Safeer Ladha

Evaluation of a PC-based LSI search engine

Dr. Berry

Computer Science 411

April 8, 1998

Michael W. Berry
Department of Computer Science
University of Tennessee
Knoxville, TN 37996-1301

Searching through a database for relevant information is a nontrivial task that is undertaken by millions of people everyday. The search for information is often tedious and time consuming. For many reasons, the user may not be able to access the information he or she is looking for. Some databases are restricted in vocabulary, so certain terms have to be used to find certain information. If the user is unable to express his or her query appropriately, the information will go unfound. Even if the vocabulary is not restricted, different relevant documents use different vocabulary, and thus require different search terms to uncover them. Though the document could easily have synonyms of the terms in the query, only the terms in the query are searched for, and the document may not be returned as relevant even though it is. Mixed in with the relevant documents that are retrieved, there will be documents that have terms in the query but are not what the user intended since many terms have more than one definition ("Latent Semantic . . . TREC-2", 1).

For instance, a query on "light waves" could retrieve documents on physics papers concerning light and waves, or stories concerning ships on the water following the light from the North Star. Users may have trouble finding information if they are unable to express the query with enough important terms to find the relevant documents. Latent Semantic Indexing (LSI) overcomes these problems.

LSI uses a multidimensional space to place all the documents and terms. The underlying topics and meaning of the document will be encoded in a vector, and that vector contains the coordinates in the multidimensional space. Since each individual term has little impact on the entire document, but the combination of all the terms determines the document's place in the vector space, LSI can usually get an accurate assessment of the topic of the document. So, a document will may not rank as high if LSI has determined that the topic of that document and the topic determined from he query are different, even if the document contains the terms from the query. The document's rank will be higher than if it did not have any terms from the query, but the other terms in the document pull the rank of the document down since it inherently has non-relevant terms in it since it is a non-relevant document. Usually, this does not happen since the terms in a document place where the document is in the vector space, and the documents a term is in help place where that term is in the vector space. Thus, problems such as the need to use synonymous terms to retrieve different relevant documents in one topic will be fixed. LSI will also lower a document's rank if it is not relevant but has key terms in the query ("Latent Semantic . . . TREC-3", 1-2).

In this thesis, the results of the LSI search engine will be tested with varying *k*s against a Lexical search engine. The database consists of 619 documents. Under the guidance of Dr. Berry during the Fall of 1997, Computer Science 460 collected these documents from the Bureau of Labor Statistics, and several other government organizations. The collection was parsed for bad html code, copied into one file separated by blank lines. The collection of 619 documents is a very small set to do testing on, but the LSI engine and search application can be run on any database. Dr. Berry then chose 5838 terms to use for searching probably by creating a stop word list, and then adding all terms not found in the stop word list and found in the documents to the terms list.

Lexical matching search engines retrieve documents that include the most terms entered in the query. Though, these search engines have become more effective with increased processing power, and can search the entire document for the query terms, this also allows terms in the document unrelated to the subject of the document to be retrieved. For example, some techniques were used to help users find relevant documents. Search engines can add synonyms to the query to make sure it has a chance to retrieve as many relevant documents as possible. However, the increased terms bring more non-relevant documents for the user to search through. Some search engines ask the user which synonyms to add to the query to help with the search, thus decreasing the number on non-relevant documents retrieved ("Using Latent Semantic . . . ", 2). However, this adds considerable time to the search.

Latent Semantic Indexing puts documents in a multidimensional space depending on the terms in the document. Documents that have many terms in common will end up closer together in the multidimensional space than documents that don't have many terms in common. This way, the documents can be separated not in subsets but into an analog space. Similar documents won't have all terms in common, but will have more terms in common than documents on different topics. When the user enters a query, the search engine projects the query into the multidimensional space, deems the documents closest to it as relevant, and returns them.

The number of dimensions that is chosen for the dimensional space is very important. As many dimensions as wanted up to the number of documents can be computed beforehand, and then, at search time, the user can choose how many of these dimensions to use. Higher dimensions allow the searches to be more literal. When k is near the number of documents, the search is basically a Boolean search. As rank k gets smaller, the search becomes more abstract. A rank of about two to three hundred *k* is desirable because it puts similar documents close together but not so close that non-similar documents end up clustered together. For the calculations, a maximum *k* of 300 was chosen, so for subsequent searches, *k*s ranging from 1 to 300 can be chosen. Any *k* below 50 is useless, since documents with different will cluster together in the vector space. For my testing with only 619 documents, a *k* of 50 may not be too low. For *k*s under 25 though, many documents will have vectors that have no data and they will be at the origin in the vector space. These documents will have no topic and the user will not be able to be retrieved them with any search.

Important terms are determined by a function that weights each term on its frequency through the collection and its frequency through each document. A term that is in each document approximately the same number of times should have a lower weight since it disambiguates the documents poorly. In addition the more documents a term is in the lower its weight should be. The more documents a term is in, the less LSI discriminates between documents when that term is in a query. Therefore that term should not have as big an impact on the query as terms that are less frequent or have different frequencies through the documents.

Methods

To create the multidimensional space, many steps are needed. A list of important terms must be created. This can be done electronically by searching the database and add to the term list any term that is not in the *stop word list* or not allowed by a *stop word function*. A *stop word list* is a list that encompasses the words that should not be indexed. This usually includes articles such as *the* and *a*, pronouns, adverbs, HTML tags, and other terms that have very little contribution to the underlying subject of a document. A stop word function does the same thing as a stop word list, but uses a function to remove words that are added to the term list. It may remove words that are only one or two letters, numbers or words with numbers in them.

Once the list is created, the database that is going to be indexed must be parsed through to determine the frequency of each term in each document; this information is used to create a term-by-document matrix of these frequencies. Next, the weights must be computed. There are two different weights; there is a weight for each term, which is the global weight, and a weight for each term within a document which is called the local weight. There are several different ways to compute these weights.

The simplest computational method is to make the local weights can be the frequencies themselves. This has the drawback that longer documents are more likely to have more terms, but those documents should not have a higher weight just because of the document's length. To get around that, the local frequencies can be binary, 0 or 1, based on whether or not the term is found in the document. The local weights can also be computed by the ln(frequency + 1). The plus assures that the local weight is not undefined when the frequency is zero. A fourth method is a little more complicated. It involves averaging the binary frequency with a normalized frequency. For the testing, the natural logarithm was used for the calculations ("A Semi-Discrete Matrix . . . ", 2-4).

Global weights are optional. They can either be left as one, so the calculations will not be altered by global weights or one of two formulas can be used. The Inverse Document Frequency weight is calculated by log(total_number_of_documents)/(number_of_documents_term_is_in) for each term to get is global weight. This scheme is similar to the Probabalistic Inverse. It is ln(number_of_total_documents -

number_of_documents_term_is_in)/(number_of_documents_term_is_in). That quantity can also be expressed as the Inverse Document Frequency and subtracting one before the natural logarithm is taken. Thus, it is understandable that the two weighting schemes produce similar results. Both weighting schemes give higher weightings to terms that appear in fewer documents, and lower weightings to terms that appear in more documents on a logarithmic scale. The Probabilistic Inverse weighting scheme was used since it does not let the weights skyrocket as easily as the number of documents the term is in decreases (“A Semi-Discrete Matrix . . . “, 4).

There are two matrix decompositions that can be used, the Semi Discrete Decomposition (SDD), and the Singular Value Decomposition (SVD). To express this multidimensional space, a vector is used for each term and for each document. The number of dimensions in the space is *k*, and thus the number of indices in the vector is *k* (“A Semi-Discrete Matrix . . . “, 4).

The two matrix decomposition algorithms which can be used for computing the vector space are similar. Both store an approximation of the original term-by-document matrix by computing a lower-rank approximation, which requires less storage space. The final matrix is then approximated by using only *k* vectors of the SDD matrices. The Semi-Discrete Decomposition (SDD) was used. The SDD only approximates the original term-by-document matrix even if *k* = n. The final matrices that approximate the term-by-document matrix have a value of {-1, 0, 1} in each index. The value represents whether that term or the documents depending on which matrix is being used is similar to the to the topic that is encoded into each of the *k* vectors (“A Semi-Discrete Matrix . . . “, 4).

When a user enters a query, the query must be put in the multidimensional space. After the query is parsed for terms, a *k* length vector qt is computed. It is the sum of all the (term vectors * term's global weight) from the terms in the query. Then a vector qd is computed which is the sum of all the document vectors used in the query. Searching with documents enhances a search to bring it closer to the information that is being searched for. In the testing however, It appears that document searches changed the final rankings of each document negligibly. Next, q is computed. It is the multiplication of the Diagonal matrix from the Semi Discrete Decomposition (SDD) multiplied by qd and then added to the vector qt. The SDD matrix is used for weighting the qd if documents are being searched for or weighting qt if terms are being searched for. This gives the vector of the point the query is in the multidimensional space. Now, the cosines between the document vectors and the query vector can be computed. Cosine is used because it returns 1 when two vectors are equal, 0 when they are orthogonal, and -1 when they are in opposite directions. Thus higher cosines mean better similarity between the document and the query (Kowalski, 153).

Latent Semantic Indexing (LSI) does not create the multidimensional space straight from the frequencies of the terms in the documents, but by an algorithm that separates the documents out by content. The terms in a document make up the content of that document, not individually, but the combination of all the terms in the document. Any single term can occur in many different documents over a wide range of topics, but LSI relies on the combination of many terms pointing to the underlying meaning of the page. Thus the documents can be separated out not manually, but using the Semi Discrete Decomposition, or the Singular Value Decomposition algorithm

(“Latent Semantic . . . TREC-2”, 1). For example, a document concerning *legislation* and a document concerning *construction* both may contain the term *house*. And a document concerning *legislation* and a document concerning *money management* may both contain the term *taxes*. However, combining all the terms together in the document, the sum of the terms will end up putting the document in the multidimensional space near documents that have the same terms. So, if one document has eighty percent of the terms the same as the second, the second has eighty percent terms similar to a third, and the third has eighty percent similar to a fourth, the fourth and the first document may have as little as forty percent similar terms, but probably refer to the same topic.

Benefits and Drawbacks

According to Susan Dumais, “ . . . the LSI method has equaled or outperformed standard vector methods and other variants in almost every case, and was as much as 30% better in some cases” (“Latent Semantic . . . TREC-3”, 1). This is due to its main feature, which is that it clumps similar topics together based on the terms in those documents.

LSI has many other benefits as well. Since it uses a vector space model rather than a lexical comparison, documents and terms are both placed in the same medium. This way, relevance feedback, or using documents to search rather than terms or in addition to terms, is possible, and it is just as simple as using only terms. Also, terms rather than documents can be ranked and returned. Thus, one can search and find terms that are similar to a term, or find terms that characterize a particular document (“Using Latent Semantic . . . “, 1).

LSI has its drawbacks as well. Location of terms in a document is not taken into consideration. A document that has two separate topics on it, like a newspaper homepage, will get a vector encoding in between the two topics, and will not rank very high if a user searches for either one or the other topic. A second drawback is that LSI may fail to encode documents properly. When testing, returned documents that should not have ranked as high as they did will be a major concern to see if the LSI engine is working properly. If the encoding fails, the document may never be retrieved when a search for that topic is executed.

Another problem is the size. The sparse matrix for my 5838 term to 619 document collection of frequencies would require .05 MB, and the local weights file would require .08 MB. The LSI encoding needs global term weights, stored as four byte floating points, a 5838 by 300 term 1.5 bit vector representation, a 619 by 300 document 1.5 bit vector representation, and a 300 four byte vector for the diagonal matrix S. That comes to almost .6 MB of data that must be loaded and used for a very small collection of 619 documents and 5838 terms. Though .6 MB is not that large, for the internet with approximately 60 million web pages, that size could be over 60 GB, while the lexical search engine could require a sparse matrix of only 12 GB with storing weightings and term location in the document. Since web search engines store these matrices in RAM, the difference between 60 GB and 12 GB is all the more important. Both methods would also require a term and document lookup file, but that was not taken into consideration in sizes since both would require it, and the size would be hard to estimate.

With many Lexical search engines, as opposed to LSI, the user can specify documents that have certain terms, rather than looking for a topic, which finds relevant documents much more easily. The user does not have the tools to increase the number of relevant documents found, but that may not be what the user is looking for. The user may only be looking for a limited number of documents, to find a specific piece of information. The searches do not exploit the power of LSI, and therefore both a lexical search engine and a LSI engine should be available. Term proximity is another feature LSI cannot do; however, LSI overcomes the need for term proximity, assuming the user enters a good number of terms in the query. LSI searches work far better when more terms are entered. A LSI search with one term is unlikely to find much relevant information, or to divide the rankings into different topics very well.

Also, instead of getting a ranked cosine between the query vector and document vectors, Lexical searches can return a rank based on the number of terms from the query were found in each of the documents. Users are better able to understand how the Lexical rank calculated, and do not wonder why a document was ranked high even if it is irrelevant since it contains terms from the query. The major feature of LSI can also be considered as a drawback to many users. When irrelevant documents are ranked high, they sometimes don't have key terms, making the user wonder why a document was returned. They also will have trouble reforming their query as easily to compensate for their shortcomings or the shortcomings of the algorithm. In this small sample of documents or terms, however, irrelevant documents always had key terms, which made it easy to understand why those documents were ranked high. Despite their irrelevance, these documents were the basis for the vector encoding for the terms. With a small database and few relevant documents that have many of these terms, non-relevant documents can throw off the vector encoding for these terms, and bring them closer to those non-relevant documents. In larger databases, the non-relevant documents will cancel out other non-relevant documents, making the impact of the non-relevant documents negligible, and the centroid for each topic will center closer to the center of the relevant documents.

Results

Document Searching

In order to test the Semi Discrete Decomposition (SDD), a Lexical-based search algorithm to test the results from the SDD algorithm was written. The Lexical-based search algorithm used mainly data and functions from the SDD process. For my comparison data, local weights were used, which was a logarithmic calculation on the frequencies. When a query is entered, the lexical search engine creates an empty query vector. The vector has one index for each term, and then it puts a one in the index for each term entered in the query. Then it simply does a cosine calculation from the rank class to compare the query vector and the document vector from the local weights. Thus the Lexical search engine uses the cosine as its ranking, just like the SDD algorithm. This allows for an easy comparison between the rankings of each algorithm, since they both have the same range, one to negative one, as their output, and the numbers basically mean the same thing. The SDD engine allows for using documents in the query, or ranking terms as its output. However, creating a Lexical search engine that does that is another project, and thus those search features will not be compared to the Lexical search engine, but will have to be tested independently later.

For testing, *k*s of 50, 100, 200, and 300 were used to have a large range of dimensions in the multi-dimensional space, and to compare them with each other and the lexical search engine. Comparing the first ten documents from each search and the location of all the known relevant documents to rank each search result. Relevant documents are located by scanning all the titles in the documents and then reading documents that sounded like they may be relevant, and by reading the first twenty documents returned from each search.

For the first search, documents already known to be in the database were searched for. The search was for information about current statistics on house construction. Specifically, the query entered was, "I want statistics for current house construction."

	Lexical	SDD			
		300	200	100	50
1	X		X	X	/
2	X	X		X	
3	X	X	X	X	
4	X	X	X	X	
5	X	/	X		
6				X	
7	X		X	/	
8		/	/	X	
9		/	X	X	
10		/	X		/
X - relevant, / - related, - non-relevant					

With a *k* of 300, the second document described the value of all the *residential*, nonresidential, public, and private construction currently being undertaken. The SDD's main purpose is to find documents that not only have terms that were in the query, but also to find documents that are relevant and have terms not used in the query. This document never used the term *house* but used *residential*. It was relevant and came near the top of the query. The third and fourth document had statistics on current housing construction, and seven of the next thirteen documents had topics about construction. No relevant documents were found past the sixteenth ranked document. Of the remaining thirteen documents, seven were about construction; and the seventh, thirteenth, fifteenth and sixteenth were also reports on current construction of houses. Six documents in the top ten were not about housing construction. Document 5 was about declining military budgets; it talked a little about construction but never mentioned the term. Document 6 was an economic definition and example for marginal analysis. It has nothing to do with housing construction or anything close to it. Document 8 was about funding of research projects. Often the term *project* is associated with *construction*, so it is understandable that this document was returned. Document 9 was about funding and construction at Schools that do not have Doctoral programs, while document 10 was about construction projects at schools that are started later than needed.

With a *k* of 200, the relevant documents actually ranked higher. It is possible that this is due to fewer dimensions that separate the query and the document. The lower *k* would raise not only the similarity measure of relevant documents, but also the similarity measure of all documents, which would be sort of like inflation. Thus, actual similarity measure between documents with different *k*s is not important, but the order in which the documents appear is. The document on the value of residential, non-residential, public, and private construction came to the top. The document that came first in the search with a *k* of 300 came second in this listing. The document was about legislation the House of Representatives is undertaking, and had nothing to do with house construction. All seven documents were found in the top ten. The remaining three non-relevant documents in the top ten were the first three non relevant documents on the search with a *k* of 300.

With a *k* of 100, all seven relevant documents came in the top nine. The remaining three documents in the top ten included a document that was about legislation the House of Representatives was passing for construction. Both House and construction were mentioned several times, so it is interesting that the document was unrelated despite having key terms and failed to appear in searches with *k*s of 200 and 300. It came as document number 18 with a *k* of 300, and 13 with a *k* of 200. This is the other benefit of Latent Semantic Indexing; besides finding relevant documents without key terms, but similar terms, it weeds out non-relevant documents with key terms. The document ranked quite high comparatively, ranking eighteenth and thirteenth out of 619; but comparatively to documents with key terms, it ranked very low. Most of the documents above it mentioned the term *house*, but quite a few did not mention construction, but mentioned the term *repair* and ranked higher. The two remaining documents concerned research funding of projects and construction of buildings at schools.

With a *k* of only 50, none of the seven relevant documents came in the top 10. They were ranked from sixteenth to twenty third, all bunched together. The relevant documents must have been very close together in the *k*-dimensional space. As the *k* got smaller, the space between the relevant documents got smaller much quicker than the space between relevant documents and non-relevant documents. The lower *k*s except for 50 separated out the documents into better groups than the higher *k*s. This could be due to larger clusters of documents, or it could be because abstracting the search works better sometimes, and other times a more literal search is needed. The top ten documents on this search were widely varied. Many of the documents were from the top tens of previous searches, but a few new ones appeared. Some of the other topics from the documents were animal research and testing, an appendix to a government research project or paper, and a document about construction on historically black schools.

With the Lexical search engine, six relevant documents appeared in the top seven, and the seventh relevant document, which only mentioned the term *house* once, and used *residential* all the other times, was ranked eighteenth. Obviously, all the documents up until the eighteenth had the terms from the query, but the LSI searches did far better because they could weed out unimportant terms with weights, while the Lexical search did not have term weights. The terms that were found in the query by the searches were *want*, *statistics*, *current*, *house*, and *construction*. The term *want* can easily throw a Lexical search engine since it gives no meaning to a document, while the term *current* can have several meanings itself.

The search with a *k* of 100 and 200 were about the same and the best, followed by a *k* of 300 and the Lexical search being about the same, and worst of all was the search with a *k* of 50. The search with a *k* of 50, did not fail by bunching too many documents together, but for some reason was not able to exactly locate the relevant documents in the space. The *k* of 100 and 200 were very effective in finding relevant documents and weeding out non-relevant ones.

For the second test search, the sentences, "I want some information on hotels and other lodging. Specifically, I would like to know the different hotel franchises and the size of their hotels in rooms." The resulting searches used the terms *want*, *information*, *hotels*, *lodging*, *specifically*, *size*, *hotels*, and *rooms*. The terms used in this search are far from vague. All the searches should be able to find all the relevant documents. What will make the difference is where the documents are ranked in the output.

	Lexical	SDD			
		300	200	100	50
1	X	X	X	X	X
2		X	X	X	X
3					
4		/	/		/
5		/			/
6		/			
7			/	/	/
8			/	/	/
9					/
10		/		/	
X - relevant, / - related, - non-relevant					

With a *k* of 300, 200, 100, and 50, the first two documents were relevant, giving statistics on hotels. The third document was the exact same document as the second, so it was left out of the results. Those were the only relevant documents in the database. The Lexical search engine returned only one of the two relevant documents near the top, and the other relevant document was ranked at eighty second. The document used very few of the search terms causing it to get a low score. Surprisingly, the LSI searches turned up the document very easily. The lack of many documents in the collection covering the topic that was searched for should have hampered LSI's ability to group relevant documents near each other, but it had no trouble.

For my third search, a search that contained terms that could have led the LSI engine to think the query meant two different topics using polysemic terms, or words that have more than one meaning, would have been good. Since the LSI engine uses neither grammar or term placement for searching, documents from both topics would have ranked high. In the vector space, the two topics may not overlap, but would be close. An ambiguous search using terms that were in both topics would put the query in between the two topics and return documents from each. Unfortunately, with such a small collection, two documents that had different topics but shared at least three good terms between them could not be located.

Term Searching

Looking into what is placed where in the vector space is a good way to tell how well LSI separated the documents and terms. Searching for documents with other documents, or searching for terms with other terms lets the user know what the LSI engine considers similar for documents or terms. If the LSI algorithm failed to put similar terms next to each other or similar documents next to each other, then the whole LSI algorithm has failed to encode the terms adequately. Terms are deemed similar if they are in many of the same documents. Documents are deemed similar if they have many of the same terms within them. Inversely, terms that never occur in the same document or in any set of documents that have similar terms, will be far apart in the vector space and one will get a low score if the user searches with the other. The same is true with documents. Documents that don't have the same terms, nor have similar terms with documents that do have similar terms to each other, will not be close together. Though a *k* of one to three hundred can be used, using a *k* space of 300 to compare the full encoding of the vector is sufficient for testing.

Searching on the term *house* should put the term *representatives* high on the rankings. Representatives ranked 189th out of 5838 terms with a relevance score of only 0.23 on a scale of –1 to 1. That is not as high as it ought to have ranked, but it is understandable that a document could easily mention the term *house* and not the term *representatives* and vice versa. Terms that did rank high were *hall*, *albany*, *maresi*, *branch*, *alison*, *root*, *avenue*, and a myriad of others with ranks around 0.8. Only one document in the database contained both terms, which makes the results even more confusing. However, if the terms are in documents that have similar terms, then the two terms will be put closer together in the document space.

Using the term *disabled* for the query, the term *handicap* should rank high. A single document would probably only use one of the terms, but the documents containing either of the terms would have many other terms in common. Unfortunately, *handicap* was judged as non-relevant by the LSI engine, since its rank was 0.0. The term *disabilities* was 23rd with a rank of 0.56. However, *disabling* was given a rank of 0.00, and *disability* was given a rank of –0.12. My only conclusion is that with a small database synonyms usually do not occur in the same document since the authors of the documents usually chose to stick with one. Beyond that chance of getting similar encodings for the terms, there was not enough relevant documents in the database to get the two vector encodings similar. Terms that did rank high, excluding names, were *underparticipation*, *talent*, *youth*, *corporation*, *lost*, *population*, and *opportunities*. It turns out that several documents contained all or almost all of the terms, and the terms really could easily be put into sentences or phrases that would be found in documents with topics concerning the *disabled*.

LSI encoding failures

Though relevant documents are appearing in each of the searches, often documents that are highly ranked will have a different topic than desired. Trying to get those documents to come up on with a query for their topic will show whether or not these documents were correctly encoded by LSI.

Eight different queries were conducted. Some queries had few relevant documents, and some even had only a few documents containing query terms. Other searches had five to ten relevant documents, and as many as twenty to thirty documents had terms from the query. However, not one of the searches turned up a document that was neither relevant nor had a term from the query in it with a rank greater than 0.25. The query that returned non-relevant documents and documents not containing terms from the query actually had no relevant documents in the database. Interestingly, LSI failed and it is easy to see why. The query used was *birth control*. Both terms were in the LSI terms list and thus were used for searching. Of the first three documents returned, two had topics concerned with immigration, and both had the term *birth* and *control* throughout the document. The third document had the term *control* in it three times. The fourth document had a rank of about 0.24, and was about with immigration. Since no documents in the database concerned birth control, the LSI algorithm was unable to create a subspace in the space for those documents. The encoding for the terms *birth* and *control* are determined by the documents that contain them. Thus documents containing those terms and documents similar to those documents were returned. On a larger database containing documents for a wider variety of topics, this would not have been a problem. Unfortunately, this is a problem with the automated indexing that has no real solution. Since the encoding for *birth control* was made with documents unrelated to the phrase *birth control*, the LSI engine will then return those documents with a high rank despite the fact that they are not relevant. The documents would have had an even higher rank except, 45 documents contained either the term *birth* or the term *control* and two contained both terms, moving encoding for *birth control* in between them.

Even more interesting is the fact that the LSI rankings for the two documents were actually higher than the lexical rankings. This is probably due to the programming of the lexical engine. It reused the Cosine code and took the cosine between the query vector and the document vector full of all the terms, instead of returning documents that contained the largest number of terms. The Lexical engine therefore is not the traditional one, but it does serve as a good benchmark.

Windows

The software for the whole LSI process was written in C programming language for the DOS operating system. Much of the LSI software uses common code, and will compile and run on a UNIX computer, but portability was not an issue in this work. The software can be run on any computer with DOS. Currently, for convenience of file tracking, DOS must have Windows running for long filename support. This is the only reason Windows is needed, and the software can easily be changed to allow the 8.3

filename format or the long filename format. If any user would like to test the software on any database, the database must first be formatted in a certain way. The documents must be saved into a single file separated by blank lines called **combined.docs**. A file named **combined.terms** must have all eligible terms in it, one per line, and a file named **combined.titles** must be supplied with only one title per line. Each executable prints out what arguments it needs to run when it is run with the wrong number of arguments or bad arguments. First, **Sparse.exe** will search through the **combined.docs** file and create a document by term matrix. That matrix must be transposed with **transpose.exe** to create a term by document matrix. Next, local and global weights, and the A matrix are created with **weights.exe**. Once again, the local weights must be transposed for the next step. Then, the SDD algorithm is run from the **SDD.exe** file. It creates the *k* by document and *k* by term decomposition matrices, and the *k* weight vector. Both matrices must be transposed. Then **Parser.exe** can be run, which outputs the ranked scores of the terms or documents. The Lexical engine uses the same data, and is just an optional toggle input to the **Parser.exe**. The actual software must be changed to reflect the number of documents and terms in the database and the maximum k to be calculated.

Conclusion

The LSI vector space worked very well to classify documents and search for them. On almost every search, relevant documents were all at the top, and it worked well or better than the Lexical searches. The search for relevant terms from other terms was perplexing. The terms returned from searching did not seem to be related.

However, judging from the documents the terms were paired in, it seems the algorithm for placing terms in the vector space did not fail.

The LSI engine may have worked better on this database than other databases due to the small and diverse topics the documents had. With a huge vector space of 300, and only 619 documents, the documents were spread out very well, and there were many distinct topics in the document database. In the future, a larger document database with less distinct documents would allow a better test of LSI. With many documents having different topics in one subject, there would be many irrelevant and relevant documents containing terms from the query. Also using documents to search for similar documents would be more meaningful. In this database, so few documents were relevant, that they were all highly ranked, without any need for using documents to help the search.

Sources Cited

Deerwester, Scott, Dumais, Susan T., Furnas, George W., Landauer, Thomas K., Harshman, Richard R. "Indexing by Latent Semantic Analysis". Downloaded from http://zeus.polsl.gliwice.pl/agrest-cgi/plconv/isoHnopl/superbook .bellcore.com/~std/LSI.papers.html, October 1997. Journal of the Society for InformationScience, 41(6), 391-407.

Dumais, Susan T. "Latent Semanti Indexing (LSI): TREC-2". Downloaded from http://zeus.polsl.gliwice.pl/agrest-cgi/plconv/isoHnopl/superbook.bellcore.com /~std/LSI.papers.html, October 1997. In: D. Harman (Ed.), The Second Text Retrieval Conference (TREC2), National Institute of Standards and Technology Special Publication 500-215 , pp. 105-116.

Dumais, Susan T., Furnas, George W., Landauer, Thomas K., Deerwester , Scott, Harshman, Richard. "Using Latent Semantic Indexing Analysis to Improve Access to Textual Information". Downloaded from http://zeus.polsl.gliwice.pl/agrest-cgi/plconv/isoHnopl/superbook.bellcore.com/~std/LSI.papers.html, October 1997. In Proceedings of CHI'88: Conference on Human Factors in Computing, New York: ACM, 281-285.

Kowalski, Gerald. Information Retrieval Systems. Kluwer Academic Publishers, Boston, MA. Ed. W. Bruce Croft, 1997.

Kolda, Tamara G., O'Leary, Dianne P.. "A Semi-Discrete Matrix Decomposition for Latent Semantic Indexing in Information Retrieval." Tech Report: UMCP_CSD CS-TR-3724. December 5, 1996.