Semantic Search of Unstructured Data using Contextual Network Graphs

Maciej Ceglowski, Aaron Coburn, and John Cuadrado

National Institute for Technology and Liberal Education
Middlebury College, Middlebury, Vermont, 05753 USA
{mceglows, acoburn}@middlebury.edu
drjlc@acm.org

Abstract. The authors present a graph-based algorithm for searching potentially large collections of unstructured data, and discuss its implementation as a search engine designed to offer advanced relevance feedback features to users who may have limited familiarity with search tools. The technique, which closely resembles the spreading activation network model described by Scott Preece, uses a term-document matrix to generate a bipartite graph of term and document nodes representing the document collection. This graph can be searched by a simple recursive procedure that distributes energy from an initial query node. Nodes that acquire energy above a specified threshold comprise the result set. Initial results on live collections suggest that this technique may offer performance comparable to latent semantic indexing (LSI), while avoiding some of that technique's computational pitfalls. Both the algorithm and its implementation in a production Web environment are discussed.

1. Introduction

The rapid growth of online information has proved both a blessing and a burden, as information retrieval systems struggle to keep up with a flood of new material. This information overload is particularly acute in the academic community, where domain experts need to perform advanced searches, but frequently lack the expertise and training needed to make productive use of tools like Boolean query syntax and regular expressions. Current metadata creation standards for the humanities, such as SCORM [1] or Dublin Core [2], require such intensive human effort that, for many institutions, the costs of data markup are prohibitive.

This situation has created an urgent need for automated search tools that can search large data collections based on semantic content. While techniques such as latent semantic indexing (LSI) have shown great promise in enabling this kind of content discovery, it is not clear that they can scale well to large, dynamic document collection.

The National Institute for Technology and Liberal Education (NITLE), with generous support from the Andrew W. Mellon Foundation, has been working to make usable, advanced search tools available to students and scholars. Our initial work on adapting LSI for general use has led to the rediscovery of a search algorithm first described in 1981 [3] which shows great promise in offering results comparable in quality to LSI, without the concomitant computational overhead.

In this paper we describe both the new algorithm and its initial implementation as an open-source, Web based search service designed for maximum usability.

2. Latent Semantic Indexing

Latent semantic indexing (LSI) is a vector-space technique that has shown great promise in addressing the problems of polysemy and synonymy in text collections, offering improved recall over standard keyword searches. One of the most striking features of LSI is the ability to return relevant results for queries even when there is no exact keyword match [4]. Because the LSI vector model is a purely mathematical representation, the indexing technique is not limited to any particular language. Indeed, recent work has shown that LSI can be usefully extended to non-text domains, including log file analysis [5] and protein structure prediction [6].

Several features of vector space models in general, and LSI in particular, allow for the design of relevance feedback features that would be difficult to implement in a standard search engine. These include 'find similar' links for individual documents in a result set, as well as the ability to run an iterative search by re-querying the engine with a set of relevant results. Suitably implemented, these kinds of feedback features can reduce or eliminate the need to train users in an elaborate query syntax.

For all its advantages, LSI also presents some drawbacks. The poor scalability of the singular value decomposition (SVD) algorithm remains an obstacle to indexing very large collections. While techniques have been developed for making incremental updates to a scaled collection, these changes typically cannot exceed a certain threshold without triggering a rebuild [7,8]. These constraints make LSI ill suited to the kinds of large, rapidly changing document collections typically found on the Web.

A further disadvantage to LSI is the difficulty in interpreting the underlying reduced term space [4]. This makes it difficult to select an optimum number of singular values to retain in the SVD for a given collection, or allow domain expert adjustment of relevance values in the reduced space once the SVD has been calculated.

These shortcomings have led us to seek out an indexing scheme that retains the advantages of LSI, while presenting fewer computational and conceptual obstacles.

3. Contextual Network Graphs

A standard step in LSI is the creation of a term-document matrix (TDM), which is essentially a weighted lookup table of term frequency data for the entire document collection. In LSI, this matrix is interpreted as a high-dimensional vector space.

An alternative interpretation of this matrix is possible, however, with the TDM representing a bipartite graph of term and document nodes where each non-zero value in the TDM corresponds to an edge connecting a term node to a document node. In this model, every term is connected to all of the documents in which the term appears, and every document has a link to each term contained in that document. The weighted

frequency values in the TDM correspond to weights placed on the edges of the graph. We call this construct a *contextual network graph*.

As an example, consider the miniature document collection in table 1, and its associated term list in table 2.

Table 1. A sample document collection with associated node labels

Node	Document content
1	*Glacial ice often appears blue.*
2	*Glaciers are made up of fallen snow*
3	*Firn is an intermediate state between snow and glacial ice.*
4	*Ice shelves occur when ice sheets extend over the sea.*
5	*Glaciers and ice sheets calve icebergs into the sea.*
6	*Firn is half as dense as sea water.*
7	*Icebergs are chunks of glacial ice under water.*

Table 2. A labeled term list derived from the collection in table 1. All terms occur across at least two documents in the parent collection.

Node	Term	Occurrence count
a	glacial ice	3
b	ice	5
c	glacier	2
d	snow	2
e	firn	2
f	ice sheet	2
g	sea	3
h	water	2
i	iceberg	2
j	sheet	2

We can represent this collection of terms and documents as a contextual network with the topology shown in figure 1.

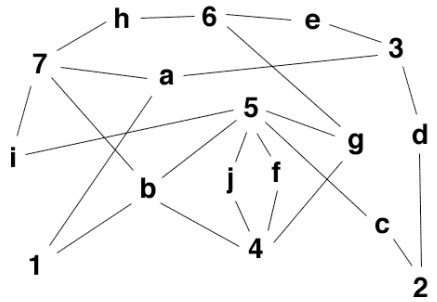

Fig. 1. Sample contextual network graph indicating connections between the documents in table 1 and content terms in table 2

While similar in spirit to IR approaches like conceptual graphs [10], the contextual network graph does not encode any information about grammatical or hierarchical relationships between terms. Its structure is determined purely by term co-occurrence across the collection.

Each edge in the graph has a strength assigned to it whose magnitude depends on our choice of the local and global term weighting scheme used in generating the TDM. The only constraint on weighting schemes is that all edge weights must fall in the interval (0,1).

We can search the collection represented by this graph by energizing a query node and allowing the energy to propagate to other nodes along the edges of the graph based on a set of simple rules. The total energy deposited at any given node in the graph will depend both on the number of paths between it and the query node, as well as the relative strength of the connections along those paths. This corresponds to the intuition that documents that share many rare terms are likely to be semantically related. It also enables the same kind of enhanced recall provided by LSI, since a query on a particular keyword may still reach a document that does not contain the word itself, but is closely linked to other documents that do.

Because the initial energy dissipates as it spreads over the graph (the requirement that energy dissipate is the reason for the constraint on edge weights), results for any search are localized to a single region of the graph, with important implications for scalability.

In the example above, a search on 'iceberg' would begin by activating the node corresponding to the query term, in this case node *i*. This query node is assigned a default starting energy *E*, which is distributed to neighbor nodes according to the following algorithm:

```
1  procedure energize( energy E, node n_k ) {
2       energy(n_k) := energy(n_k) + E
3       E' := E / degree of  n_k
4       if ( E' > T ) {
5            for each node n_j in N_k  {
6                 E" := E' * e_jk
7                 energize( E", n_j )
8            }
9       }
10 }
```

Where N_k is the set of all neighbor nodes of n_k, e_{jk} is the weight of the edge connecting nodes n_j and n_k, T is a constant threshold value, and energy(n_k) is a data structure that stores node energy values for the duration of a query. Note that this version of the algorithm performs a depth-first traversal of the graph. In the case where N_k is sorted by decreasing edge weight, the traversal is also best-first. The algorithm may be suitably modified to perform a breadth-first traversal; the optimal traversal strategy is a topic for further study.

In our example, the query consists of a single term, and the search terminates after the energy from the initial node has been distributed as far as it can go before dipping

below the threshold *T*. In the case of queries consisting of multiple nodes, the procedure would be repeated for each query node in turn, with the final energy values for nodes in the graph a superimposition of the individual searches. Note that the query may consist of any combination of term and document nodes.

Although our graph does not include nodes for singleton terms that occur in only one document, it is a straightforward matter to keep a list of these terms and substitute the appropriate document node for any singleton term in a query.

Once all the nodes in a query have been processed, results are gathered in a collection step, with nodes sorted by reverse order of accumulated energy.

Nodes with the highest energy values will correspond to documents and terms that are semantically closest to the original query. This result set will consist of both document and term nodes. The search interface may choose not to display nearest term results, or may use them as a relevance feedback feature.

In addition to the traversal order, there are several other tunable parameters in the algorithm. The starting energy *E* and threshold energy *T* are both arbitrary values; the larger the starting energy, and the lower the limit threshold, the further the initial node energy will spread in the graph. The mechanism for distributing energy among neighbor nodes in line 3 and the weighting step in line 6 may also be altered.

3.1. Preliminary Observations

Work is ongoing to evaluate the quality of contextual network search (CNS) compared to keyword and LSI approaches. Preliminary results on live collections of Civil War articles [10] and protein sequence data [11] indicate that the CNS offers comparable results to an LSI search engine built from the same term-document matrix. Both CNS and LSI can return semantically related documents that do not contain an exact keyword match; however, CNS tends to favor keyword over non-keyword matches more than LSI.

3.2. Distributed Search

The algorithm described above can be distributed in two ways, both to cope with very large collections, and to improve performance for queries containing a large number of nodes. The two distribution techniques are complementary.

The first technique involves partitioning the graph into subgraphs, and distributing the subgraphs across a computing cluster controlled by a master server. The master server must have a full map of graph connectivity, but need not store any information about node energies or edge weights. It distributes queries by passing them along to the slave server responsible for the portion of the graph containing the query node, as well as handling cases where searches cross subgraph boundaries. The master server collects and collates results from all slave servers involved in a given search into a single result set.

For cases where queries contain a large number of document or term nodes (this can occur frequently in relevance feedback searches, as well as searches that involve large segments of text pasted into a Web form), it is possible to improve search speed by searching in parallel over multiple copies of the graph, and superimposing the

resulting node energies. This approach overcomes one weakness of CNS compared to vector space techniques: while the limiting factor for vector space comparisons is the time required to parse the query into a pseudo-document vector, search time in the contextual network graph model grows as a function of the number of nodes in the query.

In principle, it is possible to combine both the graph segmentation and parallel query approach given a sufficiently large computing cluster.

3.3. Insertions and deletions

Insertion and removal of documents into the collection does not pose the same kinds of problems for CNS as it does for vector model approaches. For additions, the graph server simply has to parse the new documents, and add additional connections between document nodes and existing term nodes. Because singleton terms that occur in only one document are not included in the graph, the list of singleton terms has to be checked when a new document is parsed, to see if any of the terms need to be promoted to full term node status.

Documents can also be removed from the collection on the fly, by excising the appropriate document nodes and checking the graph for any new singleton term nodes, which are relegated back to the singleton list.

Depending on the document normalization and term weighting parameters used in generating the term-document matrix, addition or removal of nodes from the graph may require that edge weights be recalculated for the entire collection. However, this calculation is much less onerous than the kind of full rebuild required in LSI, and there are several ways to mitigate its impact. Edge weights can be updated on a rolling basis if necessary.

4. Implementation

The techniques described in this paper have been successfully implemented in a live Internet search engine covering a collection of Civil War-era newspaper articles, as well as on several test collections of both text and biological data [10,11,12]. The search engines are designed to be modular and to run on multiple servers and platforms, if necessary, to maximize flexibility. The relevance feedback features made possible by CNS form an important part of the user interface.

4.1. System Architecture

Search, indexing, and interface components in the system are decoupled from one another, and communicate across a series of network interfaces. This means that the document data, search engine and user interface need not be located on the same network.

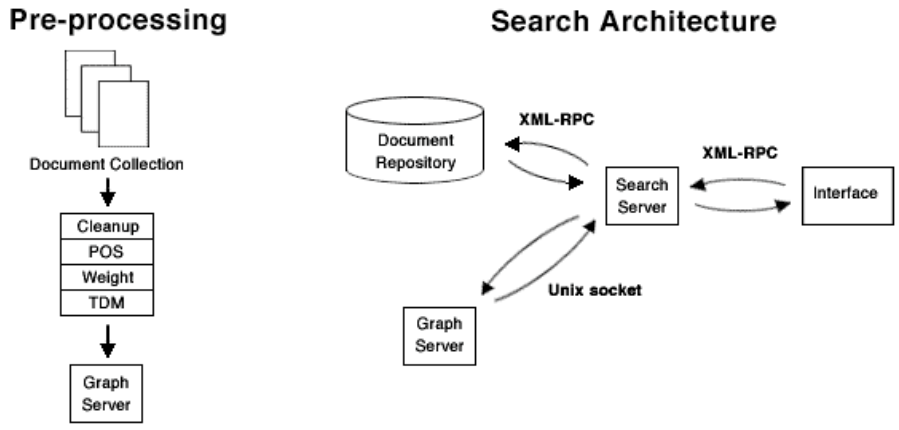

Fig 2. Schematic of document processing and search component design

The system is built around a central document repository, which stores all of the document and term data for the collection. The documents in this repository are processed in an initial indexing step, described below, to create the graph server, after which the search component is completely independent of the document repository. The repository is implemented as a MySQL database with a Perl interface that runs as a Web service, listening for document requests over XML-RPC.

The graph query server is a C++ program that loads the actual contextual network graph, and searches the collection. This server uses a very simple protocol to minimize overhead – it listens on a Unix socket for a list of one or more query nodes, and returns a list of result nodes and energy values in response. Nodes are represented as strings in the form 'd123' or 't456', depending on whether the node is a document or term node. The graph server can also be asked to raise or lower its minimum energy threshold, to control the sensitivity of the search.

The search server correlates results from the graph server with document data obtained from the document repository, and organizes the data into a format suitable for display by the human interface. The search server is responsible for converting natural language queries to node lists to pass along to the graph server, as well taking care of details such as providing short extracts of long documents, and paginating result sets for the benefit of the human interface. This server is implemented in Perl and also runs as a Web service, listening for queries over an XML-RPC protocol.

The user interface can take a variety of forms; in its current implementation, it runs as a Perl Web application. This consists of a single query box and search button, with relevance feedback features provided as part of each result set. Documents in a result set are displayed with a clickable 'find similar' link, allowing for effective horizontal navigation through search results. Users are also given a 'document basket' – a session-based store they can populate with documents encountered in the course of their search. The basket has its own find similar link, letting users search on the aggregate of all documents in the saved set. The basket metaphor, familiar from

online shopping sites, gives users an intuitive method for performing an iterative search on the collection.

The most unusual relevance feedback item is a list of similar terms displayed with each result set. Since CNS returns both document and term nodes, displaying a list of 'nearest terms' for a query is trivial. While these terms are clickable, their more important function is in keeping users oriented. By glancing at the list of similar terms, users can get a feel for the semantic neighborhood of their query.

4.2. Pre-processing

We assume that the raw documents in the repository are in plain text or in a Web markup format, such as XML or HTML. Documents are cleaned using a suite of Perl regular expressions to strip formatting and remove markup. They are then sent through the appropriate parser. For English language text, we use a part-of-speech (POS) tagger in tandem with a regular expression for finding maximal noun phrases to generate our term list [13]. Unlike more traditional stop list + stemming approaches, this approach allows us to discriminate between polysemous words whose meaning changes depending on the part of speech (e.g., 'fly' as a verb or noun), as well as extract multi-word noun phrases and proper names ('hot rod', 'Rip Van Winkle'). Noun phrase extraction is greedy and recursive, so that a single noun phrase can be parsed into multiple terms.

Our POS tagger [14] uses a stored lexicon with statistical information about word and part-of-speech usage for English derived from the Penn Treebank Project [15], an annotated corpus of text developed by the Linguistic Data Consortium. The tagger uses stored probability data in conjunction with a bigram Hidden Markov Model (HMM) of POS occurrence, assigning a tag t_i to a word w_i according to the following formula:

$$t_i = \underset{j}{argmax}\, P(t_j \mid t_{i-1})\, P(w_i \mid t_j) . \quad (1)$$

A metric based on word morphology supplements the HMM for words not appearing in the lexicon.

The term list derived from the parsing step is used to generate a standard term-document matrix (TDM). In our implementation, we weight TDM values according to the formula:

$$e_{ij} = g_j ln(f_{ij}) n_i c_j , \quad (2)$$

where g_j is a global term weight determined by inverse document frequency, f_{ij} is the frequency of term j in document i, n_j is a document normalization factor, and c_j is the number of words in term j.

6. Further Work

We are currently evaluating CNS against both LSI results and human-classified collections from TREC to obtain empirical estimates of search quality. In addition to these ongoing tests, we are pursuing several promising directions of study.

6.1. Scalability.

In its current implementation, the graph server requires approximately 1GB of RAM for every 20K documents (95K terms), running on a desktop Linux server.

We are currently working on implementing the distributed search techniques described earlier, using a cluster of commodity PCs over a Gigabit Ethernet connection to field queries on very large document collections. At this time, available memory rather than processing speed seems to be the limiting factor on graph size.

6.2. Representing Internal Links

The graph data model allows for the potential inclusion of document-to-document links, whether in the form of internal references to other documents, or as explicitly defined links added to the collection model by a human curator. While patterns of internal citations in a document collection have previously been studied [14], contextual networks provide a natural framework for combining content and link-based similarity. Both inherent and user-defined internal links are features that would be difficult to represent in a vector-space model.

The addition of intra-document links causes two major changes to the model - the graph is no longer bipartite, and part of the graph becomes directed. The possible implications of these changes are unclear.

It is also possible to conceive of internal links between term nodes, for example, as an implementation of cross-language search.

6.3. Clustering

Several techniques are available for segmenting the network graph and assigning documents to content clusters. We are currently experimenting with ways to segment very large collections into smaller sub-collections in an effort at auto-categorization, both in the text and protein domain, as well as approaches to graph clustering that will form the subject of a future paper.

6.4. Non-text collections

Our work with LSI in the protein domain has shown encouraging results for context-based prediction of protein structure based on sequence. We have every reason to believe similar results will obtain from a contextual network model, while allowing us to work with larger data collections.

References

1. SCORM: http://www.altrc.org/specification.asp

2. Dublin Core: http://dublincore.org

3. Preece, Scott. "A spreading activation network model for information retrieval" PhD thesis, CS Dept., Univ. of Illinios, Urbana, IL. 1981.

4. Deerwester, S., Dumais, S., Furnas, G., Landauer, T. and Harshman, R., "Indexing by latent semantic analysis", *Journal of the American Society for Information Science*, 1990, pp. 391-407.

5. Quesada, J., Kintsch, W. and Gomez, E. "A Computational Theory of Complex Problem Solving Using Latent Semantic Analysis" In W.D. Gray & C.D. Schunn (Eds.) *Proceedings of the 24th Annual Conference of the Cognitive Science Society, Fairfax, VA*. Lawrence Erlbaum Associates, Mahwah, NJ, 2002, pp.750-755.

6. Ceglowski, M. and Cuadrado, J. Slide presentation, O'Reilly Bioinformatics Conference, February 2003. http://conferences.oreillynet.com/cs/bio2003/view/e_sess/3406

7. Berry, M., Drmac, Z. and Jessup E. "Matrices, Vector Spaces, and Information Retrieval", *SIAM Review*. Vol. 41, No. 2, pp. 335-362.

8. Simon, H. and Zha, H. "On Updating Problems in Latent Semantic Indexing", *Technical Report No. CSE-97-011*, Department of Computer Science and Engineering, Pennsylvania State University, 1997.

9. Montes-y-Gómez, M., López-López, A. and Gelbukh, A. "Information Retrieval with Conceptual Graph Matching", *Lecture Notes in Computer Science*, Vol. 1873, Springer Verlag, 2000, pp. 312-321.

10. University of Virginia Valley of the Shadow Project search engine. http://mojave.cet.middlebury.edu/cns/uva.pl

11. PDB Toxin search demo. http://mojave.cet.middlebury.edu/cns/toxins.pl

12. Steven Johnson research notes. http://mojave.cet.middlebury.edu/cns/sbj.pl

13. Bader, R., Callahan, M., Grim, D., Krause, J. and Pottenger, W., "The role of the HDDI™ collection builder in hierarchical distributed dynamic indexing", *Workshop on Text Mining*, Chicago, 2001, pp. 23-30.

14. Lingua::EN::Tagger, a Perl part-of-speech tagger for English text. http://search.cpan.org/author/MCEGLOWS/

15. Marcus, M., Santorini, B., Marcinkiewicz, M.A., and Taylor, A. TreeBank-3. Linguistic Data Consortium, 1999.