HyPursuit: A Hierarchical Network Search Engine that Exploits Content-Link Hypertext Clustering

Ron Weiss, Bienvenido Vélez, Mark A. Sheldon
Chanathip Namprempre, Peter Szilagyi, Andrzej Duda, David K. Gifford
Programming Systems Research Group
MIT Laboratory for Computer Science
545 Technology Square, Cambridge, MA 02139
USA
Tel: 1-617-253-6264
E-mail: rweiss@lcs.mit.edu

ABSTRACT

HyPursuit is a new hierarchical network search engine that clusters hypertext documents to structure a given information space for browsing and search activities. Our *content-link* clustering algorithm is based on the semantic information embedded in hyperlink structures and document contents. HyPursuit admits multiple, coexisting cluster hierarchies based on different principles for grouping documents, such as the Library of Congress catalog scheme and automatically created hypertext clusters.

HyPursuit's *abstraction functions* summarize cluster contents to support scalable query processing. The abstraction functions satisfy system resource limitations with controlled information loss. The result of query processing operations on a cluster summary approximates the result of performing the operations on the entire information space. We constructed a prototype system comprising 100 leaf World Wide Web sites and a hierarchy of 42 servers that route queries to the leaf sites. Experience with our system suggests that abstraction functions based on hypertext clustering can be used to construct meaningful and scalable cluster hierarchies. We are also encouraged by preliminary results on clustering based on both document contents and hyperlink structures.

KEYWORDS: Network Resource Discovery, Hypertext Clustering, Hyperlink Structures.

INTRODUCTION

The World Wide Web's vast collection of servers can be viewed as a distributed hypertext database containing a wealth of information. Searching this information space is a difficult problem due to the size and diversity of the data it contains. The general search problem spans a spectrum of activities ranging from a well-defined search for a specific document to a non-specific desire to understand what information is available. To support all these activities, a system must impose some semantic organization on the information space. To organize the information space well, the system must use all available knowledge about the data. In a hypertext environment, this includes both document content and hyperlink structures.

The HyPursuit prototype is a scalable system that uses *content-link* hypertext clustering, based on document contents and link information, to structure the information space and to support the entire range of search activities. Content-link clustering automatically computes sets of related documents called *clusters*. HyPursuit admits multiple, coexisting cluster hierarchies based on different principles for grouping documents, such as the Library of Congress catalog scheme and institutional structures. These hierarchies may be constructed automatically or manually.

Clusters are important for two reasons:

- Clusters can be used to group hypertext nodes into more complete documents that can be searched or combined into larger clusters. In hypertext environments, hypertext nodes are generally not independent documents. This is especially true on the Web where authors create many small pages, rather than single monolithic documents. Authors are motivated to create small pages to keep retrieval latencies low. This also permits retrieval of only relevant information.
- Clusters organize an information space for the user and the system by grouping related subspaces together. Subspaces may be clusters of documents or clusters of clusters. The partitioning of the information space provides convenient abstraction barriers for both the user and the system. The cluster abstraction allows a large information space to be treated as a unit, without regard for the details of its contents. A user exploring the portion of the information space relating to biology may

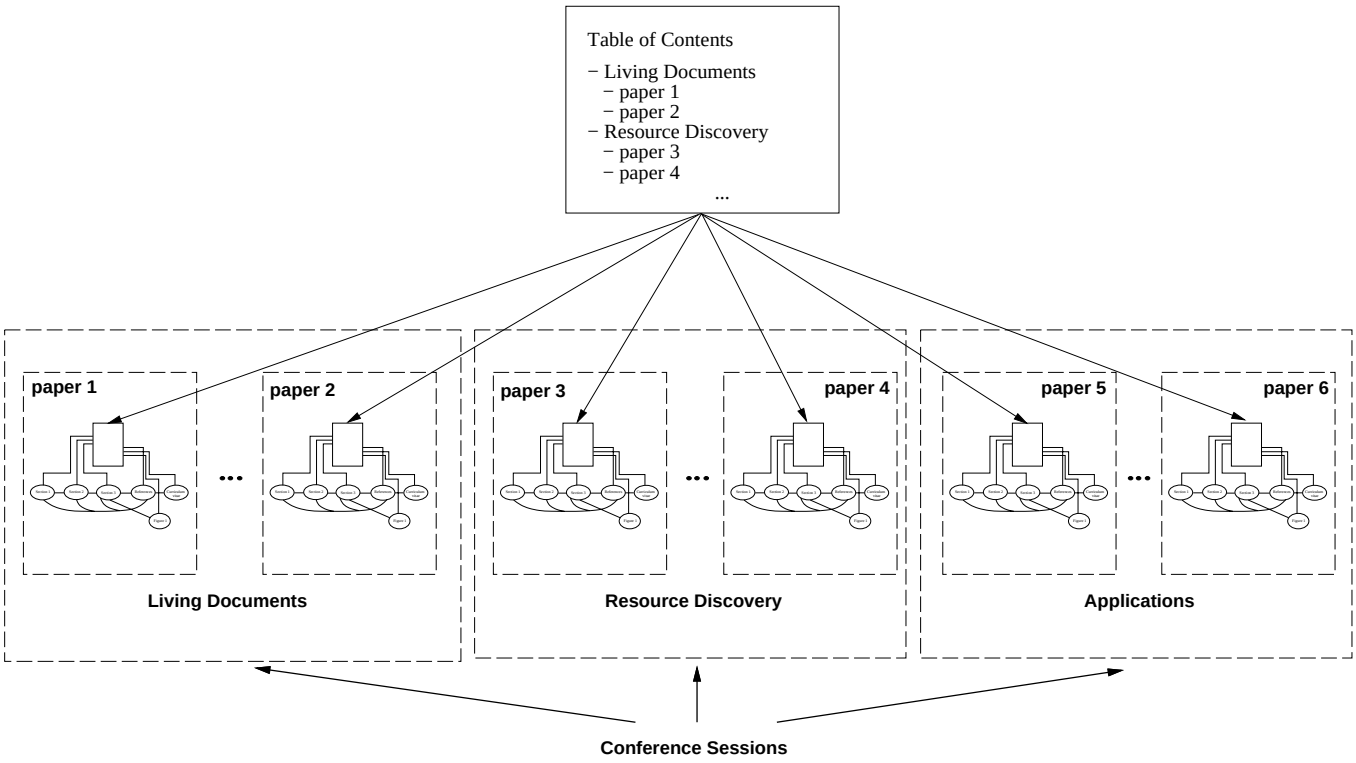

Figure 1: Content-Link Hypertext Clustering Example

want to identify all clusters (not all documents) that are related to DNA computation. Thus, the user may interact with the system at a level of granularity that is appropriate to the specificity of the information need and the complexity of the information space. Clusters also provide convenient units for the partitioning of work and resource allocation among the distributed components of the system. For example, a separate information server on a separate host may represent each individual cluster, performing operations on its local data.

HyPursuit is the first system known to the authors that combines information about document contents and hyperlink structures to cluster documents. Most previous approaches to hypertext clustering have focused on using link structures alone to group nodes. Most traditional information retrieval clustering techniques focus on the text of the individual documents. Because hyperlinks and document contents provide valuable semantic information, we hypothesize that incorporating both will improve document clustering. The strength of the relationship between documents in HyPursuit is proportional to the number of terms, ancestors, and descendants in common, as well as the number of direct links between the documents.

Figure 1 provides an example of a possible hypertext organization of conference proceedings stored on a web site. The site includes a Table of Contents page that points to the cover page of every paper in the conference proceedings. Each paper is organized according to some general guidelines, but the exact configuration and the number of the nodes across the papers varies depending on the number of sections, number of figures and number of authors. One clustering approach would be to first group together all nodes that comprise individual papers. The clustering algorithm can rely solely on the link information to reconstruct the papers. However, any further clustering that attempts to cluster the papers into related groups, such as the sessions in which the papers were presented, must exploit the content information in the nodes because the graph structure alone does not reveal the grouping implied by the sessions. The links to the papers from the Table of Contents were placed in separate lists to reflect the session structure. Therefore, the clustering algorithm could have grouped together papers that were in the same list in the HTML document. However, in general, the clustering algorithm must rely on term similarities between the clusters in order to perform better clustering.

To support scalable query processing, HyPursuit uses manageable summaries of cluster contents, called *content labels*, to approximate complete knowledge of the information space. A *manageable* summary refers to a data structure that satisfies the given resources limitations. HyPursuit's *abstraction functions* compute the content labels, which are then transmitted up the cluster hierarchy as input for the abstraction functions of higher-level clusters. An abstraction function summarizes the contents of a cluster in support of system operations while controlling information loss to satisfy the resource limitations of a particular information server.

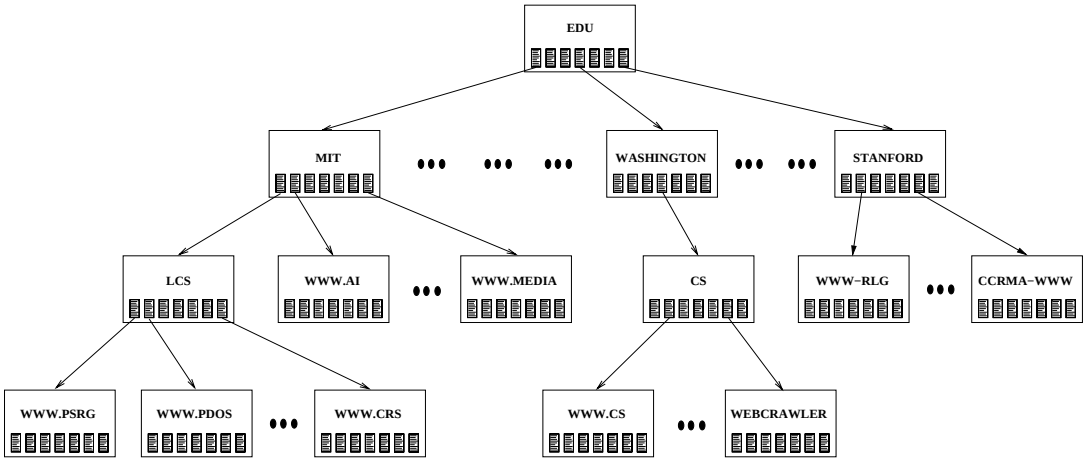

Figure 2: Content Routers Organization

For example, one abstraction function in the HyPursuit prototype ranks terms from a cluster's index structures and extracts highly ranked terms to identify appropriate clusters for particular queries. In the event that storage is limited, lower ranked terms may be dropped from a summary. The result of performing an operation on a content label approximates the result of performing the operation on the entire information space described by the content label.

HyPursuit uses content-link clustering to provide cluster-based information *browsing*, scalable *query refinement*, *result set expansion*, *query routing* and *result set clustering*. The HyPursuit interface allows users to browse the information space by traversing the cluster hierarchy, and retrieving documents as well as content labels. HyPursuit provides query refinement by dynamically computing and suggesting recall- and precision-enhancing terms for a given user query to help guide the user in further query formulation. To improve recall, HyPursuit expands a query result set with additional relevant documents that do not match the query but which are clustered with the query-selected documents. To support a variety of query processing operations, HyPursuit uses *query routing* to identify relevant clusters, forward queries to the information servers for those clusters, and merge the results.

We have built an experimental HyPursuit prototype comprising 100 Web sites organized in a four-level hierarchy of 42 content routers. The hierarchy, shown in Figure 2, is constructed to reflect the structure of the domain name system (DNS) [17]. Experience with this configuration suggests that hierarchical clustering provides a valuable discovery service to end users, and our data supports the ability of the system to scale with modest increases in content label sizes.

In the remainder of this paper, we review related work, describe the design of HyPursuit, introduce content-link hypertext clustering, discuss our experimental system, provide experience and performance statistics, and offer our conclusions and directions for future work.

RELATED WORK

Related work can be classified into the following categories: *hierarchical network search engines*, *centralized network search engines*, *subject-based directories*, *query refinement* and *hypertext clustering*.

Hierarchical network search engines address issues of scalability in terms of storage requirements and network communication for the task of resource discovery. The three examples of hierarchical search engines that are known to the authors include Harvest [4], GLOSS [14] and our work on content routing [8, 22, 21]. These systems reduce storage requirements at any of the distributed components by generating succinct descriptions (*content labels*) of the contents of leaf servers. The succinct descriptions allow the resource discovery systems to selectively access manageable sets of information providers that are believed to contain information relevant to the user's needs. None of these systems exploit hypertext clustering to provide additional information retrieval services similar to the ones HyPursuit provides.

Discover [21] implements *query refinement* using a refinement database that consists of WAIS document headlines. Discover suggests additional refinement terms for a given query based on term collocation in the document headlines. This requires every content router to keep term collocation information on a per document basis. HyPursuit supports query refinement based on term collocation on a per cluster basis. In addition, HyPursuit scans entire documents for term information.

GLOSS [14] uses a probabilistic scheme to predict the size of query result sets for each subsidiary server and forwards queries to those servers likely to have matching documents. The prediction is based on a histogram of the occurrences of words within a server. GLOSS's estimates are not accurate because they rely on the assumption that terms appear *independently* of other terms in documents. GLOSS offers several alternatives for us-

ing the histogram data, but it does not support query refinement.

Centralized Network search engines such as the Web Crawler [18], WWW Worm, ALIWEB [16], and the RBSE Spider [9], gather information about resources on the Web for query-based access. However, these systems are not scalable because they use a global indexing strategy, *i.e.,* they attempt to build one database that indexes everything. They do not provide any organization of the search space, and they do not give the user any guidance in query formulation. These systems overburden network resources by transmitting entire documents, rather than the index data, or better still, content labels. Furthermore, a HyPursuit system allows greater autonomy to each information provider and content router to tailor its indexing mechanisms and facilities using local knowledge. Veronica [15] is a discovery system that maintains an index of document titles from Gopher [1] menus, and it suffers from the same limitations as the Web search systems. HyPursuit provides a coherent framework for the integration of diverse centralized search engines.

Subject-based directories of information, *e.g.,* Planet Earth[1], Yahoo [2], and the NCSA Meta-Index[3], provide a useful browsable organization of information. These would be useful paradigms for organizing particular hierarchies in a HyPursuit System. As the information content grows, it becomes cumbersome to browse them and discover relevant information in these systems without query routing and refinement. At present, these systems are rather *ad hoc* and static, requiring manual update and maintenance. Yahoo [10] classifies documents manually and supports content-based access to the collection of documents gathered from either users' submissions or web robots.

Document clustering has been previously studied as a mechanism to improve both searching and browsing. Salton [19] presents a summary of recent document clustering techniques that are used to improve collection *searching*. Scatter/gather [7] dynamically clusters collections of documents for *browsing* large information spaces. It presents summaries of clusters to the user, who can then select a subset of these clusters for further reclustering. The selected clusters are scattered into the original documents and re-clustered. The new clusters reveal their contents in more detail, since the total number of re-clustered documents is reduced. Unlike scatter/gather, HyPursuit defines a framework for information retrieval services, such as query routing and refinement, in a hierarchy of servers.

Botafogo [2] proposes a *hyperlink-based document clustering* algorithm based on k-edge-components to generate clusters of hypertext documents. The similarity between hypertext nodes is proportional to the number of independent paths between them. Botafogo *et al.* [3] use biconnected components and strongly connected components for hypertext clustering. First, the hypertext is converted into an undirected graph by adding links. Then, edges adjacent to so called reference and index nodes are removed. The algorithm then finds biconnected subgraphs and the entire process is recursively applied to each resulting subgraph until no more biconnected components can be isolated. Each final biconnected component becomes a cluster. Neither approach uses term information or implements any information retrieval service exploiting hypertext clustering.

[12] combines terms and hyperlinks to rank nodes that match a query in a hypertext document. In contrast, HyPursuit uses terms and hyperlinks to cluster large collections of hypertext documents. The strategy proposed by [12] represents a promising paradigm for ranking query results that may be incorporated in future implementations of HyPursuit. However, the algorithms require modification to handle arbitrary hyperlink structures such as cyclic graphs. Also, the algorithm may not be suited to process queries in very large, distributed collections because it requires the dynamic propagation of weights for every node in the hierarchy on every query.

DESIGN OF HYPURSUIT

HyPursuit is a new content routing system prototype that takes advantage of content-link hypertext clustering to provide cluster-based information browsing, scalable query refinement, result set expansion, as well as query routing. This section discusses the hierarchical organization of a HyPursuit system and explains how the content routing system architecture provides a framework for multiple coexisting cluster hierarchies in a distributed network resource discovery environment. Then, it describes the abstraction of information spaces into manageable data sets in order to provide scalable services. Finally, the section describes the services supported by HyPursuit together with their corresponding user interfaces.

Hierarchical Organization of a HyPursuit System

HyPursuit's architecture admits multiple, distributed, coexisting cluster hierarchies in a network resource discovery environment. Individual cluster hierarchies are useful for browsing and searching large document collections [5, 7] because they organize the information space. Because no single organization can meet all user needs, HyPursuit supports arbitrary cluster topologies, including multitrees[13]. Users browse through a hierarchy and perform searches that exploit its organizational structure. Each hierarchy corresponds to a method of grouping related documents into clusters. Leaf nodes within the hierarchy are single documents, and interior nodes correspond to clusters of documents and clusters of clusters. The clustering method depends on the context and the motivation for the organization of the hierarchy. For example, an article by author Smith about video databases can be grouped with articles about video databases, or with articles that Smith authored.

[1] `http://white.nosc.mil/info.html`

[2] `http://www.yahoo.com`

[3] `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/MetaIndex.html`

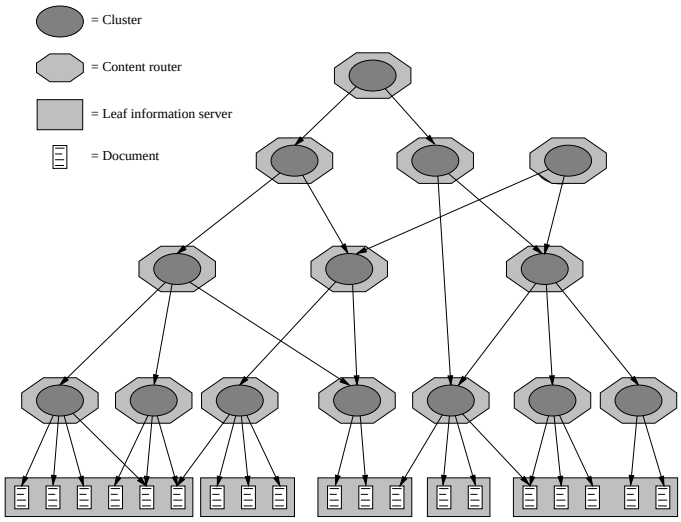

Figure 3: Content Routing as Cluster Hierarchies

The same condition exists in higher levels of the cluster hierarchy. For example, documents can be clustered based on institutional boundaries or based on Library of Congress catalog subjects.

As shown in Figure 3, HyPursuit organizes the information space into a cluster-based hierarchy. Leaf nodes are the documents of the information space, and access to leaf nodes is via leaf information servers. Internal nodes contain clusters of related documents (and other clusters), and access to internal nodes is provided by content routers. Content routers support a number of information retrieval services including query forwarding to relevant information providers, query refinement, result set expansion, browsing of content label summaries and clustering of result sets. Content routers can index and return pointers to leaf documents that reside on leaf information servers such as WAIS, Gopher and World Wide Web sites.

In a hierarchical network search engine, it is more efficient to store a document and its associated index structures in close physical proximity. Therefore, content routers that index leaf documents should be close to, if not on, the same host computer as the documents. This will ensure that the indexing process consumes less network bandwidth. In addition, if abstraction functions (see below) produce summaries that approximate the contents of the information space, then minor modifications to the contents do not necessarily result in changes to content labels. Thus reindexing operations remain local until changes are significant.

Abstraction Functions Summarize Information Spaces

To support operations like query processing in a scalable way, HyPursuit uses manageable summaries of cluster contents, called *content labels*, to approximate complete knowledge of the information space. HyPursuit's *abstraction functions* compute these content labels, which are then transmitted up the hierarchy as input for the

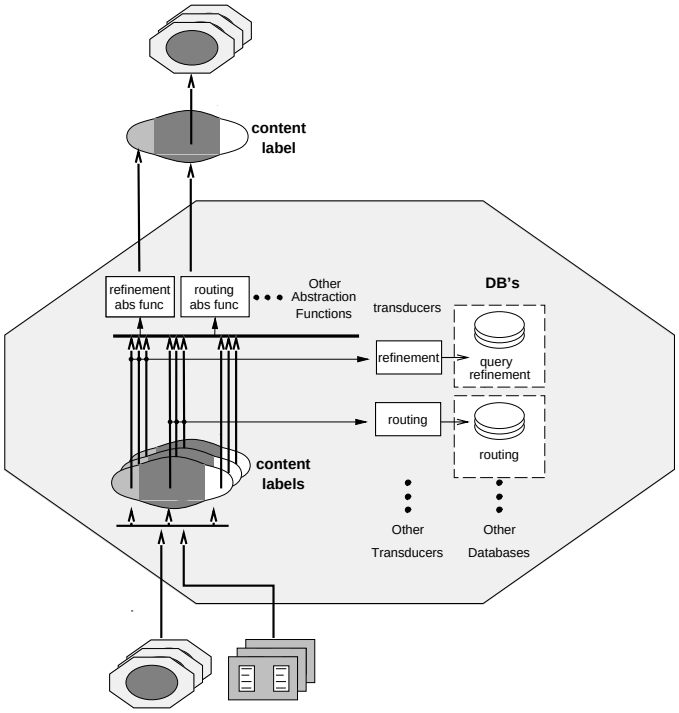

Figure 4: Abstraction Functions

abstraction functions of content routers that contain higher-level clusters. An abstraction function summarizes the contents of a content router's cluster in support of a system operation while controlling information loss to satisfy the resource limitations of a particular information server. The result of performing an operation on a content label approximates the result of performing the operation on the entire information space described by the content label.

Each service provided by a router may require a different view of the information space contained by the router. For example, query refinement may use probabilities of term collocation in documents, whereas query routing may require information about whether documents that satisfy a particular query exist. Different services may also use the same abstraction function.

Each content router uses its abstraction functions to compute a content label that summarizes its associated cluster. Figure 4 illustrates how information flows up the hierarchy as content labels consisting of information generated by the abstraction functions. A content router uses the content labels of its children to support the information retrieval services it offers. When a child of a content router is a document, then the content label for that document is computed from the document's contents (including outgoing links). To construct its own content label, a content router applies its abstraction functions to the content labels generated by its children rather than to the entire information space. This eliminates the need to transmit very large data sets to higher level content routers. In addition, a content

router can compute and transmit a different content label to each of its parents, based on the requirements of that parent.

The following paragraphs describe the three abstractions functions used in the HyPursuit prototype:

• HyPursuit's abstraction function for *query refinement* computes a data structure that enables the system to assist users in formulating queries. This abstraction function summarizes the content router's cluster as a set of sub-clusters. The resulting data structures consist of a manageable set of sub-clusters that represent groupings of related documents in the information space that is reachable from a content router. The sub-clusters do not necessarily correspond to the organizational structure of the content routers. The information stored for each sub-cluster is the set of the most heavily weighted terms from documents in that sub-cluster. The abstraction function drops the least weighted terms in order to satisfy given resource limitations. The abstraction function computes a set of sub-clusters by starting off with the set of sub-clusters received from the children routers, and reclustering until it reaches a specified resource budget. See below for a discussion of how this sub-cluster information is used to compute terms suggestions for query refinement. Both the result set expansion and the result set clustering services also rely on the output of the query refinement abstraction function.

• The abstraction function for *query routing*, on the other hand, computes a manageable set of terms that are used for identifying portions of the information space relevant to particular queries. The abstraction function uses term and term frequency information in the children's content labels to compute term weights. The abstraction function then selects the most heavily weighted terms for generating the content router's content label. The abstraction function may also choose to add additional terms that characterize the information space but were not among the terms transmitted up the hierarchy. For example, the abstraction function could add a term describing a poetry cluster as `literature` even though none of the poems mention `literature` explicitly.

• The abstraction function for *browsing* content labels computes a summary of the information space suitable for human comprehension. This includes extracting information from the query routing summary such as the number of documents, the total size in bytes, a small set of the most heavily weighted terms and links to a sample of documents in the cluster. In addition, the abstraction function similarly summarizes each sub-cluster computed for query refinement.

Figure 5 is an excerpt from an actual content label HyPursuit automatically generated from a collection of documents on a Web site. The content label in the figure shows three sub-clusters that are used for computing query refinement suggestions. The sub-cluster summary always includes precise information about the number of documents it contains, the total size in bytes, and the

```
((version: 1)
 (url: "http://www.psrg.../cgi-bin/crs/www-eecs.mit.edu")

 (cluster:
  ((cluster-num: 1) (size: 6763) (num-docs: 5) (num-terms: 458)
  (url: "http://www-eecs.../committees.html")
  (url: "http://www-eecs.../hq/index.html")
  (url: "http://www-eecs.../staff.html")
  (url: "http://www-eecs.../stuorg.html")
  (url: "http://www-eecs.../test'pictures.html")
  (term: ((attribute: header) (value: administrative) (tf: 1) (df: 1)))
  (term: ((attribute: header) (value: committees) (tf: 2) (df: 1)))
  (term: ((attribute: header) (value: computer) (tf: 5) (df: 5)))
  (term: ((attribute: header) (value: eecs) (tf: 5) (df: 5)))
  (term: ((attribute: header) (value: department) (tf: 11) (df: 5)))
  ...
  (term: ((value: committee) (tf: 13) (df: 1)))
  (term: ((value: eecs) (tf: 27) (df: 5)))
  (term: ((value: officer) (tf: 4) (df: 1)))
  (term: ((value: organizations) (tf: 2) (df: 1)))
  (term: ((value: personal) (tf: 3) (df: 1)))
  ... ))

 (cluster:
  ((cluster-num: 2) (size: 17235) (num-docs: 8) (num-terms: 1030)
  (url: "http://www-eecs.../AY94-95/announcements/index.html")
  (url: "http://www-eecs.../AY95-96/announcements/1.html")
  (url: "http://www-eecs.../AY95-96/announcements/2.html")
  (url: "http://www-eecs.../AY95-96/announcements/3.html")
  (url: "http://www-eecs.../AY95-96/announcements/index.html")
  (url: "http://www-eecs.../current/announcements/index.html")
  (term: ((attribute: header) (value: ay95) (tf: 1) (df: 1)))
  (term: ((attribute: header) (value: computer) (tf: 8) (df: 8)))
  (term: ((attribute: header) (value: current) (tf: 1) (df: 1)))
  (term: ((attribute: header) (value: department) (tf: 17) (df: 8)))
  ...
  (term: ((value: announcements) (tf: 22) (df: 8)))
  (term: ((value: department) (tf: 32) (df: 8)))
  (term: ((value: eecs) (tf: 38) (df: 8)))
  ...
  (term: ((attribute: title) (value: announcements) (tf: 8) (df: 8)))
  (term: ((attribute: title) (value: eecs) (tf: 8) (df: 8)))
  ))

 (cluster:
  ((cluster-num: 3) (size: 6766) (num-docs: 3) (num-terms: 1105)
  (url: "http://www-eecs.../class-materials.html")
  (url: "http://www-eecs.../comment-form.html")
  (url: "http://www-eecs.../web-pages.html")
  (term: ((attribute: header) (value: comment) (tf: 1) (df: 1)))
  (term: ((attribute: header) (value: course) (tf: 1) (df: 1)))
  (term: ((attribute: header) (value: czars) (tf: 1) (df: 1)))
  (term: ((attribute: keyword) (value: mail) (tf: 1) (df: 1)))
  (term: ((attribute: keyword) (value: name) (tf: 1) (df: 1)))
  ...
  (term: ((attribute: title) (value: form) (tf: 1) (df: 1)))
  (term: ((attribute: title) (value: home) (tf: 1) (df: 1)))
  (term: ((attribute: title) (value: pages) (tf: 3) (df: 3)))
  ))

 ...
)
```

Figure 5: Sample Clustered Content Label

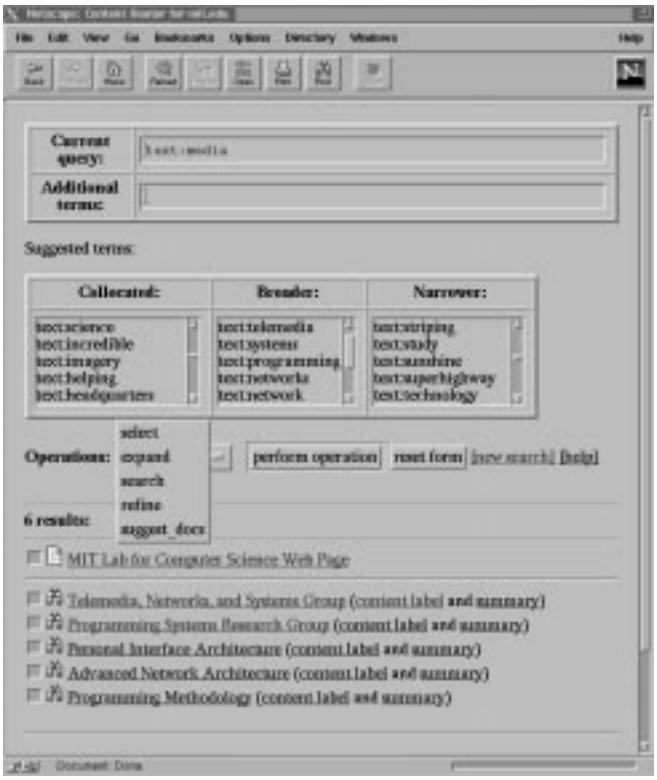

Figure 6: Query Refinement Suggestions

total number of terms in the sub-cluster. It also includes URLs of documents in the sub-clusters and the terms they contain. For each unique term, the content label stores the term frequency (`tf`) and document frequency (`df`). The query refinement abstraction function may decide to store only a subset of the URLs and terms depending on available resources.

Information Retrieval Services

HyPursuit provides a set of information retrieval services including query routing, query refinement, result set expansion, cluster-based browsing, and result set clustering. The user explicitly invokes all but the latter service via user interface operations. HyPursuit automatically clusters result sets before they are shown to the user. This section describes HyPursuit's information retrieval services as well as how the HyPursuit user interface provides a coherent set of user-level operations based on these services.

Query Routing HyPursuit's user interface allows the user to search for relevant information with query-based operations that automatically traverse the cluster hierarchy. Our previous work[22] describes in detail how these content routing operations prune the information space and provide progressively finer-grained views of the relevant information. Relevant information may be either clusters (*i.e.,* content routers) or documents.

Figure 6 shows the HyPursuit user interface, including the query processing operations. The current query appears at the top of the window, followed by a text entry region for adding terms to the query. The query routing operations, shown in the pop-up menu in the center of the figure, are `select`, `expand`, and `search`. `Select` prunes the current result set to the clusters and documents that match the given query. `Expand` selects children documents and clusters that match the given query. `search` recursively searches clusters for all documents that match the query by selectively traversing the cluster hierarchy down to the leaf nodes. Note that the result set includes both a leaf document and several content router clusters.

HyPursuit uses query routing to support the search operations. Query routing uses the content labels stored in the content router to determine which of the child servers are likely to contain documents related to the user query. The query is forwarded to these servers, and the results from each server are merged into a single result set. Documents returned by more than one child server are displayed only once.

Clustering Result Sets To help users browse and comprehend the information space, HyPursuit organizes and presents the result set documents according to their sub-clusters. Documents that belong to the same sub-cluster are placed together in the user screen. Figure 7 illustrates a clustered result set that contains five documents grouped into three sub-clusters. Note that two of the documents have the same title. These results have different URLs that point to the same underlying documents. The HyPursuit prototype removes duplicate URLs, but cannot detect this type of aliasing. However, result set clustering helps the user detect duplication because duplicate documents will appear in the same sub-cluster.

Query Refinement HyPursuit uses term information about sub-clusters to dynamically compute recall- and precision-enhancing terms related to a user query. Figure 6 shows the interface of our system after an interaction with the search facilities to produce a result set and a subsequent query refinement operation. The region titled `suggested terms` in Figure 6 contains three scrollable lists of terms. A content router suggests query refinement terms using the sub-clusters in the content labels of its child servers. `Collocated` terms are the highest weighted terms from the sub-clusters that match the query. HyPursuit's term weights approximate conditional probabilities of term collocation. Term collocation in sub-clusters approximates term collocation in documents.

`Broader` and `narrower` terms are suggested by a thesaurus-based query refinement mechanism. Broader terms represent general concepts related to the terms in the query, and are expected to improve recall. They provide a means of exploring the information space. Narrower terms can improve precision by allowing specialization of queries. The thesaurus mechanism is based on the automatic construction of thesaurus classes built offline using a Forsyth-Rada algorithm [11] modified to consider sub-clusters rather than

documents. For a given collection of sub-clusters, we generate a two-level hierarchy of term classes by gathering term frequencies, calculating weights and grouping high-frequency and low-frequency terms. We then establish the broader/narrower relationships between the high-frequency and low-frequency terms based on similarities between the term frequency distribution functions.

Cluster-Based Browsing As shown in figure 6, users can browse through the information space by examining clusters, content labels, and cluster summaries. To see the contents of a cluster, *i.e.,* its child documents and clusters, a user clicks on the cluster's name. To see a cluster's content label, the user clicks on the text `content label` next to the cluster. To see the cluster's summary, the user clicks on the text `summary` next to the cluster.

HyPursuit summarizes the information space of a content router in a format that is suitable for human comprehension. A cluster summary includes two parts: the most heavily weighted terms in the cluster and a summary of each of the sub-clusters computed by the abstraction function for query refinement. The summary for each sub-cluster includes a selection of the most heavily weighted terms and a list of some of the documents in the sub-cluster. In the current implementation, HyPursuit arbitrarily selects the sample list of documents. A future implementation may choose to select documents that are more representative of the sub-cluster. For example, the system can suggest precomputed centroid documents [7] for each sub-cluster based on both the terms in the sub-cluster and the link structures.

Result Set Expansion To improve recall, HyPursuit suggests additional related documents that, though not selected by the query, are collocated in sub-clusters with query-selected documents. Figure 7 shows the result of executing a `suggest docs` operation after processing the query `text:file text:semantic text:system`. The list labeled `5 results:` contains query selected documents. The list of documents labeled `Similar Documents` consists of documents that appear in the same sub-clusters as the query-selected documents. Currently, HyPursuit suggests all documents that appear in the same sub-cluster as any result set document. A future implementation of HyPursuit may suggest only certain documents from the sub-clusters, such as those nearest the sub-cluster centroid. A document will be compared to the centroid based on both the terms in the sub-cluster and the link structures. We also plan to provide users with a graphic representation of the relevant sub-cluster.

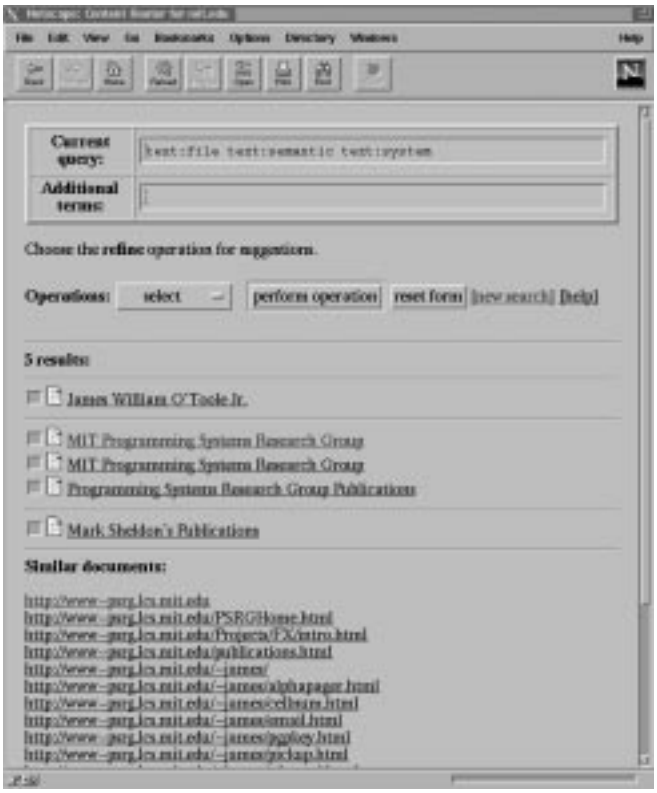

Figure 7: Suggesting Additional Documents

CONTENT-LINK CLUSTERING

Content-link hypertext clustering organizes documents into groups of related documents called *clusters* based on the terms they contain and their hyperlink structures. We first describe the generic content-link clustering algorithm. The algorithm uses a new document similarity function based on both term similarity and hyperlink similarity factors. We then describe a novel hyperlink similarity function that assigns higher similarities to documents that have ancestors and descendants in common, as well as documents that point (directly, or indirectly) to one another. Finally, we describe how term weights are factored into the computation of hypertext document similarities using a traditional term-weighting scheme.

Similarity-Based Clustering

Our clustering is based on the *complete link method* [19]. Although faster clustering algorithms exist [6], we chose the complete link method because it was easy to implement. The complete link method starts with a set where each original document represents an independent cluster. The algorithm iteratively reduces the number of clusters by merging the two most similar clusters until `max_clusters` clusters remain. The algorithm uses pair-wise similarities of component clusters to compute the similarity of two compound clusters. The similarity of the compound clusters is the minimal similarity between any of these pairs. The complete link method avoids generating very large clusters.

Our content-link hypertext clustering uses a hybrid similarity function that includes hyperlink and term components. The first component, S_{ij}^{links}, measures the similarity between hypertext documents d_i and d_j based on their hyperlink structures. The second component, S_{ij}^{terms}, measures the similarity between hypertext documents d_i and d_j based on the document terms. The

similarity between two hypertext documents, S_{ij}^{hybrid}, is a function of S_{ij}^{links} and S_{ij}^{terms}, as shown in equation 1:

$$S_{ij}^{hybrid} = \mathcal{F}(S_{ij}^{terms}, S_{ij}^{links}) \quad (1)$$

The similarity measurements proposed in the following sections capture *qualitative* notions about how link structures and document contents imply relationships between documents. The design of HyPursuit includes parameterization of these similarity functions to allow experimentation and customization based on the information space. For example, in the HyPursuit prototype, the function $\mathcal{F}$ that combines the hyperlink and term similarity values is `max`. This ensures that if either the link similarity or the term similarity is high, then the hybrid similarity is also high. In the future, we plan to investigate the *quantitative* behavior of the proposed hyperlink similarity function and other alternatives.

A Simple Hyperlink Similarity Function

Our measure of the hyperlink similarity between two documents, S_{ij}^{links}, captures three important notions about certain hyperlink structures that imply semantic relations: a path between two documents, the number of ancestor documents that refer to both documents in question, and the number of descendant documents that both documents refer to. Other notions, such as the number of independent paths between the two nodes, are also important but currently not considered by the HyPursuit prototype.

For our discussion, we use the following definitions:

$spl_{xy} \equiv$ length of a shortest path between d_x and d_y

$spl_{xy}^{\overline{z}} \equiv$ length of a shortest path between d_x and d_y not traversing d_z

Direct Paths We hypothesize that the similarity between two documents varies inversely with the length of the shortest path between the two documents. A link between documents d_i and d_j establishes a semantic relation between the two documents. If we assume that these semantic relations are transitive, then a path between two nodes also implies a semantic relation. However, as the length of the shortest path between the two documents increases, the semantic relation between the two documents tends to weaken. Because the hypertext links are directional, we consider both shortest path $d_i \rightsquigarrow d_j$ and $d_j \rightsquigarrow d_i$. If there is no path between d_i and d_j, we do not add any weight to the similarity function for this component. Equation 2 shows S_{ij}^{spl}, the component of the hyperlink similarity function that considers shortest paths between the documents:

$$S_{ij}^{spl} = \frac{1}{2^{(spl_{ij})}} + \frac{1}{2^{(spl_{ji})}} \quad (2)$$

The denominator ensures that as shortest paths increase in length, the similarity between the documents decreases.

Common Ancestors The similarity between two documents is proportional to the number of ancestors that the two documents have in common. The analogy comes from bibliographic citations: when two or more articles $a_1, a_2, \ldots a_n$ cite some set of common articles $c_1, c_2, \ldots c_n$, then this likely implies a semantic relation between the c_i's. As with S_{ij}^{spl}, the semantic relation tends to weaken as the paths between the citing articles a_i's and the cited document c_i's increases. Equation 3 shows S_{ij}^{anc}, the component of the hyperlink similarity function that considers common ancestors:

$$S_{ij}^{anc} = \sum_{x \in \substack{common \\ ancestors}} \frac{1}{2^{(spl_{xi}^{\overline{j}} + spl_{xj}^{\overline{i}})}} \quad (3)$$

S_{ij}^{anc} considers the length of the shortest path between a common ancestor and both d_i and d_j. As the shortest paths increase in length, the similarity decreases. Also, the more common ancestors, the higher the similarity. The computation normalizes S_{ij}^{anc} to lie between 0 and 1 before it is included in S_{ij}^{links}. The weight contribution from each ancestor x is divided by the number of ancestors in the same "level" as x. The level of x with respect to d_i and d_j is the minimum distance from either d_i or d_j.

A common ancestor a_1 does not contribute to S_{ij}^{anc} when the only path that reaches d_j from a_1 is through d_i. If d_i is an ancestor of d_j, then all ancestors $a_1, a_2, \ldots a_n$ of d_i are automatically ancestors of d_j. This would imply that any document that cites d_i, directly or indirectly, adds to the similarity between d_i and d_j. A path between d_i and d_j is already considered in the similarity measurement with the S_{ij}^{spl} component. Therefore, S_{ij}^{anc} does not include ancestors that are not proper common ancestors. However, if there is another path $a_1 \rightsquigarrow d_j$ that does not traverse d_i, then a_1 is considered a proper common ancestor of d_i, and its weight contribution is added to the similarity function.

Common Descendants The similarity between two documents is also proportional to the number of descendants that the two documents have in common. The situation is analogous to the semantic relation implied by common ancestors. If articles $a_1, a_2, \ldots a_n$ cite some set of common articles $c_1, c_2, \ldots c_n$, then this likely implies a semantic relation between the a_i's. Equation 4 shows S_{ij}^{dsc}, the component of the hyperlink similarity function that considers common descendants:

$$S_{ij}^{anc} = \sum_{x \in \substack{common \\ descendants}} \frac{1}{2^{(spl_{ix}^{\overline{j}} + spl_{jx}^{\overline{i}})}} \quad (4)$$

The computation normalizes S_{ij}^{dsc} to lie between 0 and 1 before it is included in S_{ij}^{links} in the same manner as the normalization for S_{ij}^{anc}.

Complete Hyperlink Similarity The complete hyperlink similarity function between two hyperlink documents d_i and d_j, S_{ij}^{links}, is a linear combination of the above components:

$$S_{ij}^{links} = W_d \cdot S_{ij}^{dsc} + W_a \cdot S_{ij}^{anc} + W_s \cdot S_{ij}^{spl} \quad (5)$$

Term-Based Document Similarity Function

The similarity between hypertext documents also relies on the traditional method of using weighted terms. The term-weight function should favor terms that are representative of the documents, but should also discriminate between the documents and the servers that hold them. The best known term weighting approaches [20] use compound normalized weights with three factors: term frequency, inverse document frequency and a factor inversely proportional to the size of documents. *Term frequency* is the number of occurrences of a term in a document. *Document frequency* is the number of documents within the global information space in which the term appears. The *document size factor* compensates for high term frequencies of terms in large documents.

The distributed nature of hierarchical search engines complicates the task of defining an appropriate term weighting function because global collection frequencies are not available. For example, consider the task of generating a content label for the Laboratory of Computer Science (LCS) server at MIT. The term `computer` appears in hundreds of documents (high collection frequency) and is very frequent in many of these documents (high term frequency). Is this a good term to keep in the content label for LCS? If we weigh the term with the three factors above it may end up with a low weight because of its high collection frequency. However, if the global document space includes documents in broad areas unrelated to computers, this term happens to be a good discriminator for the LCS server, and therefore should be kept in its content label.

The current weight function uses term frequency and document size factors, but does not include collection frequency. However, we are investigating possible alternatives that yield content labels with less high global collection frequency terms.

Term weights also consider term attributes. The weight function assigns a larger factor to terms with attributes `title`, `header`, `keyword` and `address` than the weight factor assigned to `text` terms.

The total weight w_{ki} of a term t_i in document d_k is calculated based on the term similarity function proposed by [20], with the omission of the collection frequency factor, as follows:

Let

$$\begin{aligned} tf_{ki} &\equiv \text{term frequency of } t_i \text{ in } d_k \\ w_{ki}^{tf} &\equiv \text{contribution to weight from frequency } tf_{ki} \\ w_{ki}^{ds} &\equiv \text{contribution to weight from size of } d_k \\ w_{ki}^{at} &\equiv \text{contribution to weight from term attribute} \end{aligned}$$

then

$$w_{ki}^{tf} = (0.5 + 0.5\frac{tf_{ki}}{\max_j \{tf_{kj}\}}) \quad (6)$$

$$w_{ki}^{ds} = \frac{1}{\sqrt{\sum_i \left(w_{ki}^{at} w_{ki}^{tf}\right)^2}} \quad (7)$$

$$w_{ki} = w_{ki}^{tf} \cdot w_{ki}^{ds} \cdot w_{ki}^{at} \quad (8)$$

The weight factor w^{at} is configurable on a per server basis, but defaults to 10 for titles, 5 for headers, keywords, and addresses, and 1 for text attribute types.

The term-based similarity function S_{ij}^{terms} between documents d_i and d_j is the normalized dot product of the terms vectors representing each document.

$$S_{ij}^{terms} = \sum_k w_{ik} \cdot w_{jk} \quad (9)$$

IMPLEMENTATION

The HyPursuit prototype consists of a distributed set of content routers that interact with each other through HTTP POST queries and with users through the HTML FORM interface. Each content router is implemented as a CGI script that performs content-routing operations. HyPursuit runs on Sun SparcStations under SunOS 4.1.3. All modules in the system were implemented using GNU C++.

Architecture

The set of executable modules comprising each content router in the current HyPursuit prototype includes *abstraction* modules, *transducer* modules, and a single CGI *engine* module. The engine module is responsible for processing requests generated from users through their WWW browsers or from higher level HyPursuit routers. The HyPursuit engine is stateless and does not currently maintain any cache or history of previous requests.

The abstraction modules are responsible for summarizing an information space by generating content labels. HyPursuit's abstraction module for query refinement uses both term information and link information to generate the document sub-clusters included in content labels for WWW servers. However, the abstraction function for higher level routers currently uses only term information to generate query refinement sub-clusters

for higher level routers. Future HyPursuit abstraction functions will take advantage of link information at all levels of the hierarchy.

HyPursuit includes a set of transducer modules that use the abstraction function output to generate databases required by the content routing information retrieval services. Every service may potentially require a different database, but all necessary databases must be created from the information provided in content labels. Various services may share the same database. The current HyPursuit prototype has four different databases: one for query refinement based on sub-cluster collocation of terms, another for query refinement based on thesaurus classes, a third for routing queries and a fourth for suggesting similar documents. Result set expansion and result set clustering rely on the term collocation database.

Configuration

Leaf content routers provide content-based access to collections of individual documents on the World Wide Web. These routers may run anywhere on the network, although for the sake of efficiency they should run in close physical proximity to their data, preferably on the same machines. Each such router invokes a Web robot that gathers the full text of documents on the corresponding web server and generates a full text index mapping terms to document URLs. The router's robot also generates a content label for the web site that includes sub-clustering information used by the daemon for providing the content routing services. For the sake of our experiments, the HyPursuit leaf content routers all run on our computers. They communicate with each other via HTTP.

The HyPursuit prototype uses a simple content label size budgeting scheme based on maximum numbers of terms and sub-clusters per content label. This scheme was chosen for its simplicity and ease of implementation. We are investigating other approaches.

Figure 2 shows a portion of a content routing hierarchy that was constructed for our experimental system. The system consists of 100 leaf servers (that index particular Web sites) and a series of higher-level content routers organized in a hierarchy that resembles the structure of the Domain Name System [17]. For instance, the root router's name is "edu", and some of its children are "mit.edu" and "cmu.edu". The pattern follows until leaf servers with full domain names are reached (e.g. www.psrg.lcs.mit.edu).

EXPERIENCE

This section presents experience with the HyPursuit prototype. The section first compares the clusters generated by each variation of the content-link hypertext clustering algorithm on a particular collection. Then, the section discusses the HyPursuit prototype storage requirements and the performance of the query processing operations.

Clustering Example

We ran an experiment to compare CNN's manual clustering of pages on their `www.cnn.com` web site with our automatic clustering techniques. CNN manually clusters documents into predefined categories, including technology, health, business, weather, and others. The manual clustering of the documents is reflected in their URLs. For example, `http://www.cnn.com/TECH/apple/windows_hype/` is categorized as a `TECH` document, i.e. it is clustered with other documents about technology.

Table 1 shows the clustering result obtained by running the term-based, link-based, and content-link clustering algorithms on a collection of documents from the `www.cnn.com` web site. The example collection includes 195 documents retrieved by following a breadth-first-search starting with the root `http://www.cnn.com`. Each table entry represents a cluster generated by one of our algorithms. An entry lists the documents in the cluster, giving names of their CNN categories and the number of documents from each category. The category names are literally extracted from the URL.

A subjective examination of the table reveals that both the term-based and the link-based algorithms approximate CNN's manual clustering reasonably well, but the hybrid algorithm tends to agree most closely with CNN's scheme. For example, the term-based algorithm clusters 13 documents from the `HEALTH` category with other documents from `BIZ`, `POLITICS`, `TECH`, and `US`. The links algorithm also clusters `HEALTH` documents with other documents. In contrast, the hybrid algorithm is able to group `HEALTH` documents in a separate cluster. We are investigating approaches that will allow us to quantify these observations.

Storage Requirements and Performance of Operations

HyPursuit's leaf content routers contain the first 100 documents in a breadth-first-search traversal of each HTTP server starting at the node whose URL has a null pathname (e.g. `http://www.psrg.lcs.mit.edu/`). We choose to index only 100 documents from each site because of storage limitations and to allow us to experiment with different clustering techniques without overburdening the web sites. This limitation does not reflect any constraints of the design or implementation. The abstraction module that generates content labels for leaf servers considers the first 500 terms from each document, as well as the 500 most frequent terms appearing afterwards in the document. Again, we index a limited number of terms to satisfy our resource limitations. We choose the first 500 terms because they possibly characterize the document well. The content label budget scheme fixes the maximum number of subclusters for query refinement: `max_clusters` is 30 for leaf servers and 50 for higher-level routers.

Table 2 shows the storage requirements of the HyPursuit content label, query routing, and query refinement components. Level 0 indicates the storage requirements of an average leaf content router. The table reflects

	Algorithm		
cluster #	Terms	Links	Hybrid
1	BIZ (22), HEALTH (13) POLITICS (17) TECH (14), US (9)	Programs (2), HEALTH (13) POLITICS (16), feedback (1)	HEALTH (13)
2	WEATHER (7)	WEATHER (9)	WEATHER (7)
3	EARTH (1), HLN (1) SHOWBIZ (14), Studio (1) US (10), WEATHER (2) WORLD (1)	SHOWBIZ (17), INDEX (1)	SHOWBIZ (16)
4	SHOWBIZ (2), SPORTS (2) networks (1)	BIZ (19)	BIZ (22)
5	US (5), WORLD (11)	WORLD (18)	WORLD (18)
6	AtWork (1), Radio (1) INDEX (2)	homepage (1), Airport (1) AtWork (1), BIZ (3), Programs (5) Radio (1), EARTH (1), HLN (1) INDEX (3), PressRelease (1) SEARCH (3), Studio (4) networks (1)	Airport (1), AtWork (1) Programs (3), Radio (1) Studio (3), networks (1)
7	Studio (3)	Programs (1), TECH (14)	TECH (14)
8	SPORTS (8)	SPORTS (9)	SPORTS (8)
9	STYLE (17)	STYLE (17)	STYLE (17)
10	Programs (6)	US (22)	US (20)
11	WORLD (5)	POLITICS (1)	INDEX (2), POLITICS (17)
12	PressRelease (1) SHOWBIZ (1)	feedback (1)	homepage (1), EARTH (1), PressRelease (1), SEARCH (3) SHOWBIZ (1), US (1), INDEX (2)
13	Airport (1)	US (1)	HLN (1), Studio (1), US (1) WEATHER (2)
14	Programs (3)	Programs (1)	Programs (6)
15	homepage (1), WORLD (1) INDEX (2)	SPORTS (1)	SPORTS (2)
16	SEARCH (3)	US (1)	US (2)
17	feedback (3)	feedback (1)	feedback (3)
18	feedback (1)	feedback (1)	feedback (1)
19	feedback (1)	feedback (1)	feedback (1)
20	feedback (1)	feedback (1)	feedback (1)

Table 1: Clusters by Algorithm on `http://www.cnn.com`

the different storage limitations placed on leaf content routers versus higher level content routers. These constraints also limit the sizes of the routing and refinement databases.

Table 3 shows the average processing time of the HyPursuit prototype operations, measured in whole seconds. The `select` operation is local, and therefore there is no difference in performance between content routers on different levels. The `expand` operation contacts only children routers, and therefore the level of the router does not affect the processing time of the operation. The `search` operation requires communication with descendent routers that match the given query, and therefore the processing time increases with the levels.

The `refine` and `suggest_docs` operations consult the local query refinement sub-cluster database. For these operations and our example queries, as the level of the content router increases from zero to two, the average number of matching sub-clusters increases from four to twelve. Currently, the parsing of sub-clusters takes over a second for each sub-cluster that consists of approximately 500 terms. Therefore, the processing time for the `refine` and `suggest_docs` operations increases with the content routing levels due to the increase in the number of matching sub-clusters. A better data structure that maps sub-cluster identifiers to terms and reduces sub-cluster parsing times would greatly improve the performance of these operations.

	Databases		
Level	content labels	routing	refinement
0	410	3509	3378
1	678	3937	4003
2	630	4005	4057

Table 2: Average Sizes of Databases (in KB)

	Operation				
Level	select	expand	search	refine	suggest_docs
0	1	1	1	6	5
1	1	3	2	11	10
2	1	2	6	18	18

Table 3: Average Processing Time (in seconds)

CONCLUSION AND FUTURE WORK

We have built a new HTTP-based prototype content routing system that exploits content-link hypertext clustering to provide access to over 100 web sites. This system was used to examine the feasibility of content-link clustering for the construction of meaningful and scalable cluster hierarchies. The HyPursuit prototype supports query processing operations with abstraction functions that summarize information spaces and cluster documents based on both term and link information. Our limited experience with the system shows that the prototype delivers adequate performance. The results are also promising with respect to the ability of the system to scale up to a very large number of web sites. It is clear that a full scale system will require better performance in both offline and online operations. In addition, we are concerned that Content Routing may be subject to performance hotspots due to intense transient interest in a particular subject. Preliminary results illustrate that combining term and link information offers benefits to hypertext document clustering.

We are continuing to investigate alternative approaches to hypertext clustering and methods to measure the performance of various clustering algorithms. We plan to build larger, automatically constructed hierarchies both to test our hypothesis about the scalability of the system as well as to apply the clustering algorithms to more diverse information spaces. Finally, we plan to compare the performance of using content-link hypertext clustering for the query refinement abstraction function versus other non-clustered approaches.

ACKNOWLEDGMENTS

The authors would like to acknowledge the contributions of others in the original writing and formatting of this document. We are grateful to David Karger, James O'Toole, Jr., and our reviewers for valuable comments. This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense contract number DABT63-95-C-0005.

REFERENCES

1. Bob Alberti, Farhad Anklesaria, Paul Linkner, Mark McCahill, and Daniel Torrey. The Internet Gopher protocol: A distributed document search and retrieval protocol. University of Minesota Microcomputer and Workstation Networks Center, Spring 1991. Revised Spring 1992.

2. Rodrigo A. Botafogo. Cluster analysis for hypertext systems. In *ACM 16th Annual International SIGIR '93*, Pittsburgh, PA, June 1993.

3. Rodrigo A. Botafogo and Ben Schneiderman. Identifying aggregates in hypertext structures. In *Hypertext '91*, December 1991.

4. C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The harvest information discovery and access system. In *Proceedings of the Second International World Wide Web Conference*, pages 763–771, Chicago, Illinois, October 1994.

5. Donald B. Crouch, Carolyn J. Crouch, and Glenn Andreas. The use of cluster hierarchies in hypertext information retrieval. In *Hypertext '89*, November 1989.

6. Douglass R. Cutting, David R. Karger, and Jan O. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *16th Annual International SIGIR'93*, Pittsburgh, June 1993.

7. Douglass R. Cutting, David R. Karger, Jan O. Pedersen, and John W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *15th Annual International SIGIR*, pages 318–329, Denmark, June 1992.

8. Andrzej Duda and Mark A. Sheldon. Content routing in networks of WAIS servers. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 124–132, Poznan, Poland, June 1994. IEEE.

9. David Eichmann. The RBSE spider – balancing effective search against web load. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

10. David Filo and Jerry Yang. Yahoo frequently asked questions. World Wide Web Document. URL http://www.yahoo.com/faq.html.

11. R. Forsyth and R. Rada. *Machine Learning—Applications in Expert Systems and Information Retrieval.* Ellis Horwood, 1986.

12. Mark E. Frisse. Searching for information in a hypertext medical handbook. *Comm. ACM*, 31(7), July 1988.

13. George W. Furnas and Jeff Zacks. Multitrees: Enriching and reusing hierarchical structure. In *CHI 94 Human Factors in Computing Systems: Celebrating Inderdependence*, Boston, April 1994.

14. Luis Gravano, Anthony Tomasic, and Héctor García-Molina. The efficacy of GlOSS for the text database discovery problem. Technical Report STAN-CS-TR-93-2, Stanford University Department of Computer Science, October 1993.

15. Harley Hahn and Rick Stout. *The Internet Complete Reference.* Osborne McGraw-Hill, Berkeley, California, 1994.

16. Martijn Koster. ALIWEB – archie-like indexing in the web. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

17. P. Mockapetris. Domain names - concepts and facilities. RFC 1034, 1987.

18. Brian Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

19. Gerard Salton and Jose Araya. On the use of clustered file organization in information search and retrieval. Technical Report TR 89-989, Cornell University, April 1989.

20. Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. Technical Report TR 87-881, Cornell University, November 1987.

21. Mark A. Sheldon, Andrzej Duda, Ron Weiss, and David K. Gifford. Discover: A resource discovery system based on content routing. In *Proceedings of The Third International World Wide Web Conference*. Elsevier, North Holland, April 1995. To appear in a special issue of Computer Networks and ISDN Systems.

22. Mark A. Sheldon, Andrzej Duda, Ron Weiss, James W. O'Toole, Jr., and David K. Gifford. Content routing for distributed information servers. In *Fourth International Conference on Extending Database Technology*, pages 109–122, Cambridge, England, March 1994. Available as Springer-Verlag LNCS Number 779.