Computing Related Terms in a Corpus of Documents

James W. Cooper and Herbert A. Chong
IBM T J Watson Research Center, PO Box 704, Yorktown Heights, NY 10598
jwcnmr@watson.ibm.com
herbchong@compuserve.com

Abstract

We describe the practical details of a method for computing relations between co-occurring terms in a collection of documents based on their frequency and proximity using mutual information and other statistical measures to rank their importance. In this work, we used a part of speech tagger and shallow parser to obtain noun phrases and verb groups. This approach also allows us to recognize noun-verb-noun triples and relationships deduced from an abbreviation detection module.

1. Introduction

The problem of finding important and relevant documents in an online document collection becomes increasingly difficult as documents proliferate. Our group has previously described the technique of Prompted Query Refinement and Lexical Navigation [1] to assist users in focusing or directing their queries more effectively. However, even after a query has been refined, the problem of having to read too many documents still remains.

We have also previously reported the details of the "Avocado" summarization system we developed for producing rapid displays of the most salient sentences in a document. [2].

Users would prefer to read or browse through only those documents returned by a search engine that are important to the area they are investigating. One way to find related documents of interest is to start with an important term and ask the system what terms are related to that term. Then, by using these related terms to focus your query or by asking which documents show that relation, you can quickly narrow the number of documents you need to read in detail. This is, in fact, the essence of Lexical Navigation. [1]

We report here a new approach to the computation of relations that does not directly rely on the collection level statistics required by previous approaches[1]. Further, this method is completely scalable because it uses a relational database for term storage. In this new approach, we accumulate all of the terms, along with their document keys and paragraph and token offset values into a single DB2 table. It is then possible to obtain all of the unnamed relations using a single DB2 query.

2. Background

Finding documents in a collection is a well-known problem and has been addressed by any number of commercial search engine products, including Verity, IBM Intelligent Miner for Text, and Google.

There have been a number of approaches to solving document retrieval problems in recent years. For example, Fowler [6] has described a multi-window document interface where you can drag terms into search windows and see relationships between terms in a graphical environment. Local feedback was used by Buckley [7] and Xu and Croft, [1] who also utilized local context analysis using the most frequent 50 terms and 10 two-word phrases from the top ranked documents to perform query expansion. Schatz *et al*.[8] describe a multi-window interface that offers users access to a variety of published thesauri and computed term co-occurrence data.

3. The Talent Toolkit

In approaching these document retrieval problems, we have applied a number of technologies developed by our project. In particular, we used the suite of text analysis tools collectively known as Talent (Text Analysis and Language Engineering Tools] for analyzing all the documents in the collection.

Talent is a chain of tools for recognizing multi-word terms and proper names. It reduces related forms of a term to a single *canonical form* that can then be used in computing term occurrence statistics more accurately. In addition, it recognizes abbreviations and finds the canonical forms of the names they stand for.

Using occurrence statistics, it is possible to compute a collection-level importance ranking called IQ or Information Quotient [11]. IQ is effectively a measure of the document selectivity of a particular term: a term that appears in only a few documents, and appears in some fo them more than once, is highly selective and has a high IQ. On the other hand, a term that appears in many documents is far less selective and has a low IQ. IQ is measured on a scale of 0 to 100, where a value of X means that X% of the vocabulary items in the collection have a lower IQ.

We refer to two versions of this Talent software package in this paper, Textract 4.1, which was completed in 1999, and Talent 5.1, which is a complete object-oriented revision of this system with facilities for adding more linguistic plug-ins which operate on a stream of document annotations.

The Textract 4.1 research program we have described previously computes all of these statistical parameters by accumulating them into dynamic tables while each document is processed. This system has an upper limit to scalability, and in this paper we describe a more scalable system, using the new Talent 5.1 package.

Features of Talent 5.1

The Talent package is a new version of the text mining system, which concentrates almost entirely on document level recognition rather than collection-level recognition, but is organized as a series of plug-ins, each of which carries out specific operations on the document text. Like the previous versions, Talent discovers names [4] and technical terms[5] and assigns them to categories as shown in Table 1.

Table 1 –Categories assigned to terms by Talent

UWORD	Unknown word
UTERM	Unknown term
UABBR	Unknown abbreviation
UNAME	Unknown type of name
PLACE?	Probably a place
PERSON?	Probably a person
PLACE	A place
PERSON	A person
ORG	An organization
CARDINAL	An integer value
DATE	A date pattern
CURRENCY	A currency pattern

Additional operations in Talent 5.1 include part-of-speech tagging, shallow sentence parsing and abbreviation recognition.

The Talent system *annotates* each document. The annotations consist of the words, phrases, sentences and paragraphs that make up the document. They include parts of speech, sentence parts, and noun types. The annotations are thus nested and can be represented as a tree.

The Talent annotation list contains several types of information:

Document – Sentence and paragraph boundary information.

Syntax – The part of speech of each term, including the term offset boundaries.

Sentence parsing information – Parts of the sentence, including subject, verb, object, noun phrase, and active or passive verb group.

Vocabulary – The word itself, including sentence offset.

4. Building a Java Interface to Talent

To make it easier to access the Talent system, we developed a Java interface to a library version of Talent. This JTalent package is a set of Java classes that allow us to write relatively simple Java programs to create and examine annotation lists in Java.

The connection between these components is illustrated in Figure 1. The JTalent classes all make a connection through a TalentCalls class which contains native declarations of the C functions in the JTalentDLL. Then the JTalentDLL calls various functions in the simplified TAL_API layer that then makes calls directly into Talent.

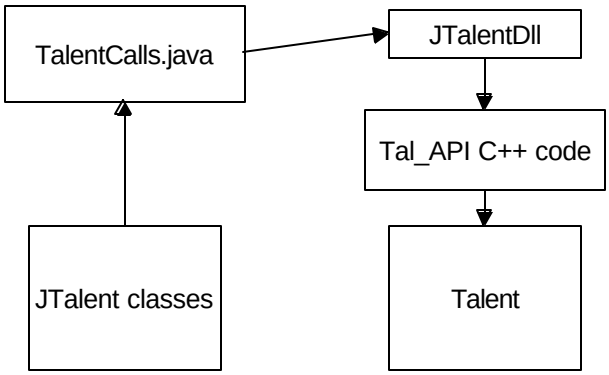

Figure 1 - The JTalent connection to the Talent library

We started by designing a set of Java objects that represented the abstractions of Annotation lists and Vocabulary lists at a high level, and then designed the API calls to support the construction of these objects from the underlying Talent objects. Then we designed the Java JNI [15] layer to communicate with the C++ layer. Essentially the JNI layer decomposes the C++ objects into fundamental types (integer, string, float, etc.) and the Java code reconstructs Java objects from these fundamental types.

Figure 2 shows the shallow parse tree for a small document on the left side where the largest unit is a sentence and the smallest noun and verb groups. The right panel shows terms discovered within the document by the term recognition plug-ins. This figure represents a GUI interface written in Java which calls the JTalent library to obtain the annotation list.

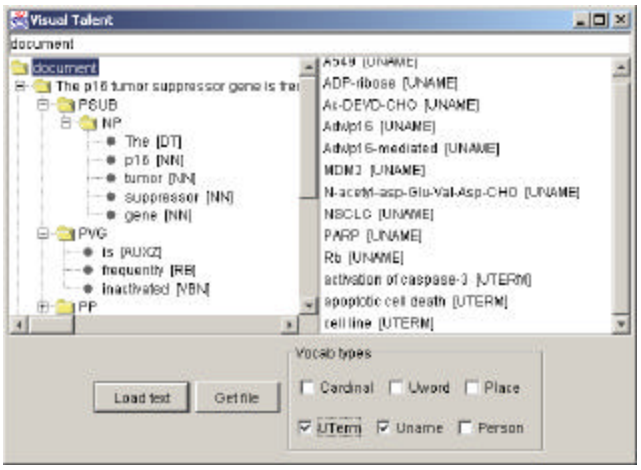

Figure 2- A visual representation of Talent output.

Computing Unnamed Relations

We can use the information in these annotations to compute unnamed relations more directly and in a more scalable fashion than in Textract 4.1. The unnamed relations computed by Textract 4.1 are a measure of the how frequently pairs of terms occur near each other. If salient terms are found near each other on a recurring basis through a number of documents, the strength of the relation is quite high. If, on the other hand, the terms are literally found in every document, the relation may be strong, but the salience of the terms is much reduced. Thus, we should only report relations between salient terms, or terms which have an IQ above a selectable threshold.

The problem of searching term by term through a collection for terms which co-occur is one which intrinsically is compute-bound, because it would seem to require about $(n/2)^2$ comparisons. It is this computational bottleneck that this new approach seeks to reduce.

The Textract 4.1 system performed its collection-level computations using a series of increasing memory buffers and a few files. This approach was not further scalable, and the approach of Talent 5.1 has so far to concentrate on improving the sophistication of phrase detection in individual document processing. Accordingly, we developed a new, scalable approach for the computation of these relations using the Talent annotation list.

5. Building a Database Interface

The fundamental breakthrough that allows scaling of the relations computation to a collection of any size is the use of a relational database for storing most of the parsed document information. Using the Java JDBC connection [16], we wrote a set of classes to create and manipulate database tables at a high level. This set of classes were named KSS (for Knowledge System Server) and have been described previously [13]. We created a series of tables in a DB2 database and wrote code using JTalent to create load files for those tables [14]. These tables are described below.

A Document-Terms Table

Using the syntax and vocabulary annotation items it is possible to compute the paragraph, sentence and offset of every multi-word phrase in the document and store it in a database. In these experiments we generated a DB2 load file and loaded it after a large number of documents were processed. This table then contains the major multiword terms in each sentence in the document along with the computed paragraph, sentence and offset information.

A Unique Terms Table

We can then issue a simple SQL query against the DocTerms table to generate a list of unique terms and compute their frequencies in the document collection. This is done using a query of the form

```
Insert into TermList (Term, Numdocs)
SELECT DOCTERMS.TERM,
Count(DOCTERMS.DOCKEY) AS
CountOfDOCKEY
    FROM DOCTERMS GROUP BY
DOCTERMS.TERM;
```

We can also compute the number of documents a term appears in and the number of documents a term appears in more than once using the query

```
SELECT Count(*) AS DocCount,
      DOCTERMS.TERM, DOCTERMS.DOCKEY
      FROM DOCTERMS GROUP BY
DOCTERMS.TERM, DOCTERMS.DOCKEY
      ORDER BY DOCTERMS.TERM;
```

By looping through the results, we can compute the NumDocs and NumDocsGT1 columns.

We can then compute a measure of the salience of each term in the collection, based on the term's frequency, the number of documents it appears in and the number of documents it appears in more than once. This measure, developed by Prager [10], is called the term's IQ or Information Quotient.

We now have a table of all the terms in the collection along with the frequencies and IQs. However, in order to compute relations effectively, we can reduce the number of database joins if we copy the IQs, frequencies and Term keys back into the TermDocs table. Fortunately, this also can be done in a single SQL update-select statement.

6. Computing the Unnamed Relations Table

We now have all the data we need to compute all the unnamed relations in the collection *in a single SQL statement!* This is much faster than the previous approach and it is easily tuned for the nature of the collection.

Basically we look for all instances of pairs of terms that occur within a set number of token positions from each other, and which have an IQ greater than a pre-selected minimum. Since we store paragraph and sentence information, we can use this to further restrict the relationships we return if this seems desirable. In our initial experiments we look for all pairs of terms that occur within 20 tokens of each other. After some experimentation, we settled on a formula where the relations must occur in the same paragraph and sentence, and the terms must have IQs greater than a pre-selected but collection-dependent value. We also restricted the number of relations further by only using terms with a frequency greater than 2, and a maximum distance of 6 tokens from each other.

In detecting multiword terms generated by the shallow parser in Talent, we store both noun phrases (NP) and active and passive verb groups (VG, and PVG). However, for computing unnamed relations, we further restrict our query to use only nouns and noun phrases.

This unnamed relations query produces all occurrences of pairs of terms. However, the query is sufficiently complex that it is not efficient to sort the huge number of terms into order in the same query. Instead, we insert the results of this query into a temporary table and then sort the results by term key in a second query to gain further speed.

To determine the strength of the relation, we need only go through this new result table and count these occurrences, which are now adjacent in the sorted result.

Computing the Mutual Information Value

We compute the mutual information value, or strength of relation by

$$mutInfo = \log\left(\frac{(totalTerms * pairCount)}{(freq1 * freq2)}\right)$$

where

- totalTerms is the number of terms in the collection
- pairCount is the number of times the terms appear as pairs
- freq1 and freq2 are the individual term frequencies

Note that the strength of unnamed relations grows with the number of times they co-occur in the collection.

These data are written into a preliminary DB2 load file and the maximum value of the mutual information relation determined. Then we rewrite this file to scale these values to lie between 0 and 100. We can eliminate those with a strength below some predetermined threshold, say 50, and write the rest of these pairs and their strengths into the final DB2 load file to be loaded into a Relations table for rapid computation of relations in information mining.

7. Named Relations

Named relations are relations detected by simple parsing of the text to determine relations between noun phrases. There is no weighting in our calculation for named relations that occur more than once.

In this version of our code, we use the output of Talent's abbreviator module to find "same-as" relations between an abbreviation and its expansion.

A number of such relations are shown as a result of a database query in Figure 3. One such abbreviation (NO is the same as nitric oxide) is shown in Figure 4.

Anatomy and Developme	AD	100	same as
C02 Acquisition Membra	CAM	100	same as
Dendritic Alloy Solidifica	DASE	100	same as
Science Applications Int	SAIC	100	same as

Figure 3 - Named relations found as abbreviations.

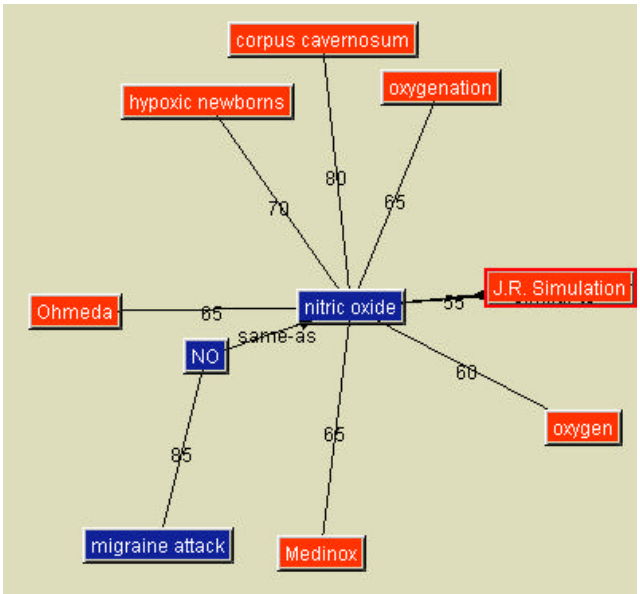

Figure 4 - A Lexical Network of Relations

8. Computing and Displaying Relations

Figure 4 shows a network of unnamed and named relations. The weight of unnamed relations is shown over the arrows, and where the relations have been assigned names, the name of that relation is shown over the arrow. Such network drawings can be initiated by finding the salient terms in a single document and suggesting them as entry points for exploring the term relations space. We have called this exploration process Lexical Navigation.

The Relations table consists of

- Left term
- Right term
- Relation strength (0-100)
- Relation name
- Left IQ
- Left frequency
- Right IQ
- Right Frequency

To compute relations that might be found to any term, we simply query the Relations table for any right terms related to a given left term, above a given strength. To filter the result more, we can limit the results to those above a given IQ or frequency, as well.

9. Computing More Types of Relations

The Textract 4.1 system computed named relations based on appositives, parentheticals and comma-separated lists. [6]. Detection of these features has not yet been completely implemented in the Talent 5.1 system, but appositives are found and marked correctly by the shallow parser. Consider the sentence:

Lou Gerstner, CEO of IBM, said today that Sam Palmisano would be his successor.

This sentence is parsed as shown in Figure 5. The parse tree [17] shows an appositive phrase (CNP) and the noun phrases (NP) that comprise it.

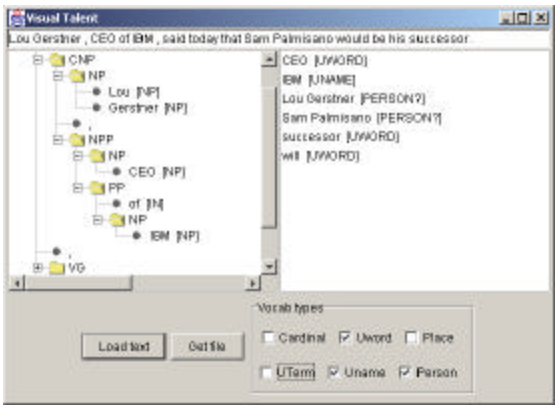

Figure 5 - Parsing of an appositive phrase using JTalent.

Noun-Verb-Noun Groups

In addition, with the information on term occurrence and proximity, we can find other useful types of named relations directly.

Since Talent produces a complete parse of every sentence in the document, we can use this information to deduce subject-verb-object relations as well. We simply write an SQL query to give a list of noun phrases separated by a n active verb group (VG), where the token distance is less than, say, 6 tokens. Such a query has the form

```
SELECT a.TERM, b.TERM, c.TERM,
a.DOCKEY, b.offset-a.offset AS Expr1,
c.offset-b.offset AS Expr2, a.DOCKEY
FROM docterms AS a, docterms AS b,
docterms AS c
WHERE (((a.DOCKEY)=[b].[dockey] And
(a.DOCKEY)=[c].[dockey]) AND
(([b].[offset]-[a].[offset])<6) AND
(([c].[offset]-[b].[offset])<6) AND
((a.PARAGRAPH)=[b].[paragraph] And
(a.PARAGRAPH)=[c].[paragraph]) AND
((a.SENTENCE)=[b].[sentence] And
(a.SENTENCE)=[c].[sentence]) AND
((a.OFFSET)<[b].[offset]) AND
((b.OFFSET)<[c].[offset]) AND
((a.TERMTYPE)=1) AND ((b.TERMTYPE)=2)
AND ((c.TERMTYPE)=1) AND
((Val([a].[term]))=0))
ORDER BY a.DOCKEY;
```

A representative example of the results of this query is shown in Figure 6.

an expression vector	harboring	murine ZPK cDNA
These findings	suggest	ZPK gene
NIH 3T3 fibroblasts	were	transfected
cell growth	were	an expression vector
ZPK	exhibits	cell growth
ZPK	is	recently described serine/threon
PG	is	recently described serine/threon
NIH 3T3 fibroblasts	were	an expression vector
cell growth	were	transfected
that ZPK	could play	development
mouse brain sections	revealed	specific association
mouse brain sections	revealed	ZPK mRNA
various organ systems	including	stomach

Figure 6- Result of a query for finding noun phrases separated by a verb group.

"Is-a" Relations

If we can detect noun-verb-noun groups by a simple SQL query, we can further restrict these queries to those separated by parts of the verb "to be." We can store these as "is-a" relationships. Some partially filtered results are shown in Figure 7. Clearly some additional filtering is desirable here, but the algorithms are fairly plain.

chromosome 1	is	cause
anhidrosis	is	an autosomal recessive hereditary
gp110proto-trk	is	mature form
human trk locus	is	novel tyrosine kinase cell surface r
gp110proto-trk	is	further glycosylated
This molecule	is	mature form
This molecule	is	further glycosylated
its primary translational product	is	110,000-dalton glycoprotein

NIH 3T3 fibroblasts	were	an expression vector
NIH 3T3 fibroblasts	were	transfected
cell growth	were	an expression vector
cell growth	were	transfected
Residues 1	were	trk oncogene

Figure 7 - Noun groups separated by "is" and "were"

10. Incremental Updates

It is not necessary to reload the Document-Terms table completely when more documents are processed. However, we do need to load any new terms into this table. We can then add these new terms to the unique Terms table, but need only compute the IQs of any new terms by computing them only for the rows that have been added. Then we recompute the relations as before. Since there is no requirement that we reanalyze past documents or recomputed old term IQ values, this system has fewer scalability limits. However, once the collection has changed markedly in size, it is advisable to recompute all of the IQs to make sure that they have not changed markedly with the advent of a significant number of new documents.

11. Summary and Conclusions

We have constructed a Java system based on JNI, JDBC along with the Talent text mining system that allows us to find the major multi-word terms in a collection of documents and compute how strongly the terms are related base on their repeated proximity throughout the collection. Since the system uses a relational database, it is highly scalable. We propose new methods of finding types of named relations based on abbreviations, and noun-verb-noun groupings.

12. Acknowledgements

We want to thank Bhavani Iyer for helping develop some of the more complex database queries, and Mary Neff, Roy Byrd and Bran Bogaraev for technical support in developing the Java code that interfaces to Talent. We also want to thank Bob Mack and Alan Marwick for their continuing support of our research.

13. References

1. Cooper, James W. and Byrd, Roy J. "Lexical Navigation: Visually Prompted Query Expansion and Refinement." Proceedings of DIGLIB97, Philadelphia, PA, July, 1997.

2. Neff, Mary S. and Cooper, James W. 1999a. Document Summarization for Active Markup, *in Proceedings of the 32nd Hawaii International Conference on System Sciences*, Wailea, HI, January, 1999.

3. Byrd, R.J. and Ravin, Y. "Identifying and Extracting Relations in Text," *Proceeedings of NLDB 99*, Klagenfurt, Austria.

4. Ravin, Y. and Wacholder, N. 1996, "Extracting Names from Natural-Language Text," IBM Research Report 20338.

5. Justeson, J. S. and S. Katz "Technical terminology: some linguistic properties and an algorithm for identification in text." *Natural Language Engineering*, 1, 9-27, 1995.

6. Fowler, Richard H., Wilson, Bradley A., and Fowler, Wendy A.L. "Information Navigator: An information system using networks for display and retrieval." Report NAG9-551, No.92-1. Department of Computer Science, University of Texas, Pan American, Edinburg, TX.

7. Buckley, C., Singhal, A., Mira, M & Salton, G. (1996) "New Retrieval Approaches Using SMART:TREC4. In Harman, D, editor, Proceedings of the TREC 4 Conference, National Institute of Standards and Technology Special Publication.

8. Schatz, Bruce R, Johnson, Eric H, Cochrane, Pauline A and Chen, Hsinchun, "Interactive Term Suggestion for Users of Digital Libraries." *ACM Digital Library Conference, 1996.*

9. Xu, Jinxi and Croft, W. Bruce. "Query Expansion Using Local and Global Document Analysis," *Proceedings of the 19th Annual ACM-SIGIR Conference*, 1996, pp. 4-11

10. Prager, John M., Linguini: Recognition of Language in Digital Documents, *in Proceedings of the 32nd Hawaii International Conference on System Sciences*, Wailea, HI, January, 1999.

11. Cooper, J.W. and Prager, John M.,"Ant-Serendipity: Finding Useless Documents and Related Documents," in *Proceedinds of the 33rd Hawaii International Conference on System Sciences*, Wailea, HI, January, 2000.

12. Cooper, J. W. "The Technology of Lexical Navigation," Workshop on Browsing Technology, *First Joint Conference on Digital Libraries*, Roanoke, VA, 2001.

13. Cooper, J.W., Cesar, C., So, Edward, and Mack R. L., "Construction of an OO Framework for Text Mining," *OOPSLA*, Tampa Bay, 2001.

14. Cooper, J W, "Loading Your Databases," *JavaPro*, May, 2000.

15. Liang, Sheng, *TheJjava Native Interface*, Addison-Wesley, 1999.

16. White, S., Fisher, M., Cattell R., Hamilton, G. and Hapner, M., *JDBC API Tutorial and Reference*, Addison-Wesley, 1999.

17. Bird, Steven and Liberman, Mark, 199. "Annotation graphs as a framework for multidimensional linguistic data analysis." In *Towards Standards for Discourse Tagging, 37th Annuanl Meeting of the Association for computational Linguistics*, 1-10, Baltimore, MD.