracer, however, VTrace performs some of the state reconstruction at trace time. Specifically, it keeps an internal table of recently seen file handles. Any handle not in the table seen during tracing is resolved to a full file path, and a special FILENAME trace entry is made to map the file

System Call	Description
READ()	Reads data from a file through a file handle
WRITE()	Writes data to a file through a file handle
PROCESSCREATE()	Create a new process from a given executable file
FILENAME	Indicates the full file path of a newly seen file handle

Table 4.2: Traced calls in Microsoft Windows

handle to the resolved path. This removes the need for Connections to track file handles as they are duplicated or passed to child processes.[2] Thus, instead of reconstructing the entire system state, Connections only keeps a mapping from ⟨pid, file handle⟩ pairs to file names, updated by FileName entries. The file names for all other entries can then be identified by consulting the file handle table.

Filtering

The Windows version of Connections implements a *read/write* filter that acts similarly to the Linux read/write filter. The read/write filter treats files accessed by READ() calls as inputs and files accessed by WRITE() calls as outputs. Although it would have been possible to implement other filtering schemes, the purpose of the Microsoft Windows implementation was for use in the online study described in Chapter 5, and thus it was optimized for the chosen parameters.

4.2 Database layout and interaction

Connections implements the relation graph using BerkeleyDB to store and retrieve the relationships. BerkeleyDB implements tables using a fast, b-tree implementation that maintains ACID properties.

Connections maintains five tables in the database. The *FileID* table maps full file paths (stored as arbitrary strings) to node IDs (stored as 4 byte integers). The *IDFile* table maps node IDs to full file paths. Note that the data storage for the FileID and IDFile tables are shared, with separate indexes (one index on the node ID, the other on the file path). The *NodeOut* table maps node IDs to a structure describing the outgoing links from that node. The *NodeIn* table maps node IDs to a structure describing the incoming links to that node. The *PageRank* table maps node IDs to the PageRank of that node in the graph (stored as a 4 byte floating point value). The structure used to store link information in both the NodeOut and NodeIn tables is an array of tuples containing a 4 byte weight and 4 byte node ID.

[2]Although already present in VTrace, this file name reconstruction also simplified the implementation of user-defined directory filtering.

During indexing, Connections updates the tables to reflect the new inter-file relationships identified using the algorithm described in Chapter 3.2.3. The FileID and IDFile tables are updated only when new files are accessed by the user, with node IDs being assigned in sequential order. During trace analysis, Connections maintains a cache of identified relationships. After trace analysis is complete (or if the indexing system runs low on memory) the NodeOut and NodeIn tables are updated from the cache. Once the NodeOut table has been fully updated, Connections uses it to run PageRank analysis, updating the values in the PageRank table.

During querying, Connections consults the tables to run the searching and ranking algorithms described in Chapters 3.3 and 3.4. The first step of a query is to convert the list of results from content analysis from file names into node IDs using the FileID table. The second step is to run the search algorithm, consulting both the NodeOut and NodeIn tables to perform weight cutoff while performing the breadth-first expansion. The third step is to run the ranking algorithm, using either the NodeOut table (in the case of Basic-BFE and HITS) and/or the PageRank table (in the case of PageRank). The fourth step is to convert the node IDs back into file paths to return to the user.

4.3 Content-only search tools

Although Connections's context-enhanced search is algorithmically agnostic to the underlying content analysis tool, its current implementation is designed to take input from Indri (version 2.1) [49]. The Indri toolkit is a research-driven content-only analysis system developed primarily for web search. I chose Indri because it uses many of the latest techniques in information retrieval, has performed well at the TREC conference for several years [2, 1], has a publicly available code-base, provides confidence values on its ranked results, and provides a mechanism to tag documents with searchable metadata. Also, Indri's flexible query language allows exploration of alternate content analysis techniques without having to change the underlying tools; and, although designed for the web, Indri performs no link analysis, making it an ideal candidate for use in Connections.

The drawback of Indri is that its web-oriented design makes assumptions about data contents and the utility of information such as file names and paths. As such, Connections required two modifications to Indri's indexing process: identifying which files contained indexable content (to avoid polluting the index with binary data) and indexing not only a file's contents but also its file name and path (this is especially important in the case were a file's contents cannot be indexed).

To identify which files were content-indexable, I made use of the Unix *file* command. The *file* program identifies a file's contents using a combination of file system metadata information to identify special file types (e.g., symlinks or empty files), magic number tests to identify file types with common header formats, and content analysis to identify files with console-readable contents. Any files that were identified as containing parsable text[3]

[3]This includes Indri-parsable formats such as text, PDF, MS PowerPoint, MS Word, HTML and XML.

were included in the content-indexing process. All other files had only their file name and path included in the index.

To index a file's name and path, I made use of Indri's searchable metadata feature. I modified Indri's indexing process to take the path of the file and decompose it into a set of words broken on non-alphanumeric character boundaries. Each of these words were included into the file's metadata under the "filepath" heading, providing a searchable "filepath" index that could be used during querying.

To make queries agnostic to the underlying content analysis technique at use, queries were provided as a set of keywords k_1 through k_n and an optional set of file types. For example, a user searching for photos of a birthday party might use the keywords ⟨birthday party⟩ and the file types ⟨jpeg, gif, bmp⟩. The methodology for mapping *keyword queries* to the underlying query technique are described below for the three techniques explored in Chapter 7.2.

$$\#weight(1.0\ \#combine(k_1,\ ...,\ k_n)\ 2.0\ \#combine(k_1.filepath,\ ...,\ k_n.filepath)) \tag{4.1}$$

Equation 4.1 describes Connections's default mechanism for mapping keyword queries to Indri's underlying query language. The #combine() operator considers each of the keywords with equal weight when satisfying the query. The #weight() operator considers each portion of the query with different, specified weights. In this case, all keywords are considered equally, and the files' contents are given half as much weight as the files' names. The file type filter was applied following the search using a boolean OR operation (i.e., any file with one of the matching file types was displayed).

$$\#band(k_1\ ...\ k_n) \tag{4.2}$$

Equation 4.2 describes how Connections maps keyword queries into an Indri query that simulates the behavior of Glimpse [54]. This query uses a boolean AND operator to find files that have contents matching each of the keywords in the query. Just as in Glimpse, this query ignores filenames entirely. The file type filter was applied following the search using a boolean OR operation (i.e., any file with one of the matching file types was displayed).

The last content analysis technique explored uses the query translation described in Equation 4.1, but adds Indri's implementation of pseudo-relevance feedback [79, 10]. Pseudo-relevance feedback uses content similarity to identify additional keywords that are relevant among top-ranked results. These additional keywords are then used to modify and re-rank the content results before they are passed to the relation graph. The two parameters required are the number of results to consider for content similarity and the maximum number of additional terms to generate, for which Connections uses Indri's default values: 10 and 20, respectively.

4.4 Wrap-up

Connections is a flexible, cross-platform context-enhanced search tool designed specifically for comparing context-enhanced search to traditional content-only search. The next two chapters evaluate Connections using offline analysis methods and an online user study.

5 Utility evaluation

The performance of any search tool has two aspects. The first is the utility of the search tool (can users find what they are looking for?), and the second is the speed with which it performs. This chapter deals with the the utility of context-enhanced search both in terms of result quality and user experience, Chapter 6 evalutes the performance overheads of context-enhanced search, and Chapter 7 evaluates the sensitivity of the system to changes in the underlying parameter settings and algorithms.

The utility evalution includes the results of two user studies: an offline study performed with Linux users and an online study performed with Windows users. The advantage of offline analysis is that a wider array of schemes can be compared with far less user interaction than is required in online analysis. The advantage of online analysis is that aspects of a user's interaction with the tool can be studied, as opposed to strict recall and precision based metrics.

This chapter is divided into two sections. The first section describes the evaluation methodology for both studies, including how user information is gathered and the metrics used. The second section overviews the results from each study, describing the advantages of context-enhanced search.

5.1 Methodology

The methodologies for the two user studies differ significantly. The first study, which I refer to as the *offline* study, was designed to gather information required for traditional information retrieval metrics (i.e., recall and precision). This study also focused on metrics that could be evaluated independently of user behavior, providing a way to analyze the sensitivity of the system to different parameters and algorithms. The second study, which I refer to as the *online* study, was designed to gather information required to answer questions more related to the user's interaction with the system over time (e.g., average success rate, average search time), focusing more on the utility experienced by the user.

5.1.1 Offline study

To compare the utility of content-only search to context-enhanced search, the offline study borrowed and adapted evaluation techniques from information retrieval [10]. Traditionally, content-only search tools are evaluated using large public corpora of data, such as archived newswire data or collections of publicly accessible websites. Queries are generated by experts and evaluated by individuals familiar with the material. These "oracle" results are then compared to the results generated by the system under evaluation.

Unfortunately, three subtle differences make file system search, and especially context-enhanced search, more difficult to evaluate. First, because the data is personal, only its owner can create meaningful queries and act as expert for evaluating a query and generating "oracle" results. Second, the nature of the queries (searching for old data) demand that traces exist over a long period of time; Connections cannot provide context-enhanced results if it has no traces that describe the usage of the desired data. Third, queries for data that are not accessed during the trace period provide no information about the accuracy of context-enhanced search. Because we already know that context-enhanced search cannot assist in these cases, it is preferable that users generate queries for data accessed during the trace period, placing additional constraints on the query generation process.

Metrics

Recall and precision measure the effectiveness of the search system in matching the oracle results. A system's *recall* is the number of relevant documents retrieved divided by the total number specified by the oracle. A system's *precision* is the number of relevant documents retrieved divided by the total number of documents retrieved.

Unfortunately, only the user of a system knows their data well enough to act as oracle for their queries, and the users in this study were not willing to examine every file in their systems to find all of the correct results for each query. To account for this, I used a technique taken from information retrieval known as pooling [10]. Pooling combines the results from a number of different search techniques, generating a set of results with good coverage of relevant files. Users then specify the correct results from this smaller set, generating the oracle results for each query. The potential disadvantage of pooling is that there may be valid results that are not included in the set presented to the user and, thus, will not be marked as correct by the user. Fortunately, while this will potentially skew the recall of all of the systems under comparison higher (since they will appear to include a higher percentage of correct results), the relative comparison between systems is still valid (since they are all measured against the same standard).

For this study, I pooled the first hundred results from a number of different runs of Connections with wide variety of indexing and searching parameters, including the default settings being evaluated, and presented them to users.[1] Users then chose the relevant documents from this pooled set of files to create the oracle.

I compare the recall and precision of different systems using two techniques. The first technique is to examine the recall/precision curve of each system. This curve plots the precision of the two systems at each of 11 standard recall levels (0% - 100% in 10% increments) [10]. At each recall level n, the curve plots the highest precision seen between n and $n + 1$. If a given recall level is not achieved for a given query, its precision at that level is

[1]Specifically, all of the parameters explored in section 7.

reported as 0%. To calculate the user-averaged recall/precision values over a set of queries, the precision of each query at a given recall level is calculated, the precision of the queries for a given user are averaged, and then the individual user results are averaged across all users. Examining this curve shows how well a system ranks the results that it generates.

The second technique is to examine the recall and precision of each system with fixed numbers of results.[2] Most search systems present only a few results to the user at a time (e.g., a page with the first 10 results), requiring prompting from the user for more results. For example, result cutoffs of 10, 20, and 30 may map to 1, 2, or 3 pages of results, after which many users may give up or try a different query. Examining the recall and precision at low result cutoffs indicates how quickly a user could locate relevant data with the system. Examining the recall and precision at the last returned result shows how many relevant results can be located using the system. In this case, results are examined until either (1) recall reaches 100% or (2) all results returned by the system are considered.

Experimental setup

To gather context data, I traced the desktop computers of six computer science researchers for a period of two years. Users submitted and evaluated queries at two points during this time, once after six months of tracing and again after two years of tracing, for a total of 52 queries. After evaluation, 18 of these queries were discarded because neither Connections nor Indri returned correct results within the first 30, leaving a total of 34 usable queries.

Queries were in the keyword query format used by Connections and were mapped to Indri using the technique described in Chapter 4.3. The default parameter settings of Connections's relation graph search algorithm were a path length of 3 and a weight cutoff of 2%. The default ranking algorithm was Basic-BFE with an α setting of 0.5 and the percentile-detect super-node detection algorithm with $\beta_{95} = 10^{-2}$ and $\beta_{99} = 10^{-4}$.[3]

Using the traces, I generated two relation graphs for each user, one from the first six months of traces and one from the entire two years of traces, using the following default parameters: a 30 second relation window, a directed edge style, and a read/write operation filter.[4]

To match the content index with the context index, I built two content indexes for Indri, the first using the state of the user's data after the initial six months of tracing and the second using the state of the user's data after the entire two years of tracing. I chose to only index files from the user's specified home directories, with the assumption that most users create and store their files in that location. Furthermore, indexing this data alone, as opposed to the entire file system, reduces Indri's false positive rate.

[2] Assuming that the system returns this many results and hasn't already reached 100% recall before reaching the cutoff.

[3] I chose these settings after performing the sensitivity analysis described in Section 7.1.

[4] Again, I chose these settings after performing the sensitivity analysis described in Section 7.1.

5.1.2 Online study

The online study moves away from the traditional information retrieval metrics of precision and recall and instead focuses on metrics that measure user-percieved improvements. This choice was precipitated by two aspects of the online system. First, because the system's interface was designed to act as a traditional file search tool, I wanted to minimize any additional interactions required from the user. Because having the user mark all correct results in the system is more effort-intensive than allowed, it was impossible to get accurate precision and recall numbers for a search. Second, gathering all of the results required for pooling can often be quite computationally intensive and, if the search tool performed poorly, users would be far less likely to use it.

Metrics

The online study uses four metrics to measure how a user percieves the search tool's utility. The first metric (and perhaps most relevant) is whether or not a search was successful; i.e., did the user find what they were looking for? If the user can more often find what they are searching for, then the tool is more useful. The second metric is the amount of time spent examining results. This indicates how quickly the system can find correct results for the user; if two systems have identical success rates, but system A allows the user to succeed in half the time, then system A is likely percieved as being more useful. The third metric is the number of results examined while scanning the results. Similarly to the amount of time spent searching, if the number of results examined is fewer for successful searches, then it is likely that the user found what they were looking for with less effort. The fourth metric is the retry rate: how many unsuccessful searches are performed (in sequence) before a successful search is completed? Often, a user performing an unsuccessful search will try the search again with slightly different, but synonymous, keywords. This metric helps to measure how accurately a system can match the keywords provided by the user to the data they desire.

Exerimental setup

The online study was run in three phases. During the first phase, users were presented with the results of content-only search; by mimicking the behavior of traditional file system search tools, this phase allowed the users to become familiar with the interface of the tool without altering expectations about the tool's returned results. During the second phase, users were presented with the results of the context-enhanced search; this phase measured the utility of the returned results at a point when users were already familiar with the search interface. During the third phase, users were again presented with the results of the content-only search; this phase measured the effect of switching back to content-only search, ensuring that the utility seen in phase two was not due to increased familiarity with the tool. Users were informed about the techniques being studied and the study methodology, but were not informed as to the length of phases or which phase was active.

Entry	Description
OPEN	Start time of the query.
VIEW	Viewing a particular range of results.
CLICK	Double-clicking or copying a particular result.
SURVEY	User specified success/failure, and re-query attempts.

Table 5.1: Log entries generated by the online user study interface

Given that keyword queries are agnostic to the underlying retrieval technique (as discussed in Chapter 4), I was able to provide a query interface for Microsoft Windows that could transparently run a query on either Connections or Indri. Once users had submitted a query, results were displayed as sets of ten results, with "next" and "prev" buttons for navigation. Users could double-click results to open a file or copy a result's location into the clipboard. When the user closed the result list, a final window would open, surveying them to determine if the query was a success or failure and offering them a chance to perform a new query. Simultaneously, the query interface logged all user actions to gather information required to calculate the metrics described above.

Table 5.1 lists the different log entries generated by the query interface. When a user first submits a query, the interface generates an OPEN entry, marking the time of the query. Whenever the interface displays a set of results to the user (directly after a query or whenever the "next" or "prev" button is clicked), it generates a VIEW entry, specifying which results were displayed. Whenever a user double-clicks or copies a result, the interface generates a CLICK entry, specifying which result was accessed. After the user completes the survey, the interface generates a SURVEY entry, specifying success or failure and if the user attempted another query.

These four entries are sufficient to calculate each of the four metrics described above. The SURVEY entry logs success rates as specified by the user. The amount of time spent examining the results of a particular query can be estimated by subtracting the timestamp of the OPEN entry (always the first entry in the log) from that of the SURVEY entry (always the last entry in the log). The number of results examined is calculated with two values: the number of items accessed by the user, as logged by CLICK entries, and the number of items viewed by the user, as logged by VIEW entries. The retry rate is calculated as the number of successful queries that required more than one query to attain success. Any SURVEY entry that registers failure is checked for a retry, and the combination of a retry with a second query log containing an OPEN entry within one minute of the retry is considered a re-query.

5.2 Overview

The objective of these studies is to explore the utility of context-based inter-file relationships as applied to search. This section discusses a direct comparison of Connections's context-

Cutoff	Recall %		Precision %	
	Indri	Connections	Indri	Connections
10	24	32	44	52
20	28	40	38	43
30	32	45	34	39
50	39	54	32	36
100	43	58	30	34
Last returned result	45	80	26	27

Table 5.2: Precision and recall at set cutoffs from the offline user study

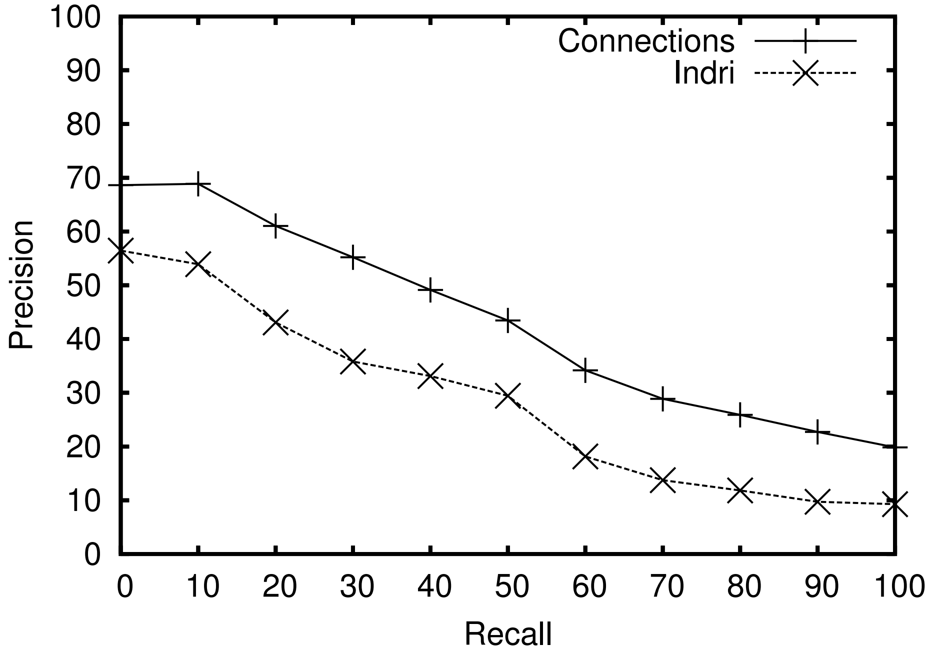
enhanced search to Indri's content-only search from the two user studies.

5.2.1 Offline results

Figure 5.1 illustrates the benefits of adding context information to traditional content-only search, as shown by a precision/recall curve user-averaged over the the 34 queries for both Indri and Connections. At each recall level, Connections's context-enhanced search improves precision by 10% to 20%. A paired t-test across the queries shows a p-value less than .05 for the precision at each recall level and for the average increase in precision, indicating that the precision of Connections is "significnatly" higher than that of Indri alone. The result is two-fold. First, because precision is higher, the user is likely to find what they are looking for more quickly, since correct results are presented to the user sooner. Second, because 100% recall is achieved more often (thus the higher precision at this level), the user is more likely to find everything that pertains to a given query.

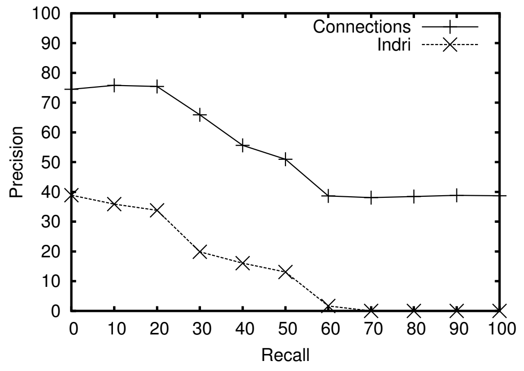
The advantages of context-enhanced search can also be seen when looking at the precision and recall at different results cutoffs. Table 5.2 lists the user-averaged precision and recall of Indri and Connections for the 34 queries at six different result cutoffs. Again, this shows the improvements provided by context-enhanced search; Connections improves both recall and precision at each cutoff. Particularly interesting are the results at a cutoff of 10 results and at the last returned result. At a cutoff of 10, Connections improves precision from 44% to 52%, making it more likely for a user to see a correct result on the first "page" returned by the system. At the last returned result, Connections improves recall from 45% to 80%, while still maintaining the same level of precision. This means that, with the same false-positive rate across results, Connections was able to return more of the results that the user was interested in for the queries. This makes it more likely that a user could find all of the data relevant to a given topic.

One of the key advantages of Connections is that it does not rely on file contents to identify relevant data. As such, it is able to locate relevant results, even when the desired results do not contain parsable contents (e.g., images, music). Figure 5.2a shows the user-averaged precision/recall curve for both Indri and Connections over just those queries where

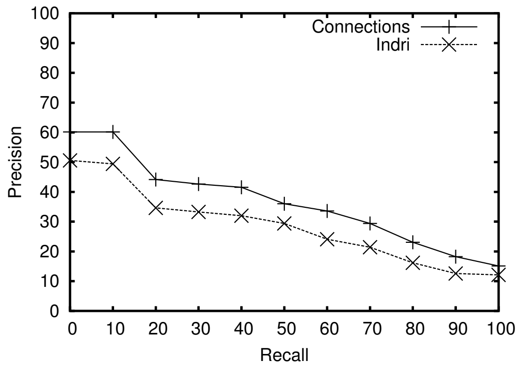


Recall %	**Precision %**	
	Indri	**Connections**
0	56	69
10	54	69
20	43	61
30	36	55
40	33	49
50	29	43
60	18	34
70	14	29
80	12	26
90	10	23
100	9	20
average	28	43

Figure 5.1: Average precision/recall curve over all queries in the offline user study



(a) Overview of non-text searches



(b) Overview of text-only searches

Figure 5.2: Breakdown of non-text versus text-only queries

the correct results include non-texual files. In these cases, Connections's ability to locate files that contain no parsable content increases precision by 30% to 40% across all recall levels.

Conversely, figure 5.2b shows the user-averaged precision/recall curve for both Indri and Connections over just those queries where the correct results include only files with parsable, textual contents. In these cases, the benefits of Connections is far less significant, increasing precision by only 3% to 10%. Although most of Connections's benefits are derived from its ability to correctly identify relevant non-texual results, the contexual relationships formed from repeated user behavior can help to identify the most relevant results from a set of files. So while it does not find significantly more correct results during text-only searches (as shown by its only slight improvement in precision at 100% recall) it is able to provide a slightly more accurate ranking in some cases (thus the improvement in precision at lower recall levels).

The benefits of Connections, in both cases, stem from its ability to locate results that are relevant to the user specified keywords, but do not contain the keywords themselves (or make very infrequent use of them). Connections locates these files because the user accesses them together with files that do contain the specified keywords. By identifying which of these new results are most relevant and placing them correctly in the rankings, Connections is able to successfully combine the new results with the content results, providing not only higher recall but also higher precision. While this technique is far more effective in the case where the desired data has no keywords (i.e., non-text data), there are still cases in text-only queries where the user's specified keywords do not match the contents of the desired files. Examples of this behavior from individual queries are discussed below.

Category Description	**Query Num**	**Indri**			**Connections**		
		Results	**R %**	**P %**	**Results**	**R %**	**P %**
Non-text queries	1	14	0	0	30	100	37
	2	8	0	0	29	100	7
	3	30	0	0	30	47	23
	4	2	25	100	12	38	25
	5	0	0	0	18	100	89
	6	30	23	10	30	46	20
	7	30	12	87	30	13	100
	8	30	47	53	30	47	53
	9	30	31	37	30	31	37
	10	30	50	57	30	50	57
	11	30	56	73	30	56	73
	12	30	13	27	30	13	27
Text queries	13	30	17	3	30	33	7
	14	30	20	60	30	24	73
	15	30	22	93	30	22	93
	16	30	20	60	30	24	73
	17	30	31	17	30	31	17
	18	23	100	30	26	100	27
	19	30	0	0	30	1	7
	20	30	57	27	30	57	27
	21	30	100	3	30	0	0
	22	30	20	7	30	60	20
	23	30	8	57	30	10	67
	24	30	50	3	30	50	3
	25	30	15	13	30	15	13
	26	30	17	97	30	17	97
	27	30	5	30	30	6	37
	28	6	75	50	6	75	50
	29	30	40	7	30	40	7
	30	30	5	17	30	5	17
	31	2	100	100	2	100	100
	32	1	100	100	1	100	100
	33	30	80	40	30	80	40
	34	30	25	3	30	25	3

Table 5.3: Overview at cutoff of 30 results

Category Description	Query Num	Query Terms	File Types (optional)
Non-text queries	1	postal service	mp3
	2	songs artist Led Zeppelin	mp3
	3	15-768 storage systems homework	ps, sxw
	4	comic strips	gif, jpg, png
	5	continuous reorg visit day	eps, bmp, wmf, ai, pdf
	6	dreamcast games	
	7	attack	
	8	florence	
	9	rome	
	10	venice	
	11	noel	
	12	mother	
Text queries	13	able ICAC	ps, pdf
	14	pasis read write	ps, pdf
	15	erasure code	ps, pdf
	16	optimistic versioning	ps, pdf
	17	USENIX Anaheim California	pdf
	18	bus ridegold schedule	pdf
	19	anneka winters	
	20	wedding marriage 2004	
	21	train yoga sport	
	22	ahmer noah	
	23	Dushyanth	
	24	Evans	
	25	fellowship	
	26	machine learning	
	27	magpie	
	28	communists socialists	
	29	latex guide	
	30	project management	
	31	austin cover letter	
	32	dan ellard protocol	
	33	relocation package	
	34	srds lazy verification	

Table 5.4: List of query terms for each of the 34 queries

Individual queries

To get a sense of how context is improving individual query results, I consider each of the queries in isolation. Table 5.3 shows the precision and recall of each query at a cutoff of 30 results and table 5.4 lists the query terms for each query. In the case of typed queries (i.e., queries 1 through 5 and 13 through 18), the content-only system might have returned no results of the correct type, but remember that all returned results are used as the basis of the context-enhanced search, not just those of the matching type. It is also important to remember that, while many of the queries have identical recall and precision at a cutoff of 30 results, most queries have higher recall under context-enhanced search when more results are considered (e.g., a cutoff of 100).

For several of the queries, Connections is able to dramatically improve the recall and precision of the query. This is particularly true in the case of non-textual queries, where traditional content-analysis techniques are severly limited in their available information. For example, in queries 1 and 2, the user was searching for music files they had downloaded off the web. Before the user downloaded the files, the mozilla browser had read cached copies of the pages the user was visiting. These cached HTML pages were later indexed by Indri, and lead Connections to identify the desired files. This behavior exhibited by the mozilla browser also accounts for the success of queries 4, 6 and 22. While it is fortuitous that (1) the standard mozilla cache was large enough to capture at least one day's worth of activity, allowing it to be indexed and (2) the behavior of mozilla was such that it lends itself well to temporal relationships, these results indicate that application specified relationships could be a fruitful area of future work.

Another example of how Connections was able to leverage context to identify non-textual data is shown in query 5. In this case, the user was searching for the image of a poster they had created. The images and text on the poster in many cases had been taken from already create text and images (in many cases figures from papers which already had contextual clues). These files were read in and used to update the poster, creating relationships that allowed Connections to identify both the poster and related images. A similar connection from a user's homework assignment to the desired images is seen in query 3.

In queries 13, 14, 16, and 27 the users were searching for documents related to a particular topic or project. Despite these documents being content-indexable, Connections was able to improve both the recall and precision on these queries. The reason for this is that, while Indri often found all of the desired documents, they did not show up until far beyond the 30 document cutoff. By leveraging usage information, Connections could determine which files were more relevant, and rank them higher in the results. For example, in query 14, the user was searching for papers related to a particular project. In some cases, the word "pasis" appeared only in the pathname to the desired file, and was not directly referenced in the text (and the terms "read" and "write" were very common throughout the user's documents). Conversely, "pasis" would show up in the text of related work that referenced the project, sometimes multiple times, forcing the rank of these less-related documents higher than the desired documents. Connections was able to leverage the usage patterns to strengthen the desired documents through the relationships formed between the many files in the "pasis" directories and the resulting PDF created when the paper was built. A similar result is seen in query 27, where the user is looking for information about the "magpie" project.

In query 23 the desired files are related to a co-authored paper. While many of the relevent files are content indexable, just as with the user searching for the poster, some of the indicated files did not contain the desired keyword (the author name), and thus were not returned by the content-only system. For example, in this query the user appears to also be interested in scripts used to generate results for the co-authored paper. The feedback loop of generating results, creating a graph, and then refining the scripts seen in the traces allows Connections to identify the scripts as part of the context for the paper, even though they contain no relevant keywords and reside in a separate directory.

In several of the queries, the benefits of Connections are not with its increased precision, but instead are related to its increased recall. For example, in queries 8, 9 and 10 the user was searching for images and text related to particular parts of a trip they took to Italy. In these queries, Connections identified the most relevant documents as being those returned by Indri (identified primarily thanks to the user's well-formed naming scheme), thus the identical recall and precision at a cutoff of 30 results. However, later results included a number of additional, related images that were created at the same time, but whose names did not include the given query term.

Examining some of the queries where Connections was unable to improve search effectiveness (or in some cases reduced it) provides interesting insights. For query 20 the most relevant files located by Indri were mailbox files. Such "meta-files" are composed of several smaller sub-units of data. Because the trace data cannot distinguish among relationships for individual sub-units, these files often have misleading edges, making it difficult for Connections to provide accurate results. This problem indicates a need for some level of application assistance (e.g., storing individual emails in separate files). Another important aspect of this is that because these files often contain a large number of links (due to the diversity of the context of individual emails), the results returned by Connections are not significantly modified from the original Indri ranking. Thus, at a cutoff of 30, the performance of the two systems are very similar.

In query 19, the desired files are related to an individual that the user knows. In this case, again, the content results are dominated by email cache files, resulting in poor recall and precision for both systems at a cutoff of 30. Connections has slightly higher recall because of of the relevant files are updated in conjunction with accesses to many of the other relevant files. Specifically, these files appear to contain contact information that includes the requested individual. Updates to other related contacts in these files are made in conjunction with accesses to other files returned by content. This, in turn, pushes up the relevance of these two files.

For query 21 one of the search terms had multiple meanings within the user's data. The user specified "training" to refer to their workout schedule, but they also happened to be working on a project related to machine learning that often contained the word "training." These disjoint uses of a single word indicate that some level of result clustering could be useful in presenting results to users. By clustering contextually related results together, the ranks of disjoint sets could potentially be adjusted to include some results from each cluster.

5.2.2 Online results

The online study was run for a total of four months, with the three phases taking two months, one month, and one month, respectively. Users were located through means of distribution lists and bulletin board systems at Carnegie Mellon University. Twenty-two users signed up for the study, although only six used the tool throughout all three phases of the study.[5] The demographics for these six users are listed in Table 5.5.

Table 5.6 describes the statistics gathered during each phase of the study. The first part of the table lists the total number of queries and the number of query sequences. A query sequence combines queries that make up a series of retries. The second part of the table lists the average time spent, pages viewed and results clicked for each query sequence in a phase. The third part of the table lists the success rate for each sequence, the number of successful sequences, the percentage of successful sequences that included retries, and the average number queries in a successful sequence.

While there is not enough data from this study to show significant improvements from context-enhanced search, the results do suggest three possible take-aways. First, during phase one of the study, the success rate is lower than during phase two or three. Given that phases one and three used the same underlying query technique, this suggests that during phase one, users may have felt unfamiliar with the tool and, over time, became better at querying with the tool (thus the improvement in success rate and the drop in number of pages viewed).

Second, the success rates of the context-enhanced phase and the second content-only phase are the same. The similar success rates between the context and content phases suggest that, in the Windows environment, context is not significantly increasing the number of files that a user can find. Anecdotal evidence through user surveys seems to indicate that this is largely due to the kind of data stored and searched for by users. Most of the users in the study seemed to be searching for content-indexable data (e.g., Microsoft Word files), with most of the failures coming from the tool's inability to search through web and email data.

Third, despite having the same success rates, the phases two and three have different retry rates. Although most user data could be indexed by content, several of the user content queries required refinement to achieve success. Context may be reducing the need for this refinement by identifying the set of files that were most relevant to the user for those keywords, presenting the correct results to the user without the need for query term refinement. This is important because, while it does not reduce the time required for an individual query, it does reduce the number of queries required to find a piece of data, thus reducing the overall time, page views, and result clicks for a query sequence. This reduction in time and effort is one of the goals of any search engine and, while more data will be needed from future studies to confirm these results, the available data is suggestive of results similar to those of the offline study.

[5] None of the users from the offline study were included in the online study.

Gender	**Count**
Male	5
Female	1
Computer skill	**Count**
1	0
2	0
3	1
4	1
5	4
Use of computer in profession	**Count**
1	0
2	0
3	0
4	0
5	6
Age	**Count**
18-25	1
26-35	3
36-45	1
46-55	1
Use of this computer	**Count**
Personal	1
Business	2
Both	3
Hours per day spent using this computer	**Count**
< 1	1
1-3	0
3-5	1
5-7	0
7+	4

Table 5.5: Reported demographic information

Metric	**Phase one (Content-only)**	**Phase two (Context-enhanced)**	**Phase three (Content-only)**
Total queries	41	19	18
Query sequences	30	15	12
Time spent per seq.	389s	182s	286s
Pages viewed per seq.	6.8	2.9	5.0
Results clicked per seq.	2.4	1.8	3.2
Sequence success rate	27%	40%	42%
Successful sequences	8	6	5
Sequence retry rate	63%	17%	60%
Queries per success	1.7	1.2	1.8

Table 5.6: Results of the online user study

5.2.3 Wrap-up

The overall results of these two studies is that context information can improve the quality of a user's searching. As shown in the offline results, context can improve the recall and precision of searches by finding more of the data a user is interested in and ranking it more accurately. Particularly, the largest benefits from Connections was its ability to locate documents with non-parsable contents (such as images and music) and the increased recall of related documents that do not directly contain the requested keywords. Similar benefits of context-enhanced search are suggested by the results of the second study where, even when most user data is content indexable, the system is able to find the results the user is interested in and rank them more appropriately with less required user input. These results also answer the implicit question posed by my thesis statement: Connection's automatically gathered context information provides it with more information about each file, which in turn improves the effectiveness of its file indexing and search.

6 Performance evaluation

In traditional content-only search, the performance costs include time required to scan the file data to create the index, the space required to store the index, and the time required to query the index. Similarly, there are three added performance costs of context-enhanced search: the additional indexing time required to build the relation graph from traces, the additional disk space required to store the relation graph, and the additional time required to search the relation graph during a query. In this chapter, I measure each of these overheads, discuss the source of the overheads, and describe potential methods for reducing these overheads further.

All performance experiments were run using Linux 2.4.22 on an Intel Xeon 3Ghz with 2GB RAM using a Seagate Barracuda 7200RPM, 250GB disk drive. In all cases, BerkeleyDB was configured to use a 64MB buffer cache and 2MB log cache. The traces used in these experiments are the same as those described in Chapter 5 for the offline experiment.

6.1 Indexing performance

In Connections, the indexing phase consists of both content analysis and merging file system traces into the relation-graph. Although the startup cost of content indexing is high, the incremental costs are low enough that many users already accept them. Furthermore, content-only indexing times can vary widely based on the technique (e.g., 30 to 700 seconds per gigabyte) and are an area of related but tangential ongoing work by a number of groups [20], and so are not analyzed here. Indri's reported indexing times were just under 3 minutes per gigabyte [20].

With context analysis, there is no startup cost, because no record of temporal relationships exists initially. To evaluate the incremental cost of context indexing, I measured the time required to merge each day of traces into the corresponding machine's relation graph over the six months of traces. The average indexing time for a single day of traces was 24 seconds with a standard deviation of 24 seconds. The longest observed indexing time was 370 seconds.

These overheads are low enough that we believe most users will find them acceptable. Even if the machine is constantly in use while powered on, a worst-case 7 minute per-day indexing time is likely low enough to run in the background with little impact on foreground work.

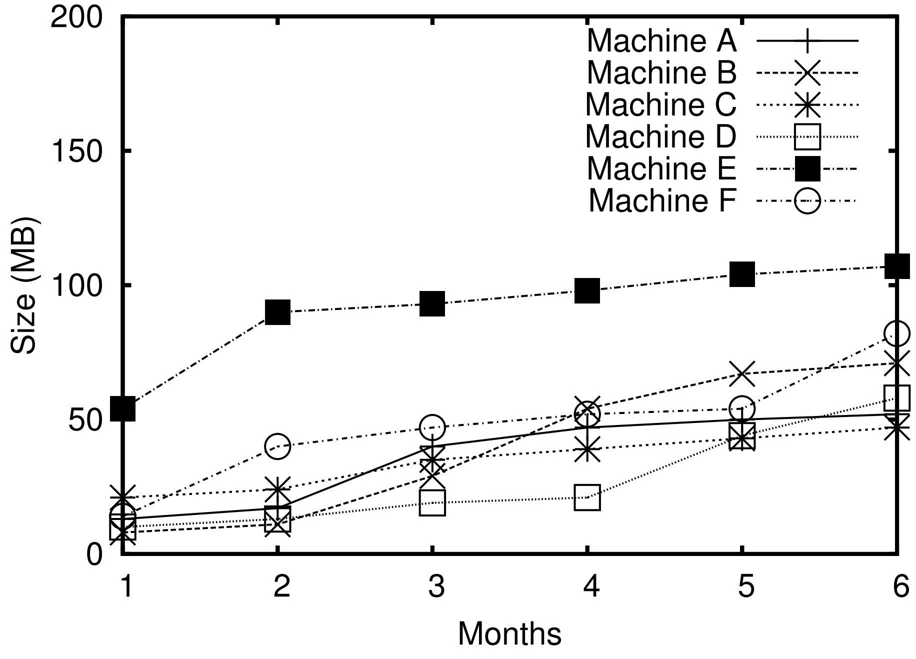


Figure 6.1: Growth of the relation graph space utilization

6.2 Space utilization

The space utilization of Connections is a combination of the space required by Indri's content-only index and the space required to maintain the relation graph. Similarly to indexing times, the size of content-only indexes can vary widely based on the technique and are an area of related but tangential ongoing work by a number of groups, and so are not analyzed here. Indri's reported index sizes were between 25% and 33% of the original data size [22].

Because the relation graph is a representation of how a user's files are connected to each other, its size cannot grow larger than $O(n+n^2)$, where n is the number of files in the user's file system. This limit represents a worst-case scenario, where each file is related to every other file in the system. Thus, the questions to answer are: (1) how much space is required to maintain the relation graph for a graph of n nodes and m edges, and (2) what are the growth characteristics for the ratio between n and m? To examine these two questions, I ran the six months of traces for each system through Connections's indexing algorithms and measured the number of nodes, edges, and total space utilization after each day.

Figures 6.1, 6.2, and 6.3 illustrate the size of the relation graph, the average number of links connected to a node (i.e., link density), and the number of nodes in the relation graph, respectively, for each traced machine after each month of tracing was added to the graph. The most noticable result is that the context index is quite small. With user data

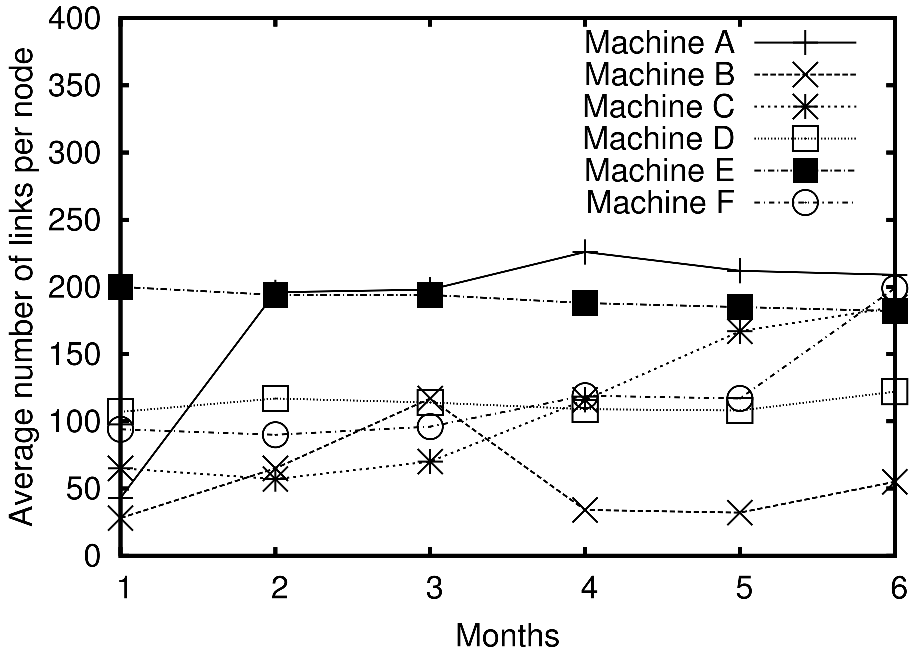


Figure 6.2: Growth of the number of links per node in the relation graph

sets ranging from 10 GB to 14 GB in size, even the largest index is only 1% the size of the smallest data set.

Given the naive design of Connections's relation graph database, the expected storage overhead for a node is the size of the mappings from nodeID (8 bytes) to file name (approximately 128 bytes) and back, and the expected storage overhead for an edge is the space required to store the link (8 bytes) and its weight (8 bytes) for both the incoming and outgoing node. This totals to $144 + (m \cdot 32)$, where m is the average number of links per node in the graph. However, this estimate is below the reported sizes; for example, examining machine E, the estimate at 6 months is 70 MB while the measured size was 107 MB.

The cause of this descrepency in expected size to actual size is a side-effect of BerkeleyDB's storage mechanisms. First, indexing the tables incurs a small overhead, although this is expected to be 5% or less of the data size. Second, because BerkeleyDB uses sector sized pages, nodes with few links (e.g., a single link) still require 512 bytes to store their link information. Given the high variability in link counts, this inefficiency accounts for the additional space used by the database, indicating that this would be an important aspect to consider in the design of a more space-conserving link database.

In regards to the growth characteristics of the graph, Figure 6.2 indicates that the link density does not have worst-case linear growth and instead is fairly stable. After some warm-up period (which is different for each user), the average number of links in the graph seem to stablize, indicating that hysteresis will likely be more important for result quality

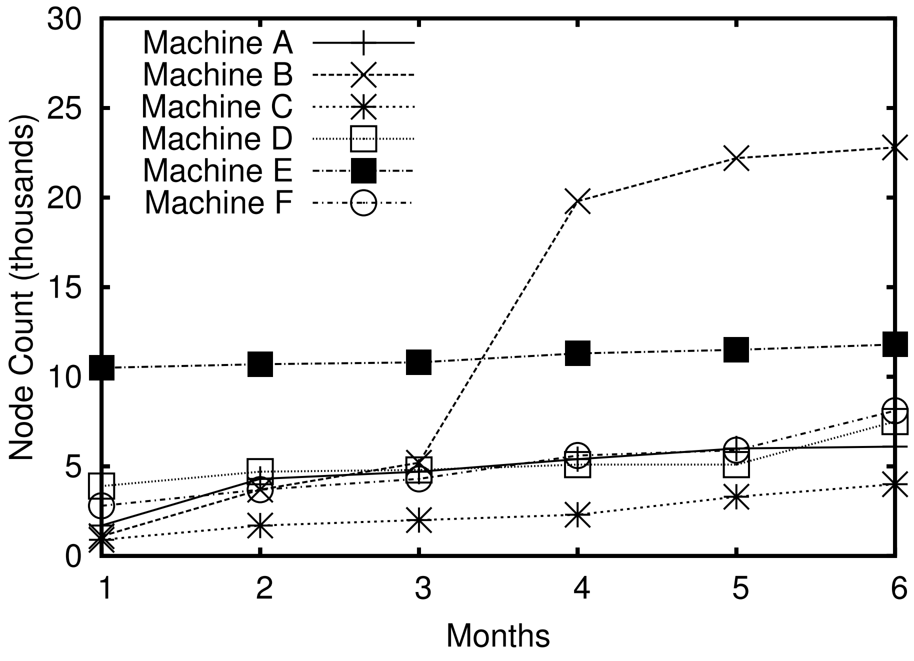


Figure 6.3: Growth of the number of nodes in the relation graph

than for graph size maintenance. The reason for this non-exponential graph growth is that a user's working set size does not change much over time. In the worst case, all nodes in the system could become connected to all others, but users do not access all of their files concurrently because they can only manage a small subset of their files at once. The result is a linear growth in graph size, with a constant ratio between nodes and links.

The one exception to this is machine B, where there is a drastic spike in the number of nodes between months 3 and 4 combined with a corresponding drop in the average number of links per node. This was caused by the introduction of a large number of of new files added to the machine, as the user joined ongoing work on a new project. Because the user did not later access many of the files initially added, the ratio of nodes to links dropped overall, but the ratio among actively accessed files remained around 100.

6.3 Querying performance

A third overhead introduced by context-enhanced search is the increase in query time caused by searching through the relation graph and re-ranking results. To measure this overhead, I measured the execution time of each of the 25 queries, separating the time required for content querying and context querying. The average query time was 3 seconds, with 1 second of that used for content querying and 2 seconds of that used for context querying. Despite this 200% increase in query time, a 3 second query time may be small enough that most

Machine	Time	σ
A	1.8s	0.2
B	0.9s	0.5
C	0.9s	0.5
D	0.6s	0.1
E	5.8s	0.1
F	1.1s	0.2

Table 6.1: Average query performance for each machine

users are willing to accept it. Furthermore, examining the query times of individual relation graphs indicates that improvements in link storage could have significant improvements in query performance.

Table 6.1 lists the average context query time for each machine using the default relation graph generated from the six months of traces for that machine. Interestingly, profiling the query engine indicates that for all machines, more than 85% of the time is spent on file I/O (with the I/O time for machine E taking up more than 95% of the time), indicating that a more efficient on-disk link structure layout could provide significant improvements in query time. The much lower performance for machine E is caused by cache misses; the relation graph size of machine E is large enough that queries do not fit entirely within the 64 MB database cache. Increasing the cache size of BerkeleyDB from 64 MB to 256 MB (thus reducing the load time of the link structure to a single 107 MB disk read) reduces the average query time on machine E to 1.71 seconds.

6.4 Wrap-up

The performance analysis indicates that most of the overheads in Connections are small and would have little effect on the deployment of such a system. These results also indicate that the largest improvements in performance could all be gained by changes to the structure of the link database. By finding a layout that clusters closely related items and reduces the space required to store link information, it may be possible to reduce the I/O time involved with searching and updating the relation graph.

7 Sensitivity analysis

This chapter uses the traces and data gathered during the offline study to examine the algorithms that make up Connections, providing insight into how contextual relationships are formed and some of the potential pit-falls of such a system. It begins with a sensitivity analysis of the specific algorithms within Connections and then discusses the effect of changes to the underlying content analysis tool, the effect of using user input as a contextual clue, and the effect of graph growth over time on result quality.

7.1 Parameter sensitivity

One of the design objectives of Connections was flexibility. Providing a large number of tuning knobs allows experimentation that provides an understanding of how contextual relationships are formed and which relationships contain useful information for search. To this end, I ran a sensitivity analysis to answer two questions. First, how sensitive are the parameter settings to change, i.e., how far can the knobs be turned before they begin to affect result quality? Second, when the parameters are set outside of the sensitivity bounds, what changes occur in the graph that affect result quality?

Just as in any retrieval system, there is a tension within Connections between false-positives and false-negatives. False-positives in Connections are usually caused by super-nodes, nodes that are highly connected within the relation graph, creating irrelevant relationships that lead to incorrect results. False-positives have the largest impact on precision when they appear early in the results (i.e., at low recall levels), because precision is measured based on the number of examined results.

False-negatives in Connections are usually caused by parameters that are too restrictive, preventing the system from following relevant paths in the relation graph, thus missing relevant results. False-negatives have the largest impact on precision at high recall levels for two reasons. First, lost results are usually the least connected nodes in the relation graph and, thus, the least relevant (and lowest ranked) of the correct results. Second, missing results prevent the system from achieving high recall levels, thus dropping precision to 0% at these levels.

The introduction of super-node detection techniques reduce the sensitivity of the system to parameter settings by reducing the effect of false-positives created by super-nodes, although large deviations in the parameter settings that significantly increase link counts can make even relevant nodes look like super-nodes, thus defeating the advantages of super-node

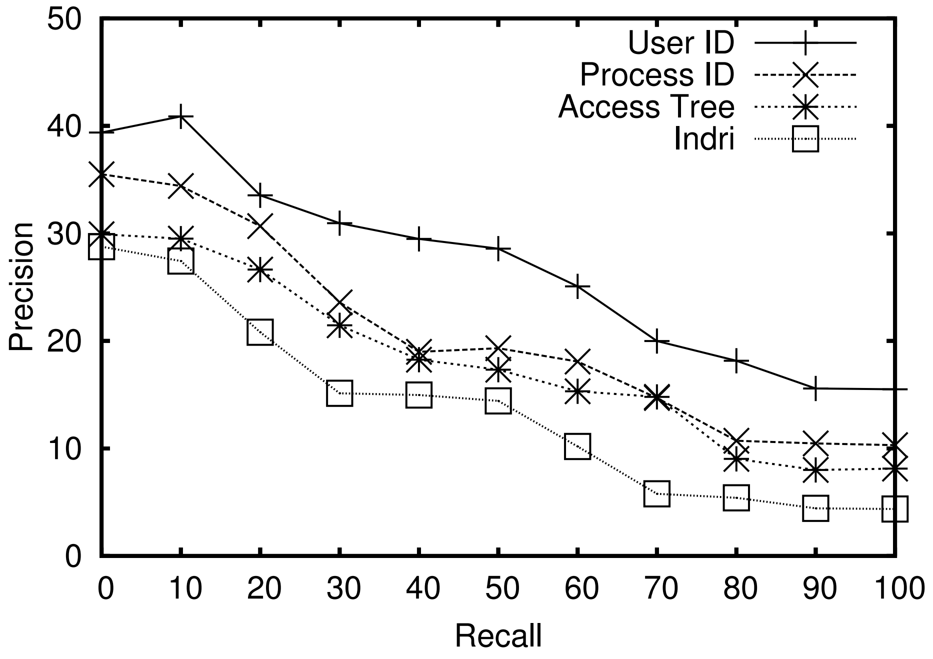


Figure 7.1: Sensitivity to the isolation level

detection techniques. The end result is that, when combined with super-node detection, parameters are surprisingly insensitive to change, although the best performing settings result in similar performance with or without super-node detection.

Although I ran the full matrix of parameter settings (more than 200,000 parameter combinations), this section examines the trade-offs within each parameter individually, using the settings listed in Section 5.1.1 as a baseline. The one exception to this is the use of stddev-detect as the default super-node detection technique. This was chosen over percentile-detect because it most effectively reduces the effect of super-nodes in the case where parameters are set incorrectly. Each of the graphs presented in this section also contains Indri's content-only search results as a second comparison point. To better highlight the differences between the parameter settings, graphs are shown with a precision scale that only goes up to 50% rather than 100%.

7.1.1 Indexing parameters

The indexing parameters determine how links are formed within the relation graph and include the isolation level, access filter, window size, and edge direction. Because relevant links are built up over time, there is a tension within these parameters between ignoring irrelevant links to avoid false-positives and including links that may become stronger over time even if they appear irrelevant now.

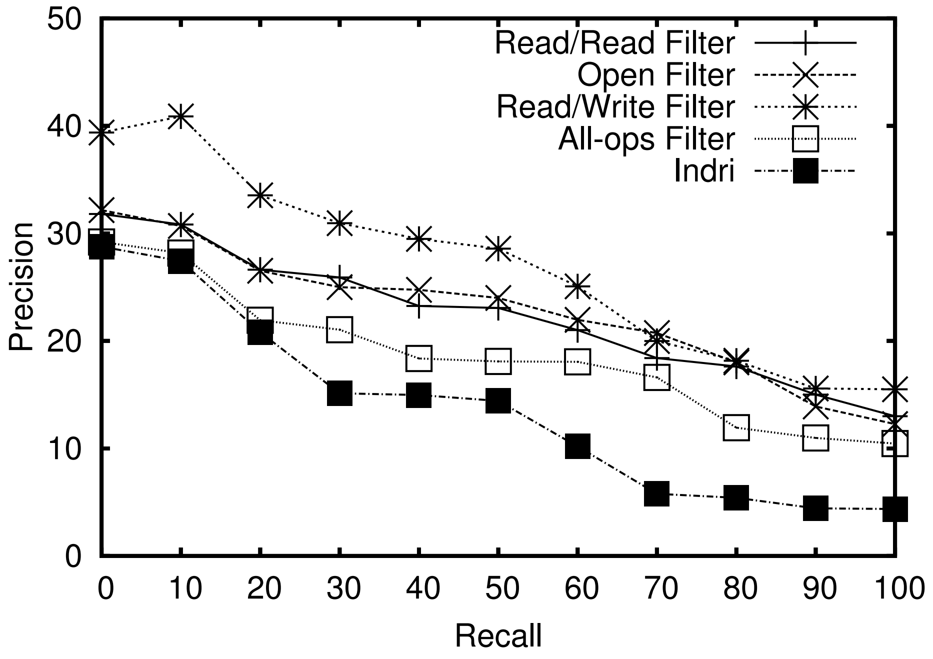


Figure 7.2: Sensitivity to the access filter

Isolation level: Figure 7.1 illustrates the results of the three different isolation levels provided by Connections: user ID (the default level), process ID, and access tree. Process ID and access tree isolation underperform user ID isolation because they miss many important inter-process interactions that the user invokes based on intent rather than based on direct application behavior. For example, a user reading PDFs while working on a word document (e.g., working on a report) would not have their context fully captured by any process-level isolation technique. Interestingly, while access tree isolation more broadly connects files in the relation graph, process ID isolation performs slightly better. This is because process ID and access tree isolation capture many of the same relationships; the output file of one process in the tree is often the input file for another. However, access tree isolation introduces many more false positives, since the application-specific files that cause most super-nodes become more strongly connected in the graph. The result is that the additional false positives drive down precision. Overall, while more intricate isolation levels may outperform any of the techniques offered by Connections, it is likely that methods that can identify complete user "tasks," rather than only application-specific behaviors, will be more successful.

Access filter: Figure 7.2 illustrates the results of the three different access filters available in Connections: read/read, open, read/write (the default filter), and all-ops. Of the four filters, the read/write filter performs the best. This is largely due to the fact that it introduces the fewest number of false-positives. Although the other filters all have the potential to increase recall further than the read/write filter, the increased noise reduces the effectiveness of this increased recall.

Interestingly, while intuitively read/read connections could be considered the most useful, in practice the large number of read/read connections in normal system operation introduce two problems that lead to significantly reduced precision. First, the increase in extra links flattens the link count distribution making some valid nodes look like super-nodes (and vice-versa). This renders the super-node detection ineffective; as an anomoly detector it begins to lump valid nodes and invalid nodes together when they have similar link counts. Second, the increase in link weights (often for irrelevant connections) reduce the impact of valid connections. This is a problem because Connections has no techniques to completely eliminate irrelevant links. The alpha parameter helps to reduce the effect of incorrect link counts, but it assumes that relative link weights are approximately correct (i.e., it assumes that links with more weight are more relevant, even if it uses less of the weight). The introduction of read-read links will sometimes invert this assumption (some of the irrelevant links have more weight).

As an example of the problems created by read-read links, consider the following two files, “core” and “submit.ps,” which were returned as results during a read-read search for query 27. The file “core” is a debugging memory dump of a failed application, and is not relevant to the search. The file “submit.ps” is a paper marked as relevant to the search by the user.

In the read-write case, the core file is rarely linked to in the relation graph, because it is written very infrequently; as such, none of the files found through content point to “core” with any strength, and so it is never returned. In the read-read case, the core file has more, heavier links in the relation graph, although not enough to mark it as a super-node. This is caused by a lengthy debugging session that spans a large number of relation windows. During the debugging session, accesses to files that are unrelated to “core,” but relevant to the search terms, cause the creation of misleading links between these two disjoint contexts. During the search, these misleading links lead from relevant content results to the irrelevant “core” file. In this particular case, web cache files, auto-save files, mail cache files, music files, trace files, several pdf files, and a number of other miscellaneous files become connected to “core,” mostly caused by concurrent tasks (e.g., listening to music, checking email, browsing the web). Of these connected files, some are marked as relevant by content search, thus adding the “core” file to the list of results. This problem is further exacerbated as connections from “core” to other, also unrelated, files are added to the results.

Conversely, the addition of read-read links to the relation graph has little effect on the connectedness of “submit.ps,” as most of its relevant links were identified from when the user created (i.e., compiled) the document. The result is that, in the read-read case, “core” is ranked higher in the results that “submit.ps” because the weight of its links is, in many cases, higher (the inversion of link weights mentioned above).

Despite this deficiency for search applications, where results must be targeted specifically to the user’s information need and too many false positives can cause both frustration and failure, read-read links may be more useful in organizational tasks. For example, read-read links could present the user with a few different contexts in which “core” or “submit.ps”

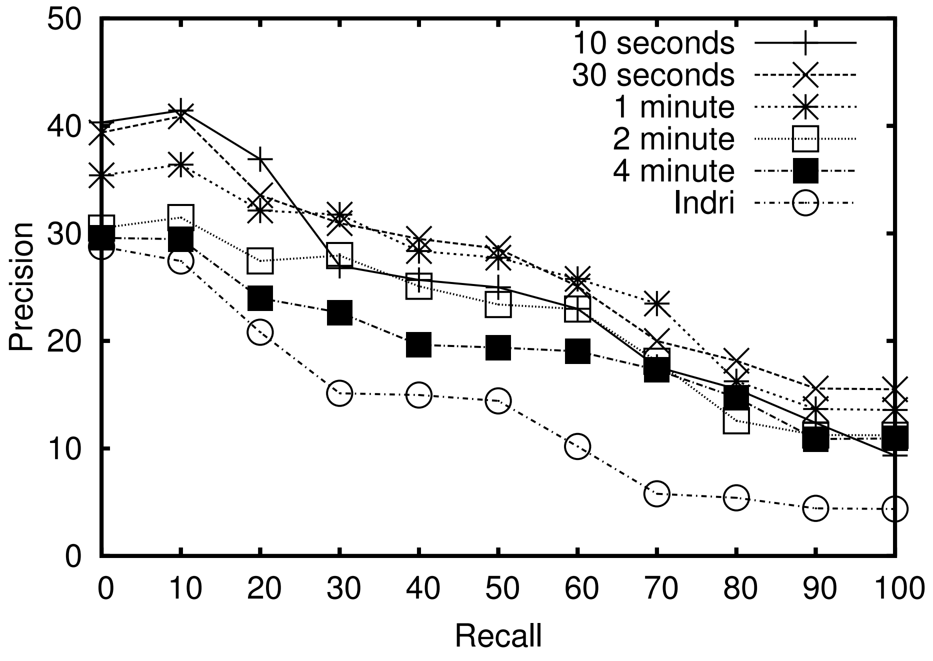


Figure 7.3: Sensitivity to the window size

could fall, and then the user could select the set of files that are most relevant. The system could then use this information to group or manage data. Exploring these options, and the viability of using read-read links in other ways for search, is an area of future work.

The open filter's reduced precision is caused by the introduction of large numbers of application-specific relationships. Applications often read a large number of application-specific files during operation that are irrelevant to the user. Because OPEN() calls are made before both READ()s and WRITE()s, the result is a combination of read-write, read-read, and write-read relationships. In particular, the introduction of read-read relationships causes many of the problems discussed above.

The all-ops filter's reduced precision is caused by irrelevant relationships formed by the extra information used from the trace. The frequency of some system calls, such as ACCESS(), increase the number of ephemeral links in the graph. This increase in average link counts impacts the super-node detection (particularly the stddev-detect scheme), increasing the number of false-positives. Furthermore, this increase in links makes the weight-cutoff more restrictive (since each link makes up a smaller percentage of the total node weight), often cutting relevant links, further reducing precision.

Despite these problems, Connections matches or beats Indri with all four filtering schemes. This indicates that the context information available within the trace contains useful relationships, and the task is to determine the best method for identifying them.

Window size: Figure 7.3 illustrates the results of using five different window sizes:

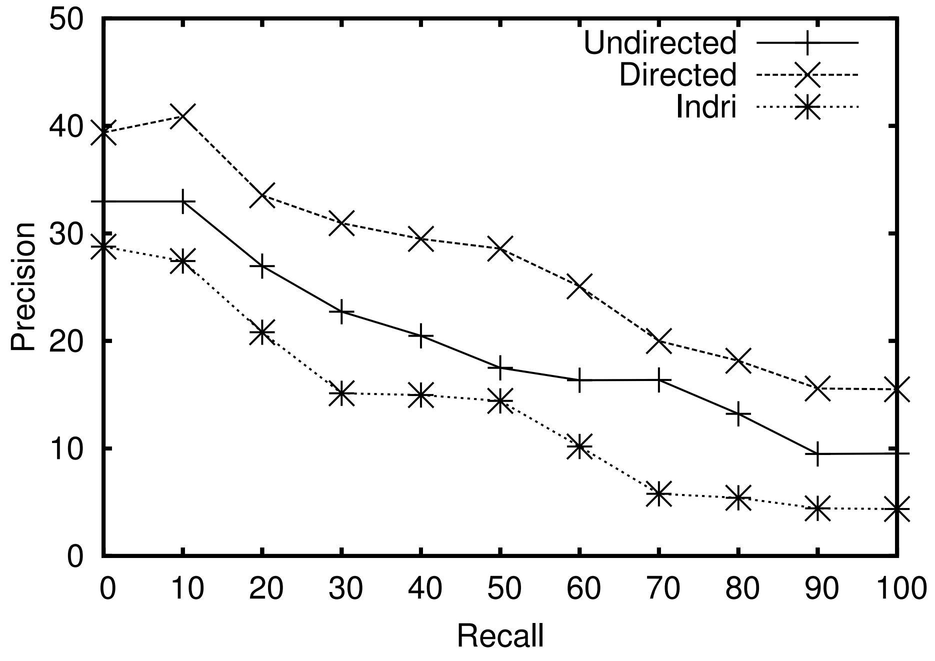


Figure 7.4: Sensitivity to the edge direction

10 seconds, 30 seconds (the default setting), 1 minute, 2 minutes, and 4 minutes. With a smaller window size, there are fewer connections between nodes in the relation graph, resulting in fewer reachable nodes. The result is more false-negatives, reducing the precision of the system at high recall levels. The reason that windows sizes as small as 10 seconds still perform well is that (1) many applications batch i/o behavior (e.g., auto-saves) and (2) the combination of a 10 second window and a path length of three steps is often enough to capture user behavior (e.g., connecting web-cache files to downloads). With larger window sizes, files become more densely connected. The increase in link counts eventually begins to reduce the effectiveness of super-node detection (e.g., 2 minutes), resulting in lower precision at low recall levels.

Edge direction: Figure 7.4 illustrates the difference between directed (the default setting) and undirected links in Connections. The drawback of undirected links is that by discarding the temporal ordering relationships within the trace, it introduces a large number of additional links in the relation graph. Just as the open filter's introduction of read-read relationships increases the number of irrelevant application-specific links in the relation graph, undirected links create a large number of irrelevant application-specific links through write-read connections (i.e., inverted read-write connections). Normally, invalid read-write connections caused by reading application-specific files are ignored, because the read files are not returned as correct by the content analysis and there are no incoming links to those nodes. Also, undirected links effectively double the number of links in the graph, making

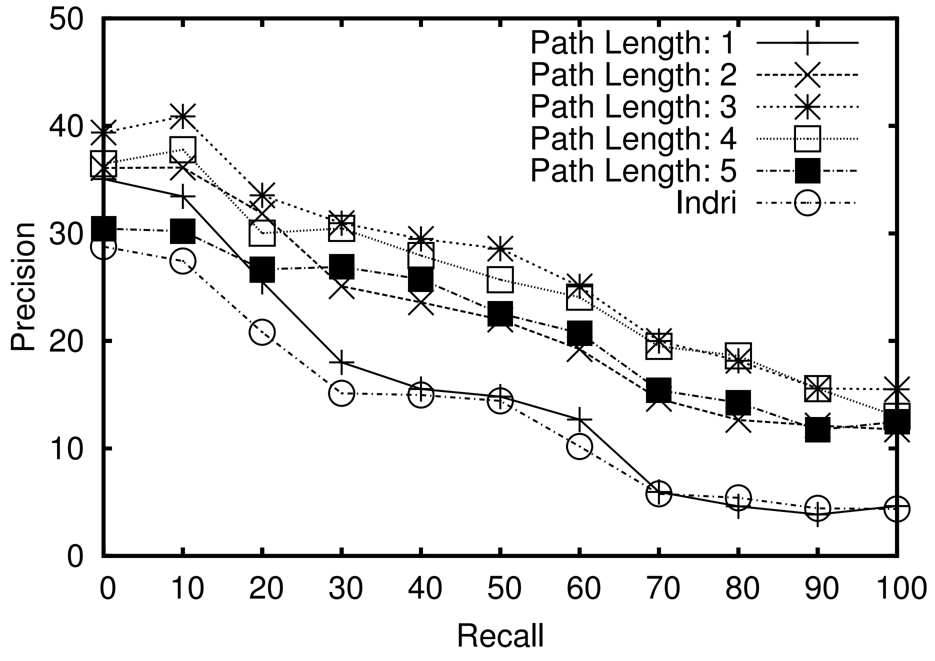


Figure 7.5: Sensitivity to the path length

the weight-cutoff more restrictive (just as in the all-ops filter), cutting relevant links. The result is an increase in both false-positives and false-negatives that reduce precision at all recall levels.

7.1.2 Search parameters

The search parameters (path length and weight cutoff) determine which links are taken from the relation graph when forming the result graph. It is likely that the relation graph will err toward including too much information, since it is difficult to identify the difference between ephemeral links and new-but-important links at link creation time. The tension within the search parameters is between including all of the relevant nodes in the relation graph without following ephemeral links or links to super-nodes.

Path length: Figure 7.5 illustrates the effect of varying the path length parameter in Connections from one to five. The default path length in Connections is three, although, as illustrated in the graph, path lengths of two and four perform similarly. The drawback of a small path length (e.g., one) is that not enough information is gathered from the relation graph to properly inform the ranking algorithm. The result is a large number of false-negatives that reduce the precision of the system, particularly at low recall levels. The drawback of a large path length (e.g., five) is that the increase in followed paths that lead through super-nodes results in large numbers of false-positives. Although the super-node detection mitigates this to a degree, when the path length increases sufficiently, the

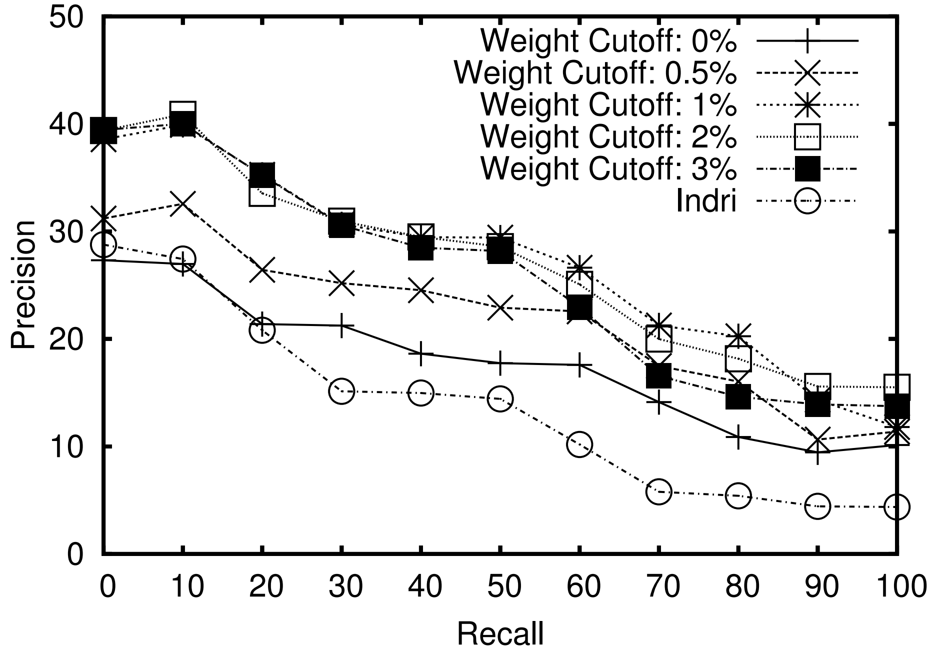


Figure 7.6: Sensitivity to the weight cutoff from 0% to 3%

penalties associated with super-nodes must be increased correspondingly to continue to be effective. Still, despite these potential pitfalls, context-enhanced search still matches or beats content-only search for all of the path lengths.

Weight cutoff: Figures 7.6 and 7.7 illustrate the effect of varying the weight cutoff from 0% to 3% and 3% to 20%, respectively. The default weight cutoff in Connections is 2%, although any cutoff between 1% and 4% performs similarly. The drawback of an overly low weight cutoff (e.g., below 1%) is that the result graph will include a large number of ephemeral links and links leading through super-nodes, resulting in high numbers of false-positives. In the case of 0%, the number of false-positives is high enough to also affect the precision at high recall levels. The drawback of an overly high weight cutoff (e.g., above 5%) is that the result graph is restrictive enough to ignore many of the informative links, resulting in more false-negatives which reduces the precision of the system at high recall levels.

7.1.3 Ranking algorithms

The ranking algorithms (Basic-BFE, PageRank, and HITS) determine the order in which nodes in the result graph are presented to the user. The objective of these algorithms is to identify the most relevant results based on the strength of relationships within the result graph and/or relation graph. Because it is easier for the search parameters to remove ephemoral links than links to super-nodes, the Basic-BFE algorithm also includes three

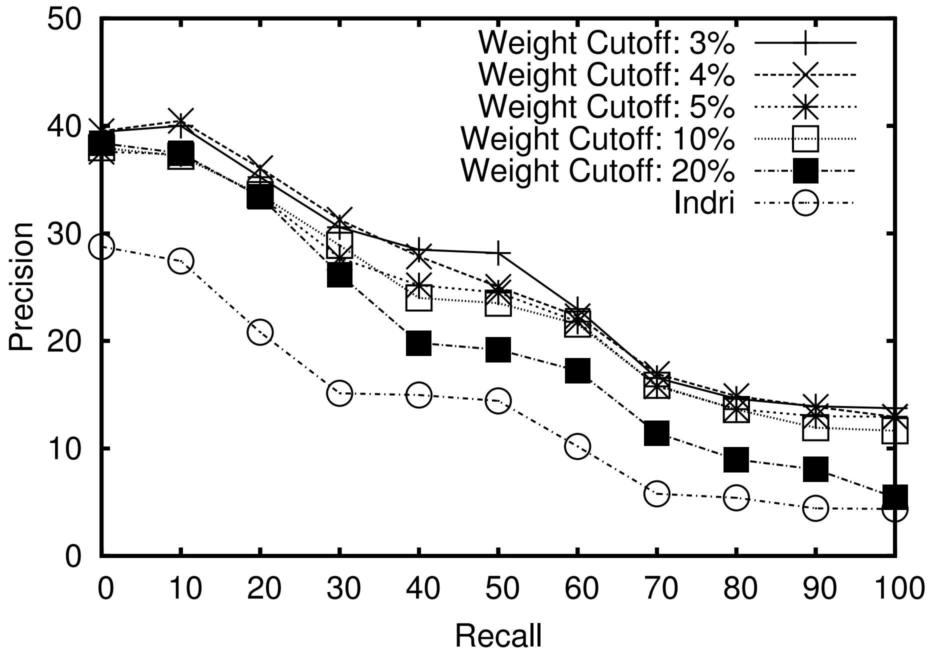


Figure 7.7: Sensitivity to the weight cutoff from 3% to 20%

methods of detecting super-nodes and reducing their effectiveness.

Basic-BFE: Figure 7.8 illustrates the effect of varying the alpha parameter in the Basic-BFE algorithm from 0 to 1. The default alpha setting is 0.5, although any setting between 0.25 and 0.75 performs similarly. In the case where alpha is too low (e.g., 0), link weights are ignored during the ranking. The result is that less relevant links pass more weight, pushing less relevant results higher in the rankings, reducing the precision at low recall levels. Interestingly, increasing the weight cutoff reduces the negative effect of low alpha parameters, further indicating that link weights are inaccurate (and perhaps unimportant) among relevant links and that the more important task is to identify the relevant links. In the case where alpha is too high (e.g., 1), the link weight variability caused by application behavior favors super-nodes, increasing false positives and reducing the rank of relevant nodes with weaker links, reducing precision across all recall levels.

PageRank: Figure 7.9 illustrates the effect of using PageRank to rank the results of a context-enhanced search, using the three different techniques discussed in Chapter 3: PR-only, PR-before, and PR-after. When PageRank is applied alone (as in PR-only), its performance is similar to that of content-only search. This is because PageRank is designed to find "authority" pages, which is often undesirable for local file system search.[1] Authority

[1]This is also less relevant on the web because web search does not include an expansion phase as is found in Connections. By only ranking pages that include the queried terms (or have incoming hyperlinks with the terms), irrelevant pages with particularly high PageRank are not included in the results.

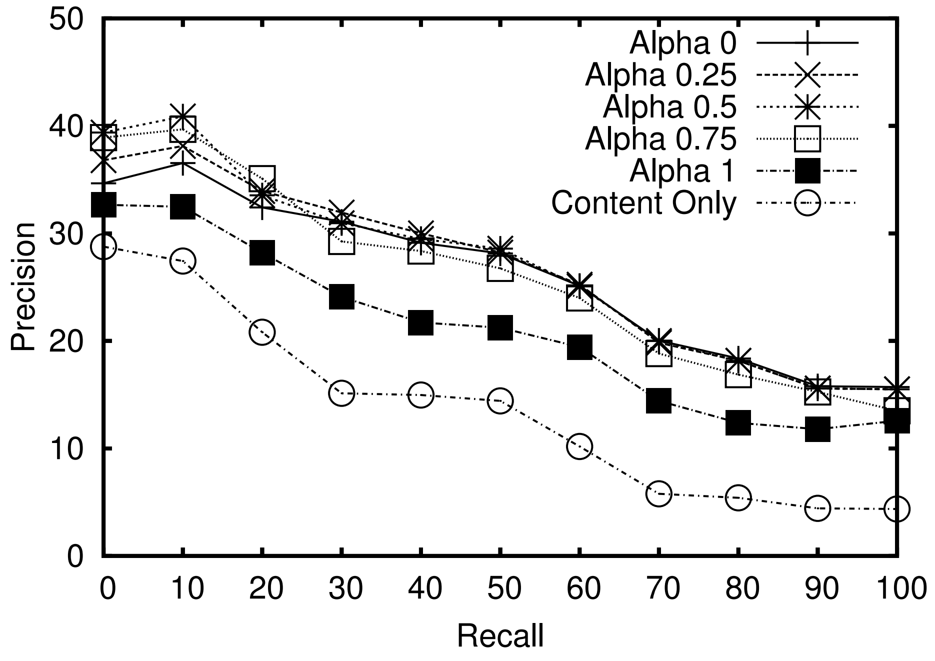


Figure 7.8: Sensitivity to alpha

nodes are usually sinks in the graph (nodes with many incoming links, but few outgoing links), which in many cases are irrelevant application-specific files that are written but rarely read (e.g., web-cache files or super-nodes).

When applying PageRank as the initial ranking (often referred to as a *non-uniform prior* relevance value) to the results of content-only search, and then applying Basic-BFE (as in PR-before), the result is similar to Basic-BFE alone. This indicates that the context information is more relevant than the content rankings and has a tendency to outweigh any non-uniform priors applied to the content rankings (this is discussed further in Section 7.2). Conversely, when PageRank is applied after Basic-BFE (as in PR-after), the less accurate rankings provided by PageRank tend to cancel out the ranking improvement provided by Basic-BFE. The result is that highly ranked results in Basic-BFE are pushed down in the ranking, giving it precision similar to that of PageRank only.

HITS: Figure 7.10 illustrates the effect of using HITS to rank the results of a context-enhanced search, using both the authority and hub rankings for two different techniques: HITS and Expanded-HITS. Both HITS and Expanded-HITS underperform Basic-BFE. The reasons for HITS's reduced performance is similar to that of PageRank's: application-specific files that have imbalanced incoming and outgoing links (for authorities and hubs, respectively), particularly super-nodes, outweigh the importance of relevant nodes. Without a method of super-node detection, these schemes push relevant results lower in the rankings. This is especially evident when looking at the results of Expanded-HITS (auth), as the

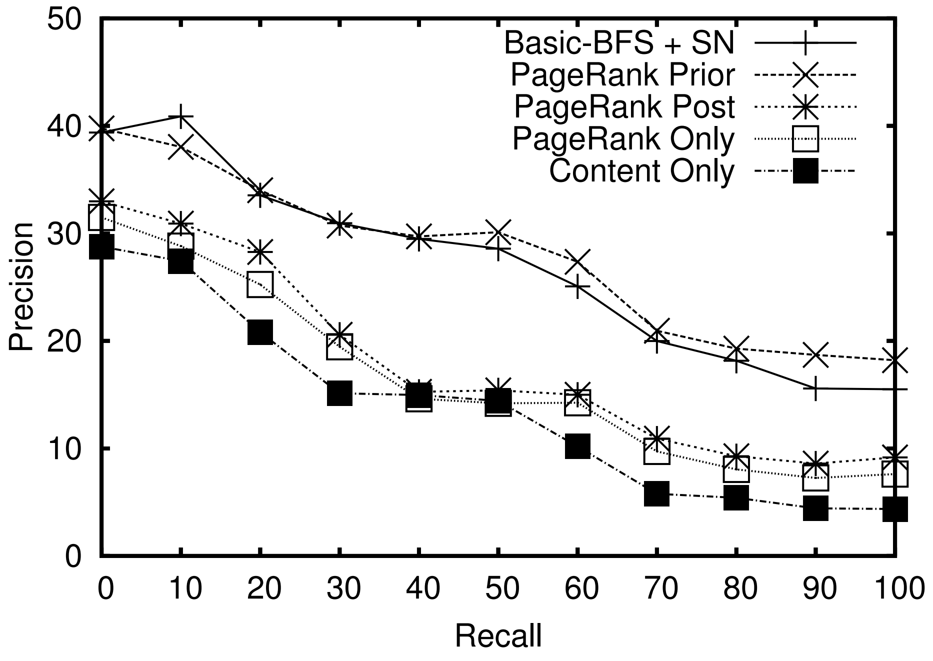


Figure 7.9: PageRank methodology comparison

precision actually goes up as more results are introduced.

Super-node detection: Figure 7.11 illustrates the effect of the four different super-node detection techniques using the default indexing and search parameters. This graph shows three interesting results. First, with parameters tuned to the best settings, Basic-BFE alone outperforms both the PR-detect and clique-detect schemes. Because the other parameter settings sufficiently mitigate most of the effects of super-nodes, these two detection techniques incorrectly identify valid nodes as super-nodes and falsely penalize them.

Second, PageRank makes a poor super-node detection scheme. PageRank tends to rank authority nodes highly, however authority nodes are only a portion of the super-nodes in the graph. Many super-nodes have a large number of both incoming and outgoing links, meaning they will not become sinks (i.e., authorities) in the relation graph, and in fact, are ranked quite poorly by PageRank. The result is that this kind of super-node is boosted by PageRank, enhancing the effects of these super-nodes.

Third, global information from the relation graph (such as used by stddev-detect and percentile-detect) is more useful than local information from the result graph (such as used by clique-detect), when the parameter settings are tuned properly. This is because algorithms using local information, such as clique-detect, will always assume that some of the nodes in the result graph are super-nodes and potentially penalize correct results. The result is that clique-detect under-performs stddev-detect with these settings.

Fourth, percentile-detect performs so well because it closely maps to the distribution

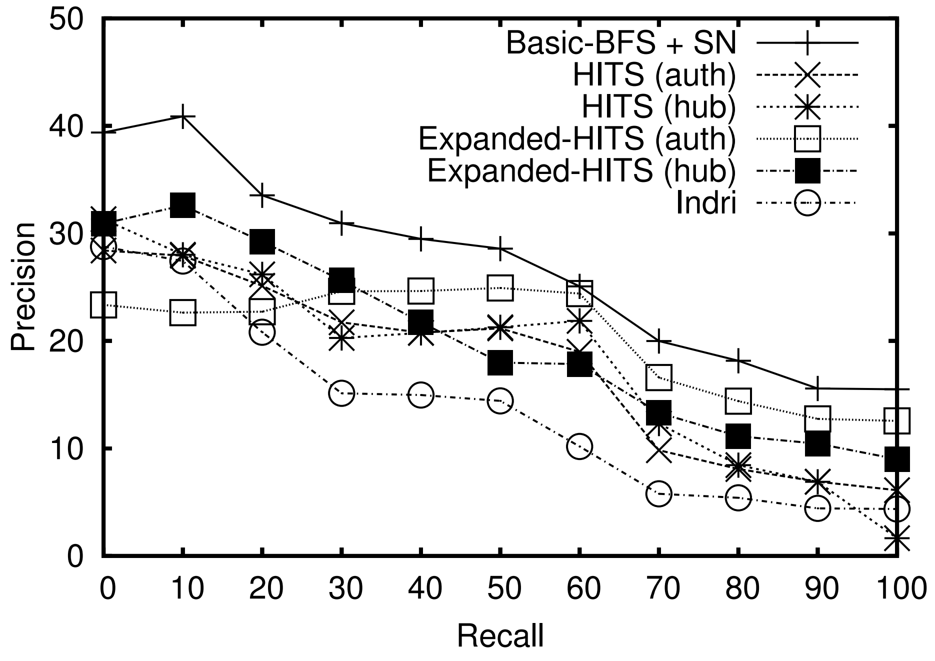


Figure 7.10: HITS methodology comparison

of super-nodes in the graphs. Figure 7.12 shows the distribution of link counts for the default two-year relation graphs of the six traced machines. The heavy-tailed nature of this distribution shows that super-nodes are only contained in the most heavily connected nodes, thus the success of using the 95th and 99th percentile to identify and penalize super-nodes.

Figure 7.13 illustrates the effect of the four super-node detection techniques when the weight cutoff parameter is dropped to 0.1%. In this case, the indexing and search parameters do not entirely mask super-node effects, causing the performance of Basic-BFE to drop significantly. However, stddev-detect and clique-detect still perform well, because they are able to properly identify and mitigate super-nodes. Percentile-detect's performance drops because its parameters need to be tuned based on the effect of super-nodes within the specific graph. Increasing the beta parameter in this case would improve the performance of percentile-detect.

By mitigating the effect of super-nodes through detection techniques, Connections is able to provide a wide range of parameter settings that improve the quality of search results. These results also indicate that many of the inaccuracies in the relation graph are formed by misleading application behavior, rather than user behavior. By exploring techniques that can identify a user's desired action from an application's behavior, many of the inaccurate links could be culled from the relation graph before search time, potentially improving search quality even further.

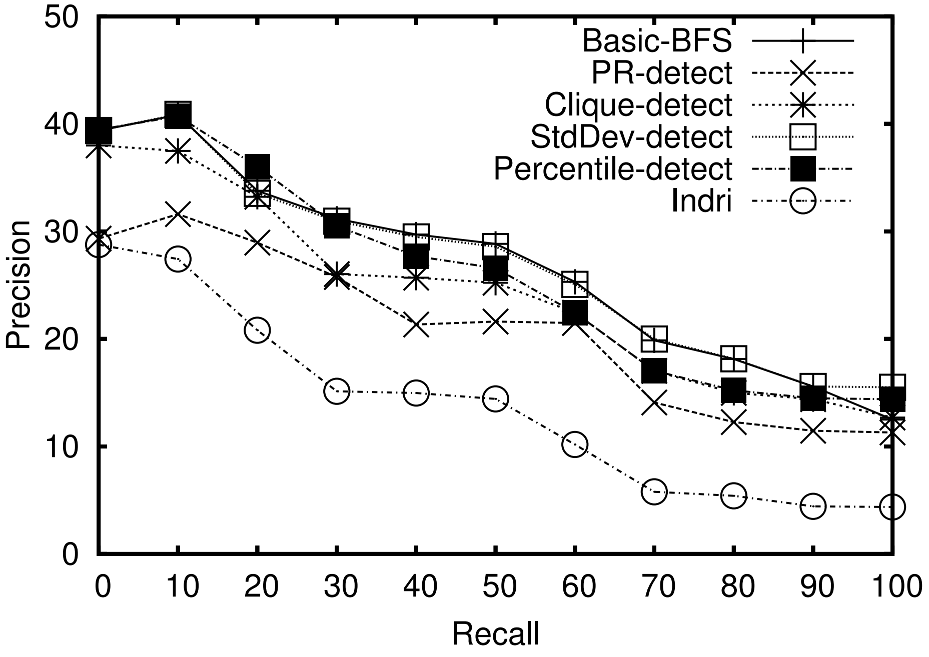


Figure 7.11: Super node detection

7.2 Alternate content analysis

While Indri provides detailed confidence information for its ranked results, most file system search tools in use today provide either no ranking information (such as Microsoft Fast Find or Glimpse) or a rank order with no provided confidence information (such as Google Desktop or MSN Desktop). To see how context can enhance these tools, I consider the following questions: (1) how much benefit is derived from context when no rank order is provided for the content-only results, and (2) given a rank order, can a naive confidence curve be applied to the order to achieve benefits similar to those achieved with full confidence information?

To answer the these questions, I compared the original Connections system that uses full confidence information to two alternate implementations. The first ignores all confidence and ranking information and assumes equal confidence for all results returned by Indri. The second ignores provided confidence information and instead applies a linear confidence function based on the provided rank order. Equation 7.1 describes how confidence C_i was calculated for each of the n nodes in the results, where $i \in \{0 \ldots (n-1)\}$ is the rank order of each node.[2] Effectively, the first result receives the most confidence, with linear spacing

[2]This particular equation was choosen to provide both linearly decreasing confidences and the property that the confidences would sum to 1. A more detailed examination of this space would likely require a more complex model [9, 11, 55].

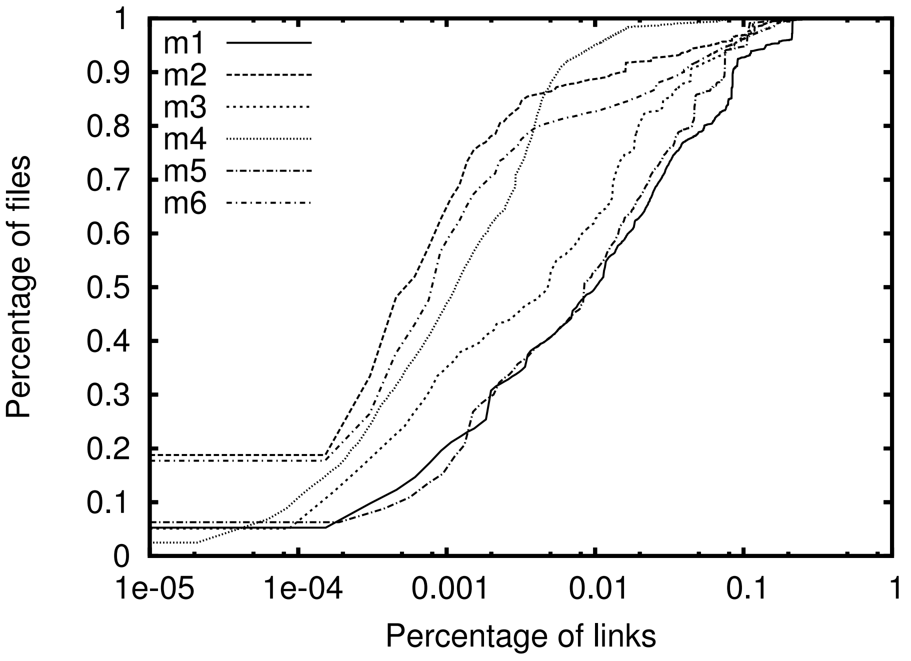


Figure 7.12: Super node distribution

between ranked results.

$$C_i = \frac{2 \cdot (n - i)}{n \cdot (n + 1)} \tag{7.1}$$

The results of this comparison are shown in figure 7.14. There are three interesting aspects of these results. First, even with little or no information about the rankings of the content analysis tool, Connections can still make informed and accurate ranking decisions based solely on context information. Second, the difference between rank order information and no information is small. While surprising, this is primarily due to the low precision of the content rankings; by identifying the most relevant items in the set based on user actions Connections is able to provide better rankings by pushing these items higher in the list. Third, after 60% recall, the precision of Connections is dominated entirely by the context ranking. This is because, in most queries, the content analysis systems do not achieve this level of recall and thus do not provide rankings for results past this point. This lack of information from content-only rankings may also contribute to the advantage of context rankings at lower recall levels.

To further explore the effect of alternate content analysis methods, I explored two other alternate content rankings: a ranking that uses a strict boolean AND to identify results and a ranking that uses pseudo-relevance feedback to alter the content results and rankings.

Figure 7.15 shows the results of using a boolean AND to identify and rank content

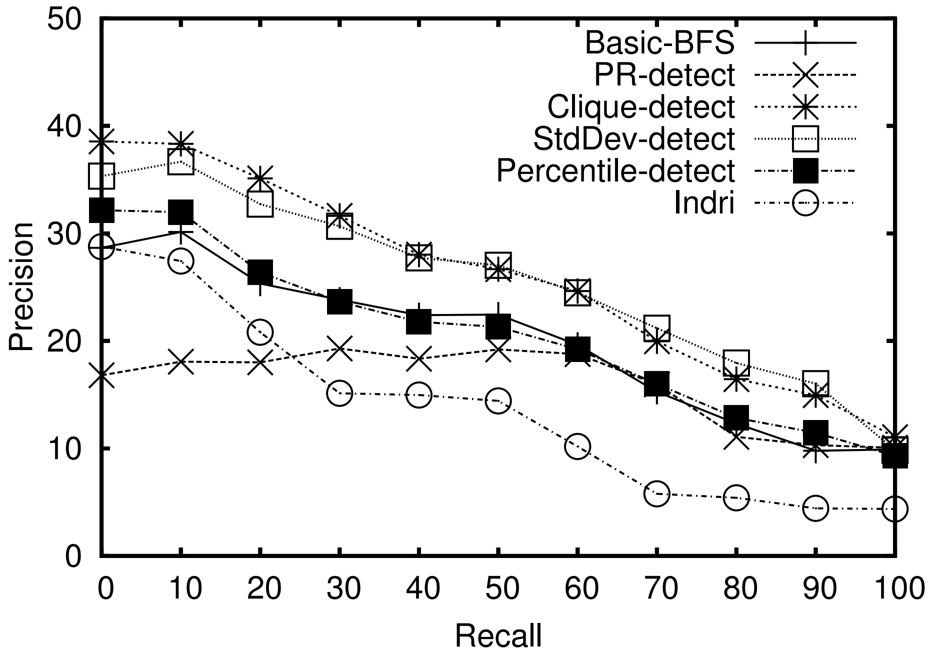


Figure 7.13: Super node detection with weight cutoff 0.1%

results, as opposed to Indri's original inference network approach. Unsurprisingly, boolean AND provides lower precision at low recall levels than Indri's default ranking. When adding context information, there are two takeaways. First, because the context ranking relies on the provided content confidences, errors in the content rankings will manifest in lower benefits for context; for example, at 10%, the benefit with boolean AND is only 8% as opposed to the 11% seen by the original system. Second, while the total number of correct results returned by boolean AND is similar to that of Indri's original results, the number of related results is significantly smaller. This is what causes the sharp drop of Connections at 100% when using boolean AND; without the related information, Connections cannot find as many additional results.

Figure 7.16 shows the results of using pseudo-relevance feedback[3] on the original content results, as opposed to Indri's original inference network approach alone. Pseudo-relevance feedback takes the first n results of the original content-only search and then, using content-similarity techniques, identifies additional keywords from these results that should be included with the user-provided keywords. This full set of keywords is then queried to get the final result list.

Interestingly, the pseudo-relevance feedback performs very similarly to original content rankings. This is likely caused by the assumption within pseudo-relevance feedback that

[3]I ran the pseudo-relevance feedback on the first ten results, finding up to 20 additional terms, as is described in Chapter 4.3.

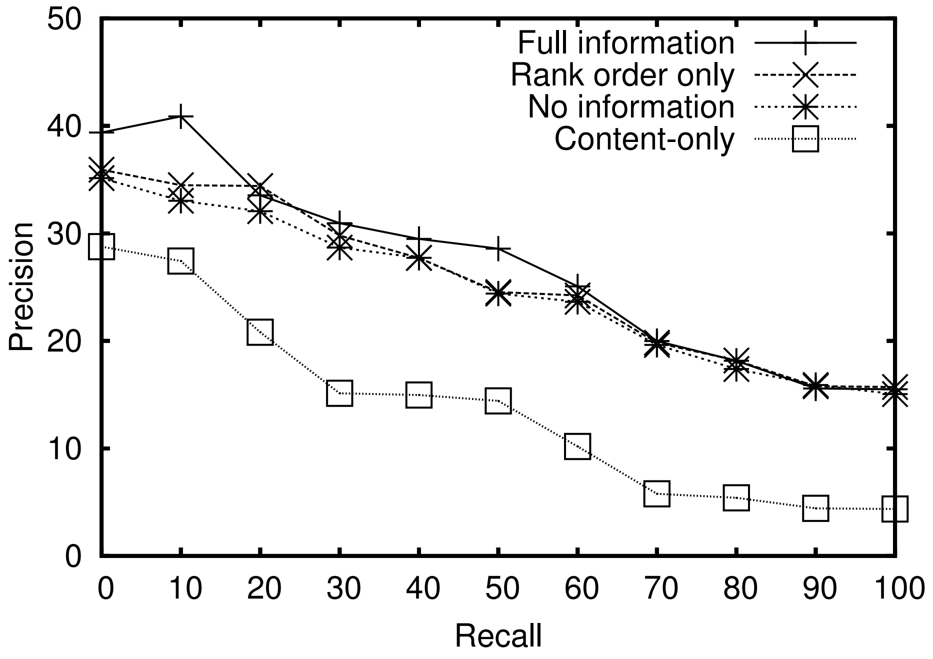


Figure 7.14: Combining context with alternate content analysis tools

the provided "feedback" results are actually relevant to the query. Unfortunately, because most of the first ten results are inaccurate, the relevance feedback finds additional terms that are also irrelevant to the query. In contrast, because Connections can find the temporal commonality among many low-ranked results to increase the ranking of relevant results, it often reduces the relevance of high-ranked results that are contextually irrelevant.

The change to using pseudo-relevance feedback in underlying content results also has a sharp effect on the total number of correctly located items in Connections. The addition of pseudo-relevance feedback increases the content similarity of the returned results, which in turn appears to reduce the context spread. In other words, some contextually related results that don't have content similarity to the first ten results are dropped, reducing the number of additional hits that Connections can locate. This matches with the increase in improvement seen from 20% to 60% recall, since the increase in items within a single context provide Connections with more information to improve the rank of those items.

Overall, these results provide two takeaways. First, context information provides useful information for ranking results in personal file search. Even without knowledge of the content rankings, context-enhanced search outperforms content-only search alone. Second, focusing the content results too strongly on particular aspects of the query terms, thus finding either too few results, or results with very similar content, can limit the effectiveness of context-enhanced search. In many cases, a term is relevant to different contextual groups, but Connection cannot identify additional relevant items from a contextual group without

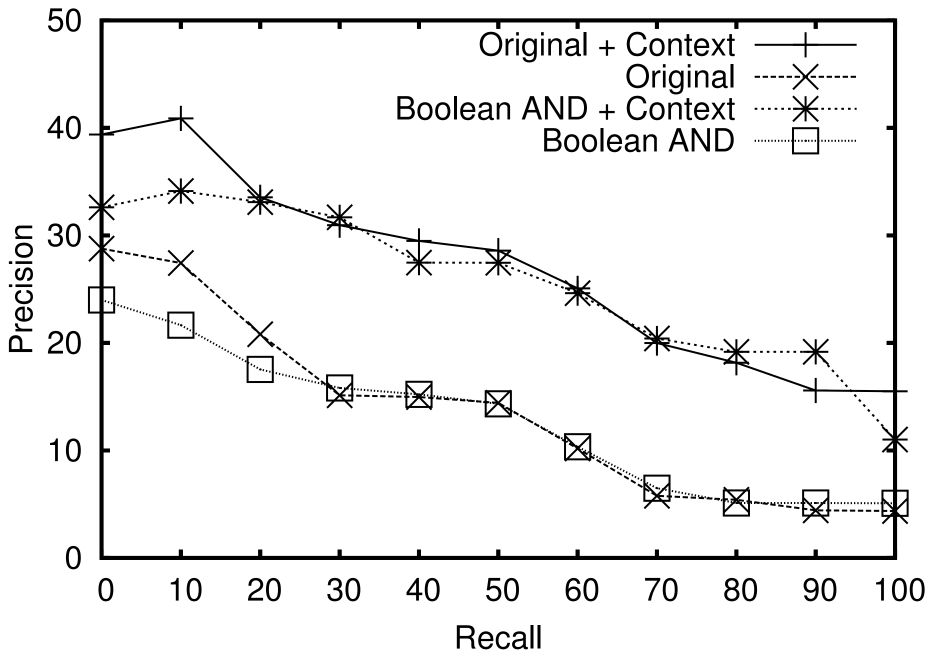


Figure 7.15: Using a strict boolean AND to identify content results

starting points from within that group.

7.3 Existing user input

In traditional semantic file systems, attributes come from two sources: content and user input. Up until now, I have only discussed using content analysis as a basis for search, mostly because user input is notoriously difficult to obtain. However, file systems already contain one source of user input: the path and name of a file. While the file name information is already used to enhance content analysis, this section discusses using the clustering aspect of directories to try to enhance context analysis.

To see how effective the directory hierarchy is at capturing the user's context, I created a version of Connections that generated its relation graph by connecting files in the same directory together with equally weighted links. Using this directory-based relation graph, I could evaluate the effectiveness of using existing user input as context. If using directory information in the relation graph can enhance content-only search, then combining directory information with temporal locality might provide additional benefits.

Figure 7.17 graphs precision against recall for each of the three systems: Connections using temporal locality, Connections using the directory hierarchy, and Indri. The result is that using the directory hierarchy information reduces the precision of the system at each recall level, performing worse than content-only search. This reduction is caused by two

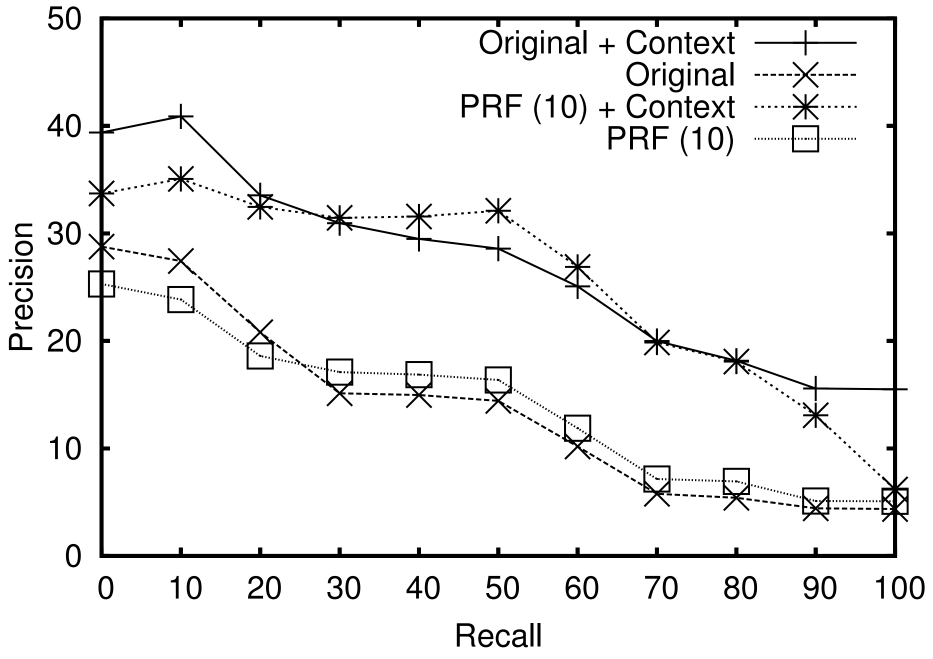


Figure 7.16: Using pseudo-relevance feedback to identify content results

different user behaviors. First, many users do not actively organize many of their files and directories. For example, a user's "downloads" directory often contains a large number of unrelated files. If a file within that directory is found using content, the result is a large number of false-positives created by the "closely related" items in the graph. Second, users will often organize their files in one way and then search for them another way. For example, a user might organize a set of papers by conference, but then later search for them by topic (or vice versa), again resulting in a large number of false positives.

7.4 Graph growth on result quality

Because Connections does not perform any hysteresis (e.g., decaying link weights over time), there is potential that misleading information stored in the graph could eventually have enough weight to have an effect on result quality. To explore this possibility, I continued to gather traces from each of the users for an additional 18 months after the time of the study. Using these traces, I built eight relation graphs for each machine, one at each three month interval throughout the lifetime of the traces, using the default settings described in Section 5.1.1. Thus the 9 month relation graph contained 9 months of traces, while the 15 month relation graph contained 15 months of traces, etc. I then re-ran each of the queries on each of the respective relation graphs to see how the changes in the graph over time would affect the precision and recall of the searches.

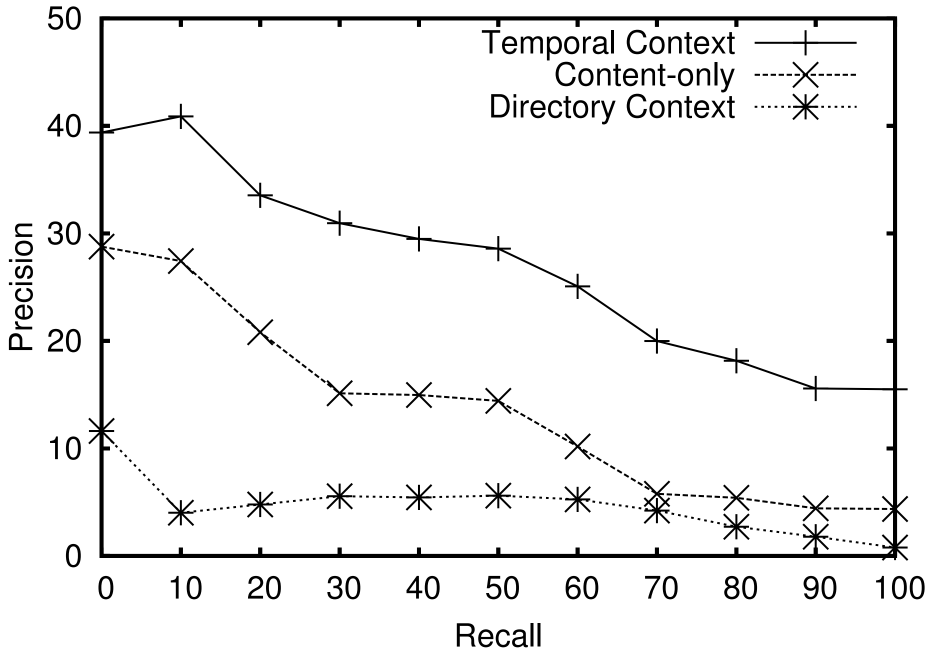


Figure 7.17: Applying existing user input from the directory hierarchy

Figure 7.18 illustrates the precision/recall curves for each of the eight time periods examined, with the results at six months matching the results presented in Figure 5.1. Because the queries were issued at six months, it is expected for the graph at six months to provide the most accurate results, since it has complete context for the queries being issued, without having additional, potentially faulty, information.

In the case where the graph contains less trace information (e.g., at three months), some of the missing information describes files relevant to a query that were created and accessed between when the graph was generated and when the query was issued. The result is false-negatives that lead to lower overall system precision.

Examining the graphs with more trace information, it is clear that the system's precision slowly degrades over time. There are two possible explanations for this behavior. One possibility is that the addition of links, over time, adds enough noise to the graph that it becomes impossible to discern between relevant and irrelevant relationships, indicating that some amount of hysteresis will be critical to any live system using these techniques. Another possibility is that, over time, additional results in the graph (i.e., newly created or accessed files) are relevant to the query and that the addition of these results is artificially reducing the precision of the system (since the standard against which results are measured was also generated at six months).

To experiment with this possibility, I was able to get 3 users to repeat the study with 14 queries using results gathered on the graph data at 24 months. By having the users re-

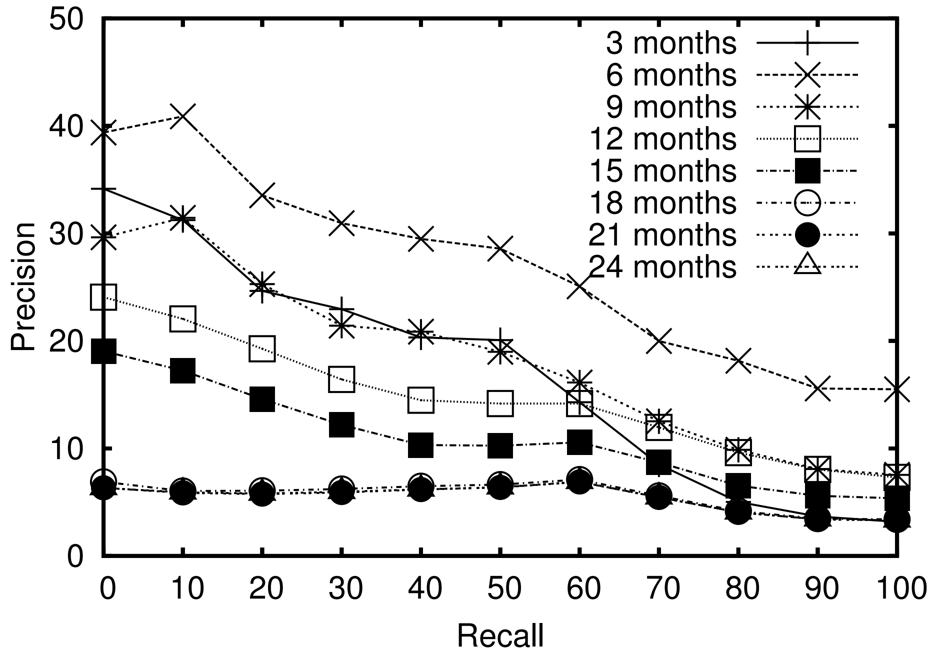


Figure 7.18: Effect of graph growth on result quality

evaluate the results, and regenerate the oracle results, it is possible to measure the benefit of any new results added by additional nodes in the graph.

Figure 7.19 illustrates the precision/recall curves for just those 3 user's queries on for different systems: Indri's content-only search, the six month graph compared against the original oracle results created at six months, the 24 month graph compared against the original oracle results, and the 24 month graph compared agains the new oracle results generated at 24 months. Just as before, the 24 month graph's precision drops significantly when compared against the original oracle results. However, when compared against the new oracle results, the precision of the system returns to nearly the level of the six month graph. This result indicates that (1) the context of a query is important, and the results of a query may change over time, (2) the graph continues to capture valuable information, even over long periods of time, and (3) the slight drop in precision, especially at low recall levels, indicates that some hysteresis could improve these results further by reducing the effect of ephemeral and irrelevant links over time, but may not be critical to implementing a working system.

7.5 Wrap-up

This sensitivity analysis provides four important takeaways. First, the algorithms described in Chapter 3 are robust to both improper parameter settings of the underlying algorithms

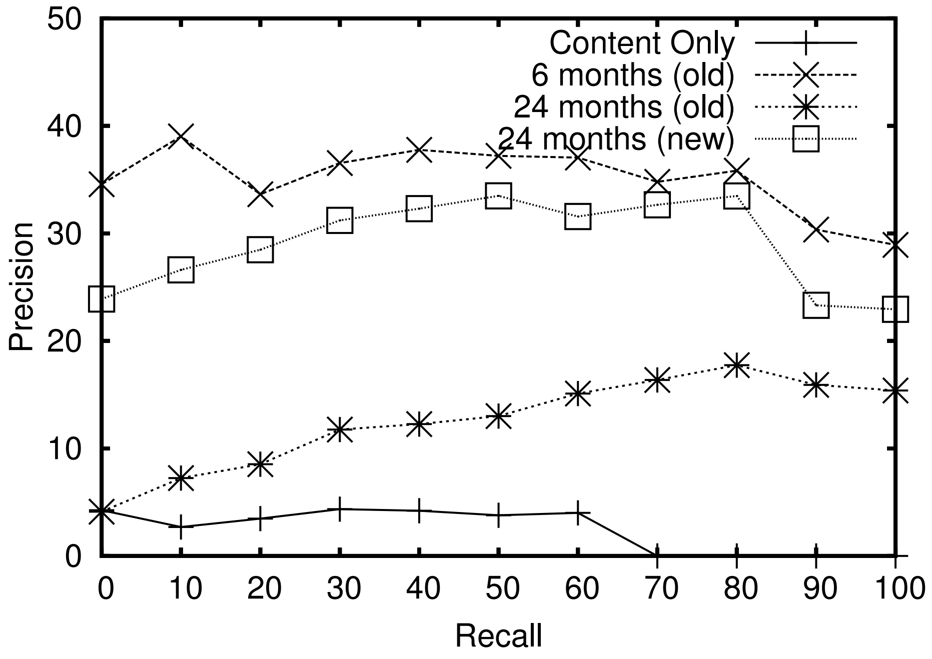


Figure 7.19: Effect of graph growth on result quality given new classification

and graph growth over time. Second, with better starting points, Connections can more accurately identify correct contextual results, thus, as content analysis tools continue to improve, so should context-enhanced search tools. Third, sources of user input, while potentially valuable, must be considered carefully before being included as contextual clues. Fourth, although the detrimental effects of additional history information are small, long-term archival data stores will need to address the currently open problem of hysteresis to reduce index sizes and prevent eventual dilution of context information.

8 Conclusions

Context is a powerful tool in assisting users to organize and locate data. It has been identified by others as a key way that users think about and locate their personal data, and its use on the world wide web has resulted in significant improvements in search quality.

This thesis describes Connections, a context-enhanced file system search tool. Connections identifies contextual inter-file relationships using temporal locality of file accesses taken from system-level traces. Using these relationships, Connections extends and re-ranks the results of traditional content-only search tools. Offline analysis of Connections shows improvements in both precision and recall, and a live user study suggests that Connections reduces the amount of user input and time required to successfully locate desired data. This confirms the thesis of this work that automatically gathered context information provides more attributes to more files that content analysis alone and that this increase in information improves the effectiveness of file indexing and search tools.

A sensitivity analysis of Connections's algorithms and components shows that, while there are potential pitfalls with Connections's design, its advantages are surprisingly robust to imperfect parameter settings. A performance analysis of Connections shows that the benefits of context-enhanced search can be achieved with minimal space utilization and performance overheads.

Overall, the largest contribution of this work is to show the relevance of context information in personal file search. Despite the simplicity of many of the algorithms presented in this work, integrating context information significantly improves a large class of queries, showing the potential of combining context information with traditional content-only analysis techniques. The remainder of this chapter describes the additional contributions and limitations of this work, and describes some areas of future work that could lead to further advancements.

8.1 Contributions and limitations

This thesis contributes three key advances in personal file search. First, it provides algorithms for identifying and utilizing temporal inter-file relationships in personal file search. The analysis of these algorithms, along with their parameter trade-offs, lays the groundwork for future advances and enhancements in this space. Furthermore, while these algorithms are very likely non-optimal, they show that content alone is insufficient to completely satisfy a large class of personal data queries. In many cases, relevant query results include data that do not contain the user-specified keywords, particular in the case of non-text queries but also in a surprising number of non-text queries. These algorithms succeed by identifying these files using context and including them in the result set. Second, this thesis provides a framework for implementing and experimenting with context-enhanced search. Most importantly, this framework provides explicit separation between content and context analysis. As new techniques for both context and content analysis emerge, they can be quickly and easily combined with and compared against existing search tools. Extending the chain of techniques within the framework beyond two (i.e., the existing content-to-context chain) could allow for combinations of multiple context analysis techniques without directly integrating the techniques. Third, this thesis provides Connections, a working context-enhanced search tool for both Linux and Windows. Connections can be used directly to identify contextual relations providing a database of file inter-relationships for use in other applications and as a drop-in replacement for existing search tools.

Connections's context-enhanced search also has advantages over traditional content-only search. First, Connections locates more relevant data for most searches, increasing average recall, particularly when examining all returned results. This increase makes it more likely for users to locate all of the desired data and, when combined with accurate ranking algorithms, provides the opportunity for improved precision. Second, Connections is able to locate relevant files that contain little or no parsable contents, increasing average recall and precision significantly when users search for results that include non-text data. Third, this ability to locate relevant files without the use of keywords extends to locating relevant content-parsable files that do not contain the user-specified keywords for a search, thus increasing recall and precision for some textual searches as well.

Despite Connections's advances and advantages, it still has several limitations, which if addressed, could further improve the effectiveness of its search. First, it requires at least a month of traces before its algorithms improve effectiveness, since without state about the user's access patterns, it can provide no additional information. Second, because the most successful algorithms only locate data that the user has created or modified Connections is ineffective in read-only data sets and for data created before Connections is installed. Third, unlike hyperlinks, the links identified by Connections do not contain associated keywords. This limit on available information reduces the effectiveness of many of the web-search algorithms that rely on these contextual clues to further improve search. Potential approaches to deal with the second two limitations, as well as other potential advancements using Connections are discussed below.

8.2 Future work

As shown in this work, temporal locality is a powerful form of context that can be leveraged to improve file indexing and search. However, despite the benefits shown here, the form of temporal locality considered in this work is crude. Connections employs system-level information, which captures application behavior rather than direct user actions. The result is that application oddities can significantly affect the witnessed temporal relationships. This is most obvious when using settings such as the read/read access filter, which introduces significant numbers of false-positives. While techniques such as super-node detection can alleviate some of these problems, a long-term solution requires a better understanding of the application's translation of user intent to file system activity. One direction for future research would be to explore forms of application profiling to help capture this translation and better inform Connections of the validity of formed relationships.

Other sources of context could both help inform Connections of relationship validity and provide additional sources of relationships. Application assistance would be one way to enhance application profiling and remove sources of faulty relationships. With a better understanding of link relevance, settings such as read/read could be employed, allowing Connections's to locate any files that the user accesses (rather than the current limitation of user created and modified files). Furthermore, application provided keywords could be assigned to created links, potentially opening the door for application of web ranking algorithms that harness hyperlink text. Furthermore, application assistance could be useful when application cache file accesses (e.g., web caches, email caches, text document buffer caches, etc.) and could provide more detailed information about sub-document relationships (e.g., individual emails from an IMAP folder, links on a cached web-page, paragraphs in a document, etc.). Other areas of consideration include hints from user interfaces, environmental factors (e.g., mobile computing), user specified contexts, etc. [25].

Connections currently keeps all gathered context information permanently, but as the number of sources of context and the time over which information is kept both grow, integrating hysteresis into Connections will become an important task. Even over a two-year span slight degradations in Connections's performance are noticeable and, with the current focus on long-term archival storage projects lasting over a lifetime of data, these degradations will eventually render context-enhanced search untenable if not addressed.

Currently, Connections identifies relationships based on information gathered from a single computer, but today's users often interact with several machines. Actions performed with a set of data on one computer will likely be relevant when accessing or searching for that data from another computer. Examining how context information should be shared among machines will be an interesting area of future work.

Similarly, as users begin to share more data, the question arises regarding inter-user context sharing. If Alice accesses data also used by Bob, should Alice's actions be used to assist Bob in organizing and searching the data? If so, should Bob be able to identify what Alice was doing, and how could one obfuscate that information while still providing relevant contextual hints?

As contextual information continues to become more relevant and accurate, the questions regarding its use will continue to grow. Through the Connections prototype, this work shows one form of context and how it can improve the utility of local file search, opening the door to explore these questions and others.

Bibliography

[1] N. Abdul-Jaleel, J. Allan, W. B. Croft, F. Diaz, L. Larkey, X. Li, D. Metzler, M. D. Smucker, T. Strohman, H. Turtle, and C. Wade. UMass at TREC 2004: Notebook. Text Retrieval Conference, 2004.

[2] N. Abdul-Jaleel, A. Corrada-Emmanuel, Q. Li, X. Liu, C. Wade, and J. Allan. UMass at TREC 2003: HARD and QA. Text Retrieval Conference, pages 715–725, 2003.

[3] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. ACM SIGMOD International Conference on Management of Data, pages 207–216, 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. International Conference on Very Large Databases, 1994.

[5] AltaVista, http://www.altavista.com/.

[6] A. Amer, D. Long, J.-F. Paris, and R. Burns. File access prediction with adjustable accuracy. International Performance Conference on Computers and Communication. IEEE, 2002.

[7] American Library Association. *Anglo-American Cataloguing Rules 2005.* American Library Association.

[8] Apple Spotlight, http://www.apple.com/macosx/features/spotlight/.

[9] A. Arampatzis and A. van Hameren. The Score-Distributional Threshold Optimization for Adaptive Binary Classification Tasks. Pages 285-293.

[10] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, 1999.

[11] C. Baumgarten. A Probabilitstic Solution to the Selection and Fusion Problem in Distributed Information Retrieval. Pages 246-253.

[12] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The worldwide web. *Communications of the ACM*, **37**(8):76+, August 1994.

[13] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, **284**(5):34–43, 2001.

[14] C. M. Bowman, P. B. Danzig, U. Manber, and M. F. Schwartz. Scalable internet resource discovery: research problems and approaches. *Communications of the ACM*, **37**(8):98–114, 1994.

[15] P. S. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. International Conference on Knowledge Discovery and Data Mining, pages 9–15, 1998.

[16] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, **30**(1–7):107–117, 1998.

[17] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY Retrieval System. International Conference on Database and Expert Systems Applications, pages 78–83, 1992.

[18] R. Card, T. Ts'o, and S. Tweedie. Design and implementation of the Second Extended Filesystem. First Dutch International Symposium on Linux, 1994.

[19] G. Chen and D. Kotz. *A survey of context-aware mobile computing research.* Dartmouth Computer Science Technical Report TR2000-381. November 2000.

[20] C. Clarke, N. Craswell, and I. Soboroff. Overview of the TREC 2004 Terabyte Track. Text REtrieval Conference. NIST, 2004.

[21] Clusty, http://www.clusty.com/.

[22] K. Collins-Thompson, P. Ogilvie, and J. Callan. Initial results with structured queries and laguage models on half a terabyte of text. Text REtrieval Conference. NIST, 2004.

[23] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/Gather: a cluster-based approach to browsing large document collections. ACM SIGIR International Conference on Research and Development in Information Retrieval, pages 318–329. ACM, 1992.

[24] M. Czerwinski, S. Dumais, G. Robertson, S. Dziadosz, S. Tiernan, and M. van Dantzich. Visualizing implicit queries for information management and retrieval. CHI, pages 560–567, 1999.

[25] P. Dourish. What we talk about when we talk about context. *Personal and ubiquitous computing*, **8**(1):19–30, February 2004.

[26] P. Dourish, K. Edwards, A. LaMarca, and M. Salisbury. Uniform Document Interaction with Document Properties. ACM Symposium on User Interface Software and Technology, 1999.

[27] S. Dumais, E. Cutrell, J. Cadiz, G. Jancke, R. Sarin, and D. CRobbins. Stuff I've seen: a system for personal information retrieval and re-use. ACM SIGIR International Conference on Research and Development in Information Retrieval. ACM press, 2003.

[28] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. ACM SIGMOD International Conference on Management of Data, pages 419–429, 1994.

[29] S. Fertig, E. Freeman, and D. Gelernter. Lifestreams: an alternative to the desktop metaphor. ACM SIGCHI Conference, pages 410–411, 1996.

[30] G. R. Ganger and M. F. Kaashoek. Embedded inodes and explicit grouping: exploiting disk bandwidth for small files. USENIX Annual Technical Conference, pages 1–17, January 1997.

[31] V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell, and J. French. Clustering large datasets in arbitrary metric spaces. International Conference on Data Engineering, pages 502–511, 1999.

[32] J. Gemmell, G. Bell, R. Lueder, S. Drucker, and C. Wong. MyLifeBits: fulfilling the Memex vision. ACM Multimedia, pages 235–238. ACM, 2002.

[33] J. Gemmell, L. Williams, K. Wood, G. Bell, and R. Lueder. Passive capture and ensuing issues for a personal lifetime store. Workshop on Continuous Archival and Retrieval of Personal Experiences, pages 48–55, 2004.

[34] D. Giampaolo. *Practical file system design with the Be file system.* Morgan Kaufmann, 1998.

[35] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole Jr. Semantic file systems. ACM Symposium on Operating System Principles. Published as *Operating Systems Review*, **25**(5):16–25, 13–16 October 1991.

[36] Google Desktop, http://desktop.google.com/.

[37] Yahoo Desktop, http://desktop.yahoo.com/.

[38] B. Gopal and U. Manber. Integrating content-based access mechanisms with hierarchical file systems. Symposium on Operating Systems Design and Implementation, pages 265–278. ACM, 1999.

[39] J. Griffioen and R. Appleton. Reducing file system latency using a predictive approach. Summer USENIX Technical Conference, pages 197–207. USENIX Association, 1994.

[40] Grokker, http://www.grokker.com/.

[41] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. ACM SIGMOD International Conference on Management of Data, pages 73–84, 1998.

[42] E. Horvitz, J. Breese, D. Heckerman, D. Hovel, and K. Rommelse. The Lumiere project: bayesian user modeling for inferring the goals and needs of software users. Conference on Uncertainty in Artificial Intelligence, pages 256–265, 1998.

[43] M. A. W. Houtsma and A. N. Swami. Set-oriented mining for association rules in relational databases. International Conference on Data Engineering, pages 25–33, 1995.

[44] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Extrapolation methods for accelerating PageRank computations. World Wide Web Conference. ACM, 2003.

[45] D. Kelly and J. Teevan. Implicit feedback for inferring user preference: a bibliography. *SIGIR Forum*, **32**(2):18–28, 2003.

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

[47] T. M. Kroeger and D. D. E. Long. Predicting file system actions from prior events. USENIX Annual Technical Conference, pages 319–328. USENIX Association, 1996.

[48] H. Lei and D. Duchamp. An analytical approach to file prefetching. USENIX Annual Technical Conference. USENIX Association, 1997.

[49] The Lemur Toolkit, http://www.lemurproject.org/.

[50] J. R. Lorch and A. J. Smith. The VTrace tool: Building a system tracer for Windows NT and Windows 2000. *MSDN Magazine*, **15**(10):86–102, October 2000.

[51] Linux Cross-Reference (LXR), http://lxr.linux.no/.

[52] Lycos, http://www.lycos.com/.

[53] U. Manber, M. Smith, and B. Gopal. WebGlimpse: combining browsing and searching. USENIX Annual Technical Conference. USENIX Association, 1997.

[54] U. Manber and S. Wu. GLIMPSE: a tool to search through entire file systems. Winter USENIX Technical Conference, pages 23–32. USENIX Association, 1994.

[55] R. Manmatha, T. M. Rath, and F. Feng. Modeling Score Distributions for Combining the Outputs of Search Engines. Pages 267-275.

[56] M. L. Mauldin. Retrieval performance in Ferret a conceptual information retrieval system. ACM SIGIR International Conference on Research and Development in Information Retrieval, pages 347–355. ACM Press, 1991.

[57] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry. A fast file system for UNIX. *ACM Transactions on Computer Systems*, **2**(3):181–197, August 1984.

[58] D. Metzler, T. Strohman, H. Turtle, and W. B. Croft. Indri at TREC 2004: terabyte track. Text Retrieval Conference, 2004.

[59] T. M. Mitchell. *Machine learning.* McGraw-Hill, 1997.

[60] B. C. Neuman. The Prospero file system: a global file system based on the virtual system model. USENIX Workshop on File Systems, pages 13–28, 21–22 May 1992.

[61] J. S. Park, M.-S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. ACM SIGMOD International Conference on Management of Data, pages 175–186, 1995.

[62] T. Qin, T.-Y. Liu, X.-D. Zhang, Z. Chen, and W.-Y. Ma. A study of relevance propagation for web search. ACM SIGIR International Conference on Research and Development in Information Retrieval, 2005.

[63] D. Quan, D. Huynh, and D. R. Karger. Haystack: a platform for authoring end user semantic web applications. International Semantic Web Conference, 2003.

[64] B. Rhodes. Using physical context for just-in-time information retrieval. *IEEE Transactions on Computers*, **52**(8):1011–1014. ACM Press, August 2003.

[65] B. Rhodes and T. Starner. The Remembrance Agent: a continuously running automated information retrieval system. International Conference on The Practical Application of Intelligent Agents and Multi Agent Technology, pages 487–495, 1996.

[66] M. Russinovich. Inside NTFS. Windows NT Magazine, January 1998.

[67] S. Sechrest and M. McClennen. Blending hierarchical and attribute-based file naming. International Conference on Distributed Computing Systems, pages 572–580, 1992.

[68] A. Shakery and C. X. Zhai. Relevance propagation for topic distillation UIUC TREC-2003 web track experiments. Text REtrieval Conference, 2003.

[69] R. Song, J.-R. Wen, S. Shi, G. Xin, T.-Y. Liu, T. Qin, X. Zheng, J. Zhang, G. Xue, and W.-Y. Ma. Microsoft Research Asia at web track and terabyte track of TREC 2004. Text REtrieval Conference, 2004.

[70] J. L. Steffen. Interactive examination of a C program with Cscope. Winter USENIX Technical Conference, pages 170–175. USENIX Association, 1985.

[71] J. Teevan, C. Alvarado, M. S. Ackerman, and D. R. Karger. The perfect search engine is not enough: a study of orienteering behavior in directed search. Conference on Human Factors in Computing Systems, pages 415–422. ACM, 2004.

[72] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. ACM SIGIR International Conference on Research and Development in Information Retrieval. ACM, 2005.

[73] Terrier Information Retrieval Platform, http://ir.dcs.gla.ac.uk/terrier/.

[74] H. Toivonen. Sampling large databases for association rules. International Conference on Very Large Databases, pages 134–145, 1996.

[75] D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, **393**:440–442, 1998.

[76] Merriam-Webster OnLine, http://www.m-w.com/.

[77] Windows Desktop Search, http://www.microsoft.com/windows/desktopsearch/.

[78] Microsoft WinFS, http://http://msdn.microsoft.com/data/ref/winfs/.

[79] J. Xu and W. B. Croft. Query expansion using local and global document analysis. ACM SIGIR International Conference on Research and Development in Information Retrieval, pages 4–11, 1996.

[80] Z. Xu, M. Karlsson, C. Tang, and C. Karamanolis. Towards a semantic-aware file store. Hot Topics in Operating Systems, pages 145–150. USENIX Association, 2003.

[81] K. Yaghmour and M. R. Dagenais. Measuring and characterizing system behavior using kernel-level event logging. USENIX Annual Technical Conference, 2000.

[82] Yahoo!, http://www.yahoo.com/.