

An Information Retrieval System for Improving Efficiency in Scientific Literature Searches: Progress Report #1

Daniel M. Dunlavy

December 18, 2002

1 Introduction

This report details the work completed in the first three months of development of the QCS (Query, Cluster, Summarize) information retrieval (IR) system. In the first section, the project as it was originally proposed is briefly presented. (For a more detailed account of the proposal, an interested reader is directed to the original proposal [Dun].) Sections 3, 4, and 5 give the details of the algorithms implemented in the current QCS system. Implementation issues are discussed in Section 6, highlighting work completed to date on the QCS system, a user interface to the system, and accompanying software tools created or utilized in facilitating the development of the QCS system. As with many software projects of this scale, some of the ideas originally proposed for the QCS system in [Dun] have been phased out for various reasons. The specific changes and the rationale behind these changes are discussed in Section 7. Finally, a plan for the completion of Version 1.0 of the QCS IR system is presented in Section 8.

2 The Proposed QCS System

Conducting scientific research most often involves a search through existing literature in order to avoid repeating research efforts, review methods already developed for solving a problem, gain a better understanding of a problem, etc. Typically, this search is performed using the Internet, which is a convenient portal to various databases of books, journal articles, technical reports, preprints, etc.

In using this approach for IR, a researcher has the advantage of being able to search through great amounts of reference material. However, along with this great access comes the challenge of efficiently retrieving and processing only relevant material. When using an IR engine to search through electronic resources, simple queries can return too many documents or documents not relevant to the intended search criteria.

For example, if a physicist is researching which methods have been used to solve problems in the area of plasma physics, a search for “*methods plasma physics*” on the World Wide Web using Google (www.google.com) yields more than 27,000 documents (out of approximately 2.5 billion, circa September 2002). The physicist could never hope to read through all of those documents. Even searching arXiv (xxx.lanl.gov), a server for preprint articles in physics, for this same information on plasma physics yields more than 350 documents (out of approximately

135,000, circa September 2002). Still, this is too much information to process. Just reading the abstracts of those 350 articles would require a great deal of time, and there is no guarantee that this document set would be representative enough to constitute a thorough literature search.

With these issues in mind, the goal of QCS project is to develop an IR system that

- retrieves documents relevant to a query,
- separates the retrieved documents into clusters by topic, and
- creates a single summary document for each topic cluster.

Ideally, such a system would facilitate more efficient literature searches. By categorizing and summarizing the set of documents that best match a researcher’s query terms, a large amount of information can be presented which is reduced in size and organized in a categorical hierarchy.

The specific computational methods proposed to carry out the tasks of retrieving a set of documents that best match a query, clustering a set of documents by topic, and creating a summary of multiple documents are Latent Semantic Indexing (LSI) [DDL⁺90], Spherical k -Means [DM01], and a hidden Markov Model (HMM) coupled with a pivoted QR algorithm [SOC⁺02], respectively. The details of these algorithms are presented in the following three sections.

3 Querying a Document Set

3.1 The Vector Space Model

A set of documents can be represented using an $m \times n$ term-document matrix A , where m is the number of terms and n is the number of documents in the set. Although a term can be defined in several ways, terms in the QCS project represent the words (white space delimited) in a document with the exception of pre-designated *stop words* (e.g. “a”, “the”, “in”, etc.).

The value of an entry of the matrix A is a product of three scaling terms:

$$a_{ij} = \tau_{ij} \cdot \gamma_i \cdot \delta_j \quad (i = 1, \dots, m; j = 1, \dots, n) \quad (1)$$

where τ_{ij} , γ_i , and δ_j , are the *local weight*, *global weight*, and *normalization factor*, respectively. These parameters are chosen so that the value a_{ij} best represents the importance (or weight) of term i in document j for a particular document set. The j^{th} column of A , a_j , is commonly referred to as the *feature vector* of document j .

The various scaling schemes used in the present QCS system are presented in Table 1 (Kolda and O’Leary present a similar table of schemes in [KO98] along with the original references for each scheme). The values f_{ij} and f_i are the number of times term i appears in document j and the number of times term i appears in the entire document collection.

The local *binary* weighting is used when it is important whether or not a term appears in a document (as in the case with a document set with very little overlap in terms across the document set), whereas the *log* weighting would be used to damp the effects of large differences in term frequencies within a single document.

The purpose of using a global weighting scheme is reduce the weight of terms that occur frequently within a document or across several documents while giving a greater weight to terms that occur infrequently. An interested reader can follow the reference links in [KO98] for the theoretical development of the global weighting schemes, as this is beyond the scope of the current report.

Local Weights (τ_{ij})		
t	Term Frequency	f_{ij}
b	Binary	$\chi(f_{ij}) = \begin{cases} 0 & f_{ij} = 0 \\ 1 & f_{ij} > 0 \end{cases}$
l	Log	$\log(f_{ij} + 1)$
Global Weights (γ_{ij})		
x	None	1
n	Normalized	$(\sum_i f_{ij}^2)^{-1/2}$
f	Inverse Document Frequency (IDF)	$\log\left(\frac{n}{\sum_j \chi(f_{ij})}\right)$
F	IDF Squared(IDF2)	$\log\left(\frac{n}{\sum_j (\chi(f_{ij}))^2}\right)$
e	Entropy	$1 - \sum_j \frac{p_{ij} \log(p_{ij})}{\log n}$ with $p_{ij} = f_{ij}/f_i$
Normalization (δ_{ij})		
x	None	1
n	Normalized	$(\sum_i (\tau_{ij} \gamma_i)^2)^{-1/2}$

Table 1: Scaling factors for a term \times document matrix

And finally, the normalization factor is used to remove any bias based on document size by scaling each document feature vector (columns of the term \times document matrix A) so that each has unit length in the Euclidean norm.

Throughout this report, all mentions to a particular scaling scheme triplet will be represented by a three letter code using the symbols in Table 1. This reflects references to the various schemes seen in the literature, with a single exception: *tfx* is sometimes referred to as *tf.idf*. This is typically the scheme that researchers use to derive performance baselines for new methods.

The representation of documents as described above (barring specific choices for what constitutes a term and scaling factors) is a *vector space model* of the document set and is the most widely used model for representing documents in IR.

3.2 Uncovering the Latent Semantic Structure of Documents

The choice of method for uncovering the latent semantic structure (or implicit higher-order structure) in the association of terms and documents is LSI. The LSI algorithm attempts to reveal implied relationships and remove term ambiguity, while preserving the most characteristic features of each document.

LSI attempts to accomplish this using the singular value decomposition of the term \times document matrix, A :

$$A = U\Sigma V^T = \sum_{i=1}^r \sigma_i u_i v_i^T \quad (2)$$

where U and V are matrices, each with columns that form orthonormal sets (u_1, \dots, u_n and v_1, \dots, v_n , respectively), and Σ is a diagonal matrix with monotonically decreasing positive

values $(\sigma_1, \dots, \sigma_r)$ along the diagonal. As is typical, it is assumed that $m \geq n$, i.e., that there are more terms than documents, and that $\text{rank}(A) = r$, where r is the number of unique documents in the document set.

The truncated SVD can be used to approximate A using a rank- k matrix:

$$A \approx A_k = U_k \Sigma_k V_k^T = \sum_{i=1}^k \sigma_i u_i v_i^T \quad (3)$$

which is the *best* rank- k approximation to A in the sense that

$$\|A - A_k\| \leq \|A - B_k\| \quad \forall B_k \text{ s.t. } \text{rank}(B_k) = k \quad (4)$$

where $\|\cdot\|$ is any unitarily invariant norm.

The choice of k depends on the size and nature of the document set, as well as the spread of terms throughout the individual documents and the entire document set, and is the number of *derived factors* used to represent each of the documents. The columns of U_k represent the derived term vectors, and the columns of V_k represent the derived document vectors. By using the truncated SVD, we have a means of capturing the underlying semantic structure of the document set while reducing the noise and variability. Furthermore, terms occurring in similar documents (documents containing many of the same words) will be close together in the k -dimensional factor space even though they may never appear in the same document. It is not uncommon to use values of $k \approx 100$ for document sets with more than 1000 documents using greater than 5000 terms as features.

Example 3.2.1. (Taken from [BDO95].) Consider the words *car*, *automobile*, and *elephant*. The terms *car* and *automobile* are synonymous, with *elephant* being an unrelated term. If none of these terms occur in the same document, then in a search for *car*, documents containing the term *automobile* have the same likelihood of being returned as documents containing the term *elephant*. However, the terms *car* and *automobile* will be close to each other in the k -dimensional factor space under the very likely assumption that they co-occur in documents with the same terms (e.g., *motor*, *engine*, *vehicle*, *model*, etc.). Therefore, using the truncated SVD, LSI increases the likelihood that documents containing the word *automobile* should be returned for queries containing the term *car*.

3.3 Matching Documents to Queries

A query can be represented in exactly the same manner as documents in the vector space model, i.e., using a query vector, $q \in \mathbb{R}^m$. A query is typically much more sparse than document vectors (contain far fewer terms than an average document) and does not necessarily use the same scaling scheme. Thus, a six letter symbol must be used to describe a vector space model used for query-based IR. For example, experiments using the scaling *lxn.bpx*, *lxn* for the documents and *bpx* for the query (where p is a global scaling not used in the QCS system) can be found in [KO98].

Once the query vector has been scaled, we can project the vector q onto the k -dimensional term space using U_k (from the truncated SVD), scale it with the k derived factors using Σ_k (the singular values), and measure how close the query is to each document in the k -dimensional document vector space. A vector of scores, s , can be computed by computing inner products of the projected and scaled query vector with the projected document vectors (columns of V_k^T):

$$\tilde{q} = \Sigma_k (U_k^T q) \quad (5)$$

$$s = \tilde{q}^T V_k^T = q^T U_k \Sigma_k V_k^T = q^T A_k \quad (6)$$

Typically, the query vector and the columns of A have been normalized, and hence, the scores in s turn out to be cosine similarity scores. That is, they represent the cosine of the angle between the projected and scaled query and each document in the k -dimensional document vector space (a space spanned by the columns of V_k^T).

Given a fixed (but usually user-specified) matching tolerance, tol , document j is considered a match to the query, q , if

$$s_j = q^T (a_k)_j > tol \tag{7}$$

where $(a_k)_j$ is the j^{th} column of A_k .

4 Clustering Documents by Topic

Assuming that we now have N documents ($N \leq n$), which are those documents returned from the query tool. We would like to find a partition of those N documents such that the documents in each partition all pertain to a single topic (in the sense that the terms are highly correlated across the documents with a given partition) with each partition corresponding to a *different* topic.

In general, solving this problem could require N partitions (or disjoint sets). However, the documents that we are considering are assumed to all have a very similar underlying semantic structure in k -dimensional factor space, and so there is a greater likelihood that we may indeed require far fewer than N partitions to cluster the documents into topic clusters. In order to discern if there is any topic similarity amongst the N documents, we return to the original m -dimensional vector space, where m is the total number of terms used to create the term \times document matrix A . Thus, we return to the original matrix A to extract the columns corresponding to the N documents of interest.

A clustering of these N documents d_1, \dots, d_N is a partitioning into k disjoint subsets, π_1, \dots, π_k . That is,

$$\bigcup_{j=1}^k \pi_j = \{d_1, \dots, d_N\} \quad \pi_j \cap \pi_l = \emptyset, \quad j \neq l \tag{8}$$

Based on cosine similarity (assuming that the extracted columns of A have been normalized), the *coherence* of the cluster π_j can be defined as

$$\sum_{d_i \in \pi_j} d_i^T c_j,$$

where c_j is the normalized centroid of cluster π_j :

$$c_j = \frac{\sum_{d_i \in \pi_j} d_i}{\| \sum_{d_i \in \pi_j} d_i \|}$$

Aggregating over all of the clusters, we can define the following combined coherence function:

$$\mathfrak{C}(\{\pi_j\}_{j=1}^k) = \sum_{j=1}^k \sum_{d_i \in \pi_j} d_i^T c_j \tag{9}$$

Now we would like to maximize this function to give us a partitioning with the optimal coherence. That is, since the inner product gives a score relating how close two unit vectors are in the m -dimensional term space, we are trying to partition the documents such that the normalized centroids of all the clusters are a minimal distance (in an average sense) away from each of the documents in that cluster. This is the traditional approach to k -means clustering.

An iterative method, called the spherical k -means algorithm (spk-means), for maximizing the combined coherence function in (9) is described succinctly in [DFG01] and detailed in [DM01]. One advantage of using the spk-means algorithm is that it employs an efficient strategy for computing the feature vector similarities (distances), which is the computational bottleneck of the classical k -means algorithm. Moreover, there is a generalized spk-means implementation of this algorithm which maximizes the combined coherence while allowing for adaptive values for k . More details will be presented in the next progress report.

5 Summarizing Topic Clusters

After clustering the N documents into topic clusters, the QCS system creates a single extract summary document for each of the clusters. An extract summary is a set of sentences extracted from a set of documents which represents a summary of all of the documents in that set. Typically, the number of sentences in an extract summary is far fewer than the number of sentences in the original set of documents.

The process of creating a summary document for each of the k topic clusters (where k is the number of partitions in the final iteration of the clustering algorithm) is inherently parallelizable. If the number of topic clusters is greater than the number of processors being used, then the work can be divided among the processors so that each processor has roughly the same amount of topic clusters to process. This is most likely not the optimal assignment of work in the context of load balancing, but it will be the simplest to implement and act as a good starting point for a parallel implementation of the algorithm described below.

Creating a single extract summary document for a set of documents in the QCS system consists of computing the probability that each of the sentences is a summary sentence for the document in which it occurs for each sentence in each document in the set, ordering the sentences in decreasing order of these probabilities (i.e., from the most likely to least likely candidates for summary sentences), and then removing redundant sentences.

A 9-state hidden Markov model, built to extract four lead sentences and supporting sentences, is used to compute the probability for each that it is a summary sentence. For an introduction to the theory behind the use of HMM's, an interested reader is referred to [Rab89]. The underlying Markov chain is constructed using the following features:

- position of the sentence in the document
- number of terms in the sentence
- number of "pseudo-query" terms in a sentence

where pseudo-query terms are the terms in each document that are more likely to occur in that document than the entire document set.

The details of using this HMM to extract summary sentences can be found in [CO01, SOC⁺02]. The details of the use of this HMM will appear in the following progress report, along with a graphical representation of the state space and an example of the HMM trained on a trivial document set.

<i>Language</i>	<i>Task</i>
Perl	Document Pre-processing
Java	User Interface
C/C++	Document retrieval (Q) Document clustering (C) Multiple-document summarization (S)
Java	Summary document formatting Creating dynamic HTML

Table 2: Specific languages being used to develop the QCS system

Once the probabilities that the sentences are summary sentences have been computed, a term \times sentence matrix is formed in a similar fashion to the term \times document matrix used in the LSI algorithm. The entries of this matrix are the term frequency counts scaled so that the Euclidean norms of the columns (sentences) are equal to the corresponding probabilities computed above. In order to remove redundant sentences, the pivoted QR algorithm is used. More details on using the pivoted QR algorithm to extract non-redundant summary sentences can be found in [CO01]. The exact number of sentences extracted depends on the size of the summary a user requests at the time of entering a query.

6 Implementation of the QCS System

Implementation of the QCS system has involved the development and integration of software tools designed to pre-process the documents, provide a graphical user interface (GUI) for the system, execute the algorithms described in Sections 3, 4, and 5, and format and display the summary documents produced by the system. The computer languages that are currently being used to develop these tools are listed in the Table 6 below in the order reflecting the flow of data through the system.

The specific document sets used for testing various components of the QCS system are ASCII-formatted SGML¹ files. Thus, the major task has been to remove the SGML tags and parse the text appearing between useful tags (e.g., `TEXT` contains useful document body text, whereas `COMMENT` does not). One complication that arose in developing the Perl scripts was how to handle nested tags. Certainly, more work could be done in this area, developing better heuristics for preprocessing the documents.

The user interface is accessed through an HTML browser. This choice was made so that the system could be accessed from the widest variety of computing platforms. This was accomplished using the Apache HTTP server, *httpd*, and Apache Java Servlet Container, *Tomcat*, both of which are released under the GNU Public License (GPL) and can be incorporated into and distributed with the QCS system. Using Java servlets over competing technologies (JavaServer Pages, ActiveServer Pages, JavaScript, mod_Perl) allows for the integration of tools built in C/C++, runtime execution within an extremely secure environment, and the ability to have the server create dynamic HTML to send to the user interface. Thus, remote users can connect to and use the QCS system in a secure environment without having to install new software on their computer.

¹Standard Generalized Markup Language

The computational server, handling the querying, clustering, and summarization of the documents, has been implemented (or will be) implemented using several existing libraries and code written in C/C++ and Fortran 77.

- **Query**

Parallel General Text Parser (PGTP), developed by Michael Berry of the University of Tennessee at Knoxville, is currently used to implement LSI. It uses MPICH , from Argonne National Laboratories, for parallel communication and SVDPACK for the SVD of the term-document matrix. The package is written in C++ and released under the GPL.

- **Cluster**

Generalized Spherical K-Means (gmeans), developed by Yuqiang Guan at the University of Texas at Austin, will be used for the clustering of documents. The package is written in C++ and released under GPL.

- **Summarize**

Currently, the only working code for summarizing documents as specified in Section 5 is a set of Matlab scripts, developed by Dianne O’Leary and John Conroy at the University of Maryland at College Park and the Center for Computing Sciences. Approximately 30% of this code has been migrated to C++ to date.

Linear Algebra Package (LAPACK) will be used for the pivoted QR decomposition. LAPACK is written in Fortran 77 and is freely available.

The output of QCS is currently in the form of dynamic HTML documents containing summaries of each topic cluster and hyperlinks to the original documents summarized. Incorporating the LSI cosine similarity scores for each document, several of the most heavily weighted terms taken from the centroids of the k topic clusters, and several of the most heavily weighted pseudo-query words produced during the summarization of the documents (to create more of an abstract than an extract) into the output presented to the user are some of the ideas currently being considered. However, as the motivation for the QCS system came from the fact that most IR systems present too much information, including this additional information would most likely need to be a choice given to a user on an individual basis and controlled through the use of parameters in the user interface.

Finally, an Application Programming Interface (API) has been developed to function as a wrapper around the various functions of the main QCS system. In the future, there may be more efficient or accurate means to solve the type of problems the QCS system is designed to handle. Thus, creating modular code and developing a robust API to allow for future integration with other IR systems has been a major focus in the development of the QCS system.

7 Changes from the Proposed System

There have been some changes to the original plans for the QCS system as detailed in [Dun].

Specifically, the hardware platform has temporarily been changed to a cluster of 16 Sun UltraSparc5 machines with 128Mb RAM, running Solaris 8 and communicating via fast Ethernet (100 Mbps). Several compilation errors prevented successful compilation of the PGTP software on the Linux cluster originally slated to house the QCS system.

Another problem with PGTP is that code has been developed for performing queries for the serial version (GTP), but not for the parallel version. Migrating this code from the serial version to the parallel version has not been straightforward, as there is very little documentation for either GTP or PGTP, crucial components of the interface are fundamentally different in the two versions, and some of the data structures are different in the two versions. To add to this list of problems, the sparse matrix format adopted for the PGTP system is the standard compressed column storage (CCS) format, but the code that stores these sparse matrices deviates from the standard format, creating problems when trying to save a sparse matrix or sharing data with functions outside of the PGTP code it as a parameter to other code.

The gmeans code for clustering documents is not well documented either. Thus, choosing the parameters for the initial partition and the parameters used for allowing the cluster size to be adaptive has been based on trial and error to date. A better understanding of the effects of the parameters on the output of the clustering algorithm will be necessary for better performance.

Finally, the Matlab prototype system for summarizing documents does not contain any functionality for detecting sentence boundaries. All testing of that system has been conducted using commercially licensed sentence boundary detection software.

8 The Road to Version 1.0

The most important task to be completed in the development of the QCS system is the C++ version of the summarization tools. This involves the development of a sentence boundary detection module, and the completion of the single document and multiple document summarization tools. Since each topic cluster contains a disjoint set of documents, the problem of producing a single extract summary document for each topic is inherently parallelizable. The plan is to complete the serial version of the tools first and then extend these tools to a parallel implementation using the message passing interface tool, MPICH, for the parallel communication.

Certainly, once the QCS computational server has been completed, code validation will be the next step. The document sets used for evaluation at the 2002 Document Understanding Conference (DUC02) are still the only set of data that is appropriate for validating the QCS system. However, some researchers believe that this may not be a reliable test set. There are other document sets (MEDLINE: abstracts from the National Library of Medicine, CISI: abstracts in library science from the Social Science Citation Index, and CRANFILED: abstract in aeronautics and related areas) which have been used to measure the effectiveness of query-based retrieval systems as well as clustering based topic categorization. At least one (but hopefully all three) of these document sets will be used for validation of the query and cluster components of the QCS system. For more information about the DUC02 data, see [Dun], and for more information about the other document sets, see [Dum91].

If time permits, implementing a parallel version of the clustering algorithm as described in [DM00] may improve the performance of the QCS system. Certainly, if the QCS system can return results more quickly with little degradation in the quality of the results, more time should be dedicated in the project to make this happen,

Finally, a user manual, installation guide, and website will be developed to document the use, development, and possible future directions for the QCS system. I have learned in the first half of the project (while attempting to incorporate software tools into the QCS system) that without documentation, future work on this system by other developers will most likely prove to be painfully difficult.

Acknowledgements

I would like to thank Dianne O’Leary, my thesis advisor at the University of Maryland and co-advisor for the QCS project, for all of her invaluable help throughout the entire development of the QCS system. I would also like to thank John Conroy, an applied mathematician at the Center for Computing Sciences and co-advisor for the QCS project, for giving me the idea to develop the QCS system and listening to all my ideas of how I wanted to change everything originally proposed.

I would also like to thank David Levermore and Bill Dorland, the instructors and project leaders of the Advanced Scientific Computation course at the University of Maryland, College Park, for which the QCS system was originally developed. Their questions about and ideas for the implementation of the QCS system proved very helpful. Input from my fellow students in the course, Jim Cooley, Darran Furnival, John Harlim, and Tomo Iida has also helped contribute to the development of the QCS system.

Finally, I would like to thank my wife Nancy and daughter Maisy for their patience and support throughout every minute of the development of the QCS system. This project is dedicated to them as they are so dedicated to helping me succeed.

References

- [BDO95] Michael W. Berry, Susan T. Dumais, and Gavin W. O’Brien. Using Linear Algebra for Intelligent Information Retrieval. *SIAM Rev.*, 37(4):573–595, 1995.
- [CO01] John M. Conroy and Dianne P. O’Leary. Text Summarization via Hidden Markov Models and Pivoted QR Matrix Decomposition. Technical report, University of Maryland, College Park, Maryland, March, 2001.
- [DDL⁺90] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by Latent Semantic Analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [DFG01] I. S. Dhillon, J. Fan, and Y. Guan. Efficient Clustering of Very Large Document Collections. In V. Kumar R. Grossman, C. Kamath and R. Namburu, editors, *Data Mining for Scientific and Engineering Applications*. Kluwer Academic Publishers, 2001. Invited book chapter.
- [DM00] Inderjit S. Dhillon and Dharmendra S. Modha. A Data-Clustering Algorithm on Distributed Memory Multiprocessors. In *Large-Scale Parallel Data Mining, Lecture Notes in Artificial Intelligence*, pages 245–260, 2000.
- [DM01] I. S. Dhillon and D. S. Modha. Concept decompositions for large sparse text data using clustering. *Machine Learning*, 42(1):143–175, Jan 2001.
- [Dum91] Susan T. Dumais. Improving the Retrieval of Information from External Sources. *Behavior Research Methods, Instruments, and Computers*, 23(6):229–326, 1991.
- [Dun] Daniel M. Dunlavy. An Information Retrieval System for Improving Efficiency in Scientific Literature Searches. *QCS IR System Project Proposal*. September, 2002.

- [KO98] Tamara G. Kolda and Dianne P. O’Leary. A Semidiscrete Matrix Decomposition for Latent Semantic Indexing in Information Retrieval. *ACM Trans. Inf. Sys.*, 16(4):322–346, 1998.
- [Rab89] L.R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77:257–285, Jan 1989.
- [SOC⁺02] Judith D. Schlesinger, Mary Ellen Okurowski, John M. Conroy, Dianne P. O’Leary, Anthony Taylor, Jean Hobbs, and Wilson Harold T. Wilson. Understanding Machine Performance in the Context of Human Performance for Multi-document Summarization. In *Proceedings of the Workshop on Automatic Summarization*, 2002. Volume 2: Draft DUC Papers.