H. Kettani and G. B. Newby, "On the ranking of text documents from large corpuses," proceedings of the 2009 International Conference on Data Mining (DMIN'09), Las Vegas, Nevada, July 2009

On the Ranking of Text Documents from Large Corpuses

Houssain Kettani[1] and Gregory B. Newby[2]
[1]ECECS Department, Polytechnic University of Puerto Rico, San Juan, Puerto Rico, USA
[2]Arctic Region Supercomputing Center, University of Alaska, Fairbanks, Alaska, USA

Abstract - *Ranking text documents based on their relevance to a topic is of great importance in information retrieval. However, giving the increasingly available avalanche of digital documents, the size of collection pool from which these documents are drawn makes this task more challenging. In addition, current computing infrastructure is unable to deal with very large corpuses directly. Thus, new algorithms are needed to seek parallel solutions and utilize more processing power to solve this problem. In this paper we propose a new algorithm that partitions a large collection of documents (a corpus) into smaller corpuses that can each be handled by a single processor for the purpose of ranking. These multiple rankings are then merged together to provide a unified listing of all selected documents from the original large corpus.*

Keywords: Text documents ranking, large corpus, latent semantic indexing, algorithms.

1 Introduction

Searching large collections of documents is challenging for computational, practical and humanistic reasons. The great success of today's online search engines has allowed us to overlook many of these challenges, since results are, generally, satisfying. Yet, only a small fraction of available electronic data is available through such search engines [8, 12]. From a practical standpoint, we would like to make increased numbers of items of all types (Web pages and other documents) available for search. In some cases, this will require interacting directly with a search service on a specific site, rather than a centralized aggregation of content (as seen in Google & others). Such distributed search might offer the only method for overcoming security and accessibility concerns for access to specific data sets [5].

If practical considerations for distributed search were overcome, there are still computational and humanistic concerns remaining. Humanistically, consider that the role of search systems – more generally called information retrieval or IR systems – is to provide a ranked list of the most relevant documents. This list is created based on matching the human expression of information need – the query – to a set of documents. The set of documents can be very large, as we see with today's Web search engines. The challenge here is that relevance is a difficult concept [10]. The list of relevant documents for a query is situational, and for a particular human information seeker the relevance of a given document will depend on what he or she is really looking for, as well as prior knowledge and expertise in the subject area. Expecting an IR system to make an effective ranked list of documents based on just a few query words is ambitious, indeed! Successful search engines today often present a variety of different results in the first set of 10 or 20 documents, so that different information seekers can quickly find high-quality documents of interest in the list. Below, we present latent semantic indexing (LSI) as one approach to better match the conceptual meaning of a query against the concepts found in a document, rather than being solely reliant on the (few) terms of a query, and the often ambiguous language used in queries and documents [2].

Computational challenges of search have, to some extent, been addressed through heroic engineering undertaken by database and Web search engine providers, among others. Being able to respond to a query in only a second or so is testament to the ability to rapidly produce results by parallelizing search across a single large corpus. In this paper, we consider the computational challenges brought about by distributed search and the merger of results from multiple collections, which has not yet been the subject of intense engineering efforts. We anticipate that the practical need to search across data sets, along with the humanistic need to get search results which are more closely targeted for individual information needs, will provide impetus to address such problems.

In this paper, we describe the LSI approach to information retrieval, and discuss some new approaches to efficiently build representations of queries, documents and document collections which attempt to overcome some of the limitations of ambiguity in human language. The basic LSI approach has been known, and demonstrated successfully, for years, but computational and practical challenges to its application have limited its use for large-scale Web search. By presenting effective algorithms for applying LSI across distributed data sets, we hope to provide a base for increasingly effective search across readily available Web-based content, as well as for content which is not currently available to general-purpose search engine harvesters.

This work was supported in part by the Arctic Region Supercomputing Center at the University of Alaska – Fairbanks, with additional support from the US Department of Defense High Performance Computing Modernization Program Office under contract G00003435 and others. This work was also supported in part by the United States Nuclear Regulatory Commission under grant number NRC-27-09-310.

2 Background

2.1 Vector space model

When addressing the issue of document ranking, it is much more convenient to reduce it to a mathematical problem. Thus, [9] introduced a vector space model (VSM) where keywords are the axis and documents are points in this multidimensional space. For example, if we have two documents $\mathbf{d_1}$ and $\mathbf{d_2}$, the keywords are the set {cat, dog}, the word "cat" occurs five and seven times in $\mathbf{d_1}$ and $\mathbf{d_2}$, respectively, and the word "dog" occurs two and three times in $\mathbf{d_1}$ and $\mathbf{d_2}$, respectively, then $\mathbf{d_1}$ and $\mathbf{d_2}$ are represented by the points (5, 2) and (7, 3), respectively, and the query vector could be (0, 1), (1, 0), or (1, 1). A pictorial illustration of this vector space model is given in Figure 1.

The ranking of the documents is achieved by calculating the distance between the documents and the query. The closer a document to a query is, the higher its rank. Alternatively, since calculating distances is time consuming, we can use the angle θ_i between each document and the query as a measure for ranking. More precisely, we can use the cosine of this angle for the purpose of ranking. Thus, we can define,

$$\alpha_i = \cos\theta_i = \frac{\mathbf{q}^T \mathbf{d}_i}{\|\mathbf{q}\| \|\mathbf{d}_i\|}.$$

Furthermore, if the vectors are normalized, this measure can be reduced to $\alpha_i = \mathbf{q}^T\mathbf{d_i}$. Thus, if all vectors are normalized, and since all the points lie in the nonnegative hypercube, minimizing the distance between $\mathbf{d_i}$ and $\mathbf{q}$ is equivalent to maximizing α_i, which is a simple dot product between the query and the corresponding document.

2.2 Latent semantic indexing

The vector space model is an excellent approach; however, it fails to address synonymy and polysemy of words in a language like "car" vs. "automobile," or like "can" in "I can" vs. "can" in "tuna can." Thus, [2] introduces latent semantic indexing (LSI) algorithm as an add-on to the vector space model, which uses singular value decomposition (SVD) to reduce redundancy in data. Accordingly, [2] points out that a corpus can be presented as a matrix $\mathbf{C}$ with non-negative integer entries representing the number of occurrence of terms in each document. The columns of $\mathbf{C}$ represent documents and its rows represent the terms as follows:

$$\mathbf{C} = \overbrace{\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & \ddots & & \\ \vdots & & \ddots & \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{bmatrix}}^{m \text{ documents}} \Big\} n \text{ terms}$$

In our case, m is in order of millions, while n is in order of hundreds of thousands. Latent Semantic Indexing suggests first obtaining the corresponding singular value decomposition as follows:

$$\mathbf{C} = \mathbf{USV}^T =$$

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ u_{21} & \ddots & & \\ \vdots & & \ddots & \\ u_{n1} & u_{n2} & \cdots & u_{nm} \end{bmatrix} \begin{bmatrix} s_1 & & & 0 \\ & s_2 & & \\ 0 & & \ddots & \\ & & & s_m \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1m} \\ v_{21} & \ddots & & \\ \vdots & & \ddots & \\ v_{m1} & v_{n2} & \cdots & v_{mm} \end{bmatrix}^T$$

Then, a rank reduction is obtained by zeroing out the singular values s_i such that $s_i << s_1 \ \forall\ i \geq k$, which introduces the following approximation of the term-by-document matrix $\mathbf{C}$:

$$\hat{\mathbf{C}} = \hat{\mathbf{U}}\hat{\mathbf{S}}\hat{\mathbf{V}}^T =$$

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1k} \\ u_{21} & \ddots & & \\ \vdots & & \ddots & \\ u_{n1} & u_{n2} & \cdots & u_{nk} \end{bmatrix} \begin{bmatrix} s_1 & & & 0 \\ & s_2 & & \\ 0 & & \ddots & \\ & & & s_k \end{bmatrix} \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1k} \\ v_{21} & \ddots & & \\ \vdots & & \ddots & \\ v_{m1} & v_{n2} & \cdots & v_{mk} \end{bmatrix}^T$$

Then, after normalizing the columns of $\hat{\mathbf{C}}$, the Vector Space Model can be applied.

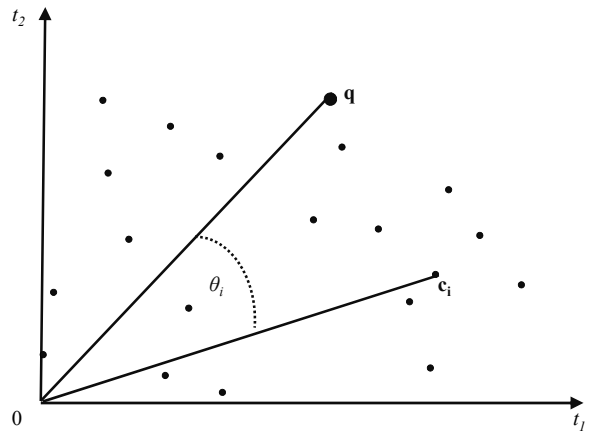

Figure 1. Pictorial illustration of the vector space model of document ranking, where $\{t_1, t_2\}$ is the set of keywords, $\boldsymbol{q}$ is the query, other points are the documents, θ_i is the angle between a document and the query.

2.3 Time complexity

The singular value decomposition of the n × m matrix $\mathbf{C}$ requires $O(nm^2 + n^2m + n^3)$ multiplications [13]. The multiplication of an $n \times m$ matrix $\mathbf{A}$ and an $n \times p$ matrix $\mathbf{B}$ requires $O(nmp)$ [13]. Thus, obtaining the matrix $\hat{\mathbf{C}}$ requires $O(\max\{nk^2, nkm\}) = O(nkm)$, since $k < m$. On the other hand, obtaining each α_i requires $O(n)$, and therefore, obtaining the score vector requires $O(n^2)$, which is the time complexity of VSM. This can further be reduced to O(n) through parallelization. Since LSI consists mainly of the aforementioned three parts, then it can readily be seen that SVD controls the complexity of LSI. Accordingly, the time complexity of LSI algorithm is in $O(nm^2 + n^2m + n^3)$.

2.4 Practical challenges

For a very large corpus, LSI is impossible to implement with current speed of processors and memory constraint. For example, the TREC Million Query Track [11], results in about 600,000 terms by about 25 million document matrix **C**. Storing this matrix is doable, since it is very sparse with about 99.99% of its elements are zero. However, applying SVD to it is even beyond the capabilities of today's large supercomputers. Thus, many researchers realized that this challenge can only be tackled with new high performance computing algorithms and infrastructure [6, 7]. This challenge is the prime motivation for this paper.

3 Proposed algorithm

Since one processor cannot perform LSI on the large corpus (due both to processing speed and the need for very large memory), we can divide the corpus to sub-corpuses and send them to different processers, then rank the documents in each sub-corpus. We then merge the ranking obtained from each processor to obtain the result to the original document ranking problem. An itemized description of the proposed algorithm is as follows. We start by using SVM to represent the query and corpus by letting:

1. **q** represent an $n \times 1$ query vector with unit norm, and
2. **C** represent an $n \times m$ term by document matrix.

We then apply LSI as follows:

3. $\mathbf{C} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathbf{T}}$ is SVD of **C**,
4. Obtain $\hat{\mathbf{S}}$ by keeping the first k diagonal elements of **S**,
5. Adjust **U** and **V** accordingly to get $\hat{\mathbf{U}}$ and $\hat{\mathbf{V}}$, by keeping the first k columns of each,
6. Approximate **C** by $\hat{\mathbf{C}} = \hat{\mathbf{U}}\hat{\mathbf{S}}\hat{\mathbf{V}}$,
7. Normalize the columns of $\hat{\mathbf{C}} = [\hat{\mathbf{c}}_1, \hat{\mathbf{c}}_2, \dots, \hat{\mathbf{c}}_m]$,
8. Define $\alpha_i = \mathbf{q}^{\mathbf{T}}\hat{\mathbf{c}}_i$,
9. Let $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_n]^{\mathrm{T}}$ be the score vector,
10. The higher the score is, the higher the rank of the document.

However, for a very large in size matrix **C**, Step 3 and/or Step 6 cannot be handled with current computing infrastructure. Thus, suppose we have p processors at our disposal to use. Then,

11. Partition $\mathbf{C} = [\mathbf{C}_1, \mathbf{C}_2, \dots, \mathbf{C}_p]$,
12. Apply LSI to each sub-corpus $\mathbf{C}_i$ to obtain the matrix $\hat{\hat{\mathbf{C}}} = [\hat{\mathbf{C}}_1, \hat{\mathbf{C}}_2, \dots, \hat{\mathbf{C}}_p]$, which is an approximation to $\hat{\mathbf{C}}$.
13. For each $\mathbf{C}_i$, calculate $\boldsymbol{\alpha}_i = [\alpha_{i1}, \alpha_{i2}, \dots, \alpha_{i(m/p)}]^{\mathrm{T}} = \mathbf{q}\hat{\mathbf{C}}_i$,
14. Put $\hat{\boldsymbol{\alpha}} = [\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \dots, \boldsymbol{\alpha}_p]^{\mathrm{T}}$, which is the estimated score vector.
15. Then, as before, the higher the score is, the higher the rank of the document.

4 Performance evaluation issues

4.1 LSI effect on ranking

The proposed algorithm stems from the fact that the order of documents in a corpus should not affect their ranking. Indeed, if SVM is applied alone, then the partitioning does not affect the ranking. However, due to LSI, we generally have $\hat{\hat{\mathbf{C}}} \neq \hat{\mathbf{C}} \neq \mathbf{C}$. Thus, performance evaluation of the proposed algorithm will involve using test data to check the algorithm's robustness to the number of partitions.

4.2 How small should k be?

Singular value decomposition (SVD) produces a diagonal matrix containing singular values of the matrix **C** in descending order. As pointed earlier, in its attempt to approximate **C**, LSI zeroes out the singular values s_i such that $s_i << s_1 \ \forall \ i \geq k$. But when is the condition "$s_i << s_1$" satisfied? In his illustrative example, [2] picked the largest two singular values and zeroed out the other seven. So the authors suppressed all singular values less than 71% of the largest singular value. Others like [3], were more cautious and argued that the condition "$s_i << s_1$" corresponds to s_i less than 10% of s_1. Thus, [3] suggests that "$s_i << s_1$" $\Leftrightarrow$ "$s_i < s_1/10$;" which we refer to here as "the 10% criterion."

4.3 The use of random matrices

To empirically evaluate the performance of the proposed algorithm, it is tempting to use random matrices to simulate the term-by-document matrix **C**. This randomness, however, makes singular values decrease very slowly even if we ensure that the matrix is very sparse. For example, to generate such sparse random matrix in MATLAB, we write the following in the command line "**C** = ceil(r – rand(m, n)," where $0 \leq r \leq 1$, which results in about $100r$% of the elements of **C** being nonzero. The slow decay of singular values will entail keeping most of them. For instance, if the 10% criterion is followed, we will need to keep 75% to 99% of the singular values. This in turn, defeats the rationale behind the use of LSI.

Nonetheless, the structure of human languages makes the content and formation of a text document far from being random. Luckily TREC Million Query Track [11] provides an

excellent, yet challenging, datasets for evaluation purpose. This dataset consists of millions of English documents and queries gathered from government domains. We constructed **C** from sub-collections of this dataset with thousands of terms and documents. More than 99% of the elements of **C** are zero. We observe an exponential decrease in the magnitude of the corresponding singular values. For example, following the 10% criterion, we keep only 0.15% to 0.69% of the singular values. Accordingly, the TREC dataset is used to evaluate the performance of the proposed method in this paper.

4.4 Time complexity

Following the discussion from Section 2.3, the time complexity for performing LSI on each partition $\mathbf{C_i}$ in Step 12 is obtained by substituting m/p for m. Accordingly, each $\mathbf{C_i}$ requires $O(nm^2/p^2 + n^2m/p + n^3)$. Since the p partitions are processed in parallel, the latter is the overall performance of the proposed algorithm. Thus, assuming that $m >> n$, as is the case with TREC data, we can see that the proposed algorithm reduces the complexity of LSI by a factor of p^2, where p is the number of processors. Furthermore, even if the partitions are processed in series, following similar analysis, the complexity of LSI is reduced by a factor of p (the complexity in this case is $O(nm^2/p + n^2m + pn^3)$). Note also that as the number of documents m grows large; LSI may not be handled with the current computing infrastructure. Consequently, for a very large m, the proposed method may be the only way to practically implement LSI.

5 Empirical analysis

5.1 Real data

To evaluate the performance of the proposed algorithm, we apply it to corpuses obtained from the TREC Million Query Track [11]. TREC is the Text REtrieval Conference, held annually at the National Institute for Standards and Technology (NIST) in Gaithersburg, Maryland. TREC is intended to provide a forum for comparison of information retrieval techniques across a variety of test conditions. Different tasks are evaluated as part of "tracks," which typically last for at least a few years before being updated, merged with other tracks, or discontinued. In the past, TREC focused on relatively structured textual data, such as abstracts, news articles and proceedings such as the Federal Register.

The growth of the Web, and interest in Web-based search engines, has resulted in numerous tracks that examined HTML, often harvested from the live Web. These harvests result in fixed test collections, with the largest holding over 20 million unique Web pages from over 6,000 unique hosts. By reusing a test collection for multiple years of TREC experiments, a set of judgments for the relevance of series of queries (known in TREC as topics) against the collection (known as a corpus) can help to build knowledge of the collection, and compare results from one year to other years.

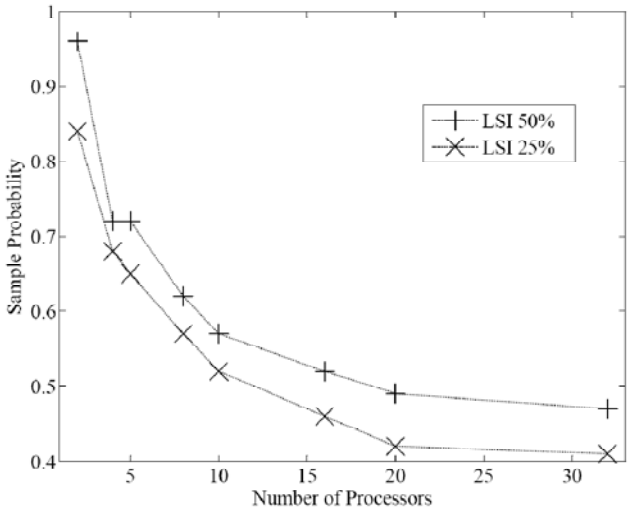

Figure 2. A plot of the sample probability vs. the number of processors using LSI with 25% and 50% of the eigenvalues.

At the Arctic Region Supercomputing Center (ARSC), we have participated in TREC's very large corpus track (VLC), million query track (MQ) and terabyte track (TB) in past years. These tracks have provided tens of thousands of unique TREC topics for evaluation against subsequently larger collections of Web-based documents.

In [4], we used an early approach to the divide and conquer method described in this paper. This followed past experiments based on techniques from [1] and other methods for merging results from separate systems. For the 2008 TREC million query track, our approach was to portray entire collections as vectors in an LSI space. For a given query, we determined the most similar collections, based on the cosine between the collection and query vectors. The fifty collections with the highest cosines were then searched, rather than the whole set of over 6,000 collections. (These collections were simply subsets of the entire 25 million documents in the TREC gov2 collection, in which Web pages from over 6,000 servers in the US .gov space were harvested.)

By applying LSI techniques at the collection level, rather than the document level, we were able to obtain greater efficiency by selecting only those collections deemed to be more highly related to a query. For practical purposes with distributed search, this is important since the incremental cost (in terms of latency and larger sets of documents to rank) for adding more collections can be avoided by only searching collections of greater interest.

5.2 Performance evaluation

From TREC data, we selected 320 documents and 100 queries that consist of at least one word. We then applied LSI algorithm and kept a certain percentage of the singular values. We simulated the case of the number of processors p = 2, 4, 5, 8, 10, 16, 20, and 32. Performance comparison is achieved

through the calculation of the *sample probability* that the top 10% of ranking without partitioning is contained in the top 20% of the ranking with partitioning. This was done for each query and then we took the average over all 100 queries. Ideally, we want this sample probability to be as close to one as possible. We run LSI using 25% and 50% of the total singular values at each stage, which was rounded to the next integer if needed using the ceiling function.

For example, with $p = 2$, and using LSI 25%, we compare the ranking obtained using LSI by keeping 80 singular values and the ranking obtained by using LSI on each partition keeping 25% of the singular values or 40 per each partition. The result of this analysis is depicted in Figure 2. Accordingly, with 25%, the average sample probability is 0.84, 0.68, 0.65, 0.57, 0.52, 0.46, 0.42, and 0.41 for $p = 2$, 4, 5, 8, 10, 16, 20, and 32, respectively. With 50%, we get 0.96, 0.72, 0.72, 0.62, 0.57, 0.52, 0.49, and 0.47. Thus, the average sample probability is inversely proportional with respect to the number of processors. This average, on the other hand, is directly proportional to the percentage of singular values used by LSI.

5.3 Remark

It is clear that if SVM is used without the add-on of LSI, then the sample probability will be one, no matter how many processors are used. The deteriorating performance of the proposed algorithm with respect to the number of processors puts LSI stability into question. This issue will be investigated in more detail in future research.

6 Conclusions and future direction

Motivated by TREC Million Query datasets, and the issue of ranking text documents drawn from a very large corpus, we have developed a divide-and-conquer algorithm implementation of the latent semantic indexing (LSI) algorithm to cope with the current speed of processors and memory constraint. The proposed algorithm reduces the time complexity of LSI by the square of the number of processors used in parallelizing the problem. We presented performance analysis of the proposed algorithm on different queries and corpuses gathered from TREC dataset to assess its performance and stability. This analysis brought the stability and robustness of LSI into question. That is ranking of documents is affected by the number of singular values retained. This should be investigated more in further research. Other study involves checking whether LSI needs to be part of the proposed algorithm and the possibility of involving other algorithms from the literature to replace LSI as a step in the proposed algorithm.

Acknowledgement

The authors wish to acknowledge constructive discussions with Chris Fallen, Kylie McCormick, and Miles Efron.

References

[1] A. L. Calvé and J. Savoy, "Database merging strategy based on logistic regression." Information Processing and Management, 2000, 36(3):341-359.

[2] S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," Journal of the Society for Information Science, 1990, 41(6):391-407.

[3] M. Efron, "Eigenvalue-based model selection during latent semantic indexing," Journal of the American Society for Information Science and Technology, 2005, 56(9):969-988.

[4] C. Fallen, G. B. Newby, and K. McCormick, "Distributed multisearch and resource selection for the TREC Million Query track." Gaithersburg: National Institute of Standards and Technology, 2008.

[5] K. Gamiel, G. B. Newby, and N. Nassar, "Grid information retrieval requirements (GFD-I.027)." Lamont, Illinois: Open Grid Forum, 2003.

[6] E. M. Garzón and I. García, "Solving eigenproblems on multicomputers: two different approaches," International Journal of Computers and Applications, 2004, 26(4):213-222.

[7] M. R. Guarracino, F. Perla, and P. Zanetti, "A parallel block Lanczos algorithm and its implementation for the evaluation of some eigenvalues of large sparse symmetric matrices on multicomputers," International Journal on Applied Mathematics and Computer Science, 2006, 16(2):241-249.

[8] S. Raghavan and H. Garcia-Molina, "Crawling the hidden web," Proceedings of Very Large Databases (VLDB), 2001.

[9] G. Salton, A. Wong, and C. S. Yang, "A Vector Space Model for Automatic Indexing," Communications of the ACM, 1975, 18(11):613-620.

[10] L. Schamber, M. B. Eisernberg, and M. Nilan, "A re-examination of relevance: toward a dynamic, situational definition," Information Processing and Management, 1990, 26(6):755-776.

[11] E. Voorhees, "Overview of TREC 2007," Proceedings of the 16th Annual Text REtrieval Conference. Gaithersburg: National Institute of Standards and Technology, 2007.

[12] J. Xu, H. Chen, Z. Zhou, and J. Qin, "On the topology of the dark web of terrorist groups," Proceedings of the IEEE International Conference on Intelligence and Security Informatics (ISI), 2006.

[13] G. Golub and A. van Loan, "Matrix Computations," Johns Hopkins University Press, 1996.