Connections: Using Context to Enhance File Search

Craig A. N. Soules, Gregory R. Ganger
Carnegie Mellon University

ABSTRACT

Connections is a file system search tool that combines traditional content-based search with context information gathered from user activity. By tracing file system calls, Connections can identify temporal relationships between files and use them to expand and reorder traditional content search results. Doing so improves both recall (reducing false-positives) and precision (reducing false-negatives). For example, Connections improves the average recall (from 13% to 22%) and precision (from 23% to 29%) on the first ten results. When averaged across all recall levels, Connections improves precision from 17% to 28%. Connections provides these benefits with only modest increases in average query time (2 seconds), indexing time (23 seconds daily), and index size (under 1% of the user's data set).

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; D.4.3 [**Operating Systems**]: File Systems Management—*File organization*

General Terms

Algorithms, Design, Human Factors, Management

Keywords

context, file system search, successor models

1. INTRODUCTION

Users need more effective ways of organizing and searching their data. Over the last ten years, the amount of data storage available to individual users has increased by nearly two orders of magnitude [16], allowing today's users to store practically unbounded amounts of data. This shifts the challenge for individual users from deciding what to keep to finding particular files when needed.

Most personal computer systems today provide hierarchical, directory-based naming that allows users to place each file along a single, unique path. Although useful on a small scale, having only one classification for each file is unwieldy for large data sets. When traversing large hierarchies, users may not remember the exact location of each file. Or, users may think of a file in a different manner than when they filed it, sending them down an incorrect path in the hierarchy.

Attribute-based naming allows users to classify each file with multiple attributes [9, 12, 37]. Once in place, these attributes provide additional paths to each file, helping users locate their files. However, it is unrealistic and inappropriate to require users to proactively provide accurate and useful classifications. To make these systems viable, they must automatically classify the user's 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 [9], use content analysis to automate attribute assignment. Search tools, such as Google Desktop [11], use content analysis to map user queries to a ranked list of files.

Although clearly useful, there are two limitations to content analysis. First, only files with understandable contents can be analyzed (e.g., it is difficult to identify attributes for movie clips or music files). Second, examining only a file's contents overlooks a key way that users think about and organize their data: *context.*

Context is "the interrelated conditions in which something exists or occurs" [43]. 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 [38] 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. These contextual relationships may be impossible to gather from a file's contents.

The focus of our work is to increase the utility of file system search using context. In this paper, we specifically examine *temporal locality*, one of the clearest forms of context and one that has been successfully exploited in other areas of file systems. Temporal locality captures a file's setting, connecting files through the actions that make up user tasks. Connections is a new search tool that identifies temporal contextual relationships between files at the time they are being accessed using traces of file system activity. When a user performs a search, Connections first locates files using traditional content-based search and then extends these results with contextually related files.

User studies with Connections show that combining content analysis with context analysis improves both *recall* (increasing the number of relevant hits) and *precision* (returning fewer false positives) over content analysis alone. When compared to Indri, a state-of-the-art content-only search tool [25], Connections increases average precision at each recall level, increasing the overall average from 17% to 28%. When considering just the top 30 results, Connections increases average recall from 18% to 34%, and average precision from 17% to 23%. With no cutoff, Connections increases average recall from 34% to 74% and average precision from 15% to 16%.

The remainder of this paper is organized as follows. Section 2 describes related work. Section 3 outlines the design and implementation of Connections. Section 4 analyzes the utility of Connections's context-based search. Section 5 discusses interesting considerations for building context-based search systems like Connections.

2. BACKGROUND AND RELATED WORK

Context is one of the key ways that users remember and locate data [38]. For example, a user may not remember where they stored a downloaded email attachment, but may remember several contextually related items: the sender of the email, approximately when the email arrived, where they were when they read the email, or what they were doing at the time they downloaded the attachment. Automatically identifying the file's context can assist the user in locating it later using any of this related contextual information, rather than only the filename or contents of the attachment.

This section begins by describing background work from file systems and web search that motivated our exploration of context-based search. It then describes work that successfully utilizes Connection's specific form of context, temporal locality, in other domains. The section ends with a description of search and organizational systems that utilize other forms of context, and how our work on identifying contextual relationships with temporal locality could enhance them.

2.1 Semantic file systems

Many researchers have identified organizational problems with strict hierarchical naming. Several proposed attribute-based naming as a solution: attaching multiple attributes (or keywords) to a file to improve organizational structure and search capabilities. The Semantic file system [9] was one of the first to explore attribute-based naming, providing a searchable mapping of ⟨category, value⟩ pairings to files. Other research [12, 37] and commercial [8] systems merge hierarchical namespaces with flat attribute-based namespaces in various ways, providing new organizational structures.

Although attribute-based naming has the potential to improve file system search, these systems focused on the mechanism to store additional attributes, not the sources of attributes. Traditionally, attributes come from two sources: users and content analysis. Understandably, most users are unwilling to perform the difficult and time-consuming task of manually assigning attributes to their files. As a result, rather than changing the underlying file system structure, recent focus has been on using content analysis to provide file system search on existing hierarchical systems.

2.2 Content-based search

The simplest content-based search tools (e.g., UNIX tools *find* and *grep*) scan the contents of a set of files for a given term (or terms), returning all hits. To speed this process, tools such as *locate* and Glimpse [29] use indices to reduce the amount of data accessed during a search. These tools work on the premise that the provided keywords can narrow the resulting list of files to a more human-searchable size.

Commercial systems such as Google Desktop [11] and X1 desktop search [44] leverage the work done in text-based information retrieval to provide more accurate, ranked search results. Although the exact methods of commercial systems are unpublished, it is likely they use techniques similar to cutting-edge information retrieval systems such as Terrier [40] and Indri [25]. These modern tools use probabilistic models to map documents to terms [41], providing ranked results that include both full and partial matches. Probabilities are generated using methods such as term frequency within a document, inverse term frequency across a collection, and other more complex language models [31].

2.3 Context in web search

Despite these tools, one could argue that it is easier to find things on the web than in one's own filespace — even with the order of magnitude larger search space and the non-personalized organization. This seems strange, but it's largely enabled by the availability of user-provided context attributes in the form of hyperlinks.

Just as in current file system search, early web search-engines [26, 30, 45] relied on user input (user submitted web page classifications) and content analysis (word counts, word proximity, etc.). More recently, two techniques have emerged that use the inherent link structure of the web to identify contextual links. The HITS algorithm [20] defines a sub-graph of the web using content search results, and then uses the link structure of the graph to identify the *authority* and *hub* nodes. The popular Google search engine [10] uses the link structure in two primary ways. First, it uses the text associated with a hyperlink to guide content classifications for the linked site. Second, it uses PageRank [5], an algorithm that uses the link structure of the web to calculate the "importance" of individual sites.

Although successful for the web, these techniques face challenges in personal file systems because tagged, contextual links do not inherently exist in the file system. Our work aims to add untagged contextual links between files using temporal locality. An evaluation of combining our approach with both HITS and PageRank is discussed in Section 4.3.3.

Another context-based approach seen in web search is personalization, using the user's current context to target search results. WebGlimpse [28] took a first step toward personalized search using the concept of a "neighborhood," the set of web pages within a certain hyperlink distance of a given page. Users could choose to search only within the current page's neighborhood, creating a directed search of potentially related pages based on the user's current context. More recent work has focused on targeting results to particular topics or interests, sometimes gathered from a user's recent activity [15, 39]. These systems remove results that do not relate to the user's current context, improving precision. Conversely, Connections extends results using context information gathered from previous activity. We believe these techniques complement each other well: removing unrelated content-based results prevents Connections from gathering files from an unrelated context.

2.4 Identifying context: temporal locality

In file systems, temporal locality can provide some of the contextual clues that are so readily available on the web. By observing how users access their files, the system can determine contextual relationships between files. Our work uses these contextual relationships to enhance existing content-based search tools.

To identify temporal relationships, we borrow from other work that uses temporal locality to model how users access their data. One example of this is using temporal locality to predict access patterns for file prefetching.

Almost all file systems use prefetching to hide storage latencies by predicting what the user will access next and reading it into the cache before they request it. Some prefetching schemes [13, 21] use temporal locality to correlate common user access patterns with individual user contexts. These systems keep a history of file access patterns using *successor models*: directed graphs 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.

The success of these schemes has led to a variety of algorithms for building successor models [2, 24, 27]. Similarly, many cache hoarding schemes use successor models to predict which files a user is likely to need if they become disconnected from the network [19, 22]. Connections uses successor models to identify relationships between files, as they have successfully identified related files in other domains.

2.5 Existing uses of context for file search

Several systems leverage other forms of context as a guide for file organization and search. Gathering context from temporal locality, as Connections does, could enhance these systems by providing additional contextual clues for classification.

The Haystack [34] and MyLifeBits [7] projects use context-based data organization at the core of their interface design, allowing users to group and assign classifications to objects more quickly. If these interfaces motivate users to provide additional classifications, they will result in improved search facilities. Our work could assist users of such a system by adding automated groupings based on temporal locality.

A few systems attempt to determine the user's current context to predict and prefetch potentially desired data. The Lumiere project provides users with help data in the Microsoft Office application suite; attempting to predict user problems by predicting their current context from recent actions [17]. The Rememberance Agent [35, 36] continually provides a list of related files (based on content similarity) to the user while they are working. By feeding recently accessed file data into a content-based search system, it locates files with similar contents that may reflect the user's current context. Our work could enhance such systems by providing additional contextually related files — those based on temporal locality rather than content similarity.

Most content-based search tools organize their search results, allowing the user to hone in on the set of files that is most likely to contain what they are searching for. For example, the Lifestreams project [6] orders search results using the latest access time of the resulting files. The Grokker search tool [14] clusters search results, grouping together files with similar contents. Our work could be used to cluster results using contextual relationships rather than, or in addition to, content-similarity or access time.

3. CONNECTIONS

Connections combines traditional content analysis with contextual relationships identified from temporal locality of file accesses. This section describes its architecture, relationship tracking and result ranking algorithms, and our prototype implementation.

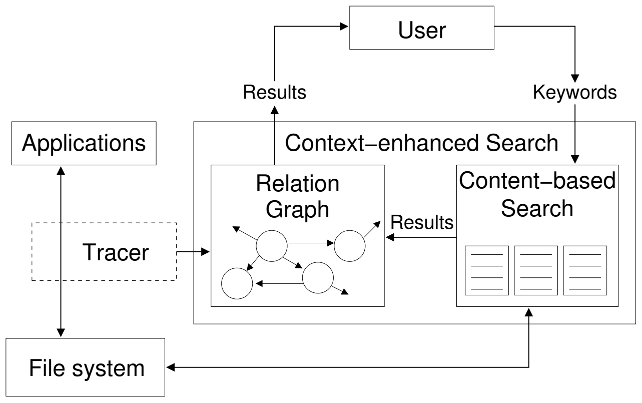

Figure 1: *Architecture of Connections.* **Both applications and the file system remain unchanged, as the only information required by Connections can be gathered either by a transparent tracing module or directly from existing file system interfaces.**

3.1 Architecture

In traditional content-only search systems, the user submits keywords to the search tool, which returns rank-ordered results directly to the user. Often, the search tool is separate from the file system, using a background process to read and index file data.

Figure 1 illustrates the architecture of Connections. From a user's perspective, Connections's context-enhanced search is identical to existing content-only search: a tool separate from the file system that takes in keywords and returns a ranked list of results. Internally, when Connections receives keywords from the user, it begins with a content search, retrieving the same results as a content-only search tool. It feeds these results into the relation-graph, which locates additional hits through contextual relationships. The combined results are then ranked and passed back to the user.

To identify and store these relationships, Connections adds two new components: the tracer and the relation-graph. The *tracer* sits between applications and the file system, monitoring all file system activity. Connections uses these traces to identify contextual relationships between files.

The *relation-graph* stores the contextual relationships between files. Each file in the system maps to a node in the graph. Edges between nodes represent contextual relationships between files, with the weight of the edge indicating the strength of the relationship. Because different users may have different contexts for a particular file, Connections maintains a separate relation-graph for each user based on their file accesses alone. This also separates user tasks from background system activity in single-user systems.

Three algorithms drive the context-based portions of Connections. The first algorithm takes the captured file traces and identifies the contextual relationships, creating the relation-graph. The second algorithm takes content-based search results and locates contextually related files within the relation-graph, creating a smaller *result-graph*. The third algorithm uses the result-graph to rank the combined set of content-based and context-related results.

System Call	Description
`open(S)`	Opens file S reading or writing
`read(S)`	Reads data from file S
`write(D)`	Writes data to file D
`mmap(S)`	Maps file S into a memory region
`stat(S)`	Reads the inode of file S
`dup(S, D)`	Duplicates file handle S to D
`link(S, D)`	Adds directory entry D for file S
`rename(S, D)`	Changes the name of file S to D

Table 1: *File system calls.* **This table lists the file system calls considered by Connections when identifying relationships from traces. Each parameter is identified as either a source (S) or destination (D).**

Each of the algorithms in Connections is specifically designed for flexibility, and as such have several tunable parameters. This flexibility allows us to study a range of options for each algorithm. An evaluation of sensitivity within each algorithm is provided in Section 4.3.

3.2 Identifying relationships

Connections identifies temporal relationships by constructing a successor model from file traces. Files accessed within a given window of time are connected in the relation-graph. Over time, a user's access patterns form probabilistic mappings between files that are the basis of Connections's contextual relationships. The specific algorithm for generating the relation-graph is described by three parameters: relation window, edge style, and operation filter.

Relation window: The relation window maintains a list of input files accessed within the last N seconds.[1] Conceptually, this captures the period of time during which a user is focused on a particular task. Too short a window will miss key relationships, while too large a window will connect files from unrelated tasks.

When the window sees an output file, it creates an edge in the relation-graph with weight 1 from each of the input files to the output file. If such an edge already exists, its weight is incremented. To avoid creating heavy weightings during long sequences of output operations to a single file (e.g., large file writes), any two files are only connected once while the input file stays in the relation window.

Edge style: The edge style specifies whether edges of the relation-graph are directed or undirected. Direction indicates how the edges of the relation-graph may be followed during a search. Directed edges point from input files to output files, while undirected edges may be followed either direction. Conceptually, undirected links reverse the causal nature of the relation-graph, allowing searches to locate the input of a given file.

Operation filter: Each entry in the trace corresponds to a file system operation. An operation filter specifies which system calls to consider from the trace and classifies the source and/or destination files accessed by each system call (shown in Table 1) as an input or an output.

In this paper, we consider three different operation filters: *open*, *read/write*, and *all-ops*. The open filter classifies the source file of an `open` call as both input and output. Conceptually, this captures strict temporal locality: files accessed nearby in time become related.

The read/write filter classifies the source file of a `read` call as input and the destination file of a `write` call as output. Conceptually, this captures causal data relationships: the data read from one file may affect the data later written to another file, relating the files.

The all-ops filter classifies the source file of a `mmap`, `read`, `stat`, `dup`, `link`, or `rename` as input. It classifies the destination file of a `write`, `dup`, `link`, or `rename` as output. This filter extends on the causal relationships of the read/write filter, adding in other access-to-modification relationships.

3.3 Searching relationships

The context-based portion of a search in Connections starts with the results of a content-based search. For each file in these results, Connections performs a breadth-first graph traversal starting at the node for that file. Files touched during the traversal are added to the result-graph, a subgraph with the results for a specific search.

In the process of building the relation-graph, incorrect edges can form. For example, when a user switches context between disparate tasks (e.g., from writing personal email to examining a spreadsheet), the edges formed during the transition could be misleading. Our algorithm attempts to reduce the number of such paths introduced to the result-graph using two tunable parameters: path length and weight cutoff.

Path length: Path length is the maximum number of steps taken from any starting node in the graph. As the system follows edges further and further from an initial file, the strength of the relationship grows weaker and weaker. By limiting the path length, one reduces the number of false positives created by a long chain of edges that leads to unrelated files.

Weight cutoff: The weight cutoff specifies that an edge's weight must make up at least a given minimum percentage of either the source's outgoing weight or the sink's incoming weight. In this manner, lightly weighted edges coming from or to files with few total accesses are still followed, but only the most heavily weighted edges are followed for frequently accessed files. This limits the effects of context-switches, removing links between oft-accessed files that are rarely accessed together.

To see how the weight cutoff and the path length work together, consider the example relation-graph in Figure 2. Assume that D was the starting point for a search using a path length of 2 and a 30% weight cutoff. Connections starts by examining D and sees that the edge DB makes up only 20% of D's outgoing weight and 20% of B's incoming weight, thus it is not followed. Following edge DE, Connections repeats the procedure. In this case, although both EB and EF make up less than 30% of E's outgoing weight, they make up more than 30% of the incoming weight of nodes B and F respectively, and both are followed. Thus the result-graph would contain only edges DE, EF, EG and EB. If the path length were increased to 3, the result-graph would also contain edges BA and BC. Similarly, if the cutoff were reduced to 15%, edge DB would be followed, the result-graph would contain all of the presented edges.

[1]We also considered using a fixed number of files as the window, but in practice we found that the burstiness of file accesses made this approach perform poorly. Related files during bursts were not connected, and unrelated files were connected over long periods of idleness.

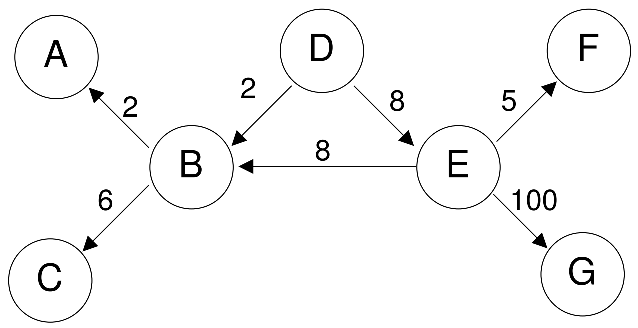

Figure 2: *Relation-graph example.* **Each of the nodes in the relation-graph map to a file in a user's system. Edges indicate related files with weights specifying the strength of the relationship. Note that the edge weights in the figure are specifically chosen for the algorithm behavior example in the text.**

3.4 Ranking results

Most modern search tools rank order results to provide their best guesses first. Connections implements three ranking algorithms: *Basic-BFS*, an algorithm that pushes weights down edges in a breadth-first manner, and two extensions to Basic-BFS based on the popular web-search algorithms HITS [20] and PageRank [5].

3.4.1 Basic-BFS

Basic-BFS 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 should match to 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.

Let N be the set of all nodes in the result-graph, and P be the path length used to generate the result-graph. 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 $w_{n_0} = 0$. Connections then runs the following algorithm 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:

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

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:

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

This sum, w_n, represents the contributions of all contextual relationship paths to node n plus the contribution of its original content ranking. The final ranking of results sorts each file from highest weight to lowest.

As an example of how the algorithm works, assume that the graph in Figure 2 is the result-graph, the path length is 2, $\alpha = 0.25$, 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:

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

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

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

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

3.4.2 HITS

The HITS algorithm attempts to locate *authority* and *hub* nodes within a graph, given a specific set of starting nodes. 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 SOSP web site), while hubs are analogous to pages with lists of links to authorities (e.g., a page with links to all ACM conference 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" [20]. These eigenvectors indicate the authority and hub probabilities for each node.

Connections implements HITS in two ways. The first implementation, HITS-original, runs an unmodified version of HITS. 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. Our evaluation in Section 4.3.3 examines both the hub and authority rankings of each implementation.

3.4.3 PageRank

PageRank is the ranking algorithm used by the Google web search engine [5]. 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*.

Within Connections, we use the Power Method [18] to calculate the PageRank of each file in the relation-graph. Unfortunately, Google's method of merging content search results with PageRank is not documented, thus we implemented three possible uses for a file's PageRank within Connections.

The first 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-BFS algorithm. The second implementation, PR-after, runs the Basic-BFS algorithm, and then applies the PageRank to the final results. The third implementation, PR-only, ignores the content rankings, and uses only PageRank to rank the files within the result-graph.

3.5 Implementation

Our prototype implementation of Connections has the three components shown in Figure 1: a tracer, a content-based search, and a relation-graph. To minimize foreground impact, only the tracer runs on the system continuously, while indexing required by content-based search and the relation-graph runs either during idle time or as background processes. The delay in indexing only affects users if they search for files created between indexing periods, a scenario that already exists with today's content-only search tools.

The tracing component sits at the system call layer in the kernel and watches user activity, tracing all file system and process management calls. Process management calls allow proper reconstruction of file descriptor activity. The tracing component is operating system specific, and Connections currently runs exclusively under Linux 2.4 kernels. Similar system call tracing infrastructure exists in other systems (e.g., Windows XP), and porting Connections should not be difficult. The performance impact of the tracing component is minimal, as with other file system tracing tools [3].

The content-based search component uses Indri [25], a state-of-the-art content analysis tool. We chose Indri because of its consistently high performance (and that of its predecessors) in several tracks of the Text REtrieval Conference (TREC) over the last few years [1, 23, 32]. TREC is an annual, competitive ranking of content-only information retrieval systems with different tracks using distinct corpora of data and queries geared toward particular retrieval tasks [42].

Connections creates the relation-graph using the algorithm described in Section 3.2 and stores it using BerkeleyDB 4.2 [33]. Connections searches the relation-graph using the algorithm described in Section 3.3 and ranks the results of the search using the algorithms described in Section 3.4.

Users specify queries in Connections as a set of keywords and (optionally) one or more file types. If file types are specified, final query results are filtered to remove other types. For example, a user searching for a copy of this paper might input the keywords *content, context* with the types *.ps, .pdf*. Connections would perform its search using *content* and *context*, and then filter the final results showing only *.ps* and *.pdf* files.

4. EVALUATION

Our evaluation of Connections has three parts. First, we evaluate the utility of Connections's context-enhanced search, comparing its precision and recall against Indri, a state-of-the-art content-only search tool; as hoped, the addition of context makes the search tool more effective. Second, we evaluate the sensitivity of the various parameters within Connections, showing that, while the settings of parameters affect search quality, using "reasonable" settings that are close to optimal is sufficient to see benefits from context. Third, we evaluate the performance of indexing and querying in Connections, finding that both space and time overheads from adding context analysis are minimal.

4.1 Experimental approach

Our evaluation compares Indri's (version 3.1.1) content-only search [25] to Connections's context-enhanced search. To compare the utility of these two systems, we borrow and adapt techniques from information retrieval [4]. Traditionally, content-only search tools are evaluated using large public corpora of data, such as archived library 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, two subtle differences make file system search, and especially context-enhanced search, more difficult to evaluate. First, 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 trace data for the desired data. Second, because the data is personal, only its owner can create meaningful queries and act as oracle for evaluating query results, especially when queries must be formed with the period of tracing in mind. Doing otherwise would render the experiment useless, since tracing would be present over the lifetime of a production system.

4.1.1 Gathering data

To gather context data, we traced the desktop computers of six computer science researchers for a period of six months. Using the traces, we generated a relation-graph for each user using the following default parameters: a 30 second relation window, a directed edge style, and a read/write operation filter.[2]

To gather content-based search results (both for the content-only system and Connections's internal use), we ran Indri over the set of all parsable document types on the users' computers (any files appearing to contain text, as well as PDF and Postscript files).

Each user submitted 3-5 queries. Table 2 lists three submitted queries as representative examples; they cannot all be listed for both space and privacy reasons. We ran queries in Indri (both alone and internally to Connections) using the "#combine()" operator. We ran Connections's relation-graph search algorithm using the default parameters of a path length of 3 and a weight cutoff of 0.1%, and used the Basic-BFS ranking algorithm with an α parameter of 0.75.

4.1.2 Evaluation

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 over the total number specified by the oracle. A system's *precision* is the number of relevant documents retrieved over the total number of documents retrieved.

Unfortunately, only the user of the system knows the data well enough to act as oracle for its queries, and our users were not willing to examine every file in their systems for each query. To account for this, we use a technique known as pooling [4] that combines the results from a number of different search techniques, generating a set of results with good coverage of relevant files. In our case, we pooled several context-enhanced searches using both broader parameter settings and the default settings being evaluated, and presented them to users. Users then chose the relevant documents from this pooled set of files to create the oracle.

[2]We chose these settings after performing the sensitivity analysis described in Section 4.3.

Query Num	Query	File Types	Description
1	osdi, background	`.ps`, `.pdf`	Papers that made up the related work of a particular paper submission
3	content, context, figure	`.eps`	Figures relating to content-based or context-based search
14	mozilla, obzerver, log	N/A	Mozilla web browsing logs generated by the obzerver tracing tool

Table 2: *Selected search queries.* **This table shows three specific user-submitted queries. Each query's search terms and file type are listed, along with an English description of the search submitted by the user.**

We 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) [4]. Examining this curve shows how well a system ranks the results that it generates. At each recall level n, the curve plots the highest precision seen between n and $n + 1$. To calculate the average recall/precision values over a set of queries, the precision of each query at a given recall level is calculated, and then averaged.

The second technique is to examine the recall and precision of each system with fixed numbers of results. 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 shows how quickly a user could locate relevant data with the system. Examining the recall and precision with an infinite result cutoff shows how many relevant results can be located using the system.

4.2 The utility of context

This section compares the recall and precision of Indri to that of Connections. First, we compare the rankings of the two systems using their recall/precision graphs. Second, we examine the interactive performance of the two systems, comparing their recall and precision at various result cutoffs. Third, we examine each of the queries in detail to get an understanding of Connections's specific strengths and weaknesses. Fourth, we present anecdotal evidence about the advantages of context-enhanced search from a user's perspective. Fifth, we discuss using another popular content-only search tool, Glimpse, in place of Indri, and the effect on search utility. Sixth, we compare automated context relationships to the relationships inherent in the existing user organization using Indri-Dir, a system that uses directories as contextual clusters.

4.2.1 Ranking performance

Figure 3 shows both the raw recall/precision data in table form, as well as a plot of the data. The most noticeable feature of this data is that Connections outperforms Indri at every recall level (as shown by its line being higher on the figure at each point). This indicates that Connections finds more relevant data (as evidenced by its high precision at high recall levels) and ranks it higher (as evidenced by its higher precision at lower recall levels) than content-only search.

Cutoff	Recall %		Precision %	
	Indri	Connections	Indri	Connections
10	13	22	23	29
20	16	29	20	25
30	18	34	17	23
50	25	40	17	21
100	28	45	17	20
inf.	34	74	15	16

Table 3: *Recall and precision at varying cutoffs averaged over 25 queries.* **This table lists the recall and precision levels of Indri and Connections at six different cutoff points. Low cutoffs show how the system performs in an interactive situation, where users request pages of results. Higher cutoffs show how the system performs when the user is trying to locate all available information on a topic.**

4.2.2 Interactive query performance

Table 3 shows the recall and precision levels of the two systems at various cutoff points. Again, the key feature of this data is that by combining content and context, Connections outperforms content-only search at every cutoff point, increasing both recall and precision.

Connections also significantly increases the total number of results found by the system. With no result cutoff (infinite), Connections increases average recall across the 25 queries by 40%. These results indicate that not only will users be more likely to find their data quickly, but that they will have a better chance of finding their data at all.

4.2.3 Individual query performance

Table 4 shows the performance of these two schemes for each of the 25 queries using a result cutoff of 1000. The most noticeable result is that, for most queries, Connections provides more correct results than content-only search with similar or better precision. To assist with interpretation, the horizontal lines partition the queries into 3 categories.

For queries 1-18, the user specified a file type. This filter reduces the number of retrieved results, improving average precision. In queries 11-14, the user specified that the file was an image (e.g., .jpg or .eps), making it much more difficult for Indri to locate relevant files. In queries 11, 12, and 13, Connections was able to leverage its contextual relationships to locate relevant images.

For queries 19-24, the user did not specify a file type. In three cases, Connections improved recall and precision. For queries 21 and 22, Connections ranked the relevant content-only results lower than Indri, resulting in lower precision. Improvements in the ranking algorithm could help Connec-

Recall %	Precision %	
	Indri	**Connections**
0	36	41
10	33	41
20	26	35
30	18	31
40	17	29
50	17	28
60	14	26
70	12	24
80	5	24
90	4	14
100	4	11
average	17	28

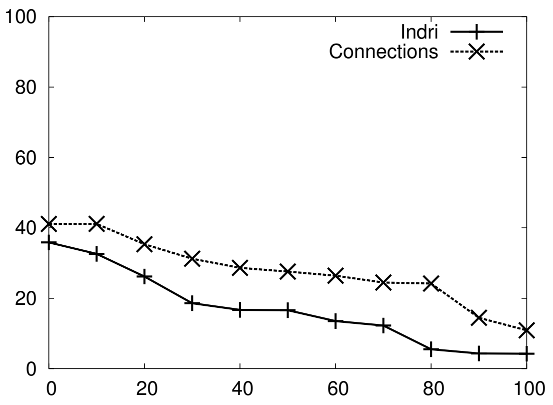

Figure 3: *Precision at 11 recall points averaged over 25 queries.* **The table on the left lists the precision of Indri and Connections at 11 different recall levels, as well as the average over all levels. This indicates how accurate the results of a system are; higher precision levels mean that more data is found more quickly by the system. The plot on the right is a graphical representation of the data in the table. A perfect system would have a line across the top at 100 for each recall point.**

tions match, if not improve upon, these queries by pushing the relevant results higher in the rankings.

For query 25, Indri's results did not exist in the relation-graph. This is a side-effect of the experimental setup; the relation-graph only contains data on files accessed during the period of tracing. If Connections was in place throughout a system's lifetime, some contextual data would exist.

Examining some of the queries where Connections was unable to improve search effectiveness provides interesting insights. For queries 22 and 24, 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).

For queries 14 and 23, the search terms had multiple meanings within the user's data. For example, in query 23, one of the words the user specified was "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 be adjusted to include some results from each cluster.

4.2.4 User satisfaction

Another important consideration for any search system is the satisfaction of the user, both with the ease of use of the system and with the provided results. Although not easily measurable, we have anecdotal evidence that indicates context-enhanced search can improve the user's satisfaction with file system search.

One improvement noted by users is that queries can be more "intuitive." For example, in query 1 (see Table 2), the keywords intuitively describe what the user is searching for, but content-based search tools are unlikely to ever provide accurate results for such a query. Often, users appear to be searching based on their context, but are instead forced to come up with content-friendly search terms.

Another improvement noted by users is the kind of results found by the system. Several users mentioned that Connections located relevant files that they hadn't remembered were on their machine. Rather than looking for a specific file that the user remembers, a user's search terms can be less directed, relying more on the search system to provide the desired data.

Although far from a scientific study of user satisfaction, such anecdotal evidence lends weight to the argument for context-based search.

4.2.5 Other content analysis tools

In exploring the utility of combining content analysis with context analysis, we also implemented a version of Connections using Glimpse [29] as the content analysis tool (Glimpse-Connections). Because Glimpse does not rank its search results (and thus neither does Glimpse-Connections), it is impossible to use any comparisons that rely on ranking, such as recall/precision curves or specific result cutoffs. Thus, recall and precision can only be compared with an infinite result cutoff.

Comparing Glimpse to Glimpse-Connections with infinite cutoff, we see results similar to those in Table 3: Glimpse has a 20% recall and 29% precision, while Glimpse-Connections has a 62% recall and 48% precision. The reduced recall and increased precision of these two systems over the Indri-based systems is due to Glimpse's strict boolean AND of all query terms, which results in fewer hits than Indri for most queries.

4.2.6 Directories as context

Traditionally, users organize their files into a directory hierarchy, grouping related files together. As such, it might seem that using these groupings as contextual relationships could provide many of the same benefits as Connections; however, in practice it does not. To explore this possibility, we built Indri-Dir, a tool that uses the directory structure to enhance search results. Specifically, Indri-Dir looks in

Category Description	Query Num	Indri				Connections			
		Total	Correct	Recall %	Precision %	Total	Correct	Recall %	Precision %
Typed queries	1	14	0	0	0	40	11	100	28
	2	8	0	0	0	30	2	100	7
	3	40	8	62	20	59	13	100	22
	4	39	3	50	8	58	4	67	7
	5	40	3	30	8	59	8	80	14
	6	116	53	71	46	138	64	85	46
	7	111	76	71	68	134	87	81	65
	8	165	55	72	33	187	65	86	35
	9	345	0	0	0	380	13	87	3
	10	2	2	25	100	31	5	63	16
	11	(1000)	0	0	0	18	16	100	89
	12	(1000)	0	0	0	27	9	100	33
	13	(445)	0	0	0	58	1	100	2
	14	(1000)	0	0	0	15	0	0	0
	15	(1000)	0	0	0	1	0	0	0
	16	11	0	0	0	1000	0	0	0
	17	47	13	81	28	1000	13	81	1
	18	23	7	100	30	36	7	100	19
Untyped queries	19	956	2	1	0	1000	42	13	4
	20	934	26	41	3	1000	28	44	3
	21	786	327	37	42	1000	354	40	35
	22	756	14	100	2	1000	14	100	1
	23	231	1	100	0	1000	0	0	0
	24	65	0	0	0	1000	0	0	0
No data available	25	(6)	0	0	0	(6)	0	0	0

Table 4: *Query result details at 1000 result cutoff.* **For the two search systems, this table shows: (1) the total number of results presented to the user, (2) from those, the total number of correct results, (3) the recall of the system, and (4) the precision of the system. When no files of the requested type are found, the number of files located before filtering is listed in parenthesis.**

the directories of content-based results for files of the requested type, assigning these files the combined weight of any content matches in the directory. If no type is specified, Indri-Dir adds all files in the directory assigning them the weight of the highest ranked content result in that directory.

Indri-Dir significantly underperforms both Connections and Indri on all metrics. The reason for this is two-fold. First, Indri-Dir relies on users organizing their files into directories in contextually meaningful ways, but many users have too many files to do this effectively (.e.g., cluttered home directories, download folders, "paper" directories, etc.). Second, Indri-Dir relies on a directory's organization to match the context of the user's search, but users often organize files in one way and then use them in another. For example, a user might download all of the proceedings for a particular conference into a single directory, but later find that one particular paper is of use in their project. Rather than finding other papers related to the project, Indri-Dir will find other papers from that conference.

4.3 Sensitivity analysis

To understand the sensitivity of different parameter settings, we examined a wide variety of parameter configurations for each of the three phases of context search. For space considerations, we present a subset of the results using three queries (those listed in Table 2) that represent the space. For each query, we present a recall/precision graph, like that shown in Figure 3. In each set of graphs, we examine the sensitivity of a single parameter, using the default settings for all other parameters.

4.3.1 *Identifying relationships*

Relation window: Figure 4 presents the recall/precision curves for Connections configured to use each of five different relation window sizes: 10, 30 (our default), 60, 120, and 300 seconds. These graphs illustrate that a larger window size tends to reduce precision. The increase in links at each node results in the weight cutoff removing some accurate links. However, as shown in query 1, too small of a window can result in missing some relationships due to edges not being formed.

Edge style: Figure 5 presents the recall/precision curves for Connections configured to use either directed (our default) or undirected edge styles. In almost every case, the directed edge style outperforms the undirected edge style. The reason for this is nuanced. Within the traces, there are many misleading input files that are related to many files (e.g., .bashrc or .emacs). Adding these as output files significantly increases the number of outgoing edges at each node, causing the weight cutoff to remove some relevant edges. Although these misleading edges may not be followed, cutting the additional edges removes paths that would have otherwise located relevant files.

Operation filter: Figure 6 presents the recall/precision curves for Connections configured to use each of three operation filters: read/write (our default), open, and all-ops. Across all queries, the open filter performs poorly; its increased number of edges result in many incorrect relationships being followed. In cases where the user specified a type (such as queries 1 and 3), the all-ops and read/write filters perform similarly. However, in untyped queries (such as query 14), the all-ops filter provides lower precision. The

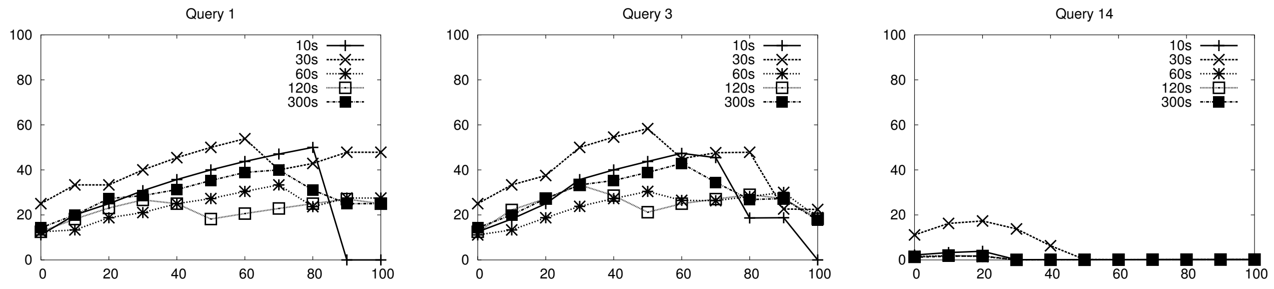

Figure 4: *Sensitivity analysis of Connections using five different relation window sizes.*

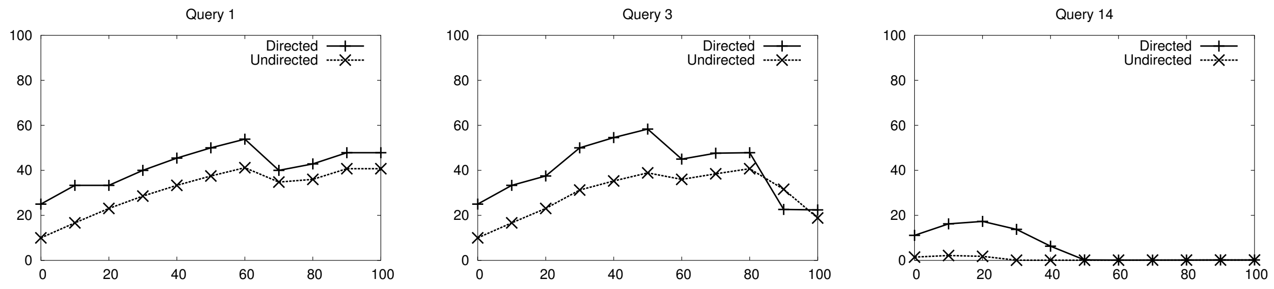

Figure 5: *Sensitivity analysis of Connections using the two different edge styles.*

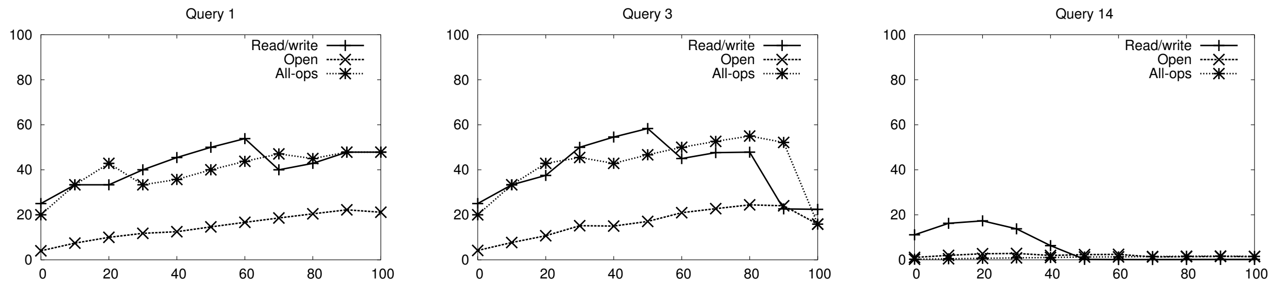

Figure 6: *Sensitivity analysis of Connections using three different operation filters.*

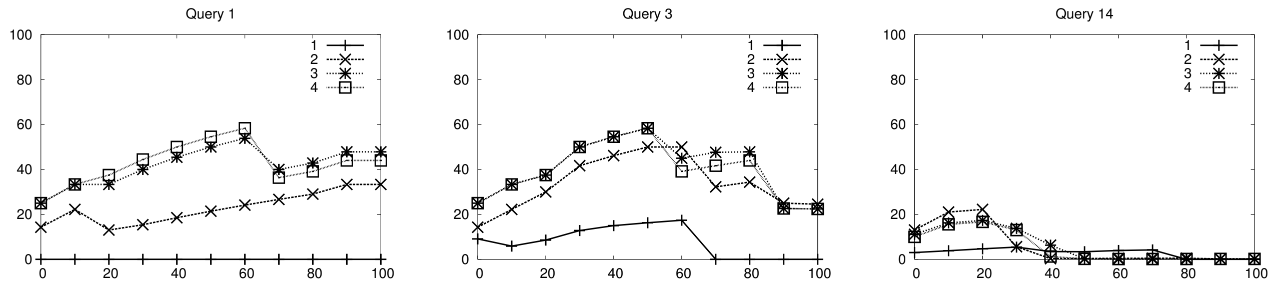

Figure 7: *Sensitivity analysis of Connections using four different path lengths.*

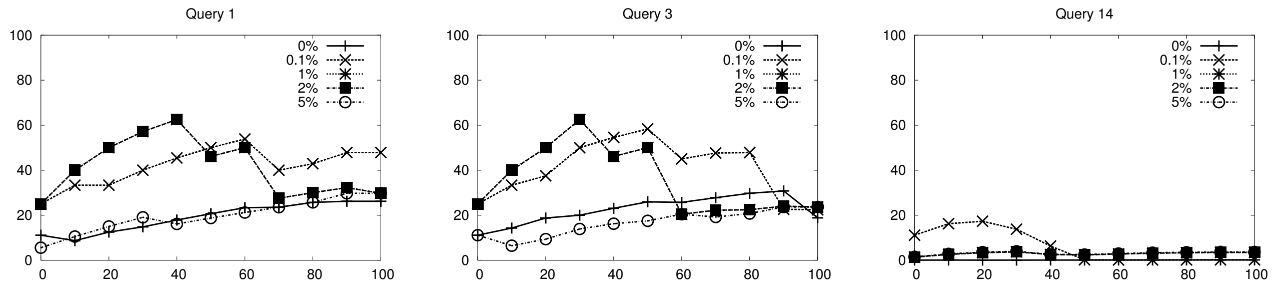

Figure 8: *Sensitivity analysis of Connections using five different weight cutoffs.*

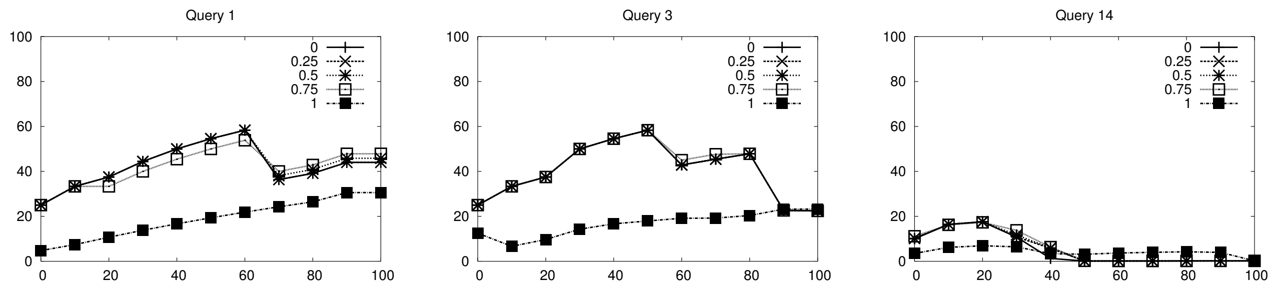

Figure 9: *Sensitivity analysis of Connections using five different α settings.*

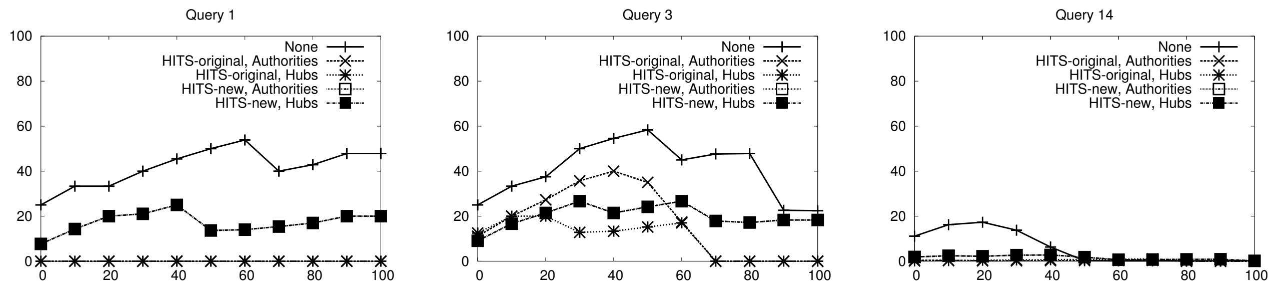

Figure 10: *Sensitivity analysis of Connections using five different HITS methods.*

discrepancy is because the user specified type filter hides most of the added, incorrect edges generated by the all-ops scheme.

4.3.2 Searching relationships

Path length: Figure 7 presents the recall/precision curves for Connections configured to use each of four different path lengths: 1, 2, 3 (our default), and 4. In most queries, a path length of 1 found few relevant files, while higher path lengths all perform quite well. Too high of a path length would reduce precision, except the ranking algorithm places nodes that are too far away on the path lower in the rankings.

Weight cutoff: Figure 8 presents the recall/precision curves for Connections configured to use each of five different weight cutoffs: 0%, 0.1% (our default), 1% , 2%, and 5%. At high cutoff levels, the recall suffers as relevant links are cut. With no cutoff, many incorrect paths are followed, reducing precision.

4.3.3 Ranking results

Basic-BFS: Figure 9 presents the recall/precision curves for Connections configured to use each of five different α parameters: 0, 0.25, 0.5, 0.75 (our default), and 1. The α parameter sets the tradeoff between individual link weight and the existence of a link. A setting of 1 specifies that only the link weight should be considered, acting as a cutoff for lightweight links in the result-graph. Other settings of α perform similarly, with 0.75 performing slightly better than the others on average. This indicates that, after creating an accurate result-graph, the individual link weights are of less importance.

HITS: Figure 10 presents the recall/precision curves for Connections configured to use each of five different HITS implementations: the default scheme with no HITS rankings, and the authority and hub rankings of both HITS-original and HITS-new. These results illustrate the difficulty of using web-based graph algorithms on temporal locality graphs.

Most web-based algorithms focus on finding either the authority or hub for a topic, however, these algorithms rely on links being a valid indicator of context. Because Connections's result-graph is only an approximation of context, many of its links are erroneous. For example, a file with many outgoing and incoming links in the result-graph (such as ".bash_history") is neither an authority nor a hub in terms of accuracy of links, however shows up as both within HITS. As a result, HITS-new performs poorly compared to Basic-BFS. HITS-original's limited result-graph further detracts from its accuracy.

PageRank: Figure 11 presents the recall/precision curves for Connections configured to use each of 4 different PageRank implementations: the default scheme with no use of PageRank, PR-before, PR-after, and PR-only. Because PageRank is a measure of a node's authority in the graph, the PageRank schemes have similar difficulties to the HITS-based algorithms. Of the three, PR-after performs best, because the highest ranked nodes all have similarly low PageRank values, resulting in only a shifting of the top-ranked results. Because Basic-BFS's rankings are additive in nature, PR-before generally performs worse than PR-only.

4.3.4 Sensitivity analysis summary

When building the relation-graph and result-graph, each parameter has settings that either increase or decrease the number of paths in the graph. Unsurprisingly, then, there is a mid-range of settings for each parameter that provides the best recall/precision trade-off. Erring on the side of too few paths results in reduced recall, while erring on the side of too many paths results in reduced precision.

Fortunately, our results indicate that this mid-range of settings is sufficiently wide that perfect tuning is not crit-

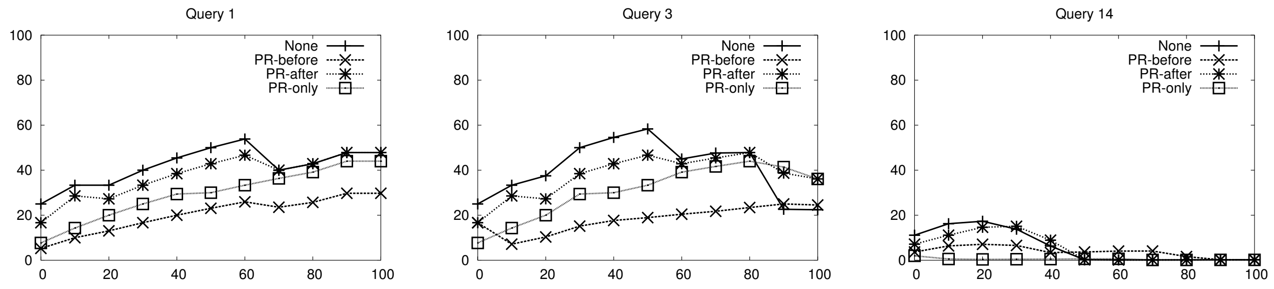

Figure 11: *Sensitivity analysis of Connections using four different PageRank methods.*

ical to the success of context-enhanced search. Using almost any set of "reasonable" parameters, Connections outperforms content-only search.

When ranking results, it is important to consider that, although the result-graph has been pruned, it still contains misleading connections. For this reason, the Basic-BFS ranking algorithm outperforms algorithms that excel in web search.

4.4 System performance

Connections demonstrates that combining content and context in file search improves both recall and precision. But few users will utilize such a system unless (1) their foreground work is not impacted by the required indexing, (2) the additional storage space required by the index does not exceed reasonable bounds, and (3) their queries are answered quickly. Although Connections's implementation is not tuned, our analysis indicates that all three requirements can be satisfied.

We ran all timing experiments using an Intel-based Pentium 3 1 Ghz processor with 384 MB of RAM and a Quantum Atlas 10K 9 GB disk drive.

4.4.1 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.

With context analysis, there is no startup cost, since no record of temporal relationships exists initially. We measured the incremental costs as the time required to merge each day of traces into that machine's relation-graph. The average indexing time per day was 23 seconds with a standard deviation of 38 seconds. The longest indexing time observed was 702 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 15 minute per-day indexing time is likely low enough to run in the background with little impact on foreground work.

4.4.2 Indexing space

Connections's index consists of both a content-only index and a relation-graph. Because the space overheads of content-only indexing are well-known [32], we examine the additional space required by the relation-graph. To do so, we incrementally added each day of tracing to the relation-graph for each of the six machines and measured the total size of the relation-graph after each month.

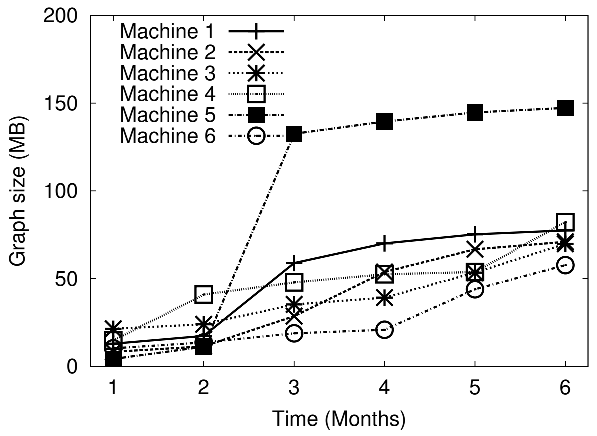

Figure 12: *Relation-graph size.* **This figure illustrates the cumulative size of the relation-graph after each month of tracing for each of the five machines in our experiments.**

Figure 12 shows the space required by the relation-graph after each month of tracing for each machine. These results indicate that relation-graph size is not a barrier to context-enhanced search. First, the final graph sizes are quite small, with none requiring more than 150 MB. On average, the index size was less than 1% of the size of the user's data set. Second, the graphs appear to grow linearly over time, for the most part matching the slow growth in user data set size. Although the worst-case for the graph is $O(N^2)$ growth, this would almost never be seen in practice because, while the total number of files in a user's system grows over time, the user's working set stays relatively constant. Edges form between constant-sized clusters of files, and as the working set shifts, the relation-graph grows at a linear rate. Only when the user drastically changes their context does the graph grow significantly in size. An example of this is seen during month 3 of Machine 5, where the user began work on a new piece of software.

4.4.3 Querying performance

Connections's average query time across all queries is 2.62 seconds, a combination of the content search (0.98 seconds) and context search (1.64 seconds)[3]. While higher than con-

[3]Query time averages are calculated across 15 runs of each query, and all standard deviations are within 1% of the mean.

tent search alone, we believe that the query times of Connections are such that users will still find the system usable.

4.5 Summary

Connections demonstrates the benefits and flexibility of combining content and context in file search. When compared to Indri, a state-of-the-art content analysis tool, Connections improves both average recall and average precision. Although parameter tuning could improve the results further, simply finding settings that are "good enough" provides most of the benefit, meaning that even non-tuned systems will perform well. Finally, the performance overheads of adding context analysis to existing file system search systems are low, even in a prototype such as Connections.

5. DISCUSSION

While designing Connections, we focused on highlighting the significant improvements that can be garnered from combining content analysis with context analysis. This section briefly discusses other considerations that could lead to further improvements in Connections.

Application assistance: Beyond temporal relationships, there are a number of other ways that the system could assign context to files. For example, many applications (e.g., e-mail) could provide context clues to the system for the files they use (e.g., sender, time, subject). Other context clues could include the user's physical location, current schedule, visible windows, etc.

Per-user settings: Different users manage and use their data in different ways. These differences mean that Connections may provide even greater benefits if the algorithm parameters are tuned for individual users. If true, one interesting question is if Connections could determine the correct parameter settings automatically, starting with broad settings, and eventually honing in on the correct values.

Deleted files: Because the relation-graph is built using trace data, over time the graph contains files that the user has deleted. Although Connections currently leaves these in place, should deleted files eventually be pruned from the relation-graph? If left in, should they be shown to users, potentially curbing needless searches for data that they have deleted, or will the increased results reduce search effectiveness? These questions merit further study.

Organizing results: As mentioned at the end of Section 4.2.3, Connections could divide results into categories based on disjoint sets in the relation-graph (e.g., disjoint uses of the word "training"), allowing users to quickly hone in on the desired set of files, even with large numbers of search results. This organization could also help the ranking algorithm by adjusting the rankings of disjoint result sets to include some results from each set.

Network storage: In many settings, it is common for users to store some of their files remotely using networked storage. One of the key advantages of such an approach is that the data can then be accessed from any of a variety of machines. This causes problems for Connections because the access patterns for a particular piece of data are now spread across the remote machines. Examining how this information could be centralized or collated will be important as mobile devices shift user data toward this usage model.

Personalization: Web and file search researchers have found that search accuracy is improved when results can be targeted to a user's current context. We believe that Connections is uniquely targeted to this task. By examining recently accessed files, Connections could use the relation-graph to identify the set of files that make up the user's current context, further targeting the search.

File system interaction: Currently, the only component of Connections that interacts directly with the file system is the tracer. As a result, Connections relies on both the existence of complete traces (such that no system state is lost) and its ability to perfectly reconstruct system state from these traces (a process that takes up most of the indexing time for Connections). Connections also uses the filename as a tag to the file's relation information, which can be faulty (e.g., if a user renames a file, or has multiple paths to the same file). By moving Connections inside the file system, these problems could be alleviated. Additionally, a Connections-like system within the file system could be used to assist with, or even automate, file organization by grouping related files into virtual directories, automatically assigning attributes from related files, and so on.

6. CONCLUSIONS

As individual data sets grow, search and organizational tools grow increasingly important. This paper presents Connections, a search tool that combines traditional content analysis with contextual relationships identified by temporal locality. We show that Connections improves both average recall and average precision over a state-of-the-art content-only search system. The results demonstrate that context information is an important complement to content analysis for file search tools.

7. ACKNOWLEDGMENTS

We thank the members and companies of the PDL Consortium (including APC, EMC, Equallogic, Hewlett-Packard, Hitachi, IBM, Intel, Microsoft, Network Appliance, Oracle, Panasas, Seagate, and Sun) for their interest, insights, feedback, and support.

8. REFERENCES

[1] 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.

[2] 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.

[3] A. Aranya, C. P. Wright, and E. Zadok. Tracefs: a file system to trace them all. Conference on File and Storage Technologies, pages 129–145. USENIX Association, 2004.

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

[5] 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.

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

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

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

[9] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole Jr. Semantic file systems. ACM Symposium on Operating System Principles, pages 16–25. ACM Press, 1991.

[10] Google, http://www.google.com/.

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

[12] 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.

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

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

[15] T. H. Haveliwala. Topic-sensitive PageRank: a context-sensitive ranking algorithm for web search. *IEEE Transactions on Knowledge and Data Engineering*, **15**(4):784–796. IEEE, August 2003.

[16] B. Hayes. Terabyte territory. *American Scientist*, **90**(3):212–216, May – June 2002.

[17] 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.

[18] 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.

[19] J. J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, **10**(1):3–25. ACM Press, February 1992.

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

[21] 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.

[22] G. H. Kuenning. *Seer: predictive file hoarding for disconnected mobile operation*. Technical Report UCLA–CSD–970015. University of California, Los Angeles, May 1997.

[23] L. S. Larkey, J. Allan, M. E. Connell, A. Bolivar, and C. Wade. UMass at TREC 2002: cross language and novelty tracks. Text Retrieval Conference, 2002.

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

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

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

[27] T. M. Madhyastha and D. A. Reed. Intelligent, adaptive file system policy selection. Frontiers of Massively Parallel Computation, October 1996.

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

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

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

[31] D. Metzler and W. B. Croft. Combining the language model and inference network approaches to retrieval. *Information Processing and Management: an International Journal*, **40**(5):735–750. ACM Press, September 2004.

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

[33] M. A. Olson, K. Bostic, and M. Seltzer. Berkeley DB. Summer USENIX Technical Conference. USENIX Association, 1999.

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

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

[36] 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.

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

[38] 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.

[39] J. Teevan, S. T. Dumais, and E. Horvitz. Personalizing search via automated analysis of interests and activities. ACM SIGIR Conference. ACM, 2005.

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

[41] H. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, **9**(3):187–222. ACM, July 1991.

[42] E. M. Voorhees. Overview of TREC 2003. Text Retrieval Conference. NIST, 2003.

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

[44] X1 Desktop Search, http://www.x1.com/.

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